Clustering through Simulated Annealing using Measurements of Extremity—the SAME Heuristic

by

Brett Allen Phillippe

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Industrial and Manufacturing Systems Engineering

College of Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

2019

Approved by:

Dr. Todd Easton
Major Professor

Abstract

Data clustering is commonly used to analyze data in numerous fields. This thesis proposes a new clustering measure, the extremity measure. The extremity measure uses two components: the extreme distance, which favors larger clusters, and the central distance, which encourages smaller clusters. By balancing these two components, the extremity measure successfully quantifies how good a clustering solution is.

In this thesis, the extreme distance attempts to estimate how extreme a point is from its remaining cluster counterparts by using linear programming. This linear program, called the Extreme Distance Linear Program (EDLP) estimates how close each point is to the convex hull of the remaining cluster points.

This thesis also proposes a new clustering heuristic called the Simulated Annealing using Measurements of Extremity (SAME) heuristic. SAME takes a current clustering solution and applies move, split, and dissolve operations to modify, and hopefully improve, the current clustering solution. SAME probabilistically chooses which of these operations to apply. These operations are based the characteristics of each cluster. SAME uses concepts from simulated annealing to decide whether a tested modification should be accepted.

SAME is applied to various benchmark data sets and tested against an implementation of the *k*-means clustering algorithm, which is the most well-known clustering algorithm. The clustering solutions generated by SAME and *k*-means are compared using the extremity measure and the silhouette index, which is a commonly used clustering measure. In 90% of instances, SAME performs better than the *k*-means algorithm. Thus, SAME is an effective new technique to cluster data.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my major advisor Dr. Todd Easton for advising me and for giving me thought-provoking work throughout my time with him at K-State. Without his wisdom and guidance, I would not be nearly the student I am today.

I would also like to thank Dr. Ashesh Sinha and Dr. William Hsu for serving on my advisory committee. I also thank Dr. Bradley Kramer for being so helpful during my time in the department and for allowing me the opportunity to pursue my master's in Industrial Engineering at K-State.

Lastly, I would like to thank all of the colleagues I have worked with and studied with during my time at K-State. Learning is best done cooperatively and it has been a privilege to work with some very bright minds.

Chapter 1: Introduction

Currently, a huge emphasis is put on tracking everything. Companies, businesses, and governments are very concerned with tracking as much data as possible. This increased emphasis is commonly referred to as the generation of "big data." The data is much larger and more complex than it was even a decade ago. In fact, there is more data moving across the internet in a single second than there was stored throughout the internet just 20 years ago (McAfee, Brynjolfsson, Davenport, Patil, & Barton, 2012).

This huge influx of information has many ramifications. First, more entities are tracking more granular types of data. For instance, a supermarket company that may have tracked its daily aggregated totals, now focuses all the way down to the individual transaction level. Additionally, the attributes or characteristics are more detailed. Companies track a customer's purchase history to keep track of what they buy, how much of it, how often they buy it, etc. With an increase in data complexity, data storage is more challenging. This challenge has led to many data-driven companies to offload their data storage into "the cloud" instead of using their own resources. Cloud computing is a way companies have combatted the increase in data by using pooled resources and huge data warehouses for storage (Mell & Grance, 2009).

Another problem with gathering more data is how difficult it becomes to analyze. Data analytics takes many different forms based on the applications or questions that one is trying to answer. Statistics can be used to create aggregated totals, or to gain confidence in results or inferences (Rice, 2006). Artificial intelligence is used to try to process and gain additional information from new data over time (Nemati, Steiger, Iyer, & Herschel, 2002). SQL or other database resources segment data into reasonably sized chunks of information that are easier to understand (Hellerstein, 1999).

Data clustering is a popular data analytics technique and is the topic of this thesis. The clustering problem takes a collection of things. A clustering solution identifies a set of clusters where each cluster consists of a group of things. The goal of clustering solutions is to group similar things, based on their characteristics, in the same cluster. Additionally, dissimilar things should belong to different clusters. This vague clustering problem definition has resulted in numerous researchers developing many clustering measures along with techniques to find quality solutions.

1.1 Importance and Complexity of Data Clustering

The importance of clustering is its ability to unlock new information. Clustering can be used to make generalizations that were otherwise challenging to make. Additionally, clustering makes inferences and predictions based on knowledge of previous similar data points. This allows for new inputs to be quickly compared to previous data or clusters.

The clustering problem typically requires numerical data as input. However, the input for many clustering problems is things, which are not necessarily numerical. Much research has been done in converting qualitative data to quantitative data (Srnka & Koeszegi, 2007). Once things are converted into numerical values, algorithms can use these as input to find quality clustering solutions.

For example, schools cluster students, which are clearly not numeric values. In kindergarten, students are clustered by class. As the students progress, the class begins to form new clusters based upon math and reading skills. Eventually, formal exams assign numbers to students. Students are then clustered into regular or advanced classes based upon their scores. These classes help the schools tailor the experience and learning for each student. Schools also use clusters to identify which students are having difficulties with specific subject matter and may need extra help (Merceron & Yacef, 2004). By high school, some students are clustered into various advanced placement or honors classes. Once students enter college, the clustering changes

and now the students are self-clustering based upon majors and/or hobbies. Clustering students in schools tends to improve results and numerous other clustering applications show similar positive results.

There are numerous other industries that use data clustering. Clustering has been applied to data streams in data mining (Berkhin, 2006), stock markets (Basalto, Bellotti, De Carlo, Facchi, & Pascazio, 2005), and computer network traffic (Silva, et al., 2013). Data mining is strongly related to "big data" as its goal is to identify trends or patterns within large data sets. Clustering is used in different fashions within image recognition. Initially, it was used to identify if people exist in a picture. Now, clustering is used to recognize specific identities of people within a picture (Wu, Zhang, Hu, & Ji, 2013). Additionally, clustering has also been implemented in different types of market research including tourism (D'Urso, De Giovanni, Disegna, & Massari, 2013) and customer market segmentation (Dolnicar, 2003).

Developing clusters that accurately fit the data is crucial for clustering to actually add value. The numerical complexity makes this a challenge. Imagine you have just 20 data points and want to group them in no more than 3 clusters. There are $\frac{3^{20}+3}{3!}$, over 500 million, unique clustering solution combinations. This number grows rapidly with the amount of data and potential number of clusters. Attempting to find the best clustering solution out of all these combinations is challenging, as complete enumeration of possibilities is too time consuming. Additionally, deciding what criteria to use to compare clustering solutions is challenging. Most clustering measurements use concepts of similarity and dissimilarity in some fashion, which are discussed later.

1.2 Research Motivation

This research's primary motivation involves developing a new clustering strategy based upon polyhedral theory. Only a minimal amount of research on this topic exists. In fact, only one master's thesis combined polyhedral theory and clustering and that research requires solving a quadratic programming problem. Quadratic programs are slow and extending this method to big data is unlikely to be successful. The overriding goal is to develop a fast method to incorporate polyhedral theory and clustering that could be extended to big data. Using linear programming instead of quadratic programming allows for one to rapidly test whether incoming data traffic fits within existing clusters and could have applications in big data.

1.3 Contributions

This thesis has three primary contributions. First, a new clustering measure, called the extremity measure, is presented. The extremity measure evaluates cluster quality based upon two components: the extreme distance and the central distance. The extreme distance estimates a point's contribution to a cluster based on its distance to the convex hull of the remaining cluster's points. One can determine the extreme distance of a single point by solving a simple linear program with a small number of variables and constraints in most cases. The central distance is a measure of how close each point is to the center of its cluster. This score favorably measures oblong shaped, spherical shaped and densely packed clusters.

The second contribution is a new heuristic to create quality clusters named the Simulated Annealing using Measurements of Extremity (SAME) heuristic. SAME uses three types of operations to generate new solutions: moves, splits, and dissolves. These adjustments are implemented within a simulated annealing environment help generate new alternate solutions.

The final significant contribution is coding and implementing SAME in a computational study. When compared to an implementation of the *k*-means algorithm, SAME performed better

in 90% of instances when using the extremity measure as a method for comparison. Additionally, an alternate implementation of SAME attempted to optimize the silhouette index. This alternate implementation also performed better than the *k*-means algorithm's results in 90% of instances.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. The second chapter details the background information required to understand this thesis. First, a formal definition of the cluster problem and the goal of clustering is provided. Next, types of clustering and their applications are presented. Afterwards, a look at measures used to quantify clusters is described. This thesis incorporates concepts of polyhedral theory, linear programming, and integer programming, so a basic understanding of these topics is given. Additionally, an introduction to general purpose heuristics and how heuristics are applied in clustering is discussed. The chapter concludes with a discussion of the quadratic polyhedral clustering algorithm, which is the most similar research to this thesis.

The third chapter explains the concepts of the extremity measure and the SAME heuristic. The chapter begins by describing how the extremity measure is calculated through the extreme distance and central distance. The heuristics subroutines for generating new solutions by moving, splitting, or dissolving clusters are described. The fourth section explains how clusters and adjustments are selected at each iteration in the SAME heuristic. The chapter concludes by describing how simulated annealing is applied to decide whether or not to accept a new clustering solution.

Chapter four gives information on implementation and computational results. This section starts by detailing the software utilized. The challenges of large data sets and upper limits for data set sizes, along with alternate implementations to SAME used in large instances, are provided. The third section details how a selected cluster's operation is chosen based on its attributes. Chapter

four concludes with comparisons to the *k*-means clustering algorithm using the extremity measure and silhouette index

In chapter five, the thesis discusses future research topics. The clustering problem has been approached in different ways, and polyhedral theory is fairly unexplored region. This chapter discusses further ways polyhedral theory can be applied to clustering. Additionally, chapter five discusses how linear programming can be applied so clustering can be used in big data instances.

Chapter 2: Background Information

This chapter details the background information required to understand this thesis. A definition for the clustering problem and the different types of clustering are provided. The second section demonstrates measures used to quantify the quality of a clustering solution. A description of general heuristics and how they have been applied to clustering is shown. In the fourth section a basic understanding of mathematical programming and polyhedral theory are described. The chapter concludes discussing previous common clustering heuristics, including the *k*-means algorithm, mean shift clustering, and the Quadratic Polyhedral Clustering Algorithm.

2.1 The Clustering Problem

Clustering is a common and frequently applied data analysis technique. Clustering things into groups allows for conclusions to be drawn that were otherwise challenging to determine. Most clustering techniques require numerical input.

In some cases, it is challenging to convert other data types, like categorical data, into numerical data (Srnka & Koeszegi, 2007). Recall, the previous school example where educators group students in order to make the experience more beneficial to them and their students. Students take math timed tests to evaluate their speed and accuracy. Books have Lexile levels to quantify a reader's proficiency based on the books they read. Teachers regularly evaluate students on these numbers to re-cluster their classes. Thus, the educational system has taken the complexity of a student and converted him or her into a number based upon a perceived math or reading skill.

This type of "data" conversion is frequently necessary prior to supplying a clustering problem with input data. Once this data is converted, heuristics or algorithms use this numerical data to develop quality clustering solutions. As such, the goal of clustering is to have each cluster homogeneously represent all its points.

This vague definition has led to many different clustering definitions. Here, partitional clustering, fuzzy clustering and hierarchical clustering are briefly presented. Other clustering problems exist, but they are less frequently studied.

2.1.1 Partitional clustering problem

Partitional clustering is the most regularly applied clustering type and is the focus of this thesis. From a set of points, partitional clustering segments the data set into disjoint sets. Each set is a cluster and is supposed to indicate similarity between the points in the cluster. However, each cluster reports no information on the degree of similarity between points in a cluster in partitional clustering.

Formally, given a set X of points $x_1,....,x_n$ in $\mathfrak{R}^m$, i.e. $x_i=(x_{i,1}, x_{i,2},...,x_{i,m}) \in \mathfrak{R}^m$ for $i=1,...,n$. A cluster C is a nonempty subset of X. Every feasible solution to the clustering problem partitions X into clusters. Equivalently, $\mathcal{C}$ is a solution to the clustering problem if and only if $\mathcal{C}$ $=\{C_1,...,C_k\}$ such that $\bigcup_{p=1}^{k} C_p = X$ and $C_p \cap C_q = \varnothing$ for all $p,q \in \{1,...,k\}$ and $p \neq q$. For notational convenience, let $N = \{1, \ldots, n\}$.

To help describe this definition, consider the 23 points in $\mathfrak{R}^2$ given in Table 2.1. Figure 2.1 depicts these points in a two-dimensional plane. A clustering solution $\mathcal{C}=\{C_1,C_2\}$ is $C_1=\{x_1, x_2, x_4, x_5, x_6, x_8, x_9, x_{10}, x_{14}, x_{18}, x_{22}\}$ and $C_2=\{x_3, x_7, x_{11}, x_{12}, x_{13}, x_{15}, x_{16}, x_{17}, x_{19}, x_{20}, x_{21}, x_{23}\}$. Observe that all 23 points are in either C_1 or C_2. These clusters are shown in Figure 2.2. Rather than providing the set notation, clustering solution are graphically depicted by a boundary where all points within the boundary are in that cluster.

Data Point	Coordinates	Data Point	Coordinates
x_1	2,7	x_{13}	9,7
x_2	2,10	x_{14}	9,9
x_3	3,2	x_{15}	10,1
x_4	3,5	x_{16}	10,3
x_5	3,8	x_{17}	10,8
x_6	3,12	x_{18}	10,11
x_7	4,3	x_{19}	11,3
x_8	4,6	x_{20}	11,5
x_9	4,10	x_{21}	11,7
x_{10}	5,7	x_{22}	11,10
x_{11}	8,7	x_{23}	12,8
x_{12}	9,5		

Table 2.1 Numerical Data for Example Data Set

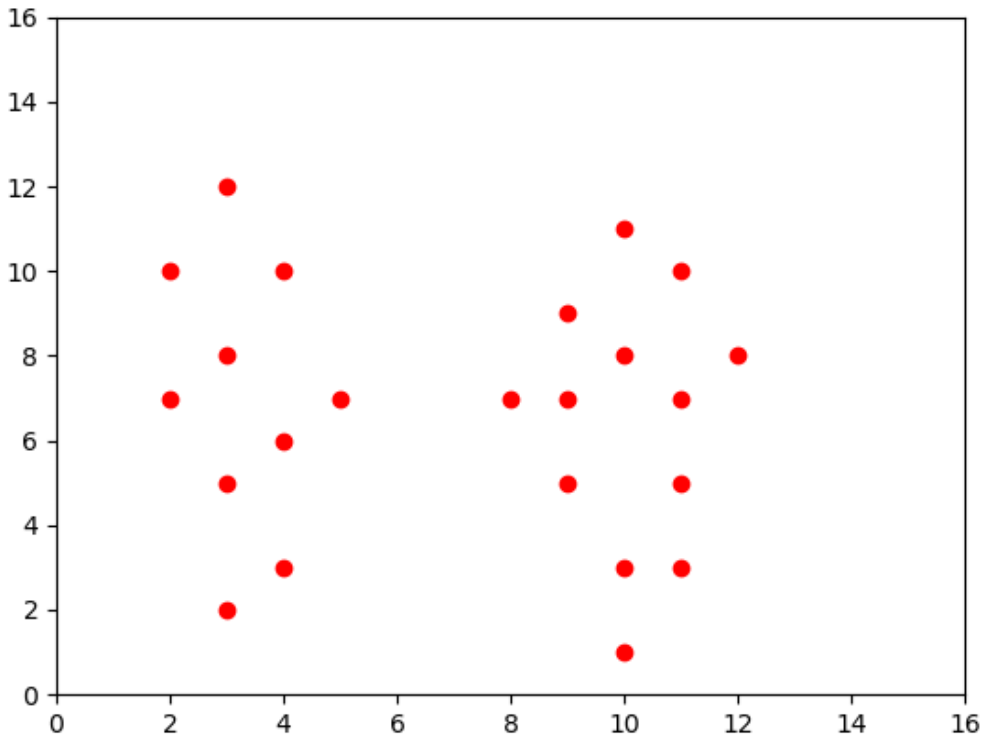

Figure 2.1 Picture of Example Data Set

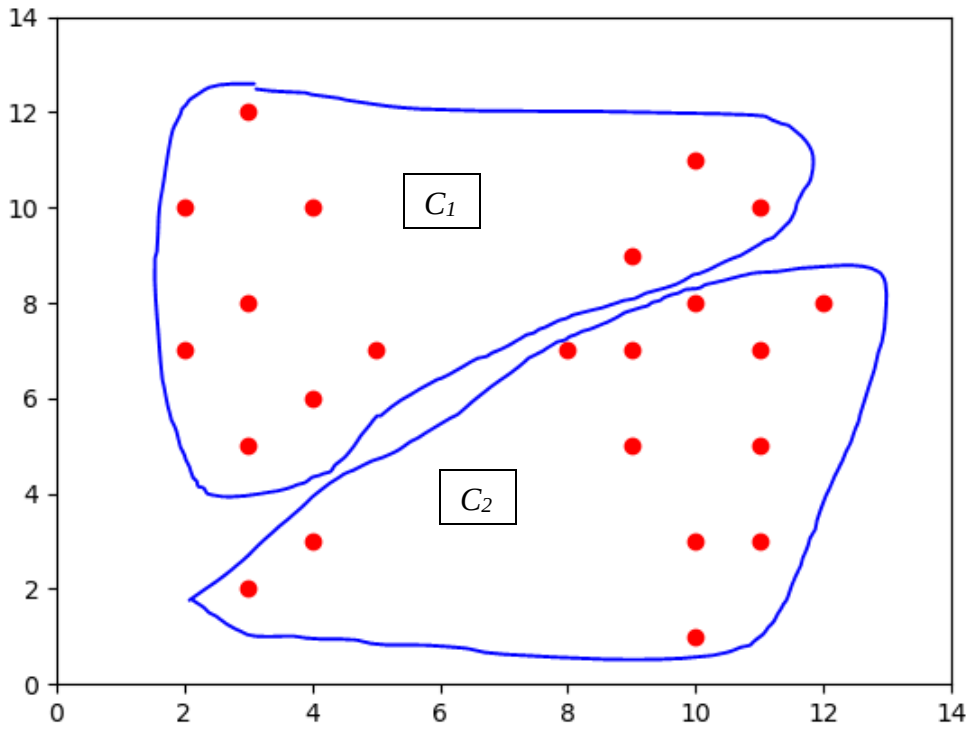

Figure 2.2 Graphical Depiction of Clustering Solution

There are a few reasons that search algorithms for optimizing partitional clusters is challenging. One is not knowing the optimal number of clusters. Another challenge involves the difficulty in comparing the quality of solutions with differing numbers of clusters. Many heuristics require the user to input the number of desired clusters. This is obviously problematic as it can create bias. Even with a set number of clusters, the partitional clustering problem with a simple objective function is an *NP*-hard problem (Aloise, Hansen, & Du Gerad, 2007).

The importance of partitional clustering is held in the applications and inferences made by the clusters. Without good clusters, the decisions made on their basis are ill-informed. Below are examples of the how good partitional clusters were beneficial in various fields.

Many companies use clustering to segment their customers in some fashion (Punj & Stewart, 1983). A study was done in Canada of supermarket customers over a 24 week period. Customers were grouped using self-organizing maps (SOMs) into 5 categories: loyal big

spenders, loyal moderate spenders, semi-loyal moderate spenders, semi-loyal potentially big spenders, and infrequent customers (Lingras, Hogo, Snorek, & West, 2005). These groupings were made based on categories measuring their frequency of visiting the store, how many transactions were made during a trip and categories measuring how much they spent during their visit. Based on these groupings, the supermarket could decide what types of deals, discounts, or advertising were most appropriate. Attrition is a widespread problem in the supermarket industry, with an increasing level of competition and a decrease sense of loyalty from customers. Since retaining customers was found to be cheaper than finding new customers, the goal was to try to reduce attrition of their customer base. To attain this goal, a measure that tries to reduce customers lost to attrition was introduced. With this measure, the company can analyze if the decisions that they are making based on the customer segmentation are actually reducing the customers lost. Over the course of the study, 50 randomly selected customers were monitored and their visiting and spending was more consistent once these clustering groups were organized. Additionally, data could be used to cluster customers for the purpose of predicting future purchases of customers.

Another study looked into quantifying the actual paths that customers took once they entered a supermarket. The goal of this study was to increase the amount of money people spent when they entered the store. This study used radio frequency identification (RFID) technology to track individual shopper paths (Larson, Bradlow, & Fader, 2005). The supermarket cart's locations were recorded every 5 seconds, which became input into determining the route taken through the store. A *k*-mediods clustering algorithm was used to cluster the paths. Note that *k*-mediods is a variation of the *k*-means clustering, which is presented in section 2.5. The clustering solution identified where groups of customers spent different amounts of time in various

categories within the store. With these clusters, a supermarket can modify their placement of products in order to place things along their current paths in an attempt to have the consumer buy new items.

There are many other sectors where partitional clustering has been applied. Partitional clustering has also been used in the image and pattern recognition to cluster pixels that are next to each other in order to distinguish different items within a picture (Matas & Kittler, 1995). Partitional clustering has had a huge impact on data mining, as well (Agrawal, Gehrke, Gunopulos, & Raghavan, 1998). Partitional clustering applications exist in the distribution center problem where one optimally places distribution centers within a network to reduce transportation costs to stores (Picard & Ratliff, 1978). Additionally, clustering has been applied within distribution centers and warehouses to reduce time spent picking orders (Kim, Heragu, Graves, & Onge, 2003).

2.1.2 Fuzzy clustering

Partitioning data sets into clusters, infers that any points that are not in the same cluster bear no similarity to each other. This was not a satisfying result to some researchers as if there exists some arbitrary cut off as to the amount of similarity required to belong to the same cluster. This fundamental issue led to the development of fuzzy clustering introduced by (Zadeh, 1965). The idea was to represent cluster strengths using functions, called membership functions, which evaluate the similarity of each point to the cluster. This also allows for the clustering solution to give insight into how strong the similarity is, which a partitional clustering problem fails to do.

A distinct difference between fuzzy and partitional clustering is that solutions of fuzzy cluster allows data points to be in multiple clusters and also allows the freedom to be in no clusters at all. Usually, these points are assigned weights to each cluster it belongs to define which cluster it is most similar to. Fuzzy clustering algorithms are generally seen as mode-

seeking algorithms, where the clusters find the densest areas in the space. This allows for fuzzy clustering solutions to be transformed into partitional clustering solutions by moving points into the clusters that they most belong. Fuzzy *c*-means clustering algorithms (Bezdek, Ehrlich, & Full, 1984) are a popular fuzzy logic technique for their obvious parallels with the logic of *k*-means clustering, which are be detailed in section 5.

Fuzzy clustering has also been used in the realm of marketing applications to group customers, but allow them to be identified in multiple target markets (Setnes & Kaymak, 2001). Fuzzy *c*-means clustering has been applied to image segmentation, like partitional clustering has (Chuang, Tzeng, Chen, Wu, & Chen, 2006). Fuzzy clustering does a much better job of dealing with pixels that appear to be different than the pixels adjacent to it.

2.1.3 Hierarchical clustering problem

Hierarchical clustering is a clustering technique that focuses on the strength of similarity between data points within a set (Olson, 1995), instead of focusing on creating disjoint sets. Hierarchical clustering uses linking criterion to join data points that are most similar. Hierarchical clustering is generally seen as a recursive method that continually nests clusters. At the lowest level each cluster is a single data point, and at the highest level every point is within one all-encompassing cluster. The solution is generally viewed as a large tree, called a dendrogram. Figure 2.3 shows an example of a dendrogram. Typically the strength is indicated by the level.

Hierarchical clustering has two different subcategories: agglomerative and divisive. Agglomerative is the more commonly used of the two methods and has a bottom up approach. Each point is initially a single point cluster. Iteratively, most similar clusters form larger clusters until all points are in a single cluster. In agglomerative methods, the linking criterion is easier to see. At each step, some linkage score is used to compare each cluster with each other cluster. In

general, the clusters with strongest linkage score between them are paired up. Agglomerative clustering tends to give more accurate results if one seeks to apply the lower-level, granular type results.

Divisive clustering is a top down method, which builds its hierarchy by starting with one large cluster. Iteratively, this cluster is broken down into sub clusters until each point is in a single point cluster. Divisive clustering typically provides better results for larger groups or a high level viewpoint.

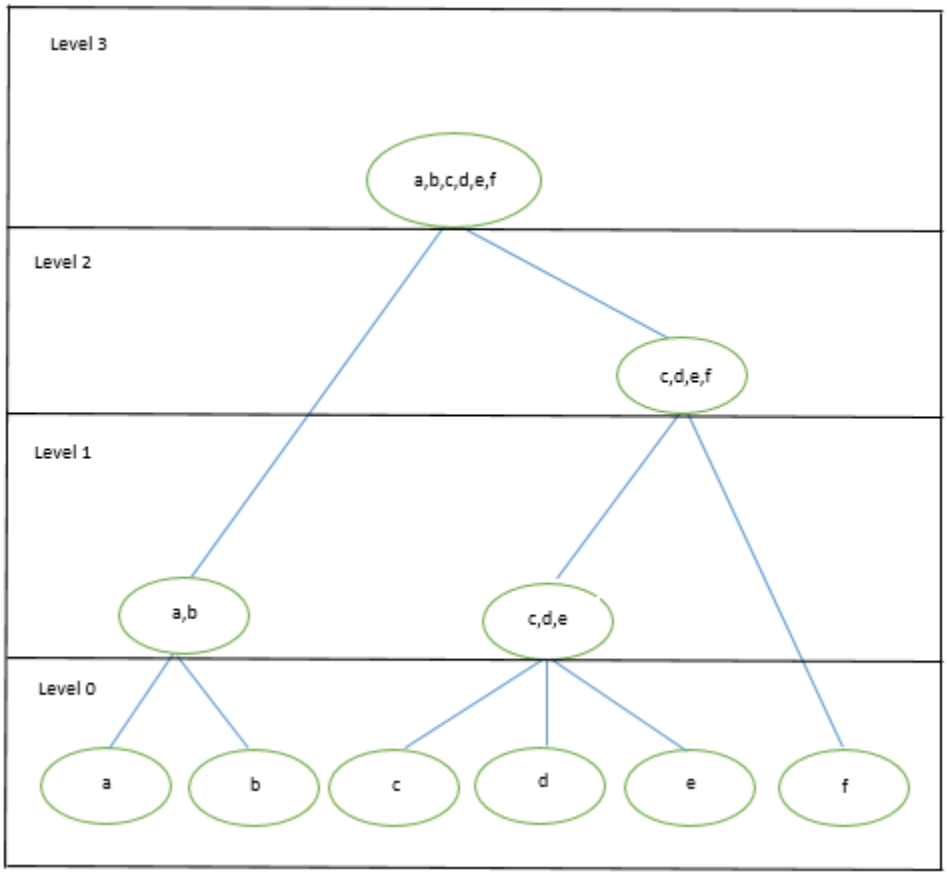

Figure 2.3

Graphical Depiction of Dendrogram

A dendrogram is viewed as the solution to the hierarchical clustering problem. Multiple clustering solutions can be generated from a single dendrogram. One could take a level 1

strength and have $\mathcal{C}$ = {{a, b}, {c, d, e}, {f}} as the clustering solution. A weaker strength level 2 would give $\mathcal{C}$ = {{a, b}, {c, d, e, f}} Thus, a dendrogram creates several different clustering solutions, which are based upon a similarity parameter associated with the depth of the tree.

Agglomerative clustering has been used in microarray/biological fields to cluster data (Chipman & Tibshirani, 2005). These fields use agglomerative clustering because those in the biological field are interested in the outcomes of small changes in samples or sample groups. Hierarchical clustering has also been applied to image segmentation to distinguish items within a picture by building up from the pixel level (Wu & Leahy, 1993). In disaster relief planning, hierarchical clustering has been used to identify where to set up make-shift hospitals or warehouses to reduce time spent picking up injured people or delivering goods to those who are most in need (Özdamar & Demir, 2012).

2.2 Clustering Measures

What is a good clustering solution? This question is the subject of much debate and researchers struggle to agree on a single measure to quantify the quality of clusters. Thus, researchers have created many alternate techniques to measure the quality of differing clustering solutions, which is the focus of this section.

The vast majority of clustering measures are based upon the distance between data points or other points of interest in the space. Even calculating the distance between two items is a subject of debate. However, most distances are typically evaluated using a *p* norm. Given two points, x_i and x_j, the *p* norm is $d_{i,j} = \sqrt[p]{\sum_{v=1}^{m} |x_{i,v} - x_{j,v}|^p}$.

The 2-norm is the most common with $d_{i,j} = \sqrt{\sum_{v=1}^{m} \left(x_{i,v} - x_{j,v}\right)^2}$. This is called Euclidean or straight line distance. This thesis uses Euclidean distance in all calculations. When $p = 1$, $d_{i,j} = \sum_{v=1}^{m} |x_{i,v} - x_{j,v}|$. This is referred to as rectilinear or Manhattan distance. This can

be viewed as how far it would take to travel from one point to the other if space is viewed in square blocks, i.e. one cannot travel in multiple dimensions at once. The infinity norm is $max\{\sum_{v=1}^{m}|x_{i,v}-x_{j,v}|\}$. It emphasizes the greatest difference between two points in any one dimension. Clustering measures frequently leave the selection of distance to the user.

Clustering measures are based on two main schools of thought. The first seeks to achieve similar points in the same cluster. This is commonly referred to as compactness and is frequently measured as the distance between points in the cluster or to the cluster's centroid. A cluster's centroid, $\bar{c_p}$, is defined as $\sum_{x_i \in C_p}\frac{x_i}{|C_p|}$. The second is commonly referred to as separation. Separation measures dissimilarity of points in different clusters. Applying compactness and/or separation leads to the majority of cluster measures. Clustering solutions 1 and 2 in figure 2.4 partition the example data set into two clusters and can be compared through clustering measures.

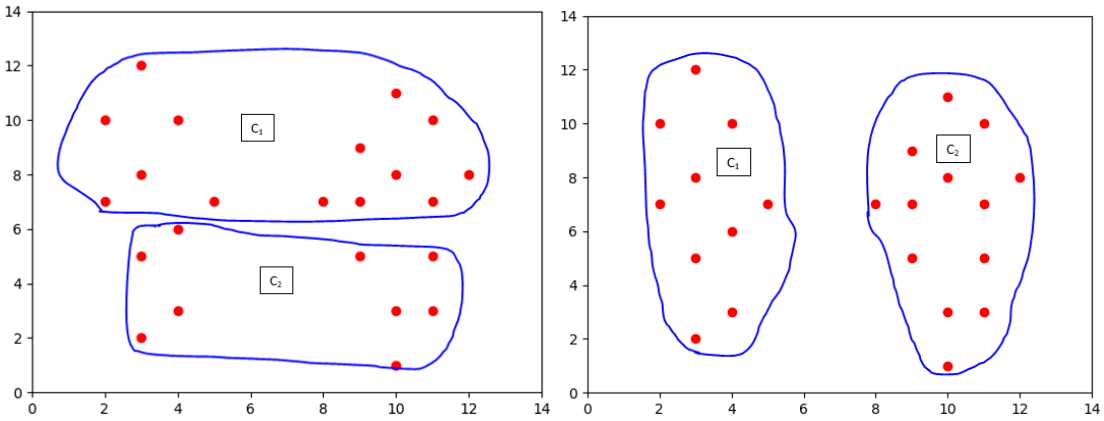

Figure 2.4 Clustering Solutions 1(Left) and 2(Right)

A standard cluster measure is the root-mean-square standard deviation (RMSSTD). RMSSTD only considers compactness. The goal of RMSSTD measures is to reduce the within cluster variance and measure the homogeneity of the formed cluster (Rujasiri & Chomtee, 2009). This measure uses the distances between the cluster's centroid and its data points as a basis to

compare solutions. For this reason, RMSSTD generally favors spherical clusters. Formally, RMSSTD=$((\sum_{p=1}^{k}\sum_{x_i \in C_p}(x_i - \bar{c_p})^2)/\sum_{p=1}^{k}(|C_p| - 1))^{1/2}$. Solution 1's RMSSTD is 4.55 whereas solution 2's RMSSTD is 5.89. Since the goal is to reduce the variance, solution 1 is better with this criteria.

The Davies-Bouldin index is more focused on the separation criteria (Bouldin & Davies, 1979). Each cluster is compared to all other remaining clusters. The highest score is its cluster similarity. The DB index is the average of all cluster similarities. Thus, the goal is to minimize these similarities and achieve higher distinctness between clusters. Formally, DB index=$\frac{1}{|k|}\sum_{p=1}^{k} C_p \, Max_{q,q\neq p}\{\frac{1}{|C_p|}\sum_{x_i \in C_p} d_{(x_i,\overline{c_p})} + \frac{1}{|C_p|}\sum_{x_j \in C_q} d_{(x_j,\overline{c_q})} / \, d_{(\overline{c_p},\overline{c_q})}\}$. Solution 1's DB index is 0.81 whereas solution 2's DB index is 1.66. Since the goal is to reduce the similarity between clusters, solution 1 is better with the DB index.

The silhouette index is a commonly used measure that takes both compactness and separation into consideration (Rousseeuw, 1987). Let a_i be the average distance between point *i* and the other points that are in the same cluster. Let b_i be the smallest average distance between point *i* and any other cluster's points. The silhouette index is define as: $s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$. By its definition, s_i, is always between -1 and 1, where a score closer to 1 means the point is close to its own clusters' points and not close to any other clusters' points. Thus the goal is to maximize $\sum_{i \in N} s_i$. Solution 1's silhouette index is 11.13 and solution 2's silhouette index is 4.76. Thus, solution 1 is better using the silhouette index.

Using all three of the above clustering measures, clustering solution 1 was better than clustering solution 2. However, many other measures exist. Several other measures are described in depth in (Liu, Li, Xiong, Gao, & Wu, 2010). So the question beckons 'Which measure is best?' In general, there is not one single measure that is agreed upon as the best, which is why so

much research has been done in clustering measures and heuristics. Once a distance measure and score criterion are established, a methodology should be used to generate solutions. The following section details general heuristics and how they can be used to generate such solutions.

2.3 General Heuristics

A basic understanding of general heuristics is needed before diving into clustering-specific ones. Since clustering is *NP*-complete, most research has focused on heuristic techniques. Heuristics are methods to solve problems. Most heuristics focus on practicality and intuitive processes. As such, most heuristics do not guarantee optimality, but instead try to find sufficiently good solutions in a reasonable amount of time. This section describes several heuristics that have been applied to clustering. To understand how heuristics generate new solutions, an explanation on the concept of neighborhoods and neighboring solutions is necessary.

2.3.1 Neighborhoods

A key component of the majority of generalized heuristics involves a neighborhood. From a solution, a neighboring solution is a feasible solution that is similar to the existing solution in some well-defined way. An existing solution's neighborhood consists of all neighboring solution.

Formally, given some problem $\prod$, let $\mathcal{S}$ be the set of all feasible solutions to $\prod$. Let $S_1 \in \mathcal{S}$. If $S_2 \in \mathcal{S}$ and $|S_1\text{-}S_2| < \varepsilon$, then S_2 is a neighbor of S_1 where $|\ |$ and ε are well defined in some fashion. Furthermore, $N_\varepsilon(S_1) = \{S \in \mathcal{S}: |S_1\text{-}\ S| < \varepsilon\}$. Finally, if problem $\prod$ has an objective function, then z_S denotes the objective value of S for all $S \in \mathcal{S}$.

Many heuristics move from a neighboring solution to a new solution by slightly adjusting the current solution in an attempt to find a good solution. Bit-swapping is an example of such a

technique. Bit-swapping methods keep all things the same about a certain solution except changing one attribute. Imagine that a pizza restaurant has six toppings and one seeks the best combination of pizza toppings. If the current solution is beef and pepperoni, two neighboring solutions by bit swapping would be beef, pepperoni, and mushrooms or just pepperoni.

For clustering, numerous neighborhoods could be defined. A neighboring solution could keep all points in the same clusters except for moving one point to a different cluster. Alternately, swapping two points in adjacent clusters could be a different neighboring solution. Removing a point from a cluster and making it an outlier, could be another example of a neighboring solution. These examples describe how clustering neighborhoods differ. Fundamentally, the neighborhood is completely based on how a heuristic defines | | and ε.

Neighborhood search techniques create adjustments to the current solution. At some solutions, called local optimal solutions, all neighbors are inferior. Formally, $S \in \mathcal{S}$ is a local optimal solution for a minimization problem if $z_S \leq z_{S'}$ for all $S' \in N_\varepsilon(S)$. Additionally, $S \in \mathcal{S}$ is a global optimal solution for a minimization problem if $z_S \leq z_{S'}$ for all $S' \in \mathcal{S}$. Clearly, researchers seek methods to leave local optimal solutions and to try to find global optimal solutions. Here hill climbing, simulated annealing and tabu search are presented in the next section.

2.3.2 Hill Climbing

Hill climbing is a neighborhood search technique. Hill climbing algorithms are based on the physical analogy of climbing a hill. Imagine a person in the mountains trying to find the highest peak in a hilly region. The person decides to only move in a direction up the hill. Every so often, the person looks around and sees if there is a better direction to move to find a higher elevation. While climbing, the person never goes in any direction that would take him back down the hill. Obviously, a peak is reached when any movement from that spot would start taking the

person down. The whole concept of hill climbing algorithms is to continue going up the hill and never accept any decrease in objective function value. This type of search terminates once a local optimal solution, but it may not necessarily be the global optimal solution.

At every iteration, hill climbing algorithms take the existing solution and seeks a neighboring solution that has a better objective value than the current solution. If such a neighboring solution exists, this solution replaces the current solution. This step repeats until there are no more improving neighboring solutions.

Hill climbing algorithms work well on problems that are convex, because there only exists a single local optimal solution, which is also the global optimal solution. If the goal is to find the highest peak in a hilly region, a hill climbing algorithm does not guarantee to find the highest of all the peaks in the region; but it would just guarantee to find one of the peaks.

Formally, hill climbing algorithms for minimization problems begin with $k = 1$ and an $S_1 \in \mathcal{S}$. Let $S' \in N_\varepsilon(S_k)$ such that $z_{S'} \leq z_{S_k}$. Assign k to $k+1$ and S_k to S'. If no such S' exists, terminate and report S_k as the solution. Thus, hill climbing algorithms guarantee $z_S \leq z_{S'}$ for all $S' \in N_\varepsilon(S)$.

Clustering is neither a convex feasible region nor does it have a convex objective functions. Some hill climbing algorithms use random restarts to generate other local optimums. With these restarts, the hope is that one of the local optimums found is also the global optimum.

2.3.3 Simulated Annealing

Simulated annealing is another frequently used heuristic to generate quality solutions. This technique is based on the principles of metal annealing (Eglese, 1990). Physical annealing melts a substance before it lowers the temperature over a long period of time. Crystalline solids are heated initially and then cooled at a very slow rate to reach a crystal lattice configuration free of defects. Without this process, these solids would not be able to reach different states. This thermodynamic analogy leads to the properties of simulated annealing algorithms.

In this analogy, different states correspond to different feasible solutions that would not be possible without simulated annealing. In simulated annealing algorithms, a neighboring solution is generated based off the current solution. Unlike hill climbing, simulated annealing accepts worse neighboring solutions in order to search a larger region of the solution space. At each iteration, simulated annealing algorithms compare the objective of a current solution, z_t, to a newly generated solution, z_{t+1}. An initial temperature, T_0, sets the cooling schedule. Like the cooling during annealing of metals, the probability of accepting worse solutions is reduced over time. After every iteration, the temperature, T, is lowered to decrease the acceptance probability over time. Usually, $T_{t+1}=T_t*c$ where c=1- ε. A common acceptance probability is as follows:

$$\text{P (Accepting } t\text{+1 solution)} = \begin{cases} 1, \text{if } z_{t+1} \text{better than } z_t \\ e^{\frac{|z_{t+1}-z_t|}{T_t}}, \text{if } z_{t+1} \text{worse than } z_t \end{cases}$$

Simulated annealing has both positive and negative features. Simulated annealing escapes local optimal solutions that hill climbing algorithms cannot. Simulated annealing guarantees optimality if ran for infinite time (Ingber, 1993). However, simulated annealing takes a longer time to terminate in good solutions than some heuristics. This is due to the probabilistic nature of simulated annealing. With this inherent randomness, bad accepted changes can lead simulated annealing algorithms to explore neighborhoods that are not near the global optimal solution.

The biggest challenge of simulated annealing is picking a temperature and a neighborhood. Picking a temperature that cools too slowly results in the heuristic taking a long time to run. If it is cooled quickly, the whole space may not be navigated effectively and the heuristic may get stuck in a single neighborhood for a long period of time. The temperature is generally problem-specific, with each type of problem having a temperature that yields better results.

Simulated annealing has been validated as a practical heuristic to generate near optimal solutions for multiple different clustering criterion (Brown & Huntley, 1992). Duczmal (2004) used simulated annealing as a way to identify clusters with arbitrary shapes and not necessarily spherical. For Duczmal's application, the clusters were viewed from a graphical perspective and additional connected subgraphs were added to clusters using simulated annealing principles. Additionally, simulated annealing heuristics have been applied to various problems with routing. One application uses simulated annealing to solve routing problems where things can only be delivered in certain time windows (Lin, Vincent, & Lu, 2011).

2.3.4 Tabu Search

Tabu search is a neighborhood search heuristic that is based on disallowing *tabu* or forbidden moves. A list tracks recent changes on a tabu list, or historical record. Tabu search is implemented mainly in nonlinear or combinatorial problems (Glover & Laguna, 1998).

Formally, given some problem $\prod$, let $\mathcal{S}$ be the set of all feasible solutions to $\prod$. Tabu search begins with $S_1 \in \mathcal{S}$, $i = 1$ and an empty tabu list $\mathcal{T}$. The main step finds a neighboring solution $S_{i+1} \in N_\varepsilon(S_i)$. If S_{i+1} is not in $\mathcal{T}$, then S_{i+1} is added to $\mathcal{T}$ and if $|\mathcal{T}| > k$, then the oldest item in $\mathcal{T}$ is removed. In performing these operations, if a solution is discovered that is better than the best known solution, this solution becomes the new best known solution. These main steps continue for a set number of iterations. The algorithm terminates and reports the best known solution.

Choosing a neighboring solution can be iteratively applied either deterministically or probabilistically. That is, some tabu search applications use hill climbing techniques, whereas others accept worse candidate solutions with some probability. One benefit tabu search has over other search heuristics is its ability to avoid cycles. For example, given S_1 and S_2 many search

heuristics have some probability of accepting S_2 even if it is worse than S_1. A few moves later, the heuristic may potentially attempt a move back to S_1. After some threshold of iterations, moves are taken off the tabu list. By removing some attempted operation from the tabu list, these operations can be reattempted and are potentially beneficial moves now that the current solution is different than it was.

Keeping track of a tabu list, increases the memory needed for a heuristic. A challenge of tabu lists is not knowing how long an item should be kept on the list. If the list is too long, then the list may be disallowing moves that would actually be good for long periods of time. If the list is too short, cycles may occur.

Additionally, there is a tradeoff for what information is stored during the tabu list. For instance, if one keeps track of potential moves the tabu list may look like this: $\{x_5, x_3, x_7\}$. This type of list shows what points were moved, but not where they were. So maybe x_5 should try a move to a different cluster than the one previously tried, but with this memory format it is not allowed to attempt such a move until x_5 is taken off the list. Additionally, one could keep a separate list with the clusters the points tried to move to. Even then, maybe x_5 should try the exact same move because the moves taken while it has been on the list could make this a beneficial move. Furthermore, the tabu list could keep track of the whole solution at every iteration instead of just the moves or clusters. This leads to the user having to decide what to track and weigh the costs of more memory with the benefits of having a more detailed history list of previous iterations.

The probabilistic nature of simulated annealing is combined with a systematic algorithm incorporating tabu lists in a tabu search hybrid for clustering in Osman and Christofides (1994). The cooling schedule aspects of simulated annealing are based on the oscillating fashion of tabu

search. An alternate to *k*-means clustering, which is discussed in section 2.5, named *k*-harmonic means uses tabu lists as a way to overcome some fundamental problems researchers have with the *k*-means algorithm (Güngör & Ünler, 2008).

Additionally, a common use of tabu lists is optimally scheduling flow in a manufacturing cell. Tabu search is used to efficiently schedule jobs within these cells by sequencing part families and jobs within families (Skorin-Kapov & Vakharia, 1993). Tabu lists have also been used to schedule jobs in a job shop environment where parts can only be run on certain machines (Colorni, Dorigo, Maniezzo, & Trubian, 1994).

2.4 Mathematical Programming and Polyhedral Theory

Besides a general knowledge of clustering, mathematical programming is necessary to understand this thesis. Basic concepts from polyhedral theory, linear programming (LP) and integer programming (IP) and are presented here.

2.4.1 Polyhedral Theory

A primary emphasis of this research involves linear algebra. The vectors, $v_1, v_2, \ldots, v_p$, are linearly independent if no vector is a linear combination of the other vectors. This occurs if, and only if, the unique solution to $\sum_{i=1}^{p} \alpha_i v_i = 0$ is $\alpha_i = 0$ for all i =1, …, p. In an n-dimensional space, at most n vectors can be linearly independent. Another important concept is the span of vectors. The span of a set of vectors is the set of linearly independent vectors. In many applications, whether the vector spans the real space is important. If the vectors do not span the real space, then the vectors are not fully dimensional.

The concept of a convex hull is central to the extremity measure. When viewing a set of points, the convex hull is the intersection of all convex sets that contain all of the points. A set X is convex, if and only if, $\alpha x + (1-\alpha)\, x' \in X$ for all x and $x' \in X$ and $\alpha \in [0,1]$. The convex hull of a

set of p points, $x_1, x_2, \ldots, x_p$, is the space defined by all linear weighted combination of these points. Formally, this convex hull is:

$$\{y \in \mathcal{R}^n : y = \sum_{i=1}^{p} \alpha_i x_i \text{ such that } \sum_{i=1}^{p} \alpha_i = 1, \alpha_i \geq 0 \text{ for all } i = 1, \ldots, p\}.$$

Affine independence is used to calculate the dimension and span of the convex hull of a set of points. A set of p points are affinely independent if, and only if, the unique solution to $\sum_{i=1}^{p} \alpha_i x_i = 0, \sum_{i=1}^{p} \alpha_i = 0$ is $\alpha_i = 0$ for all i =1, …, p. Additionally, the dimension of this convex hull is p-1. This can be viewed as letting one of the p points becoming the origin and subtracting each of the other points to create linearly independent vectors.

A polyhedron is a shape in n dimensional space that has defined edges, faces, and vertices. Formally, a half space is $\{x \in \mathcal{R}^n : a^T x \leq b\}$ where $a \in \mathcal{R}^n$, and $b \in \mathcal{R}$. A polyhedron is the intersection of a finite number of half spaces. Consequently, a polyhedron cannot have curves or any other different shape. A convex hull of a finite set of points is an example of a polyhedron. For the remainder of this thesis, the convex hull of a cluster is shown to define the boundary of the cluster. These are depicted in Figure 2.5.

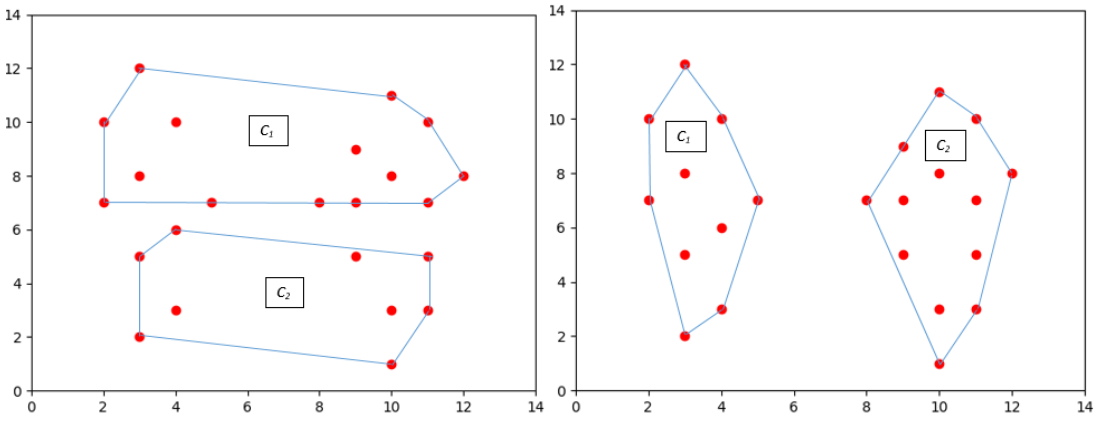

Figure 2.5 Updated Clustering Solutions 1(Left) and 2(Right)

Extreme points are vital to this research. An extreme point is a point that cannot be expressed as a convex combination of the other points in a set. Formally, $x \epsilon X$ is an extreme point if, and only if, there does not exist an x', $x''\epsilon X$ such that $\frac{1}{2} x' + \frac{1}{2} x'' = x$ and $x' \neq x''$.

2.4.2 Linear and Integer Programming

Linear programs **(**LP) are a class of problems that optimize a linear system of equations and achieve the best possible outcome subject to constraints. LPs have broadly solved many optimization problems. The feasible region of all bounded LPs are convex solution spaces that are polyhedrons.

To formulate an LP, let the decision variables be x. These variables can be changed or altered to improve the output or goal. Linear constraints must be formulated to form a feasible region that typically limits the objective values. Formally, an LP is defined as:

Maximize $c^T x$

Subject to: $Ax \leq b$

$x \geq 0.$

where $A \epsilon \mathbb{R}^{mxn}$, $b \epsilon \mathbb{R}^{m}$, $c \epsilon \mathbb{R}^{n}$.

The simplex algorithm (Dantzig, 1947) solves linear programs by moving along the edges on the feasible polyhedron until the optimal solution is obtained. The simplex algorithm always moves from one extreme point to another extreme point in the direction of best marginal improvement. Since the space is convex, there is always be a direction of improvement unless the current feasible solution is also optimal.

Interior point algorithms are another class of solvers for linear programs that search through the interior of the feasible space. Interior point algorithms run in polynomial time

(Karmarkar, 1984). Consequently, any linear program defined in chapter 3 can be solved in polynomial time.

Integer Programs (IP) take the form of an LP, but the variables are also constrained to be integer values. In IP, the spaces are not convex polyhedrons. It is challenging to define the boundaries of the integer space given the constraints which define the linear space. Instead of trying to define those edges, most integer programming algorithms use other techniques to still utilize the simplex algorithm. Usually this involves cutting the space and relaxing integer restrictions iteratively to find the optimal solutions. These techniques are time intensive because they may solve an exponential number of LPs to find the optimal solution to the IP.

2.5 Heuristics for Clustering

Besides general purpose heuristics, an understanding of commonly used clustering heuristics is needed. Researchers have used concepts from the previously mentioned measures and general heuristics to create clustering-specific heuristics. Jain (2010) and Halkidi (2001) have compiled lists of clustering methods. Here the *k*-means clustering algorithm and mean shift clustering are presented.

2.5.1 *K*-means Clustering Algorithm

K-means is the most well-known and applied clustering algorithm. This is due to its simplicity and how available applications support some variation of a *k*-means clustering. Python, Matlab, and R programing language are just a few software with *k*-means algorithm implementations. *K*-means attempts to partition data points into *k* clusters. The goal of *k*-means clustering is to reduce the squared error distances between a cluster's points and the cluster's centroid.

To start the algorithm, a number of desired clusters, *k*, is required as input as well as initial centroids of each of these clusters. Each data point is then assigned to the cluster of the

centroid it is closest to. At each iteration, the centroids are recalculated as the average of that cluster's points. Points are then reassigned if there is a closer centroid. These steps are repeated until some terminal iteration, or until no points can move to a different cluster.

For example, let's use *k*-means to cluster the points in Table 2.1 into two clusters. An intuitive implementation of the initial *k*-means with *k*=2 would be to pick the two furthest points away from each other, x_6 and x_{15}, as the initial cluster centers. All points are then assigned to the closer of these two centers, as shown in Figure 2.6. The center point of each cluster is calculated and denoted by the hollow circle in the picture. All the points are then be assigned to the closer of these two new centers, as shown in Figure 2.7. In this example, x_{22} is the only point that moves in the first iteration. This repeats until some threshold of iterations, or until no changes are made. For this example, the solution terminates in clustering solution 2 from Figure 2.5.

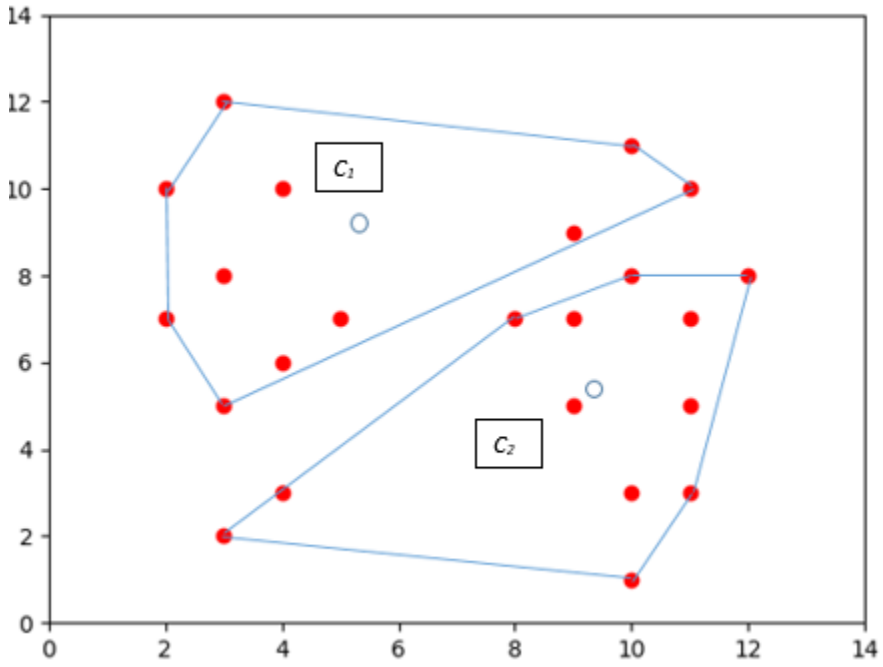

Figure 2.6 Initial *k*-means solution

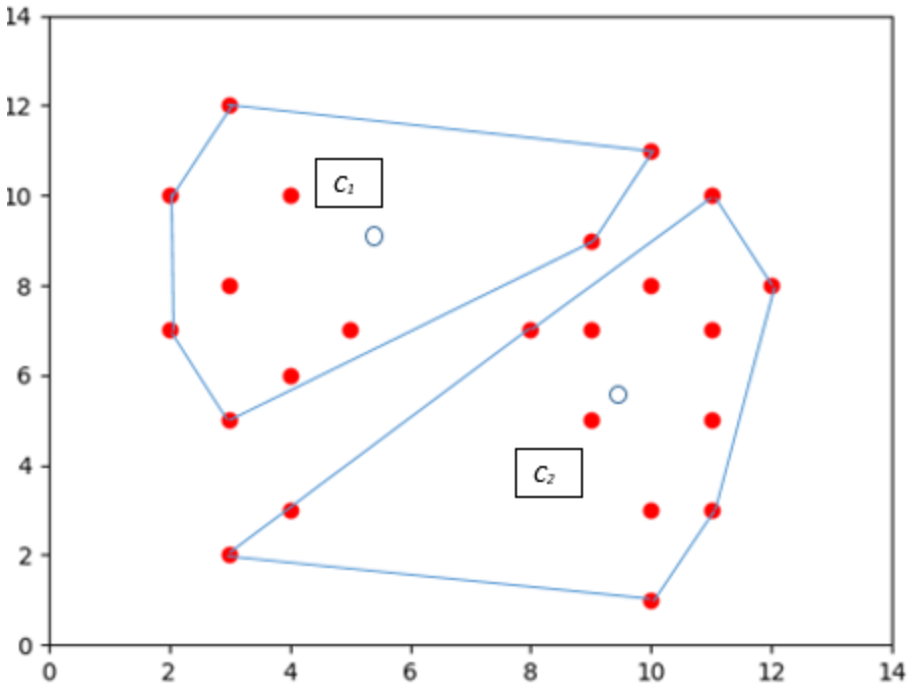

Figure 2.7 *k*-means solution after 1 iteration

K-means can be viewed as a sort of hill climbing algorithm. This is because the generic algorithm only takes changes that improve the solution. This yields a smaller region searched for solutions and does not guarantee global optimality. As such, there are different implementations of *k*-means that deal mainly with generating better initial guesses for cluster centroids, or generating random starting guesses and different values of *k*. The hope is that multiple retries with different starting points result in good clustering solutions. Much research has been done into optimizing the *k*-means algorithm to reduce time and increase cluster strength (Hartigan, 1979; Kanungo, 2002). The simplicity of the *k*-means algorithm leads to many people without optimization backgrounds to gravitate towards it for their applications. One of the biggest weaknesses with *k*-means is its inability to form "non-spherical" clusters.

2.5.2 Mean Shift Clustering

Many researchers dislike *k*-means requirement to input the number of clusters. Mean shift clustering is another common clustering heuristic created that does not require this input (Cheng,

1995). Mean shift clustering attempts to find good clusters by identifying areas of points that are dense. In addition to the data set, mean shift clustering requires kernels as input. Kernels are spheres in *n*-dimensional space with a center and radius. These kernels are used to quantify how dense a particular region is. Generally, these kernels are initially spread evenly throughout the space. At every iteration, these kernels search neighboring regions for areas of higher density. This criteria of density leads researchers to classify mean shift clustering as a mode seeking algorithm. If a kernel can no longer find an area of higher density, it no longer shifts. Once no kernels are able to shift, the heuristic terminates. Note that kernels can, and in many cases do, converge to the same location. Thus, mean shift clustering can generate a different number of clusters than the number of initial kernels. These kernel centers become the centroids of the clusters and data points are then grouped into clusters according to which kernel center each is closest. Because the kernels only shift to areas of higher density, mean shift clustering can be seen as a hill climbing algorithm.

A benefit to mean shift clustering over *k*-means clustering is the number of clusters is not needed as input. The downside of mean shift clustering is the necessary input of the number of kernels, as well as their centers and radii, which can impact the solution. Mean shift clustering can use restart techniques like some *k*-means applications, but picking an appropriate radius and initial placement for the kernels is not trivial.

2.5.3 Quadratic Polyhedral Clustering Algorithm

The quadratic polyhedral clustering algorithm (QPCA) is the most closely related research. QPCA is a clustering algorithm that uses data points distance to the closest boundary of the remaining polyhedron of the other cluster points as its clustering quality criterion (Kayarat, 2005). Each data point's score is obtained using quadratic programming to find how close a point

is to the convex hull of the remaining points. This allows for non-spherical clusters as long as each point does not add a large amount of thickness to the cluster size by itself.

QPCA uses proximity graphs to form initial clusters using the clique algorithm. That is, any two points, or vertices, whose distance between them is less than some threshold, ε, are connected via an edge. After all edges are formed, any subset of nodes that form a clique are joined into initial clusters. The clique algorithm requires O (n^4) effort.

From this initial solution, each data point's contribution to the cluster score is the minimum Euclidean distance from that data point to the convex hull of the remaining cluster points. QPCA defines $y = \{y_1, y_2, \ldots, y_v\}$ as the coordinates of the closest point in the convex hull to the test point, x_q. QPCA utilizes the following quadratic program to obtain the minimum distance from x_q to y:

$$CS_q = \text{Minimize } [(y_1 - x_{q,1})^2 + (y_2 - x_{q,2})^2 + \ldots + (y_v - x_{q,v})^2]$$

$$\text{Subject to } \sum_{x_i \in C \setminus \{x_q\}} \alpha_i * x_{i,v} = y_v \text{ for all } v = 1, \ldots, m$$

$$\sum_{i:\, x_i \in C \setminus \{x_q\}} \alpha_i = 1$$

$$\alpha_i \geq 0 \text{ for all } i: x_i \in C \setminus \{x_q\}$$

y unrestricted.

From every points distance, the total cluster score can be calculated. Each cluster's score is calculated as the sum of every point distance in its cluster. Formally,

$$CC_{c_p} = \sum_{i: x_i \in C} CS_i$$

The total clustering solutions score is the sum of the cluster scores plus a constant, λ, multiplied by the number of clusters in order to penalize the algorithm from having every cluster with only one point as the best solution. Formally, $z = \sum_{p=1}^{k} CC_{c_p} + \lambda k$. The goal is to minimize z.

QPCA utilizes merge, split, and move subroutines to generate new clustering solutions. The merge subroutine takes two clusters and combines them into one cluster. The split routine takes one cluster and splits into multiple clusters by utilizing a smaller version of the initial clique algorithm. The move subroutine moves a single point from one cluster to some other cluster. QPCA uses hill climbing logic to decide whether or not to accept solutions. That is, only a new solution with an improvement to z is accepted.

At every iteration, each point with the worst score in each cluster is tested for a move. It is removed from the cluster and tested to find the closest cluster. If it is too far away from this cluster, the point becomes a single point cluster, referred to as a singleton. If it is close enough, it makes the move to this cluster. Note that this cluster may have been its previous cluster, which can be viewed as the algorithm affirming that currently this point has nowhere better to move to. After these moves, QPCA attempts a split routine on the worst cluster and is accepted if the newly formed clusters using the modified clique formulation is better. After the split routine, merges are attempted if two clusters centroids are less than some threshold.

Note that solving a quadratic program is more time intensive than solving linear program. As such, QPCA only uses 30 iterations of these move, split, and merge routines to maintain a reasonable run time for the algorithm. This leads to challenges for QPCA in cases of big data. If only 30 iterations are run, it is more difficult to have confidence in how close the answer is to optimal. The next chapter discusses how this research resolves this issue.

Chapter 3: Extremity Measure and SAME heuristic

This chapter explains the primary contribution of this research, which are the extremity measure and the Simulated Annealing using Measurements of Extremity (SAME) heuristic. Initially, this chapter defines the extremity measure and its two components: the extreme distance and the central distance. The second section explains how neighborhoods are defined by SAME. That is, the neighborhoods for moves, splits and dissolves are explained. To begin iterating through these neighborhoods, an IP is used to generate an initial feasible solution. In the fourth section, the methods that the SAME heuristic uses to select clusters and neighborhoods for adjustment at each iteration are presented. The chapter closes with how SAME applies simulated annealing to determine whether a new solution should be accepted or rejected based on its quality.

3.1 Extremity Measure

Recall from chapter two that many clustering measures focus on cluster separation and/or compactness. The extremity measure (EM) is composed of the extreme distance and central distance of each point. The extreme distance estimates how extreme a point is to the remaining cluster. The central distance measures how close a point is to the middle of the cluster. Although both values are measures of compactness, the extreme distance encourages large clusters, while the central distance desires small clusters. Thus, these two components interact to create quality clustering solutions.

The primary concept of extreme distance involves convex hulls. Any cluster point that is an extreme point to the clusters convex hull contributes a positive value to this measure. Alternately, any point within the convex hull is more similar to the other cluster points than the extremities and should penalize nothing.

A clustering solution's extreme distance begins by the calculating the extremity of each point. The point, x_q, is removed from its cluster and tested against the convex hull of the cluster's remaining points. If x_q is in this convex hull, then it is not an extreme point and its extreme distance, e_q, is 0. If not, then x_q is an extreme point of the cluster's convex hull. An obvious measure of x_q's extremeness is the distance from this point to the remaining convex hull. The exact distance can be calculated by solving a quadratic program, which was introduced by (Kayarat, 2005). Quadratic programs are too slow and a primary goal of this research is to eliminate the quadratic part for a faster distance calculation. This research approximates the distance by solving a linear program, called the Extreme Distance Linear Program (EDLP).

Given a clustering solution with x_q in some cluster C, the concept of the extreme distance involves creating x_q as a convex combination, α_i, of x_i for all $x_i \in C\backslash\{x_q\}$. If x_q is not extreme, then such a convex combination exists. However, if x_q is extreme, then no such solution exists. Consequently, the requirement that each $\alpha_i \geq 0$ is relaxed. To achieve this, substitute α_i with $\alpha_i^+ - \alpha_i^-$ to create the formulation. The α_i's are split into two decision variables for ease of only contributing value to the objective function when a point requires a negative weight. Define $2|C|-2$ decision variables α_i^+ and α_i^- for all $x_i \in C \setminus \{x_q\}$. Formally, the EDLP is:

$$\text{Minimize} \quad e_q = \sum_{x_i \in C\backslash\{x_q\}} \alpha_i^- * d_{i,q}$$

$$\text{Subject to:} \quad \sum_{x_i \in C\backslash\{x_q\}} (\alpha_i^+ - \alpha_i^-)\, x_{i,v} = x_{q,v} \text{ for all } v = 1,\ldots,m.$$

$$\sum_{x_i \in C\backslash\{x_q\}} (\alpha_i^+ - \alpha_i^-) = 1$$

$$\alpha_i^+ \text{ and } \alpha_i^- \geq 0 \text{ for all } i\text{: } x_i \in C\backslash\{x_q\}$$

The objective function sums each negative weight multiplied by the distance between x_q and x_i. If $\alpha_i^- > 0$, then x_i is generally "opposite" of x_q in the polyhedron. By multiplying by the distance between x_i and x_q, the term $\alpha_i^- * d_{i,q}$ approximates how extremely opposite x_q is with

respect to x_i. Minimizing the sum of all such instances results in an approximation of x_q's extremeness.

Note that if the objective function attempted to merely minimize the sum of α_i^-, then the consistent solution is to pick the point that is farthest from x_q, as this point's α_i^- value could be very small. For instance, if a point is twice as far away from x_q than x_i, the contribution of x_q's extremeness would be cut in half with this objective function. Consequently, the distance term is a necessity.

The first set of constraints maintains that x_q is a linear combination of the other points in the cluster. Note that if it is within the convex hull of the other points then this combination is convex and $\alpha_i^-=0$ for all $i= 1, \ldots, p$. The second constraint line maintains that the weights must sum to 1, just like the convex hull formulation. EDLP's optimal solution never chooses a non-zero value for α_i^+ and α_i^- for some i. This is because EDLP could just reduce both variable values by the minimum of those two and maintain feasibility while reducing negative weights. As such, a constraint is not necessary to make sure that a point does not have multiple weights.

Examine C_1 in clustering solution 1 of Figure 2.5, where x_6 is an extreme point. EDLP quantifies how extreme this point is. Figure 3.1 graphical depicts this situation. Applying EDLP to x_6 results in $e_6=2.5$. The decision variable values for this solution are $\alpha_2^+=0.625$, $\alpha_9^+=0.625$, $\alpha_3^-=0.25$ with $\alpha_i^-=0$ and $\alpha_i^+=0$ for all other variables. Note that the distance between x_3 and x_6 is 10. Since x_3 is the only point with the negative weight, $\alpha_3^-* d_{3,6=} e_6=0.25*10=2.5$. Note that trivially the distance from x_6 to the remaining convex hull is 2. This gives an example of how EDLP approximates this distance with some accuracy without having to solve a quadratic program.

Now consider x_4, which is on the interior of the convex hull. A solution to EDLP for x_4 could be $\alpha_3^+=0.5$, $\alpha_5^+=0.5$ with $\alpha_i^+=0$ and $\alpha_i^-=0$ for all other variables. This means x_4 is a convex

combination of the other points and is not an extreme point. Thus, x_4's extreme distance is 0. Table 3.1 gives the extreme distances for every point in this cluster. Thus, the extreme distances of the points of C_1 contribute 9.06 to z.

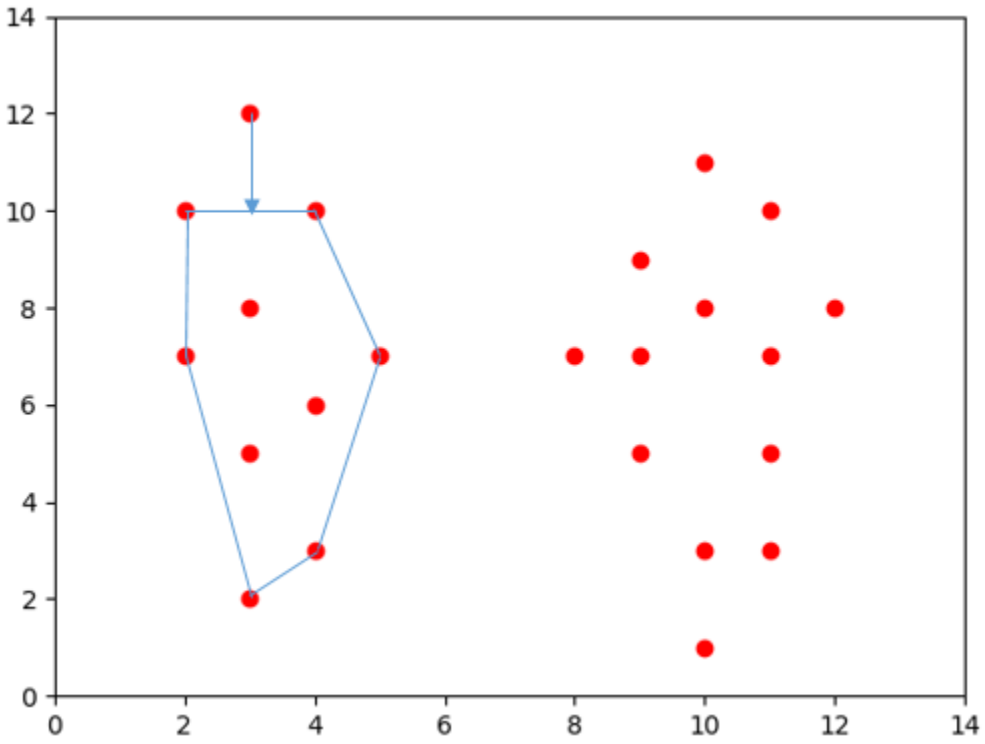

Figure 3.1 Graphical Depiction of Extreme Distance Concept

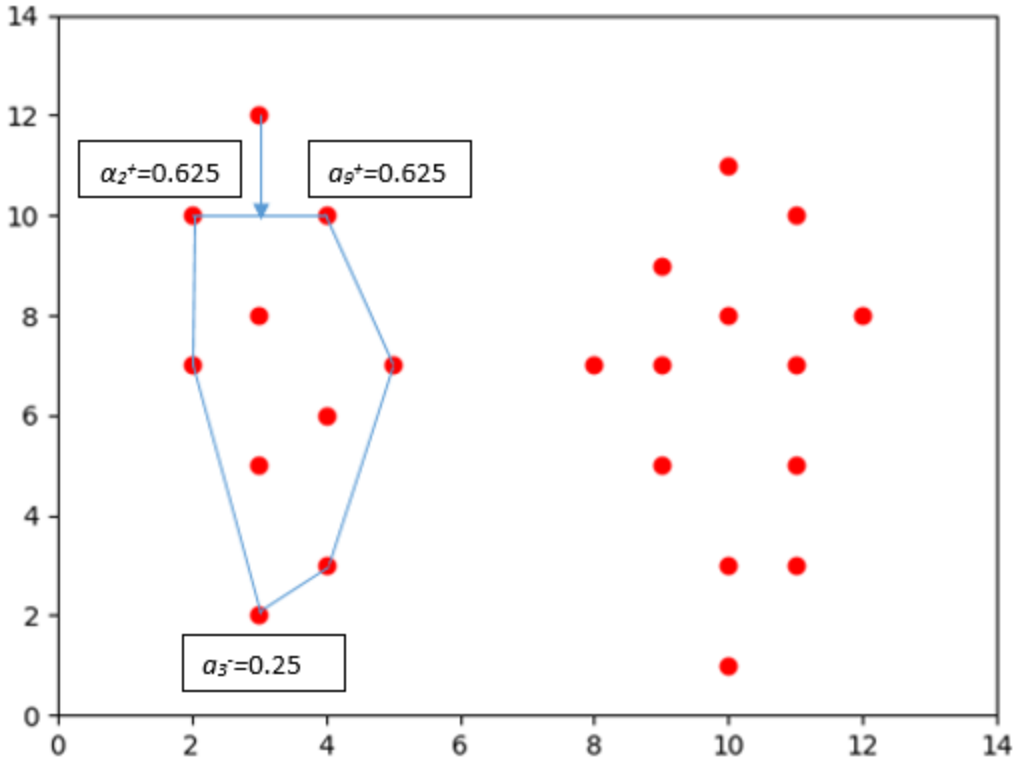

Figure 3.2 Graphical Depiction of EDLP Solution to e_6

q	e_q
1	0.42
2	0.84
3	2.69
4	0
5	0
6	2.5
7	0.89
8	0
9	0.22
10	1.5
Total	9.06

Table 3.1 Extreme Distances for C_1

A point's extreme distance has several favorable properties. Observe that a point has an extreme distance equal to 0 if, and only if, the point is not an extreme point of the cluster.

Furthermore each point has an extreme distance calculation, so a value exists to decide which points should potentially belong to other clusters.

One challenge with EDLP occurs if a cluster is not full dimensional. In this situation, a point may be tested against the remaining convex hull of points whose span does not contain the test point. In this case, EDLP is infeasible. In the event that an EDLP is infeasible, then the point is given an extreme distance of a large value, *M*. Thus, good clustering solutions avoid scenarios where points are not in the spans of the other points in the cluster.

If the extremity measure only incorporates the scores from EDLP, then problems arise. Given a cluster, adding an additional point reduces or maintains the value returned from EDLP of all current points within the cluster. The added point makes the polyhedron larger and could not make the distance from each point to the polyhedron any smaller. Thus, adding points is likely to improve the objective score. Consequently, EDLP encourages large clusters. For this reason, a second component is added to the measure to counteract this tendency.

The second component of the extremity measure, which balances EDLP's propensity to create large clusters, is called the central distance. This is accomplished by incorporating the distances between each point and its cluster centroid. Large clusters tend to have longer distances to the centroid and are less compact. Given some x_i in cluster *C*, its central penalty is defined the distance x_i to $\bar{c}$. Thus, the cluster total central distance contributes to the objective value as a function of $\sum_{x_i \in C} \; d_{(x_i,\bar{c})}$.

Intuitively, there should be some weighting factor, λ, that balances how much a clustering solution's score is attributed to extreme distance versus this central distance penalization. Given some $\lambda \geq 0$, the extremity measure of cluster *C* is: $E_C = \sum_{x_i \in C}(e_i + \lambda * d_{(x_i,\bar{c})})$. Table 3.2 gives the central distance contribution for every point in C_1 where $\bar{c_1}$ is (3.3,

7), along with the total contribution with λ=0.5. Thus, E_1=9.06+13.85=22.94. Note that using the same logic E_2 can be computed as 25.55

q	$d_{(x_i,\bar{c})}$
1	1.3
2	3.26
3	5.00
4	2.02
5	1.04
6	5.00
7	4.06
8	1.22
9	3.08
10	1.70
$0.5 * \sum_{x_i \in C} d_{(x_i,\bar{c})}$	13.85

Table 3.2 Central Distances for C_1

The quality, $z_{\mathcal{C}}$, of a clustering solution is the sum of the extremity measure of every cluster. Consequently, $z = \sum_{p=1}^{k} E_{C_p}$. Given two clustering solutions $\mathcal{C}_1$ and $\mathcal{C}_2$ with $z_{\mathcal{C}_1} < z_{\mathcal{C}_2}$, then $\mathcal{C}_1$ is a better clustering solution than $\mathcal{C}_2$. Using the concept of neighborhoods, the SAME heuristic attempts to minimize z.

A primary purpose of this research is to eliminate the requirement to solve a quadratic program. EDLP estimates this distance by solving a simple linear program with only 2 $|C|$ -2 variables. Furthermore, there are only m+1 constraints. Thus, this technique runs in polynomial time and these linear programs are quickly solved. Another research goal arises from big data. Testing how new data relates to existing clusters may be of importance depending on application. This can now be accomplished extremely fast, and EDLP may become a new effective method to classify whether or not an incoming data point belongs to an existing cluster.

3.2 Neighborhoods of SAME heuristic

This section details the types of neighborhoods that SAME considers. Moves, splits, and dissolves are modifications to an existing solution that define its neighborhood. Moves are smaller changes whereas dissolves and splits can greatly change the solution.

3.2.1 Move Operation

At times a point may clearly be assigned to an inappropriate cluster. The move operations seeks to rectify this issue. The primary concept behind the move neighborhood is to move a single point from one cluster to a different cluster.

Formally, given a clustering problem $\prod$, $\mathscr{C} = \{C_1, C_{2\ldots}, C_k\} \in \mathbb{S}$, and $\mathscr{C}' = \{C'_1, C'_{2\ldots}, C'_k\} \in \mathbb{S}$ then $\mathscr{C}' \in N_{\varepsilon_{\text{move}}}(\mathscr{C})$ if, an only if, there exists clusters C_p and C_q along with a point x_i such that $C'_p = C_p \setminus \{x_i\}$, $C'_q = C_q \cup \{x_i\}$ and $C'_r = C_r$ for all r in $\{1,\ldots,k\}\setminus\{p,q\}$. One can easily expand upon this definition of $\varepsilon_{\text{move}}$ by moving multiple points.

Consider the clustering solution 1 from figure 2.1 and examine x_1 in C_1 . Moving this point to C_2 creates a new clustering solution. Figure 3.3 depicts the C_1 and C_2 before and after the move operation. Table 3.3 shows how this move impacts the extreme and central distances' contribution to the extremity measures of the clustering solution, if λ=0.5. Notice that C_1's central distance went down while C_2's central distance went up. This is frequently the case for a cluster reducing its size and a cluster increasing its size, respectively. In this instance, x_1's move increase the central distance in C'_2 significantly. Consequently, this move may not be advantageous.

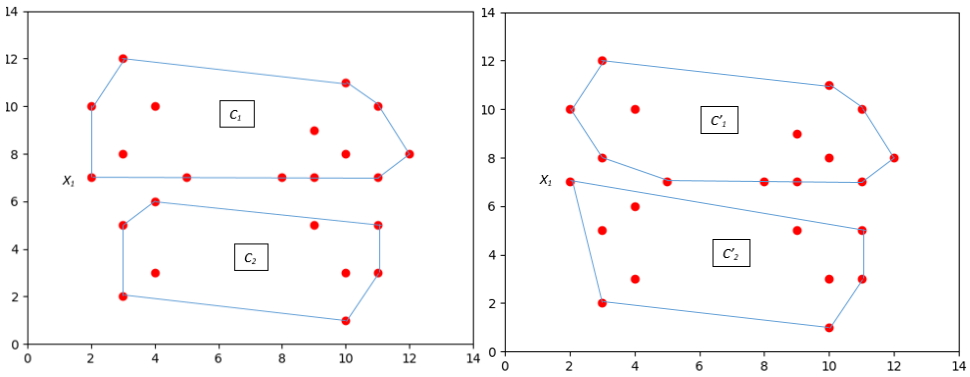

Figure 3.3 Move Operation of x_1

Solution	Contribution	C_1 Score	C_2 Score	Total Score
Before Move	Extreme Distance	15.89	10.19	72.66
	Central Distance	30.08	16.50	
After Move	Extreme Distance	18.08	10.74	75.62
	Central Distance	27.25	19.55	

Table 3.3 Comparison of Move Operation Score Change

Fortunately, the move operation can be efficiently implemented. In the majority of situations, calculating the new extreme distances requires far fewer than the anticipated $|C'_p| + |C'_q|$ EDLP solutions. More on such an efficient implementation is discussed in chapter 4.

3.2.2 Dissolve Operation

Sometimes an entire cluster is not compact. In this case, a dissolve operation may be useful. A dissolve operation moves all the points within a cluster simultaneously and assigns them to the other clusters.

Formally, given a clustering problem $\prod$, $\mathcal{C} = \{C_1, C_2, \ldots, C_k\} \in \mathcal{S}$, and $\mathcal{C}' = \{C'_1, C'_2, \ldots, C'_{k-1}\} \in \mathcal{S}$. Then $\mathcal{C}' \in N_{\varepsilon_{\text{dissolve}}}(\mathcal{C})$ if, an only if, $C_q \subseteq C'_q$ for all $q \in \{1, \ldots, k-1\}$.

To fully demonstrate the dissolve operation, examine the clustering solution with 3 clusters in figure 3.4. If C_3 is picked to dissolve, all of its points move to C_1 or C_2. Table 3.4

shows how this move impacts the extremity measures of the clustering solution when λ=0.5. Note that the total central distances after the dissolve are bigger than before the dissolve. This property holds true for the dissolve operation as more clusters leads to bigger distances between a clusters point and its centroid. Since the total score went down, this operation would be advantageous given λ=0.5.

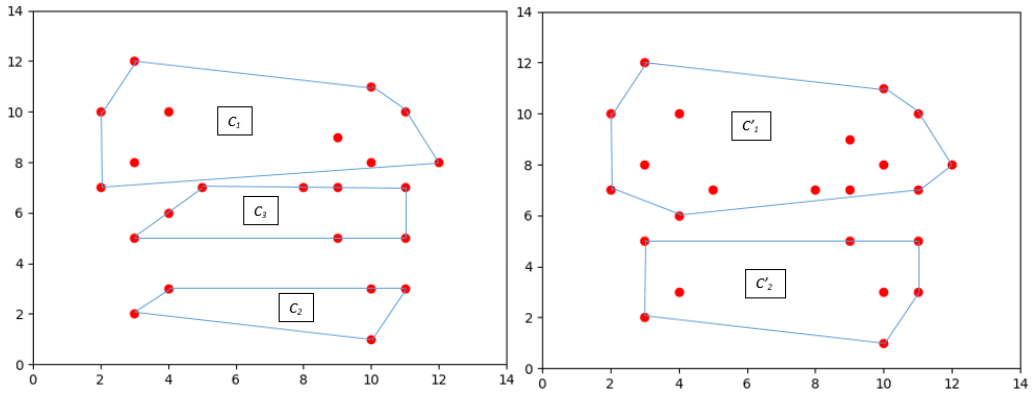

Figure 3.4 Dissolve Operation of C_3

Solution	Contribution	C_1 Score	C_2 Score	C_3 Score	Total Score
Before Dissolve	Extreme Distance	9.30	22.02	8.49	79.99
	Central Distance	20.34	8.48	11.35	
After Dissolve	Extreme Distance	9.14	12.68	Dissolved	64.28
	Central Distance	28.26	14.20		

Table 3.4 Comparison of Dissolve Operation Score Change

3.2.3 Split Operation

Sometimes a cluster is not compact and should be broken down into multiple clusters. In this case, a split operation may be useful. A split operation involves taking a cluster and partitioning the points into two new clusters.

Formally, given a clustering problem $\prod$ and $\mathscr{C} = \{C_1, C_2, \ldots, C_k\} \in \mathcal{S}$ and $\mathscr{C}' = \{C'_1, C'_2, \ldots, C'_{k+1}\} \in \mathcal{S}$. Then $\mathscr{C}' \in N_{\varepsilon_{split}}(\mathscr{C})$ if, an only if, $C_q = C'_q$ for all $q \in \{1, \ldots, k\text{-}1\}$ and $C'_k \cup C'_{k+1} = C_k$,.

To demonstrate the split operation, recall clustering solution 2 in figure 2.1. If C_2 is picked to split, some of the points split into C'_2 and the others become C'_3. Figure 3.5 depicts this situation. Note that splitting C_2 led to the sum of the central distances of C'_2 and C'_3 to be less than C_2. Table 3.5 shows in this instance, this reduction did not outweigh the increase in the extremity measures so this may not be an advantageous operation.

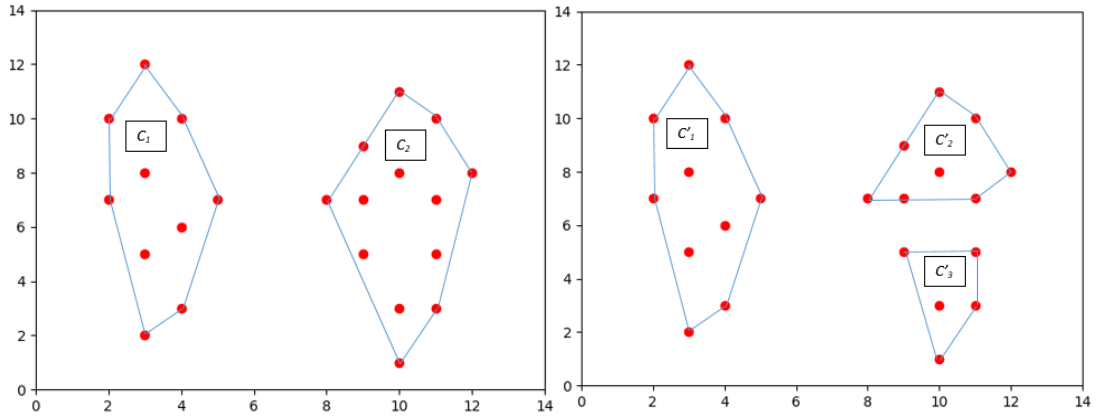

Figure 3.5 Split Operation of C_2

Solution	Contribution	C_1 Score	C_2 Score	C_3 Score	Total Score
Before Split	Extreme Distance	9.08	7.86	X	47.49
	Central Distance	13.86	16.69		
After Split	Extreme Distance	9.08	5.86	13.23	52.78
	Central Distance	13.86	6.98	3.77	

Table 3.5 Comparison of Split Operation Score Change

3.3 Initialization and Input to SAME

To start iterating through SAMEs neighborhoods, an initial $\mathcal{C}$ is required as input. A better initial clustering solution should reduce the amount of time SAME needs to generate quality solutions. If SAME just randomly generated initial clusters, then it may take a while to find a good region of neighboring solutions. As such, SAME forms initial clusters whose average distance between cluster points is below a certain threshold, ε. SAME sets ε to be such that the average cluster arc length is below 25% of the average arc length in the data set. Formally, $\varepsilon = 0.25*(\sum_{i,j:i,j \in N:i<j}(d_{i,j}))/(\frac{|N|^2-|N|}{2})$, though this threshold can be chosen arbitrarily. Using integer programming, SAME maximizes the amount of points in a cluster while maintaining this condition. Each data point has a decision variable, w_i, that is 1 if it is the cluster and 0 otherwise for all $i=1,\ldots,n$. Additionally, $y_{i,j}=1$ if i and j are both selected for current cluster and 0 otherwise for all $i,j=1,\ldots,n$ and $i<j$ The initialization IP is as follows:

Maximize: $\sum_{i \in N} \; w_i$

Subject to: $\sum_{x_i,x_j:i,j \in N:i<j}(d_{i,j} - \varepsilon) * \; y_{i,j} \leq 0$

$w_i + w_j + (1- y_{i,j}) \leq 2$ for all $i,j=1, \ldots, n$ and $i<j$

$w_i + w_j + (1- y_{i,j}) \leq 2$ for all $i,j=1, \ldots, n$ and $i>j$

$w_i \in \{0.1\}$ for all $i=1, \ldots, n$

$y_{i,j} \in \{0.1\}$ for all $i,j=1, \ldots, n$ and $i>j$.

The first constraint maintains the average distance threshold. The second and third sets of constraints deal with forcing the w_i and $y_{i,j}$ variables to link together. These constraints force that if w_i and w_j are both 1, then $y_{i,j}$ is forced to be 1 and vice versa. Any point x_i whose $w_i=1$ is assigned to a cluster and removed from the IP formulation. The IP reruns until all points are assigned to clusters or it turns infeasible. If the IP becomes infeasible, then there are some

outlying points that did not fit with the remaining cluster points. These outliers are assigned to the most similar cluster.

This Initialization IP generates quality clustering solutions. However, this IP has computational time challenges with larger data sets. Chapter 4 presents an upper bound to where this IP is practical and also presents a heuristical approach in cases where it is not.

3.4 Main Step of SAME

This section describes two important steps of SAME that occur at every iteration: new solution generation and a decision of whether or not to accept this new solution. At each iteration, a cluster must be selected. After a cluster is selected, a type of neighborhood operation is selected. A specific new clustering solution is generated based on these two decisions. This new clustering solution is accepted or not accepted using principles of simulated annealing. The flowchart in Figure 3.6 is a graphical depiction of SAME. The loop terminates at some threshold, *max iterations*.

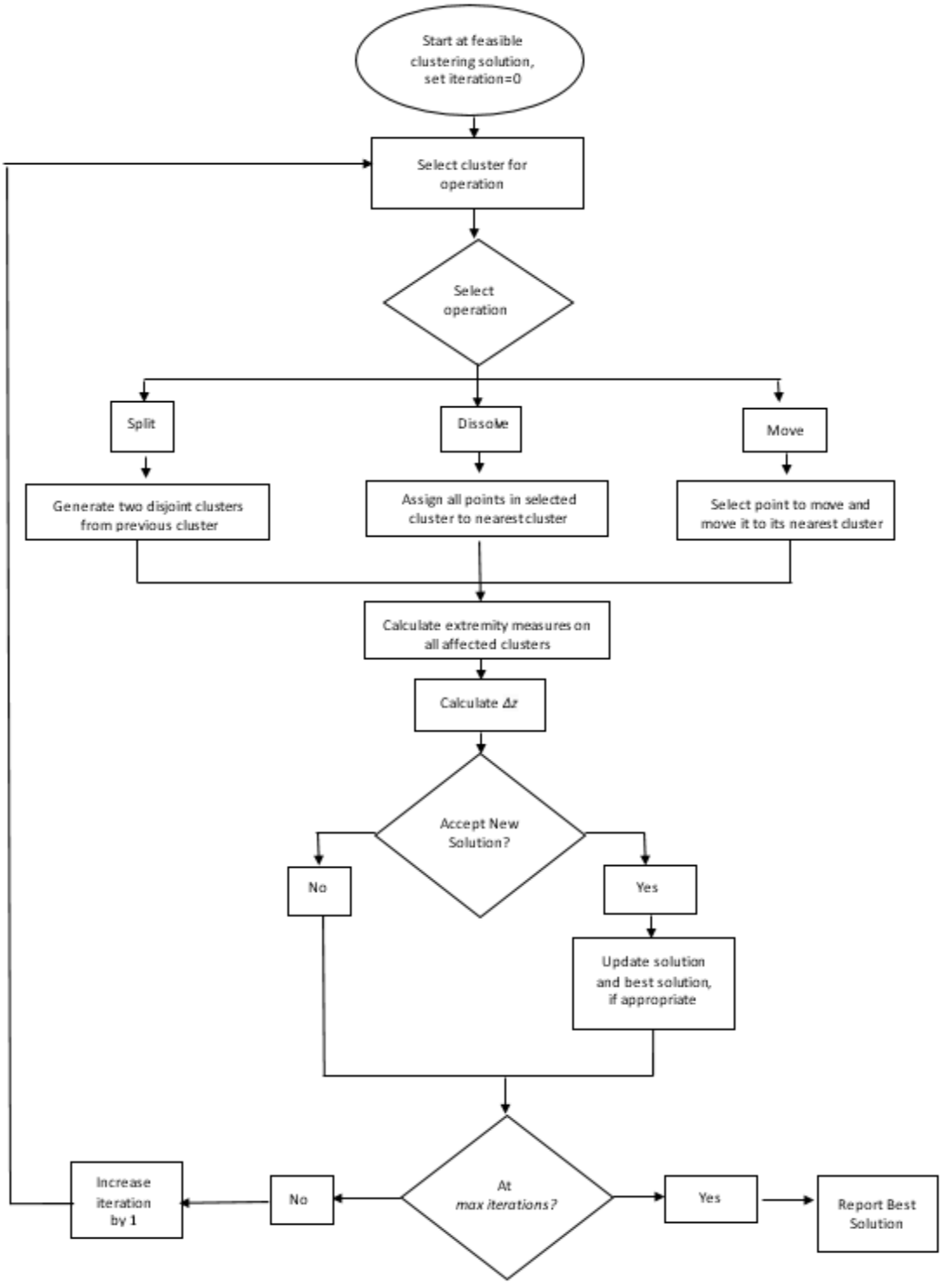

Figure 3.6 Flowchart of SAME

3.4.1 Generating New Solutions

The goal of SAME is to generate a quality solution. Thus, SAME strategically generates new solutions that are more likely to give better results. At each iteration of SAME, a cluster is selected for operation. Since the goal is to minimize the sum of each cluster extremity measure score, intuitively the clusters that contribute the most to the total score should have a higher chance of changing. As such, the probability of some cluster p being selected is: $\frac{E_{C_p}}{\sum_{p=1}^{k} E_{C_p}}$. This allows for all clusters to be potentially selected for some change, while targeting the worse clusters more frequently. For example, recall clustering solution 1. The extremity measure values of C_1 and C_2 were respectively 22.94 and 25.55 with a cumulative value of 48.49 Thus, C_1 and C_2 have a $\frac{22.94}{48.49}$ =47.30% and $\frac{25.55}{48.49}$ =52.70% chance of being selected, respectively.

Once a cluster is selected, one of three types of neighboring solutions is selected. A move, split, or dissolve operation is probabilistically chosen based on two factors: the number of points in the cluster and the cluster's extremity measure. A split operation is more likely when a cluster has more points. A split operation is also more likely if the cluster has a bad extremity measure. A dissolve operation is more likely if it has a smaller number of points, and thus the whole cluster should be disbanded. A dissolve operation is also more likely if a cluster has a bad extremity measure.

Since the probability of the three operations must sum to one, the move probability is just 1 minus the split and dissolve probabilities. In general, moves are likely when the selected cluster has a good extremity measure and should consider a smaller change and not disband the cluster entirely. One can manipulate these probabilities according to the dataset characteristics. The probabilistic functions used by SAME are discussed in Chapter 4.

A cluster that selects a move, must find which point within the cluster it should move. In SAME, the point selected to be moved is a probabilistic function based on each point's extreme distance. That is, the probability that point x_i is selected to move equals $e_i / \sum_{i:x_i \in C} e_i$. Note that this concept is similar to the initial cluster selecting probability function. If SAME chose a move operation for C_1 in clustering solution 1, x_1's chance of moving is 0.42/9.06 = 4.63%. Note that this function does not allow any interior points to be selected. For example, x_4 has no chance of being selected since e_4=0. This helps ensure that the convex hulls of our clustering solutions do not intersect each other as that would trivially be seen as a bad partitional clustering solution.

A cluster that selects a dissolve operation, divvies its points up between every other cluster. Each point in the cluster to be dissolved, finds which point it is closest to between all the other clusters and joins that cluster. A dissolve operation can be viewed as multiple move operations occurring simultaneously. Each point within the cluster moves to the most similar cluster. See Figure 3.4 for a graphical depiction of this operation.

A cluster that selects a split operation, partitions its points into two new clusters. To obtain these two split clusters, an integer program is created that minimizes the sum of all distances between points within each split cluster. Each data point in the cluster has a decision variable, f_i, which is binary and identifies which of the two new formed clusters this point is assigned. Additionally, $g_{i,j}$, is a decision variable that is also binary and takes a value of 1 if *i* and *j* are in the same new cluster and 0 otherwise. Formally, the split IP is:

Minimize $\sum_{x_i,x_j \in C: i<j} d_{i,j} * g_{i,j}$

Subject to: $\sum_{x_i \in C} f_i \geq m+2$

$\sum_{x_i \in C} f_i \leq |C|-(m+2)$

$(1- f_i)+(1- f_j) + (1- g_{i,j}) \leq 1$ for all $i,j: x_i, x_j \in C$ and $i<j$

$f_i + f_j + (1- g_{i,j}) \leq 1$ for all $i,j: x_i, x_j \in C$ and $i<j$

$f_i + (1- f_j) + g_{i,j} \leq 1$ for all $i,j: x_i, x_j \in C$ and $i<j$

$(1- f_i) + f_j + g_{i,j} \leq 1$ for all $i,j: x_i, x_j \in C$ and $i<j$

$f_i \in \{0,1\}$ for all $i: x_i \in C$

$g_{i,j} \in \{0,1\}$ for all $i,j: x_i, x_j \in C$ and $i<j$.

The objective function minimizes the sum of the distances between all points in each cluster. Recall the example shown in Figure 3.5. In this example, all distances of points within C'_2 as well as all distances between points within C'_3 contribute to the objective function. The first two constraints seek to maintain full dimensionality when running EDLP once these two new clusters are formed. If these constraints were not in place, a solution to the split IP could result in a cluster that is not fully dimensional. This situation would yield infeasible results in EDLP. The next four constraints have to deal with forcing the f_i and $g_{i,j}$, variables to link together. These constraints force that if f_i and f_j are either both 0 or if they are both 1, then $g_{i,j}$, is forced to be 1. Consequently, the objective function increases by the distance between any i,j such that $g_{i,j}$, =1. Any x_i such that f_i =0 is assigned to a new cluster, while any x_i such that f_i =1 is assigned to the other new cluster.

3.4.2 Solution Acceptance

Recall, simulated annealing is a popular technique used to generate good solutions. The general concept of simulated annealing is to have some probability of accepting a neighboring

solution that is worse than the current one. This allows for heuristics to move out of local optimal solutions. The probability of accepting the new solution is based on two things: the score difference, and the "temperature," *T*. In SAME, the score difference is defined as Δz where $\Delta z = (z_{t+1} - z_t)/z_t$.The greater the relative score increase, the lower the acceptance probability. Alternatively, if $\Delta z<0$, then the new solution should be accepted. An initial temperature, T_0, is chosen to start the simulated annealing process. As the iterations increase, the temperature decreases. Formally, $T_{t+1} = T_t * c$, where *c* is a coefficient of annealing that is slightly below 1. The acceptance probability is formally:

$$\text{P (Accepting } \mathcal{C}_{t+1}) = \begin{cases} 1, \text{if } \Delta z < 0 \\ e^{\frac{-|\Delta z|}{T_t}}, \text{if } \Delta z \geq 0 \end{cases}$$

Any accepted solution is checked to see if it is better than the current best solution. If it is better, then it replaces that solutions. That way when SAME terminates, it can report the best solution, which is not necessarily the current solution at termination. The specifics of how SAME was implemented to test its merit are provided in Chapter 4.

Chapter 4: Implementation and Computational results

SAME is a general framework to create clustering solutions. This chapter describes this research's particular implementation of SAME and efforts made to reduce its computational effort. In addition, the chapter presents some computational results of SAME on some clustering benchmark sets from Sieranoja and Fränti (2018) and Dua and Graff (2019).

SAME was implemented in the Python 2.7.14 programming language. For subroutines using linear or integer programming, Python interfaced with ILOG CPLEX Optimization Studio version 12.8. All computational studies were performed on a 3.84 GHz computer with 8 GBs of RAM.

This chapter begins with implementation adjustments that are made to SAME in large data set instances. Next, the key implementations of EDLP are described. The functions used to define the move, dissolve, and split probabilities are provided. Finally, computational results of SAME are compared to the *k*-means algorithm.

4.1 Eliminating Integer Programs

In some instances, the integer program involving the initial clustering solution can be very cumbersome. This integer program presented in section 3.3 requires decision variables for all n points in a dataset, as well as a decision variable for every i,j combination of points within the dataset such that $i<j$. This requires $|N| + \frac{|N|^2-|N|}{2}$ decision variables. Additionally, $|N|^2 - |N| + 1$ constraints are required to link these variables and maintain the distance threshold. As $|N|$ increases, this implementation is not computationally efficient enough to warrant the time spent setting up an initial feasible solution. Some large datasets studied resulted in the computer crashing while trying to solve the integer program. In this research's computational studies, datasets where $|N|>150$ implemented an alternate approach.

In these scenarios, an agglomerative clustering approach is taken. This routine requires a desired amount of clusters, *k*, as input, and some bin size, ε. The clustering routine starts with *n* clusters each of size 1. The heuristic searches through all the points, and any points whose distances are between 0 and ε are joined into the same cluster. If those two points had other points that they had previously been clustered to, then all of their previously clustered points are also combined into a single cluster. Once the distance matrix has been traversed, the current upper threshold becomes the new lower threshold and the upper threshold is doubled. If ε is set too high so all the points fit into a single cluster on the first iteration, then ε is halved. This continues until the number of clusters has been reduced to *k* or less. SAME runs many iterations to reduce the impact that this initial *k* input has on the final solution.

A similar problem arises in the split IP presented in section 3.2.3. This integer program requires decision variables for all points in a cluster, as well as a decision variable for every *i,j* combination of points within the cluster such that $i<j$. This requires $|C| + \frac{|C|^2-|C|}{2}$ decision variables. Additionally, $4*(|N|^2 - |N|) +2$ constraints are required to link these variables and maintain full dimensionality of the new clusters. Though this integer program is less time intensive than the initialization IP, as $|N|>|C|$, the split IP can be implemented numerous times during a single application of SAME. As such, this research implements an alternate approach whenever $|N|>150$.

A heuristical approach to split is taken in these larger instances. Since the split operation creates two clusters, the two furthest points within a cluster, *p* and *q*, are identified and put into separate clusters. These two points each then find the closest points to them throughout the rest of the original cluster points and join with them until each disjoint cluster could be fully dimensional. Like the IP, this requires $m+2$ points in each cluster. At this point, the heuristic

attempts to segment the rest of the points optimally. As such, each remaining point is joined to either cluster based on whether it is closer to point *p* or point *q*.

4.2 Efficiency in Calculating the Extreme Distance

During an iteration, EDLP could be run as many as *n* times if the cluster is large. If one ignored the idea of neighborhoods and reran every point in every cluster in every new clustering solution, it could take a long time to find the new objective value. This research stores all the points extreme distances and decision variable values for all α^+ and α^-. This allows SAME to only update the new clustering solution from the previous solution instead of resolving everything. Obviously, any unaffected clusters do not require recalculating their extreme distances.

To reduce the time spent evaluating the extreme distances of a cluster, the coefficient basis, right hand, and objective function are all uploaded into CPLEX prior to calculating a cluster's score. The initial basis assumes no particular x_q and instead includes all points in the initial formulation. In addition to the formula given for EDLP in section 3.1, a constraint is added $\sum_{i:\, x_i \in C} 0*(\alpha_i^+ - \alpha_i^-) = 0$ that is initially useless. To calculate the extreme distance of each point in the cluster, only three alterations per point are made to this formulation: the right hand side, the objective function and this added constraint. To test a point the right hand side of the first *m* constraints is adjusted to $x_{q,i}$ for all i=1,…,*m.* The objective function's coefficients are adjusted to $d_{i,q}$ for every α_i^- variable. Additionally, the coefficients for α_q^+ and α_q^- are set to 1 in the last constraint, which forces these variables to take a value of 0 in the solution. This is important so a point cannot take itself for a weight when being tested. Consequently, only $|C|+m+2$ adjustments are needed to the uploaded EDLP to calculate a new point's extreme distance. Thus, EDLP can rapidly calculated the extreme distance for each point in the cluster.

Since checking stored solution values is much faster than running linear programs, some preprocessing can help improve the computational speed of EDLP. One strategy is to recognize that if a point's extreme distance is 0 and the cluster is expanding through either a move or dissolve operation, then that point's extreme distance is still 0, and there is no need to solve an EDLP. For example, given a clustering solution $\mathcal{C}$ let SAME attempt to move point x_q from C_1 to C_2. Any point in C_2 with an extreme distance of 0 before the move is still 0 after the move. This is because it can maintain the same convex combination of α_i's with no weight taken for $\alpha_q^+=0$ and $\alpha_q^-=0$. This is intuitive and does not require an EDLP instance to be run.

Another strategy is to realize that any point in C_1 that did not use α_q in its stored convex combination solution from EDLP when that cluster was formed at a previous iteration still trivially can take the same convex combination of α_i's. That is, if it its optimal solution has $\alpha_q^+=0$ and $\alpha_q^-=0$, then its solution will not change. Removing one point from a large cluster does not change the boundary that much so a large majority of instances these two checks can significantly reduce the number of EDLPs solved during the move operation.

4.3 Probabilistic Functions for Selecting an Operation

Recall that a selected cluster's operation is probabilistically chosen based on two factors: the number of points in the cluster and the clusters' extremity measure. A split operation is more likely when a cluster has more points and a bad extremity measure. A dissolve operation is more likely if a cluster has less points and a bad extremity measure. Moves are likely when the selected cluster has a good extremity measure. SAME assumes a probability function based on these factors and for simplicity sake assumes they are independent. Since the probability functions are a function of two variables, three independent functional values are necessary to create a plane that can define a probability of choosing an operation based on $|C|$ and E_c. Note that these probability functions are the probabilities for attempting an operation, not necessarily

accepting the new neighboring solution. Table 4.1 provides an example clustering solution, where m=2, that is used to demonstrate these probabilities.

Cluster	$\|C_i\|$	E_c
1	4	26
2	6	26
3	10	45
4	15	48
5	15	60

Table 4.1 Clustering Solution for Operation Probabilities Example

For a split, a cluster must have at least 2*(m+2) to hopefully maintain full-dimensionality of the each cluster. As such, any cluster smaller than this should has a 0% chance of splitting regardless of the quality of the cluster. If a cluster has a great score, the probability should be lower. Consequently, the best cluster score has a 0% chance, even if it is also the largest cluster. The 3rd functional value used by SAME sets an upper bound set on the split probability, which occurs at the biggest cluster with the worst score. SAME uses 25% as the max split probability. Table 4.2 explicitly gives the three functional values. Using linear algebra a function, $P(split)=f(|C|, E_c)= \beta_{split}*|C|+\gamma_{split}* E_c +\delta_{split}$, is created that holds all 3 functional values true. In general, $\beta_{split}=\frac{0.25}{Max_p\{|Cp|\}-(2*(m+2)-1)}$, $\gamma_{split}=\frac{0.25}{Max_p\{Ep\}-Min_p\{Ep\}}$, and

$$\delta_{split}=.25\left(1-\frac{Max_p\{|Cp|\}}{Max_p\{|Cp|\}-(2*(m+2)-1)}-\frac{Max_p\{Ep\}}{Max_p\{Ep\}-Min_p\{Ep\}}\right).$$

$\|C\|$	E_c	Chance of Split
2*(m+2)-1	$Max_p\{E_p\}$	0%
$Max_p\{\|C_p\|\}$	$Min_p\{E_p\}$	0%
$Max_p\{\|C_p\|\}$	$Max_p\{E_p\}$	25%

Table 4.2 SAME's Functional Values for the Split Probability Function

For the clustering solution $\mathscr{C}$ given in Table 4.1, 2*(m+2)-1=7, $Min_p\{|C_p|\}$=4, $Max_p\{|C_p|\}$=15, $Min_p\{E_p\}$=25, and $Max_p\{E_p\}$=60. Consequently, for this example $\beta_{split}=\frac{1}{32}$, $\gamma_{split}=\frac{1}{136}$, , $\delta_{split}=\frac{-113}{544}$. Intuitively, β_{split} and γ_{split} should always be greater than 0 since the chance goes up with an increase in cluster points and extremity measure. Since there is a chance that a cluster's probability given this function is less than 0, any number below 0 is given a 0% chance. Table 4.3 gives the split probabilities for all of the clusters in $\mathscr{C}$. Note that clusters 1 and 2 have no chance since they are too small. Also note that cluster 5 has a 25% chance since it is the biggest and worst cluster. Additionally, clusters 3 and 4 are somewhere between 0-25% since they are not extreme in at least one of the criterion.

Cluster	P(Split)
1	0%
2	0%
3	2%
4	16%
5	25%

Table 4.3 Split Probabilities for Clustering Solution

In general, a cluster's extremity measure and its amount of points vary proportionally. As such, the smallest, best cluster has a 5% chance of dissolving as does the biggest, worst cluster. This creates a line that defines the "average" probability of dissolve at 5%. An upper bound is again set on the max probability. In SAME, the smallest cluster with a bad extremity measure should have a max probability of 25%. Table 4.4 explicitly gives the three functional values. Using linear algebra a function, P(Dissolve)=$f(|C|, E_c)$= $\beta_{dissolve}*|C|+\gamma_{dissolve}* E_c +\delta_{dissolve}$, is created that holds all 3 functional values true. In general, $\beta_{dissolve}=\frac{-(0.25-0.05)}{(Max_p\{|Cp|\}-Min_p\{|Cp|\})}$, $\gamma_{dissolve}=\frac{(0.25-0.05)}{(Max_p\{Ep\}-Min_p\{Ep\})}$, $\delta_{dissolve}= 0.25 + \frac{(0.25-0.05)(Min_p\{|Cp|\})}{(Max_p\{|Cp|\}-\mathrm{Min}\{|Cp|\}} - \frac{(0.25-0.05)(Max_p\{Ep\})}{(Max_p\{Ep\}-Min_p\{Ep\})}$.

$\|C\|$	E_c	Chance of Dissolve
$Min_p\{\|C_p\|\}$	$Max_p\{E_p\}$	5%
$Max_p\{\|C_p\|\}$	$Min_p\{E_p\}$	5%
$Min_p\{\|C_p\|\}$	$Max_p\{E_p\}$	25%

Table 4.4 SAME's Functional Values for the Dissolve Probability Function

Consequently, for the clustering solution $\mathcal{C}$ given in Table 4.1 $\beta_{dissolve}=\frac{-1}{55}$, $\gamma_{dissolve}=\frac{1}{180}$, $\delta_{dissolve}=\frac{-113}{3740}$. In general, $\beta_{dissolve}$ is less than 0 and $\gamma_{dissolve}$ is greater than 0 because the probability decreases with more points and increases with a bigger extremity measure. Table 4.5 gives the dissolve probabilities for all of the clusters in $\mathcal{C}$. Note that cluster 1 has a higher chance than cluster 2 because it is both smaller with the same extremity measure. Also note that cluster 5 has a higher chance than cluster 4 because it has a worse score with the same number of points.

Cluster	P(Dissolve)
1	5%
2	1%
3	6%
4	4%
5	5%

Table 4.5 Dissolve Probabilities for Clustering Solution

Since the probabilities must add to one, the probability of attempting a move operation is just a function of the split and dissolve probabilities. For the clustering solution, these probabilities are given in Table 4.6. Because of the above methods, SAME attempts many more moves than splits and dissolves since there is at least a lower bound of 50% probability of choosing a move. Based on experimentation, moves were attempted close to 90% of the time whereas splits and dissolves were each attempted in approximately 5% of iterations.

Cluster	P(Move)
1	95%
2	99%
3	92%
4	83%
5	70%

Table 4.6 Move Probabilities for Clustering Solution

4.4 Computation Results

SAME was tested on benchmark data sets from (Sieranoja & Fränti, 2018) and (Dua & Graff, 2019). The data sets and their characteristics are contained in Table 4.7. SAME was compared to a *k*-means implementation that uses *k* random points in the data set as the starting centroids. For comparisons sake, the same parameters were chosen for the same data set. That is, *max iterations*=5000 and λ is some constant for both *k*-means and SAME. The weighting constant, λ, varies depending on different characteristics of the data set. It particularly has to do with the density of a cluster. As such, each data set has some range of λ that give better results. This research recommends $0.01 \leq \lambda \leq 10$. The particular value used per data set is given in Table 4.7. Additionally, all runs of SAME used T_0=20 with c=0.99 for its cooling process. On the first iteration, a 1% increase in objective function value has a 99% acceptance probability. At iteration 720, a 1% increase has an acceptance probability of 50%. By iteration 5000 this probability has dropped to <0.0001%. These parameters, T_0 and c, can be adjusted to change how quickly this cooling process occurs.

SAME and *k*-means were both compared using the extremity measure and silhouette index. SAME, initially was implemented using the extremity measure to accept new solutions and generate new solutions. The principles of SAME could be optimized for any of the other criterion as a basis for accepting or rejecting new solutions and storing the best solution.

Data Set	Data Points	Dimensions	Number of clusters	λ used	Source(s)
A1	3000	2	20	0.01	(Sieranoja & Fränti, 2018)
32dim	1024	32	16	0.1	(Sieranoja & Fränti, 2018)
Buddy Move	249	6	Unknown	1	(Anjali & Renjith, 2014)
Breast Cancer	116	9	Unknown	1	(Patrício, et al., 2018)
Breast Tissue	106	9	Unknown	1	(Dua & Graff, 2019)
Faculty Perceptions	913	53	Unknown	0.5	(Dua & Graff, 2019)
Forest Fires	517	9	Unknown	2	(Morais & Cortez, 2007)
Happiness Survey	143	7	Unknown	0.5	(Dua & Graff, 2019)
Immunotherapy	90	8	Unknown	0.5	(Khozeimeh, et al., 2017)
Iris	150	4	3	0.01	(Sieranoja & Fränti, 2018)
Istanbul Stock Exchange	536	8	Unknown	1	(Akbilgic, Bozdogan, & Balaban, 2013)
Sales Transactions	811	53	Unknown	10	(Tan & San Jau, 2014)
Seeds	210	7	Unknown	0.5	(Dua & Graff, 2019)
Stone Flakes	79	8	Unknown	1	(Dua & Graff, 2019)
Travel Reviews	980	11	Unknown	2	(Shini, Sreekumar, & S, 2018)
User Knowledge Modeling	403	5	Unknown	1	(Dua & Graff, 2019)
Vertebrae	310	6	Unknown	5	(Dua & Graff, 2019)
Wholesale Customers	440	6	Unknown	0.5	(Dua & Graff, 2019)
Wine	178	13	3	1	(Sieranoja & Fränti, 2018)
Yeast	1484	10	8	0.5	(Sieranoja & Fränti, 2018)

Table 4.7 Experimental Data Sets

SAME was implemented to optimize the extremity measure. Many of these data sets required the agglomerative approach to form initial clusters as the size was too big to run the initialization IP. In data sets with a known number of clusters, this was selected as the initial number of clusters. In sets with an unknown *k*, some initial random *k* was chosen between 4 and 12. Since SAME was run for 5000 iterations, the impact these initial clusters had on the final solution has minimal. For a like-for-like comparison, *k*-means was implemented with the same *k* value as the clustering solution generated by SAME. SAME and *k*-means are compared in Table 4.8.

		SAME		*k*-means		
Data Set	Number of Clusters	Time (s)	EM	Time (s)	EM	% Decrease
A1	20	4841	136846	36	163711	16%
32dim	16	8601	78088	25	84283	7%
Buddy Move	8	2291	14580	2	15018	3%
Breast Cancer	3	2380	26135	2	29964	12%
Breast Tissue	3	1892	690105	1	1736472	60%
Faculty Perceptions	3	49012	28848	83	27771	-4%
Forest Fires	11	7905	53561	6	57843	8%
Happiness Survey	3	3322	277	2	280	1%
Immunotherapy	3	1303	4640	1	7951	42%
Iris	3	1331	132969790	2	288284098	54%
Istanbul Stock Exchange	3	7106	21	9	22	5%
Sales Transactions	4	17461	7345	49	16894	56%
Seeds	3	4905	268	2	289	7%
Stone Flakes	3	1679	4357	1	4886	11%
Travel Reviews	4	8267	758	5	779	3%
User Knowledge Modeling	15	403	197	3	199	1%
Vertebrae	14	1527	36623	18	39216	7%
Wholesale Customers	3	440	1318096	5	1408957	7%
Wine	3	4623	13073189	3	13201260	1%
Yeast	5	1484	186	21	70	-165%
Average	**6.5**	**6538**		**14**		**17%**

Table 4.8 Comparing SAME to *k*-means using the Extremity Measure

In 90% of the instances the final clusters provided by SAME were better than that of the *k*-means algorithm. Though these instances took approximately 2 hours on average to run versus less than a minute, better clustering solutions were generated. In the instances where SAME was better, an average extremity measure improvement of 17% was seen with the % improvement defined as $\frac{EM_{k-means}-EM_{SAME}}{EM_{k-means}}$*100%. However, in some cases where the final clustering solution generated by SAME was better using the extremity measure, it was not better in other common

clustering measures. As such, an additional versions of SAME can be created based on the measure.

In an alternate implementation, SAME attempted to optimize the silhouette index. In this implementation, Δz was a function of the difference in the silhouette index, rather than the extremity measure, between the current and previous clustering solutions. However, the extreme distance and extremity measure concepts were still used to generate new solutions; the measure was just used as a new criterion to accept or reject new solutions and update the best solutions. For a like-for-like comparison, *k*-means clusters were run on the same *k* values as the optimal clusters generated by SAME. Table 4.9 gives the time and score comparison of silhouette index values generated between SAME and *k*-means. In 90% of instances SAME performed better. Though these instances took approximately 4 hours on average to run versus less than a minute, better clustering solutions were generated. In the instances where SAME was better, an average silhouette index improvement of 53% was seen with the % improvement defined as $\frac{SI_{SAME}-SI_{k-means}}{SI_{k-means}}$*100%.

In summary, using both the extremity measure and the silhouette index, SAME produced better solutions than the *k*-means algorithm. Although SAME requires more time, this tradeoff appears to be valuable. This research has shown that estimating the polyhedral distance with linear programs can be an effective method to evaluate and generate clustering solutions.

		SAME		*k*-means		
Data Set	Number of Clusters	Time (s)	SI	Time (s)	SI	% Increase
A1	20	6201	2455	36	2335	5%
32dim	16	9872	976	25	828	18%
Buddy Move	5	4026	135	2	107	26%
Breast Cancer	6	1370	100	1	79	27%
Breast Tissue	7	1162	89	1	104	-14%
Faculty Perceptions	4	71432	370	68	365	1%
Forest Fires	6	10044	429	7	378	13%
Happiness Survey	3	2875	42	1	29	45%
Immunotherapy	4	1699	73.07	1	32	128%
Iris	11	927	104	2	24	333%
Istanbul Stock Exchange	8	6819	347	6	152	128%
Sales Transactions	3	14056	169	61	90	88%
Seeds	4	4782	130	2	129	1%
Stone Flakes	4	475	43	1	40	8%
Travel Reviews	13	15681	565	12	454	24%
User Knowledge Modeling	15	4542	279	3	231	21%
Vertebrae	9	3254	236	3	166	42%
Wholesale Customers	7	1218	91	6	74	23%
Wine	7	2039	74.9	2	83	-10%
Yeast	7	117291	888	29	763	16%
Average	**7.95**	**13988**		**14**		**53%**

Table 4.9 Comparing SAME to *k*-means using Silhouette Index

Chapter 5: Conclusions and Future Research

This thesis has three primary contributions. The first is a new clustering measure called the extremity measure. The extremity measure was developed to incorporate concepts of polyhedral theory into clustering without solving quadratic programs, so the score of a newly generated solutions could be calculated quickly. This is in part aided by the implementation of EDLP where only the number of points in a cluster plus the number of dimensions in the data set adjustments to an existing EDLP are necessary to calculate the extreme distance of each additional point in a cluster. Since EDLP approximates the distance from a point to the remaining convex hull of a cluster, oblong clusters have the ability to score well with the extremity measure, which is not necessarily the case with other measures.

The second contribution is a new heuristic named the Simulated Annealing using Measurements of Extremity (SAME) heuristic. SAME uses three types of operations to generate new solutions. One benefit of SAME is its ability to reduce or increase the number of clusters throughout the heuristic using split and dissolve operations whereas many clustering methods, like the *k*-means algorithm, require an input value for the number of clusters which cannot change.

Additionally, SAME generated quality clustering solutions. In 90% of instances, SAME had a better extremity measure score than the final formation of the *k*-means algorithm implemented. SAME was also optimized for the silhouette index by accepting, rejecting and keeping track of the best solutions based on this measure. SAME performed better than *k*-means in 90% of cases when optimizing according to the silhouette index. This leads one to determine that using SAME and its simulated annealing principles has merit in clustering and can be extended to various other clustering measures.

Using linear programming instead of quadratic programming allows for this research's concepts to be applied to big data. Notice that the datasets used were not big data instances. However, the resultant clusters can be used to test large amounts of data points versus existing clusters very fast using EDLP. An experiment was created where points were tested against a cluster with 10 dimensions and 20 extreme points. EDLP was able to test whether or not data points belonged to this cluster at a rate of 10,000 data points per second. Additionally testing whether a point is inside or outside of a current cluster's convex hull only requires the extreme points. So variables are only needed for these extreme points instead of needing variables for every point within a cluster.

5.1 Future Research

This research's main motivation was to use linear programming to quickly approximate how close a data point is from the convex hull of a set a points. Since this is already an approximation, solving EDLP quicker while maintaining some degree of accuracy may be of use. This would allow SAME to be applied to even larger datasets and generate solutions faster. EDLP could use a reduced number of variables that coincide with the extreme points of a cluster, instead of using all the points within a cluster. This should not impact the objective value much, because EDLP's solutions generally assigned the decision variables corresponding to interior points to be zero.

The extremity measure heavily relied on the λ value chosen to weigh how much the central distance versus the extreme distance should impact the overall cluster score. SAME should be applied to clustering sets with given good clustering solutions, not just their scores. Then SAME could have better approximations on what λ value should be chosen based on the size of the clusters, the amount of clusters, and how dense clusters are for a given data set. Additionally, the effects of normalizing data should be studied. Since the extremity measures

uses the raw data's distances, the values grew quite large in some of the benchmark instances studied. Normalization could also help fine tune SAME so that each parameter in a data set had a more relevant impact on the clustering solution and keep the extremity measure solution in a more normal range.

Lastly, this research could be further expanded into big data for using SAME to establish initial clusters and then use EDLP to test incoming data rapidly. If each cluster had a parallel processor responsible for it, then each incoming data point could be rapidly tested for whether it fits within that existing cluster.

References

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. *27*(2), 94-105.

Akbilgic, O., Bozdogan, H., & Balaban, M. (2013). *A novel Hybrid RBF Neural Networks model as a forecaster*. PhD Thesis, Istanbul University. doi:10.1007/s11222-013-9375-7

Aloise, D., Hansen, P., & Du Gerad, L. C. (2007). On the complexity of minimum sum-of-squares clustering.

Anjali, S., & Renjith, C. (2014). A personalized mobile travel recommender system using hybrid algorithm. *2014 First International Conference on Computational Systems and Communications*, 12-17.

Basalto, N., Bellotti, R., De Carlo, F., Facchi, P., & Pascazio, S. (2005). Clustering stock market companies via chaotic map synchronization. *Physica A: Statistical Mechanics and its Applications, 345*(1-2), 196-206.

Berkhin, P. (2006). A survey of clustering data mining techniques. *In Grouping multidimensional data*, (pp. 25-71). Springer, Berlin, Heidelberg.

Bezdek, C., Ehrlich, R. J., & Full, W. (1984). FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences, 10*(2-3), 191-203.

Bouldin, D., & Davies, D. (1979). A cluster separation measure. *IEEE PAMI, 1*(2), 224–227.

Brown, D. E., & Huntley, C. L. (1992). A practical application of simulated annealing to clustering. *Pattern recognition, 22*(4), 401-412.

Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence, 17*(8), 790-799.

Chipman, H., & Tibshirani, R. (2005). Hybrid hierarchical clustering with applications to microarray data. *Biostatistics, 7*(2), 286-301.

Chuang, K. S., Tzeng, H. L., Chen, S., Wu, J., & Chen, T. J. (2006). Fuzzy c-means clustering with spatial information for image segmentation. *computerized medical imaging and graphics, 30*(1), 9-15.

Colorni, A., Dorigo, M., Maniezzo, V., & Trubian, M. (1994). Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science, 34*(1), 39-53.

D'Urso, P., De Giovanni, L., Disegna, M., & Massari, R. (2013). Bagged clustering and its application to tourism market segmentation. *Expert Systems with Applications, 40*(12), 4944-4956.

Dantzig, G. (1947). Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 339–347.

Dolnicar, S. (2003). Using cluster analysis for market segmentation-typical misconceptions, established methodological weaknesses and some recommendations for improvement. *Australasian Journal of Market Research, 11*(2), 5-12.

Dua, D., & Graff, C. (2019). (C. U. Irvine, Producer) Retrieved from UCI Machine Learning Repository: http://archive.ics.uci.edu/ml

Duczmal, L., & Assuncao, R. (2004). A simulated annealing strategy for the detection of arbitrarily shaped spatial clusters. *Computational Statistics & Data Analysis, 45*(2), 269-286.

Eglese, R. W. (1990). Simulated annealing: a tool for operational research. *European journal of operational research, 46*(3), 271-281.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093-2229). Springer, Boston, MA.

Güngör, Z., & Ünler, A. (2008). K-harmonic means data clustering with tabu-search method. *Applied Mathematical Modelling, 32*(6), 1115-1125.

Halkidi, M. B. (2001). On clustering validation techniques. *Journal of intelligent information systems, 17*(2-3), 107-145.

Hartigan, J. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 28*(1), 100-108.

Hellerstein, J. M. (1999). Interactive data analysis: The control project. *Computer, 32*(8), 51-59.

Ingber, L. (1993). Simulated annealing: Practice versus theory. *Mathematical and computer modelling, 18*(11), 29-57.

Jain, A. K. (1999). Data clustering: a review. *ACM computing surveys (CSUR), 31*(3), 264-323.

Kanungo, T. M. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis & Machine Intelligence, 7*, 881-892.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *In Proceedings of the sixteenth annual ACM symposium on Theory of computing* (pp. 302-311). ACM.

Kayarat, D. (2005). *The Quadratic Polyhedral Clustering Algorithm: A New Method to Cluster Microarray Data.* Doctoral dissertation, Kansas State University.

Khozeimeh, F., Alizadehsani, R., Roshanzamir, M., Khosravi, A., Layegh, P., & Nahavandi, S. (2017, 2 1). An expert system for selecting wart treatment method. *Computers in Biology and Medicine, 81,* 167-175.

Khozeimeh, F., Azad, F. J., Oskouei, Y. M., Jafari, M., Tehranian, S., & Alizadehsani, R. (2017). Intralesional immunotherapy compared to cryotherapy in the treatment of warts. *International Journal of Dermatology*.

Kim, B. I., Heragu, S. S., Graves, R. J., & Onge, A. S. (2003). Clustering-based order-picking sequence algorithm for an automated warehouse. *International Journal of Production Research, 41*(15), 3445-3460.

Larson, J. S., Bradlow, E. T., & Fader, P. S. (2005). An exploratory look at supermarket shopping paths. *International Journal of research in Marketing, 22*(4), 395-414.

Lin, S. W., Vincent, F. Y., & Lu, C. C. (2011). A simulated annealing heuristic for the truck and trailer routing problem with time windows. *Expert Systems with Applications, 38*(12), 15244-15252.

Lingras, P., Hogo, M., Snorek, M., & West, C. (2005). Temporal analysis of clusters of supermarket customers: conventional versus interval set approach. *Information Sciences, 172*(1-2), 215-240.

Liu, Y., Li, Z., Xiong, H., Gao, X., & Wu, J. (2010, December). Understanding of internal clustering validation measures. *2010 IEEE 10th International Conference* (pp. 911-916). Data Mining (ICDM).

Matas, J., & Kittler, J. (1995, September). Spatial and feature space clustering: Applications in image analysis. *International Conference on Computer Analysis of Images and Patterns*, (pp. 162-173). Berlin, Heidelberg.

McAfee, A., Brynjolfsson, E., Davenport, T. H., Patil, D. J., & Barton, D. (2012). Big data: the management revolution. *Harvard business review, 90*(10), 60-68.

Mell, P., & Grance, T. (2009). The NIST definition of cloud computing. *National institute of standards and technology, 53*(6), 50.

Merceron, A., & Yacef, K. (2004, August). Clustering students to help evaluate learning. *IFIP World Computer Congress, TC 3*, (pp. 31-42). Springer, Boston, MA.

Morais, P., & Cortez, A. (2007). A Data Mining Approach to Predict Forest Fires using Meteorological Data. *New Trends in Artificial Intelligence.* Portugues: J. Neves, M. F. Santos and J. Machado.

Nemati, H. R., Steiger, D. M., Iyer, L. S., & Herschel, R. T. (2002). Knowledge warehouse: an architectural integration of knowledge management, decision support, artificial intelligence and data warehousing. *Decision Support Systems, 2*, 143-161.

Olson, C. (1995). Parallel algorithms for hierarchical clustering. *Parallel Computing, 21*, 1313-1325.

Osman, I. H., & Christofides, N. (1994). Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research, 1*(3), 317-336.

Özdamar, L., & Demir, O. (2012). A hierarchical clustering and routing procedure for large scale disaster relief logistics planning. Transportation Research. *Part E: Logistics and Transportation Review, 48*(3), 591-602.

Patrício, M. P., Crisóstomo, J., Matafome, P., Gomes, M., Seiça, R., & Caramelo, F. (2018). Using Resistin, glucose, age and BMI to predict the presence of breast cancer. *BMC Cancer, 18*(1).

Picard, J. C., & Ratliff, H. D. (1978). A cut approach to the rectilinear distance facility location problem. *Operations Research, 26*(3), 422-433.

Punj, G., & Stewart, D. W. (1983). Cluster analysis in marketing research: Review and suggestions for application. *Journal of marketing research, 20*(2), 134-148.

Rice, J. A. (2006). *Mathematical statistics and data analysis.* Cengage Learning.

Rousseeuw, P. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math, 20*(1), 53–65.

Rujasiri, P., & Chomtee, B. (2009). Comparison of Clustering Techniques for Cluster Analysis. *Nat. Sci., 43*, 378-388.

Setnes, M., & Kaymak, U. (2001). Fuzzy modeling of client preference from large data sets: an application to target selection in direct marketing. *IEEE Transactions on Fuzzy Systems, 9*(1), 153-163.

Shini, R., Sreekumar, A., & S, J. (2018). Evaluation of Partitioning Clustering Algorithms for Processing Social Media Data in Tourism. *IEEE Recent Advances in Intelligent Computational Systems, 127*(31).

Sieranoja, P., & Fränti, S. (2018, December). K-means properties on six clustering benchmark datasets. *Applied Intelligence, 48*(12), 4743-4759.

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., De Carvalho, A. C., & Gama, J. (2013). Data stream clustering: A survey. *ACM Computing Surveys (CSUR), 46*(1), 13.

Skorin-Kapov, J., & Vakharia, A. J. (1993). Scheduling a flow-line manufacturing cell: a tabu search approach. *The International Journal of Production Research, 31*(7), 1721-1734.

Srnka, K. J., & Koeszegi, S. T. (2007). From words to numbers: how to transform qualitative data into meaningful quantitative results. *Schmalenbach Business Review, 59*(1), 29-57.

Tan, S. C., & San Jau, J. P. (2014). Time series clustering: A superior alternative for market basket analysis. *Proceedings of the First International Conference on Advanced Data and Information Engineering* , 241-248.

Wu, B., Zhang, Y., Hu, B. G., & Ji, Q. (2013). Constrained clustering and its application to face clustering in videos. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, (pp. 3507-3514).

Wu, Z., & Leahy, R. (1993). An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE transactions on pattern analysis and machine intelligence, 15*(11), 1101-1113.

Zadeh, L. (1965). Fuzzy sets. *Inform, and Control, 15*, 22-32.