Region-based Crossover for Clustering Problems

by

Jeevan D’Souza

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Computer Science

Graduate School of Computer and Information Sciences
Nova Southeastern University

2012
An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Region-based Crossover for Clustering Problems

by
Jeevan D’Souza
August 2012

Data clustering, which partitions data points into clusters, has many useful applications in economics, science and engineering. Data clustering algorithms can be partitional or hierarchical. The k-means algorithm is the most widely used partitional clustering algorithm because of its simplicity and efficiency. One problem with the k-means algorithm is that the quality of partitions produced is highly dependent on the initial selection of centers. This problem has been tackled using genetic algorithms (GA) where a set of centers is encoded into an individual of a population and solutions are generated using evolutionary operators such as crossover, mutation and selection. Of the many GA methods, the region-based genetic algorithm (RBGA) has proven to be an effective technique when the centroid was used as the representative object of a cluster (ROC) and the Euclidean distance was used as the distance metric.

The RBGA uses a region-based crossover operator that exchanges subsets of centers that belong to a region of space rather than exchanging random centers. The rationale is that subsets of centers that occupy a given region of space tend to serve as building blocks. Exchanging such centers preserves and propagates high-quality partial solutions.

This research aims at assessing the RBGA with a variety of ROCs and distance metrics. The RBGA was tested along with other GA methods, on four benchmark datasets using four distance metrics, varied number of centers, and centroids and medoids as ROCs. The results obtained showed the superior performance of the RBGA across all datasets and sets of parameters, indicating that region-based crossover may prove an effective strategy across a broad range of clustering problems.
Acknowledgements

This document marks the end of my PhD in computer science at the graduate school of computer and information sciences at Nova Southeastern University. It has taken me six years to achieve this challenging and enlightening task with most of this time spent on the research process. There were many obstacles in my path over the years and there were times when I almost gave up hope of completing the program. There are a number of people who helped me finish this process and I want to sincerely thank them for their efforts.

First and foremost, I want to thank my advisor, Dr. Michael Laszlo, who has been very patient with me since September 2009. I want to thank him for his advice, time, and guidance throughout this process. I also want to thank my committee members, Dr. Sumitra Mukherjee and Dr. Maxine Cohen, who have been advising and guiding me since the start of the research process. I want to also thank Dr. Eric Ackerman and Dr. Amon Seagull, who helped me resolve minor departmental frustrations I’ve had during the pursuit of my degree.

I want to thank Dr. Ronald Krawitz, who started the CISD program with me and graduated in April 2012. Dr. Krawitz and I have been in touch throughout our time together in the program and his encouragement helped me get through the program. I also want to thank Margaret Long, a present DCIS student, and Kelly Adams, who have both been very instrumental in encouraging me when I have felt frustrated and overwhelmed.

I want to thank Andrew Reed, my student at DeVry College of New York, for his help with programming during the experimentation stage of my research. I also want to thank my Dean, Dr. Tom Schmidt, who was always willing to make administrative accommodations at work for my PhD studies and allowed me to use computer facilities at DeVry to carry out my research experiments. I also want to express my gratitude to Dr. John Gibbons for formulating the revised tuition re-imbursement program at DeVry University which enabled me to afford tuition and fees.

Last of all I want to thank my mother, Stella D’Souza, who has been constantly supporting my higher educational pursuits throughout my life. I would not have accomplished this feat without her constant encouragement for the last 16 years.
Table of Contents

Abstract iii
List of Tables viii
List of Figures x

Chapters

1. Introduction 1
   Background 1
   Problem Statement 3
   Goal 4
   Research Questions 5
   Relevance and Significance 7
     k-means 8
     Region-based crossover 8
   Barriers and Issues 11
   Limitations and Delimitations 12
   Definitions of Terms 13
   Summary 17

2. Review of the literature 18
   Clustering Algorithms 18
   Genetic Algorithms 22
   Genetic Algorithms for Clustering 24
   Genetic Algorithms for Center Selection in k-means Initialization 25
     Near-optimal Initial Seed Value Selection 25
     Genetic k-means Algorithm 27
     Evolutionary Technique for Optimal Clustering 30
     Hyper-quad Tree Approach 31
   Region-based Crossover in Genetic Algorithms 33
   Generalization of Total SSE through Choice of Metric and Cluster
   Representatives 36
     Metrics for Clustering 36
     Cluster Representatives 38

3. Methodology 40
   Program Design 40
   Experiment Design 42
   Result Analysis 43

4. Results 45
   Experiments Conducted 45
Small Datasets 47
  Data 47
  Analysis 52
Large Datasets 54
  Data 54
  Analysis 60
Running Time 61
  \(k\)-means and \(k\)-medoids 61
  Genetic Algorithms 64
Findings 65

5. Conclusions, Implications, Recommendations, and Summary 69
Conclusions 69
Implications 70
Recommendations 71
Summary 72

Appendices
A. Plots Showing Average (over all runs) of Average \(\text{SSE}_T\) per Generation for GAs 75
B. Charts Showing Best \(\text{SSE}_T\) Values of Large Datasets across all Parameter Sets 116

References 141
List of Tables

Tables
1. Datasets used in this Research 42
2. Choice of Centers for each Dataset used in this Research 45
3. Naming Convention used for the Genetic Algorithms 47
4. Best $SSE_T$ Values after all Runs for the GTD Dataset using $k$-means 48
5. Best $SSE_T$ Values after all Runs for the GTD Dataset using $k$-medoids 49
6. Best $SSE_T$ Values after all Runs for the BTD Dataset using $k$-means 50
7. Best $SSE_T$ Values after all Runs for the BTD Dataset using $k$-medoids 51
8. Best $SSE_T$ Values after all Runs for the TSP-LIB-1060 Dataset using $k$-means 55
9. Best $SSE_T$ Values after all Runs for the TSP-LIB-1060 Dataset using $k$-medoids 56
10. Best $SSE_T$ Values after all Runs for the TSP-LIB-3038 Dataset using $k$-means 57
11. Best $SSE_T$ Values after all Runs for the TSP-LIB-3038 Dataset using $k$-medoids 58
12. Approximate Running Time for each $k$-means/$k$-medoid Run for the TSP-LIB-1060 Dataset 62
13. Approximate Running Time for each $k$-means/$k$-medoids Run for the TSP-LIB-3038 Dataset 63
14. Ratio of Averages of GA0/GA2 and GA1/GA2 for Varied Metrics, Centers and ROCs for the TSP-LIB-1060 Dataset  65

15. Ratio of Averages of GA0/GA2 and GA1/GA2 for Varied Metrics, Centers and ROCs for the TSP-LIB-3038 Dataset  66
List of Figures

Figures

1. Two Dimensional Search Space Split into Quad-trees Containing Data Points 32

2. Region Based Crossover Operator 34

3. Average $SSE_T$ per Generation for the Three GAs using the GTD Dataset, $k$-means, and the Chebyshev Metric for $k=9$ 52

4. Average $SSE_T$ per Generation for the Three GAs using the TSP-LIB-3038 Dataset, $k$-medoids, and the Euclidean Metric for $k=100$ 59

5. Average $SSE_T$ per Generation for the Three GAs using the TSP-LIB-3038 Dataset, $k$-medoids, and the Manhattan Metric for $k=100$ 60
Chapter 1

Introduction

Background

Clustering is the process of grouping data into clusters based on distance between or similarities of the data objects. Clustering has applications in biology, medicine, market research, social network analysis, software evolution, image segmentation, data mining, search result grouping, slippy-map optimization, radiation therapy segmentation, grouping of shopping items, mathematical chemistry, petroleum geology, physical geography, crime analysis, and other areas (Jain, Murty, & Flynn, 1999).

The Sum of the Squared Error (SSE) is a commonly used metric to calculate the quality of the partitions created in clustering algorithms (Jain & Dubes, 1988). It is the sum of the squared distance between each point and the center of the cluster to which the point is assigned. Consider $x_i$ to be the $i$th data point in a cluster of $n$ points and $\bar{x}$ to be the centroid of its cluster. Then the SSE of the cluster would be defined as:

$$\sum_{i=1}^{n} (x_i - \bar{x})^2$$
Now consider there to be a total of $k$ clusters in the search space. Then the total SSE \((\text{SSE}_T)\) of the partition would be defined as:

$$
\sum_{j=1}^{k} \sum_{i=1}^{n_j} (x_{ij} - \bar{x}_j)^2
$$

where $x_{ij}$ represents the $i^{th}$ data point in the $j^{th}$ cluster and $\bar{x}_j$ is the centroid of the $j^{th}$ cluster. Clustering seeks to minimize $\text{SSE}_T$.

Data clustering algorithms can be hierarchical or partitional (Jain, Murty, & Flynn, 1999). Hierarchical clustering deals with forming a hierarchy of clusters based on their proximity to each other. There are two types of hierarchical clustering: agglomerative and divisive. In agglomerative clustering, each observation starts in its own cluster, and clusters are merged as one moves up the hierarchy of clusters. In divisive clustering, all observations start at a single all-encompassing cluster and splits are formed as one moves through the hierarchy.

Partitional clustering deals with simultaneously forming clusters for all data points based on their distance from a cluster center. This is done by using cluster heads or centers. Partitions are then created using mathematical models. There are three categories of partitional clustering algorithms: $k$-means and derivatives, locality-sensitive hashing, and graph-theoretic methods. Of these three methods, one of the most popular partitional clustering algorithms is the $k$-means algorithm because of its simplicity and efficiency (Jain & Dubes, 1988; Jain, Murty, & Flynn, 1999).
\(k\)-means partitions a set of data points into \(k\) clusters. It uses mean distances of points from the cluster center as the chief criterion for forming partitions. \(k\)-means uses an iterative process to reach the final optimum partitions.

Initial centers are randomly selected and clusters are created by measuring the distance of each data point from the selected centers. The centroid of each cluster is then calculated using the cluster data points, and these centroids become the new centers. This process is repeated until there is no change in any of the centroids. As noted, \(k\)-means seeks to minimize \(SSE_T\).

**Problem Statement**

The main problem with the \(k\)-means algorithm is that the quality of its solutions is sensitive to the initial selection of cluster centers (Al-shboul & Myaeng, 2009). This selection could severely affect the performance of the algorithm and the number of iterations it needs to reach its steady state.

This problem of center selection has been tackled using different approaches such as meta-heuristics, genetic algorithms (GA) and simulated annealing (Perim, Wandekokem, & Varejão, 2008). This research focuses on the evolutionary genetic algorithm approach. (Babu & M, 1993; Bandyopadhyay & Maulik, 2001; Bhuyan, Raghavan, & Venkatesh, 1991; Jones & Beltramo, 1991; Krishna & Murthy, 1999; Laszlo & Mukherjee, 2006; Maulik & Bandyopadhyay, 2000; Laszlo & Mukherjee, 2007) have used genetic algorithms for initial center selection of centers for \(k\)-means clustering.
Of the approaches mentioned above, one of the most efficient genetic algorithms for $k$-means clustering has been demonstrated by (Laszlo & Mukherjee, 2007). This GA uses a region-based crossover which scales well to higher-dimensional search spaces. The crossover operator proposed by (Laszlo & Mukherjee, 2007) exchanges subsets of centers that occupy the same region of space. The premise for the region-based crossover is that better partitions can be achieved by exchanging neighboring centers from a region of space rather than exchanging random centers.

This GA was tested on a variety of known data sets involving varying numbers of points, dimensions and centers. The GA found known optimal partitions for the data sets within a few generations in repeated runs.

**Goal**

The GA demonstrated by (Laszlo & Mukherjee, 2007) used the Euclidean distance as the distance metric and centroid-based clustering as the method of determining the new centers. The distance metric and method for determining the new centers jointly determine the objective function.

The goal of this research was to assess this GA approach, with an emphasis on the region-based crossover (Laszlo & Mukherjee, 2007), using different objective functions. If the GA performs well with different distance metrics and representative cluster objects, then the Region-Based Genetic Algorithm (RBGA) can be considered a promising algorithm for initial center selection for a range of objective functions. The main measurable criterion that was used to decide the quality of the algorithm is SSE$_T$. 
Computation of $\text{SSE}_T$ depends on the distance metric and the policy for identifying cluster representatives. Both of these parameters were varied. This research aimed at assessing the quality of the region-based strategy in a wider context than was done by (Laszlo & Mukherjee, 2007).

**Research Questions**

Below is a list of research questions that were asked in order to satisfy and measure the goal of this research:

1) *Is region-based crossover effective when the number of centers represented by individuals is allowed to vary?*

An issue with the region-based crossover is that the number of centers exchanged between the two individuals may be different, so that the individuals may be formed containing other than $k$ centers. This could be solved by specifying an average value $k$ and allowing the number of centers to vary, thereby obtaining good partitions for different values of $k$ (including $k$). Another approach would be to ensure that the same number of centers gets exchanged; for instance, where $m$ is the minimum number of centers in the region $R$ for individuals $i_1$ and $i_2$, exchange exactly $m$ centers. In this research, the latter approach was used to ensure that every individual represents the same number of centers ($k$).
2) *Is region-based crossover effective in a broad set of contexts?*

(Laszlo & Mukherjee, 2007) have shown that the region-based crossover method is more effective than other methods when the Euclidean distance is used as the distance metric and the centroid is used as the representative object of the cluster (ROC). This research explored whether and to what extent this holds true for other distance metrics and ROCs.

3) *Are there methods and heuristics that can be employed to speed up genetic algorithms that employ region-based crossover?*

(Laszlo & Mukherjee, 2007) noted that the RBGA produces the best solutions as compared to other methods, but it is computationally expensive. The $k$-means operator, which is used to compute fitness, is the most computationally expensive part of the RBGA since it is run until convergence is reached. Methods for computing fitness more efficiently were studied. One method, called the evaluation-relaxation scheme (Yu, Goldberg, & Sastry, 2003), has proven successful in speeding up genetic algorithms where a costly fitness evaluation function is replaced by a cheap, less accurate one. The authors claim that although this process compromises the quality of solution, its effect on selection in the genetic algorithm can be negligible. Such methods were considered but not used in this research.
Relevance and Significance

Cluster analysis has applications in various fields. In the field of imaging, clustering is used to obtain point, region or volume classification of scanned images. These clusters prove to be useful in determining areas of heat, infection, foreign bodies and diseases. In the field of ecology, clustering is used to classify geographical spaces and communities of plants and animals. Cluster analysis is used by forestry departments and city planning authorities to classify specific areas and communities in satellite maps. Clustering is also used in plant systematics to generate clusters of similar individuals based on common characteristics at various levels of the classification hierarchy (Jain, Murty, & Flynn, 1999; Kaufman & Rousseeuw, 1990; Jain & Dubes, 1988).

In transcriptomics, clustering is used to group related genes together. These related genes are called co-expressed genes. Co-expressed genes generally contain functionally related proteins. This information is useful in determining hereditary characteristics of organisms, plants and humans. This can also be a powerful tool for genome annotation of viruses. In medical imaging, cluster analysis is used to determine types of tissue and blood (Kaufman & Rousseeuw, 1990). In this application, density of data points is more important than the actual position. But nevertheless, clustering is used to figure out the intensity of the data points. In market research large sets of data are collected from consumer surveys (Jain, Murty, & Flynn, 1999). These data are sometimes mapped onto a graph in the form of data points.
k-means

The k-means algorithm is used in most of the applications mentioned above, so the quality of partitions created by the algorithm is crucial. Since the initial selection of centers greatly affects the quality of partitions it is very important to select good centers for the k-means algorithm.

k-means clustering and its derivatives are the most widely used partitional clustering algorithms. The k-means algorithm is as follows: (Al-shboul & Myaeng, 2009)

Input: A set of data points
Output: A partition with k clusters

k distinct data points are chosen as cluster centers.
Repeat the following steps until the centroids do not change (steady state is reached)
   Every data point is assigned to only a single closest center based on its distance from the center using a distance metric
   Once these k temporary clusters are determined, the centroids of these clusters are calculated using a distance metric
   The centers are repositioned to the location of the centroids

Region-based Crossover

Region based crossover is a genetic algorithm crossover technique introduced by (Laszlo & Mukherjee, 2007) used to find good initial centers for the k-means algorithm. Genetic algorithms (GA) are useful in generating solutions to search problems. GAs use concepts of evolutionary biology like crossover, mutation, selection and inheritance to generate “fit” offspring.

In GAs a population of strings called chromosomes is initially selected (Jones & Beltramo, 1991). These chromosomes encode candidate solutions (individuals) to the problem. In most cases individuals are represented as binary strings, but other representations are also used. The search spans successive generations. In each
generation, the fitness of the individuals is measured. Genetic operations like crossover and mutation are carried out on these individuals to generate new offspring. The selection process ensures that the fittest offspring are most likely to be used as parents in the next generation. This process is continued until a termination condition is reached. Since genetic algorithms are constantly looking for fittest solutions, they often rapidly generate solutions to search problems as compared to other methods.

A simple generational genetic algorithm looks like this:

Select an initial population of individuals
Evaluate the fitness of each individual in that population
Repeat the following steps on this generation until termination condition is reached
   Select individuals for reproduction based on fitness
   Breed new individuals (offspring) through crossover and mutation
   Evaluate the fitness of new individuals
   Replace least-fit individuals in the population with new fittest individuals

In this research, an individual represents a collection of points in space. Specifically, each individual represents a set of $k$ centers, and each center is represented by a sequence of $n$ floating-point values where the space is $n$-dimensional. An individual represents an initial configuration for running $k$-means.

Crossover, a concept from evolutionary biology, is used to combine two parent chromosomes into one or two offspring (Babu & Murthy, 1993). At its simplest, where chromosomes are represented by binary strings, two parent chromosomes are swapped at a random point. There are different types of crossovers used in genetic algorithms. Some of them are one-point crossover, two-point crossover, cut and splice, uniform crossover and half uniform crossover. Below is an example of one-point crossover:

First parent chromosome: 11001010
Second parent chromosome: 00101101
Point of crossover: 4
Offspring: 11001101 & 00101010

The crossover genetic operator is one of the two main reproduction operators in a genetic algorithm, the other being mutation. Crossover is the main operator used to vary the programming of a chromosome and deals with recombining two parents to form an offspring which typically shares many characteristics of its parents. This leads to intensification of the search. Mutation, on the other hand, diversifies the gene pool by modifying a gene. Most genetic algorithms use a high crossover probability and a very low mutation probability.

The crossover operator proposed by (Laszlo & Mukherjee, 2007) exchanges subsets of centers that occupy the same region of space. The premise for the region-based crossover is that better partitions can be achieved by exchanging neighboring centers from a given region of space rather than exchanging random centers. The authors prove that exchanging subsets of centers that are in the same region could yield much better results as compared to using random centers. They split the search space into two regions and use the crossover operator in these regions. These subsets are then combined to get good partitions in the overall search space.

The results of the experiments carried out on various data sets of varying dimensions and centers have proven to be better than or equal to the results from experiments carried out using any of the other methods mentioned above.

This research thoroughly assessed this region-based genetic algorithm and tested it against a relevant set of clustering techniques and center-selection methods. The goal
was to assess the quality of solutions produced by RBGA for center selection across a wide range of data and objective functions.

**Barriers and Issues**

Medoids are representative objects of a cluster whose average dissimilarity to all the points in the cluster is minimal. Medoids are similar to centroids, but medoids are always members of the cluster. It could be computationally expensive to exactly calculate a medoid and so heuristics are often used.

The choice of the medoid is crucial and a method of determining the medoid needed to be established. One such possible method would be to calculate the centroid of a cluster and then use the data point closest to the centroid as the medoid. Another possible method would be to select the data point whose average dissimilarity to all the data points in the cluster is minimal (Kaufman & Rousseeuw, 1990). The latter method is exact but more costly to compute; in contrast, the former method is a heuristic. The cost of the latter method is greater than that of the heuristic by a factor of \( n \), where \( n \) is the number of data points. Both these methods were tried and tested. For the sake of time complexity, the heuristic was used.

In some cases, it may not possible to reach convergence when using medoids as representative objects of a cluster because a medoid is not the true center of the cluster. This issue was dealt with, by forcefully exiting out of the iteration loop after a reasonable number of iterations were reached. This number was chosen based on the dataset, number of centers, and distance metric.
Limitations and Delimitations

Some delimitations were imposed to control the scope of the research and keep it relevant. The choice of distance metrics and representative objects of a cluster were carefully chosen. The distance metrics had to be compatible with the $k$-means algorithm and relevant to clustering problems. The same applied to the representative objects of a cluster. The relevant methods and algorithms that were used for comparison to the RBGA were carefully chosen. The datasets were intended to be benchmarks that represent a wide range of potential inputs. Datasets other than those mentioned below were not used. The following is a list of methods, datasets, distance measures, and representative objects of a cluster that were chosen as delimiters in this research:

1) Method
   a. $k$-means algorithm (centroids only) (Al-shboul & Myaeng, 2009)
   b. $k$-medoids algorithm (medoids only) (Kaufman & Rousseeuw, 1990)
   c. GA without crossover using $k$-means(Krishna & Murthy, 1999)
   d. GA with random crossover using $k$-means (Babu & Murthy, 1993)
   e. GA with region-based crossover using $k$-means (Laszlo & Mukherjee, 2007)
   f. GA without crossover using $k$-medoids(Krishna & Murthy, 1999)
   g. GA with random crossover using $k$-medoids (Babu & Murthy, 1993)
   h. GA with region-based crossover using $k$-medoids (Laszlo & Mukherjee, 2007)
2) Datasets:
   a. German town data (GTD)
   b. British town data (BTD)
   c. TSP-LIB-1060
   d. TSP-LIB-3038

3) Distance measure
   a. Euclidean metric (Minkowski metric with $p=2$)
   b. Manhattan metric (Minkowski metric with $p=1$)
   c. Chebyshev metric (Minkowski metric with $p=\infty$)
   d. Minkowski metric with a user-defined value of $p$

4) Number of centers $k$

**Definition of Terms**

*Centroid*: A point that is the arithmetic mean of all points of the object or shape (Jain, Murty, & Flynn, Data Clustering: A review, 1999).

*Chebyshev Distance*: A metric where the distance between two points is the greatest of their differences along any co-ordinate dimension. It is also known as Chessboard distance (Deza & Deza, 2006).

*Chromosome*: A population of candidates in a genetic algorithm.

*Cluster*: A set of points or objects.
*Cluster Center:* A point of the cluster whose average dissimilarity to all the objects in the cluster is minimal.

*Clustering:* The process of grouping data into clusters based on distance between or similarities of the data objects.

*Crossover:* A process within the reproduction stage of a genetic algorithm that combines parts of the parent chromosomes to form offspring.

*Distance Metric:* A distance function performed on a set of points in a defined space.

*Euclidean Distance:* The distance between two points that is represented by a straight line.

*Fitness function:* The function used to determine the quality of the solution in a genetic algorithm.

*Genetic Algorithm (GA):* An algorithm that uses concepts of evolutionary biology to find solutions to problems.

*Initialization:* A process within a genetic algorithm that generates chromosomes to participate in the genetic algorithm.

*k-means Algorithm:* A clustering method of which partitions a set of points into $k$ clusters based on an iterative process.

*k-medoids:* A partitional clustering algorithm that uses medoids instead of centroids (Kaufman & Rousseeuw, 1990).
Manhattan Distance: The distance of two points measured in a grid-based co-ordinate system. It is also known as Taxicab distance (Deza & Deza, 2006).

Medoid: A point of the cluster whose average dissimilarity to all the objects in the cluster is minimal.

Minkowski Distance: (Deza & Deza, 2006) state that for two n-dimensional points \( x=(x_1, x_2, \ldots, x_n) \) and \( y=(y_1,y_2,\ldots,y_n) \) the Minkowski distance would be given by:

\[
D_p(x, y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}
\]

Mutation: A process within the reproduction stage of a genetic algorithm that modifies a chromosome to maintain genetic diversity.

Objective Function: A function to be maximized or minimized in optimization theory. In this research the objective function is the combination of a distance metric and a representative object of a cluster (centroid or medoid). In order to test the region based crossover algorithm, various distance metrics were paired with different ROCs to create a set of objective functions. The list of metrics and objective functions are specified in the Limitations and de-limitations section above.

Partition: A division of a set into non-overlapping and non-empty parts that cover all of the set.

Region Based Genetic Algorithm (RBGA): A genetic algorithm proposed by (Laszlo & Mukherjee, 2007) that exchanges subsets of centers that occupy the same region of space.
Replacement: A process within a genetic algorithm that replaces the older group of individuals with new offspring with high fitness functions.

Representative Object of a Cluster (ROC): A point or object of the cluster chosen as the center. It is used in calculating the quality of the partition.

Reproduction: A process within a genetic algorithm that produces offspring, also known as children. Reproduction can involve various operators but crossover and mutation are the two most commonly used operators.

Selection: A process within a genetic algorithm that selects a group of individuals to participate in the next iteration of the genetic algorithm based on a fitness function

Sum of Squared Error (SSE): The sum of the squared distance between each point and the center of the cluster to which the point is assigned. Consider $x_i$ to be the $i^{th}$ data point in a cluster of $n$ points and $\bar{x}$ to be the centroid of its cluster. Then the SSE of the cluster would be defined as:

$$
\sum_{i=1}^{n} (x_i - \bar{x})^2
$$

Total SSE ($SSE_T$): The sum of the SSE of all the clusters in the search space. Consider a total of $k$ clusters in the search space. Consider $x_{ij}$ to be the $i^{th}$ data point in the $j^{th}$ cluster and $\bar{x}_j$ to be the centroid of the $j^{th}$ cluster. Then $SSE_T$ can be defined as:

$$
\sum_{j=1}^{k} \sum_{i=1}^{n_j} (x_{ij} - \bar{x}_j)^2
$$
Summary

Clustering has applications in various fields including biology, market research, image processing and social network analysis. Data clustering can be hierarchical or partitional. The most popular partitional clustering algorithm is the $k$-means algorithm, because of its simplicity and efficiency.

The problem with $k$-means is that the quality of solutions is sensitive to the initial selection of cluster centers. This problem has been tackled using several approaches. Of these approaches, the region-based genetic algorithm proposed by (Laszlo & Mukherjee, 2007) has proven to deliver very good results. This research aimed at testing the capability of this algorithm with varied objective functions and compared the results with other selected algorithms.

The two main issues/barriers in this research were medoid selection, and the number of centers that needed to be exchanged in the region-based crossover. There were some proposed solutions to these barriers and they were tried and tested.
Chapter 2
Review of the Literature

Clustering Algorithms

Clustering is the process of grouping points or objects into sub-sets called clusters. There are two main types of clustering algorithms: hierarchical and partitional (Jain & Dubes, 1988).

Hierarchical algorithms work in stages, where points are clustered based on their proximity to each other. This enables clustering in stages. The algorithm decides which points belong to a cluster as it moves from one stage (or hierarchy) to the next. A distance criterion can be set to decide the proximity between points in order to form a cluster. This distance criterion decides the total number of clusters that are formed. By changing this distance criterion the number and size of the clusters can be varied. It decides whether the algorithm should move from one stage, or hierarchy, to the next.

Hierarchical algorithms typically use a distance metric to decide distance between points. They do not make use of external data points or cluster centers to determine the next cluster stage. Dendrograms, or tree diagrams, are generally used to graphically depict the various stages of hierarchical clustering. The algorithm usually proceeds top-down with criteria for splitting clusters, or bottom-up with criteria for merging clusters, or both. The result is a hierarchy, each level of which represents a partition of particular
granularity (i.e., number of clusters). The most common metric used in hierarchical clustering algorithms is the Euclidean metric. Other metrics that could be used are Manhattan distance, maximum norm, Mahalanobis metric, inner product space and Hamming distance.

In partitional clustering, the points are grouped based on their proximity to chosen cluster centers (Jain, Murty, & Flynn, 1999). The centers could be random, calculated or chosen based on an existing data point. Centroids are typically used as calculated centers and medoids are used as existing data points.

Partitional clustering does not follow a hierarchical approach. All the cluster centers are calculated at the same time (Jain, Murty, & Flynn, 1999). Once the cluster centers are calculated the process is repeated until an optimum solution is reached. Different distance metrics could be used in partitional clustering but the Euclidean distance metric is used most often because of its simplicity. The different types of partitional clustering techniques include $k$-means algorithm, fuzzy $c$-means algorithm, quality threshold (QT) clustering algorithm, locality sensitive hashing and formal concept analysis.

The $k$-means algorithm is an iterative algorithm (Jain, Murty, & Flynn, 1999). It starts with a set of $k$ centers and computes their neighborhoods. After the neighborhoods are computed, the centers are relocated to the centroids of their neighborhoods. The neighborhoods are then recalculated based on the positions of the new centers. This process is repeated until the centers and centroids converge. This phenomenon is called the convergence criterion.
The fuzzy $c$-means algorithm is very similar to the $k$-means algorithm, but it involves concepts of fuzzy logic (Bezdek, 1981). In the fuzzy $c$-means algorithm, data points have a degree of belonging to a cluster rather than an absolute belonging to a cluster. For example, a point that is in the center of the cluster would have a stronger degree of belonging to the cluster as compared to a point located near the edge of the cluster. The formula used to calculate the centroid uses the mean of all points and also includes their degree of belonging. This formula creates clusters that are slightly different from those created by the $k$-means algorithm since the degree of belonging is taken into consideration.

In the quality threshold (QT) algorithm (Heyer, Kruglyak, & Yooseph, 1999), the maximum diameter for clusters and the minimum number of data points per cluster is chosen in advance. This is done to ensure the quality of the clusters. In the QT algorithm data points are called genes. The QT algorithm is as follows:

Repeat the following steps until the largest formed cluster has fewer genes than the user-specified number of genes

Repeat until all genes in the gene list are clustered

A gene is selected from a list of genes

The gene that is most similar to this gene is then determined

Repeat until no more genes can be added to the cluster without altering the cluster diameter

If the total diameter of the genes selected does not exceed the cluster diameter, then these two genes are clustered together

The largest cluster with the minimum number of genes is selected as the QT cluster

The genes within this cluster are removed from consideration

The result is a set of non-overlapping QT clusters

Locality sensitive hashing (LSH) is another algorithm used in clustering (Slaney & Casey, 2008). The main idea behind this algorithm is to hash input data, so that similar items are mapped to same sets with high probability. The methods used in LSH are bit
sampling for Hamming distance, min-wise independent permutations, random projections, and stable distributions. The LSH algorithm is often used to find solutions for the nearest neighbor search algorithm.

Formal concept analysis is used to cluster objects based on their properties (Ganter, Stumme, & Wille, 2005). Bipartite graphs or lattice structures are used to derive solutions. Given a set of $n$ objects and $m$ properties, a graph or table is drawn linking the objects to the properties. Then objects with similar properties are grouped together forming a cluster.

(Wu & Yang, 2001) proposed a new metric to replace the Euclidean norm in $c$-means clustering procedures. They called their method alternative $c$-means clustering. The authors claim that in non-noisy datasets the Euclidean metric works well and delivers good results. But in data sets that involve noisy points, the Euclidean metric does not seem robust enough. The authors developed a new metric that is defined as follows:

$$d(x,y) = 1 - \exp(-\beta \|x - y\|^2)$$

where $\beta$ is a positive constant and $x$ and $y$ are data points. This formula gives large weights to compact points in the data set and small weights to noisy points, thus making the metric more robust. Experiments carried out using this metric proved to have much better results as compared to the Euclidean metric in datasets involving noisy points.

(Estivill-Castro & Yang, 2004) proposed a method that uses medians instead of means, as representative objects of the cluster, in the $k$-means algorithm. Since multi-dimensional points are used, the geometric median or 1-median is used as the cluster center. Iteratively reweighted least squares method is commonly used to calculate the
geometric median of a set of \( n \)-dimensional points. Their argument is that statistically speaking, means are not robust estimators of central tendency. They are sensitive to noise and outliers. Means are also invariant only under linear transformations of a random variable. Medians, on the other hand, are invariant under monotonic transformations, and not susceptible to noise. Experiments performed by the authors proved that using medians instead of means works well for small multi-dimensional datasets.

**Genetic Algorithms**

A genetic algorithm (GA) uses concepts of evolutionary biology to find solutions to problems (Jones & Beltramo, 1991). GAs use a set of solutions, consisting of chromosomes. This set of solutions is called a population. Fit solutions from one population are modified to form a new population with a hope that the new population will be fitter than the original population. Solutions are selected based on their fitness using a fitness function. Fitter solutions are more likely to reproduce and create new offspring. The process of reproduction includes genetic operations such as mutation and crossover. This process is repeated until a terminating condition is reached. Common terminating conditions are:

- A solution is found that satisfies the required minimum criteria
- Maximum allocated number of generations is reached
- Maximum allocated computation time is reached
- A combination of the above
The four main parts of a genetic algorithm are initialization, selection, reproduction, and termination. Selection, reproduction and replacement are repeated until the termination condition is satisfied. The individuals called chromosomes collectively comprise a population.

Initialization generates chromosomes to participate in the algorithm. Initialization is traditionally done randomly covering the entire area of the search space. On some occasions, solutions are chosen in areas of the search space where the probability of finding optimum solutions is based on knowledge of the problem domain.

Selection of the powerful chromosomes is done based on a fitness function. The fitter an individual, the more likely it is to get selected. Popular selection methods include roulette wheel selection and tournament selection.

The selected chromosomes are then put through the process of reproduction. Reproduction produces offspring, also known as children. Reproduction can involve various operators but crossover and mutation are the two most commonly used operators. One standard representation for chromosomes is the bit-string but other representations are also used.

Crossover combines parts of the parent chromosome to form offspring. There are several standard crossover techniques for this representation. The most common crossover technique is one-point crossover where a crossover point is selected in the bit-string and the bits before that point in one chromosome are swapped with the bits after that point in the other chromosome. Other common crossover techniques are two-point crossover, cut and splice and uniform crossover.
Mutation modifies a chromosome to maintain genetic diversity. The most basic mutation operator simply flips a tiny percentage of bits, selected at random. After reproduction, the selection process is repeated and the fitness of each offspring is determined. This process is repeated until a termination condition is satisfied.

**Genetic Algorithms for Clustering**

Of the clustering techniques mentioned above, one of the most popular partitional clustering algorithms is the *k*-means algorithm because of its simplicity and efficiency (Al-shboul & Myaeng, 2009). *k*-means uses random initial centers to generate partitions. A problem with the *k*-means algorithm is that the quality of the resulting partition depends largely on the initial selection of centers. A poor choice of centers might lead to partitions that are much inferior to partitions generated from well-chosen centers. Solving the problem of initial center selection leads to an optimal solution, under *k*-means.

The problem of selecting an initial center selection that leads, under *k*-means, to an optimal solution, is an NP-hard search problem (Jones & Beltramo, 1991). In a search space, centers are chosen from a given set of data points. The number of combinations of initial centers for the *k*-means algorithm is exponential in the number of data points. Choosing the best data points as centers through an exhaustive process is too computationally intensive. Genetic algorithms (GA) have been used to solve this problem.

The appeal of GAs in such cases comes from their simplicity and elegance as robust search algorithms in a search space that is large, complex and not very well
understood (Perim, Wandekokem, & Varejão, 2008). GAs have the power to discover
good solutions rapidly for difficult high-dimensional problems. They are useful and
efficient when knowledge of the domain is scarce or difficult to encode or no
mathematical analysis is available.

The advantage of the GA approach is the ease with which it can handle arbitrary
kinds of constraints and objectives (Jones & Beltram, 1991). GAs only concentrate on
data points that are considered “fit” and use these points to generate “fitter” data points
which end up acting as optimum solutions. GAs can rapidly locate good solutions in
search problems as compared to other non-evolutionary methods.

**Genetic Algorithms for Center Selection in k-means Initialization**

Below is a list of relevant genetic algorithm-based techniques that have been used
to find globally optimal solutions to $k$-means clustering problems.

**Near-optimal Initial Seed Value Selection**

(Babu & Murthy, 1993) have used GAs to address the problem of center selection
in $k$-means. In a dataset with $d$-dimensional data points, they select $k$ centers, formed by
dividing each dimension into $(2^B - 1)$ segments. The length of a chromosome is $kd B$,
where $B$ is the number of bits per dimension. The resulting search space is of size $2kd B$.

The algorithm proposed by (Babu & Murthy, 1993) is as follows:

**Input:**

$P_c = \text{crossover probability}$

$P_m = \text{mutation probability}$
N = population size
G = Total number of generations
Output:
S = Solution string

Generate initial population of size N
While (G>0)
    Calculate sum of fitness values of solution strings
    While (size of present population < N)
        Perform mutation using Pm
        Perform crossover using Pc
    Calculate fitness value of each string using k-means
    If (fitness of S < maximum fitness value in population)
        Replace S by that string
    If (Error of S is near optimal or optimal value)
        Output S and terminate the algorithm
    Else G = G-1
Output the solution string S

The selection operator is based on a roulette wheel selection strategy. The
crossover operator selects a point in two strings, breaks the strings at that point, and then
exchanges the sub-strings thus formed. The mutation operator modifies selected binary
string values with the given mutation probability.

Each population has N strings, and each solution string requires running the k-
means algorithm in order to evaluate fitness. This is done for G generations. So the
complexity of the algorithm is $O(GN)$ k-means algorithms. The complexity increases
linearly as G and N increase. Since k-means is time consuming, large values of N and G
are impractical. This constrained the authors to keep N as low as 8 for the IRIS data set
and 10 for the BTD dataset. G was set to 50 for both datasets. Both these datasets
consisted of 50 four-dimensional data points. This mechanism works with small datasets
that have few centers and dimensions. Following this method, it is hard to find good
partitions with many clusters using larger data sets with higher dimensions.
Genetic $k$-means Algorithm

(Krishna & Murthy, 1999) pointed out that GA-based methods of the past are not good because they use either poor crossover operators or fitness functions that are computationally expensive. To overcome these problems, they proposed a GA-based method called GKA that uses $k$-means instead of the crossover operation of the GA. They also used a biased mutation operator, based on distance, to widen the search for the global optimum.

GKA uses a search space of $W$ matrices (Krishna & Murthy, 1999). One way of coding $W$ matrices into a string, $s_W$, is to consider a chromosome of length $n$ and allow each allele in the chromosome to take values from 1 to $k$. In this case, each allele corresponds to a pattern and its value represents the cluster number to which the corresponding pattern belongs. This type of coding is called string-of-group-numbers encoding. GKA uses a population of such strings.

GKA uses a mutation technique that changes an allele value depending on the distances of the cluster centroids from the corresponding data point (Krishna & Murthy, 1999). It may be recalled that each allele corresponds to a data point and its value represents the cluster to which the data point belongs. An operator is defined such that the probability of changing an allele value to a cluster number is greater if the corresponding cluster center is closer to the data point. To apply the mutation operator to the allele $s_{W(i)}$ corresponding to the data point $x_i$, let $d_j = d(x_i, c_j)$ be the Euclidean distance between $x_i$ and $c_j$ where $c_j$ is the cluster center. Then, the allele is replaced with a value chosen randomly from the following mutation distribution $p_j$:
\[ p_j = \Pr[s_w(i) = j] = \frac{c_m d_{\text{max}} - d_j}{\sum_{i=1}^{k}(c_m d_{\text{max}} - d_i)} \]

where \( c_m \) is a constant (usually > 1) and \( d_{\text{max}} \) is the maximum value of \( d_j \).

One problem with selecting the initial population randomly is illegal strings. Illegal strings represent a partition in which some clusters are empty, with some nonzero probability. A quick way of detecting the possibility of empty cluster formation is to check whether the distance of the data point \( x_i \) from its cluster center \( c_j \) is greater than zero. This distance is represented by \( d_{s(w)i} \). It may be noted that \( d_{s(w)i} = 0 \) even in the case of non-singleton clusters wherein the data point and the center of the cluster are the same. Thus, an allele is mutated only when \( d_{s(w)i} > 0 \). Each allele in a chromosome is mutated as described above with a probability \( P_m \), called mutation probability. The authors call this mutation DBM1. The authors claim that this mutation helps in reaching better solutions. A pseudo-code of the operator is given below.

For \( i = 1 \) to \( n \)

If \( \text{drand()} < P_m \) //uniform random number in \([0,1]\)

Calculate cluster centers, \( c_j \)'s, corresponding to \( s_W \)

For \( j = 1 \) to \( K \), \( d_j = d(x_i, c_j) \)

If \( (ds_w(i)) > 0 \)

\[ d_{\text{max}} = \max\{d_1, d_2, \ldots, d_K\} \]

For \( j = 1 \) to \( K \)

\[ p_j = \frac{(c_m d_{\text{max}} - d_j)}{\sum_{i=1}^{K}(c_m d_{\text{max}} - d_i)} \]

\( s_w(i) = \) a number, randomly selected from \( 1 \) to \( K \) according to the distribution \( \{p_1, \ldots, p_K\} \)

In the above algorithm, the selection and mutation operators might take more time to converge because the initial assignments are arbitrary and assignment changes are probabilistic. The mutation probability is forced to assume low values since higher values could lead to oscillating behavior of the algorithm. To remedy this, a simple but powerful one-step \( k \)-means operator is introduced.
The one-step $k$-means operator (KMO) is introduced by the authors instead of crossover and constitutes two stages:

1) Calculate cluster centers using a centroid formula for a given matrix

2) Reassign each data point to the cluster with the closest cluster center

The authors presented experimental results with small data sets (2- to 4-dimensional data with 50-60 data points) and for small numbers of clusters (10). Using their method, as the number of clusters increases, the problem of finding a global optimum becomes much more difficult because of the extensive computation that is involved. One cause of the extensive computation is the expensive mutation operator.

In the case of the mutation operator (DBM1), the alleles are not mutated if $d_{sw(i)} = 0$. It is required that every allele must have the potential to be mutated. For this reason, DBM1 is slightly modified to fix this problem. The modified operator is referred to as DBM2. DBM2 changes each allele, according to the distribution shown above. This may result in an illegal string, with some small nonzero probability. If this operation results in an illegal string, the above procedure is repeated until we get a legal string. This requires DBM2 to change any illegal string to any other legal string with nonzero probability. This process makes the mutation operator expensive.

In this method, the expensive mutation operator is used over and over again to advance exploration of the search space. This suggests that the GKA does not scale well to higher dimensional datasets with many points and many inherent clusters.
Evolutionary Technique for Optimal Clustering

(Bandyopadhyay & Maulik, 2001) have used GAs to find good partitions. They allow strings of variable length to search over a range of values for all clusters and use the binary representation for chromosomes. They used a specialized clustering metric $M$ in fitness computation and is computed as follows:

$$M = \sum_{i=1}^{k} M_i$$

$$M_i = \sum_{x_j \in c_i} ||x_j - z_i||$$

where $z_i$ = a center of $c_i$

The fitness function is defined as $f = 1/M$. So the smaller the value of $M$, the greater the fitness of the chromosome. In this paper, single point crossover was used by the authors. A random integer, called the crossover point, is generated and the portions of the chromosomes lying to the right of the crossover point are exchanged to produce two offspring.

Typically, in the case of binary representation of chromosomes, mutation involves picking a bit position (or gene) in a chromosome and then flipping its value. Since floating point values are used to represent centers in this method, mutation is done as follows. A number $i$ is generated between 0 and 1 with uniform distribution. If the value at the gene position is $j$, after mutation it becomes:

$$j \pm 2ij \quad \text{if} \ j \neq 0,$$
$j \pm 2i$ \quad \text{if } j = 0.$

The ‘+’ or ‘-‘ sign occurs with equal probability. The ‘2’ is used to avoid getting stuck at a positive or negative value at a particular position.

The GA is repeated up to a maximum number of iterations. The best string seen up to the last generation becomes the solution. Experimental results presented by the authors are on small-scale clustering problems. The data sets are small (containing less than 1000 points) and the number of centers and dimensions is also small (2-4 dimensions with 2-5 clusters). The method has been tested and does not scale well to data sets with high dimensions, a large number of points and many clusters.

*Hyper-quad Tree Approach*

All of the above-mentioned GA-based methods exchange random sets of centers during the crossover operation of the GA. (Laszlo & Mukherjee, 2006) used quadtrees to partition the search space. The centers and data points were grouped together based on their position (region) before running the GA. A quadtree is a tree data structure in which every internal node has exactly four children. Quadtrees are often used to divide search spaces into regions. The figure below shows a two-dimensional search space divided into regions.
While performing crossover in the GA, the authors chose a single random node in a tree, and then exchanged the two subtrees rooted at that node in the two chromosomes. This crossover alters the distribution of genes in the participating chromosomes but it does not alter the chromosomes’ tree structure. Nevertheless, the combined number of genes remains fixed. The GA proposed by the authors is as follows:

Construct the hyper-quadtree $T$ over the data points
Create the initial population of $\mu$ chromosomes
For each of $N$ generations
    Select $\mu$ parents based on the fitness
    Pair parents randomly and perform crossover using the hyper-quadtree method to produce offspring
    Mutate offspring
    Combine parents and offspring to form next generation of approximate size $\mu$

There are two problems associated with this method. First, as the dimensions increase, the trees grow exponentially broader. Second, trees can grow to arbitrary depth when required to separate points packed into an arbitrarily tight cluster. Due to these
reasons, this hyper-quad tree approach works well with low-dimensional data but does not scale well to higher-dimensional data sets.

**Region-based Crossover in Genetic Algorithms**

The method proposed in (Laszlo & Mukherjee, 2007) addressed the problem of higher dimensions, by using the region-based crossover operation. It also used a less expensive chromosomal representation which is easily implemented and yields partitions of better quality.

Most genetic algorithms use a random crossover operator that exchanges random subsets of centers between parents to generate offspring. The crossover operator presented by the authors involves chromosomes that occupy the same region of space. To split a given area of a space into regions, the authors use the hyper quad-tree method proposed by (Laszlo & Mukherjee, 2006).

The crossover operator proposed by (Laszlo & Mukherjee, 2007) exchanged subsets of centers that occupy the same region of space. This generates much fitter offspring as compared to those generated with random crossover. The crossover region is selected as follows:

1) Select a random data point $p$ and generate a random vector $n$.

2) Create a hyper-plane $h$ such that it contains point $p$ and is normal to vector $n$.

3) The crossover region is the half-space to the positive side of the hyper-plane.

Since the centers in an individual are scattered all over the search space, it is possible that the two individuals chosen for crossover could have different number of
centers in the crossover region. Let \( m_1 \) be the number of centers in the swap region for parent 1; let \( m_2 \) be defined similarly for parent 2. Let \( M = \min (m_1, m_2) \). Suppose without loss of generality that \( M = m_1 \). Then we randomly choose \( M \) centers of parent 2 belonging to the region, and swap those with all \( M \) of parent 1’s belonging to the region.

![Diagram showing region based crossover operator](image)

**Figure 2. Region based crossover operator**

Figure 2 shows a two-dimensional diagram with a plane and crossover region created by a vector. This could be extended to a hyper-plane for an \( n \)-dimensional search space.

There are two parent chromosomes and both have four centers. The centers for parent 1
are labeled $p_1, p_2, p_3$ and $p_4$. The centers for parent 2 are labeled $q_1, q_2, q_3$ and $q_4$.

After the grey crossover region is determined by a random line, corresponding to a random hyper plane, unequal number of centers from each parent lies in the crossover region. In Figure 2, $M = \min (m_1, m_2) = 1$. We therefore select the point $p_4$ from parent 1 and, at random, $q_4$ of parent 2, to swap. This generates offspring 1 with points $p_1, p_2, p_3$ and $q_4$, and offspring 2 with points $q_1, q_2, q_3$ and $p_4$.

The results of the experiments carried out on various data sets of varying dimensions (up to 100 dimensions) and centers have proven to be better than or equal to the results from experiments carried out using any of the other methods mentioned above. The convergence criterion of the algorithm is of the order of $O(nkd)$ time, where $n$ is the total number of points, $k$ is the number of centers and $d$ is the number of dimensions.

Convergence is often reached in few iterations after the initial generations since centers are moved by $k$-means, but the solutions yield different numbers of clusters since the number of centers vary from one chromosome to the next. The authors claim that early convergence is a result of intensification and diversification. In this method, crossover and selection intensified the search by focusing on a promising set of centers. In contrast, $k$-means, replacement of unused centers by random data points, and mutation through the introduction of new candidate centers were the three sources of diversification.
Generalization of Total SSE through Choice of Metric and Cluster Representatives

Metrics for Clustering

A distance metric is a function that defines the distance between two elements of a set in space (Deza & Deza, 2006). A metric could be used to determine a distance between two points or objects. If the objects possess volume, or more generally, multiple points, the distances may be measured from the objects’ centroids. Some metrics that are commonly used are the Euclidean metric, taxicab metric, Hamming metric, Minkowski metric and the Chebyshev metric. Depending on the type of problem, certain metrics prove useful. The Euclidean metric is most commonly used to calculate distances in clustering problems (Czibula & Serban, 2007). But other metrics are also useful. This research aimed at using a number of metrics and testing them on the algorithm proposed by (Laszlo & Mukherjee, 2007).

The Euclidean distance or Euclidean metric is the distance between two points that is represented by a straight line in Cartesian space. This is also the shortest distance between two points. The Euclidean distance is also called "ordinary" distance in some cases. The Euclidean distance can be calculated by the Pythagorean formula or the distance formula (Czibula & Serban, 2007). The Euclidean distance formula can be used to determine the distance between two points with any number of dimensions. If \( x = (x_1, x_2, \ldots, x_n) \) and \( y = (y_1, y_2, \ldots, y_n) \) are two \( n \)-dimensional points in Euclidean \( n \)-space, then the distance from \( x \) to \( y \) is given by:
The Manhattan distance or taxicab distance is the distance of two points measured in a grid-based co-ordinate system (Czibula & Serban, 2007). The distance gets its name from the grid-like city streets of Manhattan that taxicabs usually have to drive through to get from point A to point B. The Manhattan distance is not the shortest path between two points. In an $n$-dimensional plane, the Manhattan distance between $x$ and $y$ is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

The Chebyshev distance (Deza & Deza, 2006) is a metric where the distance between two points is the greatest of their differences along any co-ordinate dimension. In order to use this metric, the formula to calculate distance will be as follows: In an $n$-dimensional plane, the Chebyshev distance between $x$ and $y$ is:

$$D_{\text{chess}}(x, y) = \max_i \{ |x_i - y_i| \}$$

A distance metric that generalizes all of the above-mentioned distance metrics is called the Minkowski distance metric (Deza & Deza, 2006). For two $n$-dimensional points $x$ and $y$ the Minkowski distance would be given by:

$$D_p(x, y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

By varying the value of $p$, the Minkowski distance metric yields the other metrics. If $p=1$, the Minkowski metric is the same as the Manhattan distance. If $p=2$, the
Minkowski metric is the same as the Euclidean distance. If $p=\infty$, the Minkowski metric is the same as the Chebyshev distance. The Euclidean, Manhattan, Chebyshev and Minkowski distances are the most commonly used distance metrics for clustering in the Euclidean space (Czibula & Serban, 2007).

Cluster Representatives

A centroid or geometric center is the arithmetic mean of all points of the object or shape. This concept can be extended to any shape or object in an $n$-dimensional space. In an $n$-dimensional space, it could be defined as the intersection of every hyper-plane that divides the object or shape into two parts of equal moment. Consider $k$ $n$-dimensional points where $d_i$ denotes the $i^{th}$ point in the dataset. The centroid $C$ of that dataset would be given as:

$$C = \frac{1}{k} \sum_{i=1}^{k} d_i$$

Besides centroids, medoids are also a common representative for clusters. A medoid is a point of the cluster whose average dissimilarity to all the objects in the cluster is minimal. Note that a medoid, unlike a centroid, is always a data point of the cluster. The true medoid is the data point that minimizes the cluster's SSE (i.e., average dissimilarity is minimal). The data point closest to the centroid is often used as a heuristic for the medoid, to avoid time complexity.

A common application of the medoid is the $k$-medoids clustering algorithm. The $k$-medoids clustering algorithm is similar to the $k$-means algorithm and is a partitional clustering algorithm. The most common $k$-medoids algorithm used is the Partitioning
Around Medoids (PAM) algorithm (Kaufman & Rousseeuw, 1990). The goal is to find a subset \( \{m_1, \ldots, m_k\} \subset \{1, \ldots, n\} \) which minimizes the objective function

\[
\sum_{i=1}^{n} \min_{t=1,\ldots,k} d(i, m_t)
\]

Each object is then assigned to the cluster corresponding to the nearest medoid.

That is, object \( i \) is put into cluster \( v_i \) when medoid \( m_{v_i} \) is nearer to \( i \) than any other medoid \( m_w \), or

\[
d(i, m_{v_i}) \leq d(i, m_w) \quad \text{for all} \quad w = 1, \ldots, k.
\]

The PAM algorithm can be stated as follows:

Select \( k \) medoids in the data set randomly
Repeat the following steps until steady state is reached (there is no change in the medoid)
Using a distance metric, calculate the distance between each non-medoid point \( l \) in the data set and the medoids and associate the points to the closest medoid
Calculate the total cost (distance) for each non-medoid data point from its medoid in its cluster
For each medoid \( k \)
   For each non-medoid data point \( l \)
      Swap \( k \) and \( l \) and calculate the total cost of this setup
Select the setup with the lowest cost

This study aimed at proving that the RBGA is a promising algorithm using centroids and medoids as ROCs. The \( k \)-means algorithm was used when centroids were used as the ROC. When medoids were used as the ROC, the \( k \)-medoid algorithm PAM was used. The medoid was chosen by finding the centroid and then locating the closest data point to it, as compared to the method explained in the algorithm above to reduce complexity. The \( k \)-medoid algorithm was a significant part of this study and half the experiments conducted in this research used this algorithm.
Chapter 3

Methodology

Program Design

The algorithms presented in this research were tested, verified and analyzed using a computer program running on a Personal Computer (PC). The program could have been written in several languages, but C++ was used for its simplicity, ease of coding and strong mathematical libraries. The program had a command line interface and gave user the option to select parameter values at run time.

The program inputted data sets from text files stored on the hard drive and ran the chosen methods on these data by varying the number of centers, ROCs, and the choice of metric. These variables and datasets were set by the user during run-time. A list of methods, datasets, metrics, and centers that were chosen while running the program are listed in the *Limitations and Delimitations* section of chapter 1.

Based on selected parameters, the program computed the best and average $\text{SSE}_T$ and saved the information in a Microsoft Excel file. The program was then reset and fed a different set of parameters.

The C++ program used a procedural paradigm and consisted of a top-level main function for user input, mid-level functions for each method specified above and several
low level functions that performed a specific task like calculation of partitions, crossover, mutation, etc. The pseudo code for the k-means function is as follows:

Input:  
Dataset  
Metric  
Number of centers  
Number of runs

Output:  
Run number, \( \text{SSE}_T \) for the run, Best \( \text{SSE}_T \) up to present run, Average \( \text{SSE}_T \) up to present run

Randomly select centers  
While steady-state is not reached  
\quad Calculate partitions using specified metric  
\quad Calculate centroids  
\quad Replace Centers with centroids  
Send \( \text{SSE}_T \), best \( \text{SSE}_T \) and average \( \text{SSE}_T \) to main function

The pseudo code for the k-medoids function is as follows:

Input:  
Dataset  
Metric  
Number of medoids  
Number of runs

Output:  
Run number, \( \text{SSE}_T \) for the run, Best \( \text{SSE}_T \) up to present run, Average \( \text{SSE}_T \) up to present run

Randomly select medoids  
While steady-state is not reached  
\quad Calculate partitions using specified metric  
\quad Swap medoid with non-medoid  
\quad Select configuration with lowest cost  
Send Run number, \( \text{SSE}_T \), best \( \text{SSE}_T \) and average \( \text{SSE}_T \) to main function

The generic pseudo code for the GA functions is as follows:

Input:  
Dataset  
Metric  
Number of centers/medoids  
Mutation probability  
Crossover probability  
Population size  
Total number of generations

Output:  
Generation number, Best \( \text{SSE}_T \) and average \( \text{SSE}_T \) per generation
Generate initial population
Call k-means/k-medoids functions for all population individuals
Select fittest chromosomes
While Total number of generations is not reached
    Perform mutation based on probability
    Perform no/random/region-based crossover based on probability
    Call k-means/k-medoids functions for offspring
    Replace weakest individual with fittest individual
Send generation number, best SSE_T and average SSE_T per generation to main function

**Experiment Design**

The datasets that were used were the four benchmark data sets mentioned in Table 1, for which best known partitions have already been published. The details of the four data sets that were used are as follows:

**Table 1. Datasets used in this Research**

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Number of Points</th>
<th>Number of Dimensions</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>German town data (GTD)</td>
<td>59</td>
<td>2</td>
<td>(Spath, 1980)</td>
</tr>
<tr>
<td>British town data (BTD)</td>
<td>50</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>TSP-LIB-1060</td>
<td>1060</td>
<td>2</td>
<td>(Reinelt, 1991)</td>
</tr>
<tr>
<td>TSP-LIB-3038</td>
<td>3038</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

The number of centers/medoids and the number of generations were varied. These settings were dataset-dependent. The number of generations was determined by how long it took for the GA to converge and this varied by dataset. The number of centers were taken from values generally used in (Laszlo & Mukherjee, 2007) and (Laszlo & Mukherjee, 2006). The typical values for centers were between 0-500 and the typical
generations for the GAs were between 0-100. This allowed all algorithms to be thoroughly tested using all data sets. Since GAs are stochastic, they produced different partitions with the same parameter set. For this reason, multiple runs were performed on each parameter set and the average and best value of SSE$_T$ per generation was computed and stored.

Also, $k$-means and $k$-medoids were run with varied number of centers, the same total number of times as they were used in the GA. For example, if we performed $R$ runs of each GA for $N$ generations with population size $P$, then $k$-means and $k$-medoids was run $RNP$ many times, each with a varied initial set of centers.

In some cases, while using the TSP-LIB-3038 dataset and the Chebyshev metric with $k=500$, it was not possible to reach convergence when using medoids as representative objects of a cluster. This was because a few data points would switch back and forth between two clusters indefinitely. This issue was dealt with, by forcefully exiting out of the loop after a reasonable number of iterations were reached. This number was chosen after determining the typical number of iterations taken by $k$-medoids to converge across a specific parameter set.

**Result Analysis**

The analysis of results was presented via tables and plots. The main measurable criterion for determining the quality of the algorithms is the total SSE of the partitions created using a chosen method.
Every time the program was run with a set of parameters/variables, the output was sent to a Microsoft Excel file. Once the program was run repeatedly to vary all parameters/variables, it was possible to compare total SSE of the eight methods with varying datasets, metrics and centers. Plots were also made comparing the methods for each of the benchmark datasets. The plots that were generated are:

1) Bar charts comparing best $\text{SSE}_T$ after all runs for large datasets
2) Line plots showing the average (over all runs) of the average $\text{SSE}_T$ per generation for the GAs for all datasets

Published results of all methods (except $k$-medoids) using the Euclidean metric and centroid as ROC were used in this research for comparison. This information was obtained from (Laszlo & Mukherjee, 2006) and (Laszlo & Mukherjee, 2007). The rest of the results were obtained by running the C++ programs repeatedly by varying parameters since the literature did not contain these results.

The main criterion used to compare the region-based crossover GA to other methods was the optimization of the objective function, which is the function used to compute fitness values. The goal was to have this sum of the partitions (total SSE) as small as possible. Other criteria included number of generations to convergence and program efficiency (running time). This data was also captured and analyzed.
Chapter 4

Results

Experiments Conducted

Experiments were conducted on the four benchmark datasets mentioned in Table 1 of Chapter 3. The BTD and GTD datasets are smaller in size and are referred to as the small datasets. TSP-LIB-1060 and TSP-LIB-3038 are the large datasets. The number of centers \(k\) chosen for the datasets were as follows:

**Table 2. Choice of Centers for each Dataset used in this Research**

<table>
<thead>
<tr>
<th>Data set</th>
<th>Number of Centers ((k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTD</td>
<td>4 – 10</td>
</tr>
<tr>
<td>BTD</td>
<td>4 – 10</td>
</tr>
<tr>
<td>TSP-LIB-1060</td>
<td>10, 100, 150</td>
</tr>
<tr>
<td>TSP-LIB-3038</td>
<td>10, 100, 500</td>
</tr>
</tbody>
</table>

These values were chosen based on the total number of data points in the datasets and also on experiments carried out in prior research work. For each set of centers,
experiments were carried out using the eight different methods and the four different distance metrics mentioned in Chapter 1.

For the small datasets, the six GAs were run 10 times, for 100 generations with a population size of 100, with mutation probability of 0.001 and crossover probability of 0.8, for each parameter set. In addition, for comparison, \(k\)-means and \(k\)-medoids were separately run 100,000 times for each parameter set for analysis and comparison. This number is equal to the number of times \(k\)-means (\(k\)-medoids) is run by the GA; it is equal to the product of the number of generations, the population size and total number of runs used in the GAs.

For the large datasets, the six GAs were run 4 times, for 100 generations with a population size of 20, with mutation probability of 0.001 and crossover probability of 0.8 for \(k=10\), for each parameter set. In addition, for comparison, \(k\)-means and \(k\)-medoids were separately run 8,000 times for each parameter set. The six GAs were run 4 times, for 20 generations with a population size of 20, with mutation probability of 0.001 and crossover probability of 0.8 for \(k > 10\), for each parameter set. \(k\)-means and \(k\)-medoids were run separately 1,600 times for each parameter set.

The naming convention used to describe the six GA methods is shown in Table 3.
Table 3. Naming Convention used for the Genetic Algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Name used</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA without crossover using $k$-means</td>
<td>GA1-means</td>
</tr>
<tr>
<td>GA with random crossover using $k$-means</td>
<td>GA0-means</td>
</tr>
<tr>
<td>GA with region-based crossover using $k$-means</td>
<td>GA2-means</td>
</tr>
<tr>
<td>GA without crossover using $k$-medoids</td>
<td>GA1-medoids</td>
</tr>
<tr>
<td>GA with random crossover using $k$-medoids</td>
<td>GA0-medoids</td>
</tr>
<tr>
<td>GA with region-based crossover using $k$-medoids</td>
<td>GA2-medoids</td>
</tr>
</tbody>
</table>

Small Datasets

Data

The data collected for the small data sets for each parameter set of all methods included:

1) Best $\text{SSE}_T$ after all runs

2) The average (over all runs) of the average $\text{SSE}_T$ per generation for the GAs

Tables 4, 5, 6, and 7, show the best $\text{SSE}_T$ data that was collected for the small data sets. If a method generates a best $\text{SSE}_T$ value better than the other methods, it is highlighted in bold.
Table 4. Best SSE Values after all Runs for the GTD Dataset using k-means

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>$k$-means</th>
<th>GA0-means</th>
<th>GA1-means</th>
<th>GA2-means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>49600.59</td>
<td>49600.59</td>
<td>49600.59</td>
<td>49600.59</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>38716.02</td>
<td>38716.02</td>
<td>38716.02</td>
<td>38716.02</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>30535.39</td>
<td>30535.39</td>
<td>30535.39</td>
<td>30535.39</td>
</tr>
<tr>
<td>Euclidean</td>
<td>7</td>
<td>24432.57</td>
<td>24432.57</td>
<td>24432.57</td>
<td>24432.57</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>21483.02</td>
<td>21483.02</td>
<td>21483.02</td>
<td>21483.02</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>18550.44</td>
<td>18550.44</td>
<td>18550.44</td>
<td>18550.44</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>16307.96</td>
<td>16307.96</td>
<td>16307.96</td>
<td>16307.96</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>78371.25</td>
<td>78371.25</td>
<td>78371.25</td>
<td>78371.25</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>62114.71</td>
<td>62114.71</td>
<td>62114.71</td>
<td>62114.71</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>50457.03</td>
<td>50457.03</td>
<td>50457.03</td>
<td>50457.03</td>
</tr>
<tr>
<td>Manhattan</td>
<td>7</td>
<td>40458.98</td>
<td>40458.98</td>
<td>40458.98</td>
<td>40458.98</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>34840.74</td>
<td>34840.74</td>
<td>34840.74</td>
<td>34840.74</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>29301.45</td>
<td>29301.45</td>
<td>29301.45</td>
<td>29301.45</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>25918.06</td>
<td>25918.06</td>
<td>25918.06</td>
<td>25918.06</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>40408.35</td>
<td>40408.35</td>
<td>40408.35</td>
<td>40408.35</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>30722.77</td>
<td>30722.77</td>
<td>30722.77</td>
<td>30722.77</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>24023.63</td>
<td>24023.63</td>
<td>24023.63</td>
<td>24023.63</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>7</td>
<td>19099.01</td>
<td>19099.01</td>
<td>19099.01</td>
<td>19099.01</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>16424.27</td>
<td>16424.27</td>
<td>16424.27</td>
<td>16424.27</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>14275.97</td>
<td>14275.97</td>
<td>14275.97</td>
<td>14275.97</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>12530.40</td>
<td>12530.40</td>
<td>12530.40</td>
<td>12530.40</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>44808.34</td>
<td>44808.34</td>
<td>44808.34</td>
<td>44808.34</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>34777.13</td>
<td>34777.13</td>
<td>34777.13</td>
<td>34777.13</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>27163.65</td>
<td>27163.65</td>
<td>27163.65</td>
<td>27163.65</td>
</tr>
<tr>
<td>Minkowski</td>
<td>7</td>
<td>21735.81</td>
<td>21735.81</td>
<td>21735.81</td>
<td>21735.81</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>18707.41</td>
<td>18707.41</td>
<td>18707.41</td>
<td>18707.41</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>16592.78</td>
<td>16592.78</td>
<td>16592.78</td>
<td>16592.78</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14411.20</td>
<td>14411.20</td>
<td>14411.20</td>
<td>14411.20</td>
</tr>
</tbody>
</table>
Table 5. Best $\text{SSE}_T$ Values after all Runs for the GTD Dataset using $k$-medoids

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>$k$-medoids</th>
<th>GA0-medoids</th>
<th>GA1-medoids</th>
<th>GA2-medoids</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>53305.00</td>
<td>53305.00</td>
<td>53305.00</td>
<td>53305.00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>41882.00</td>
<td>41882.00</td>
<td>41882.00</td>
<td>41882.00</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>33706.00</td>
<td>33706.00</td>
<td>33706.00</td>
<td>33706.00</td>
</tr>
<tr>
<td>Euclidean</td>
<td>7</td>
<td>28462.00</td>
<td>28462.00</td>
<td>28462.00</td>
<td>28462.00</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>24347.00</td>
<td>24347.00</td>
<td>24347.00</td>
<td>24347.00</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>21563.00</td>
<td>21563.00</td>
<td>21563.00</td>
<td>21563.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>19067.00</td>
<td>19067.00</td>
<td>19067.00</td>
<td>19067.00</td>
</tr>
<tr>
<td>Manhattan</td>
<td>4</td>
<td>87362.00</td>
<td>87362.00</td>
<td>87362.00</td>
<td>87362.00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>70314.00</td>
<td>70314.00</td>
<td>70314.00</td>
<td>70314.00</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>57956.00</td>
<td>57956.00</td>
<td>57956.00</td>
<td>57956.00</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>45680.00</td>
<td>45680.00</td>
<td>45680.00</td>
<td>45680.00</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>39847.00</td>
<td>39847.00</td>
<td>39847.00</td>
<td>39847.00</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>35181.00</td>
<td>35181.00</td>
<td>35181.00</td>
<td>35181.00</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>30972.00</td>
<td>30972.00</td>
<td>30972.00</td>
<td>30972.00</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>4</td>
<td>41369.57</td>
<td>41369.57</td>
<td>41369.57</td>
<td>41369.57</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>32635.00</td>
<td>32635.00</td>
<td>32635.00</td>
<td>32635.00</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>26172.31</td>
<td>26172.31</td>
<td>26172.31</td>
<td>26172.31</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>22245.78</td>
<td>22245.78</td>
<td>22245.78</td>
<td>22245.78</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>19263.78</td>
<td>19263.78</td>
<td>19263.78</td>
<td>19263.78</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>16904.71</td>
<td>16904.71</td>
<td>16904.71</td>
<td>16904.71</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>14986.71</td>
<td>14986.71</td>
<td>14986.71</td>
<td>14986.71</td>
</tr>
<tr>
<td>Minkowski</td>
<td>4</td>
<td>46854.93</td>
<td>46854.93</td>
<td>46854.93</td>
<td>46854.93</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>36749.59</td>
<td>36749.59</td>
<td>36749.59</td>
<td>36749.59</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>29471.92</td>
<td>29471.92</td>
<td>29471.92</td>
<td>29471.92</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>25178.56</td>
<td>25178.56</td>
<td>25178.56</td>
<td>25178.56</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>21689.70</td>
<td>21689.70</td>
<td>21689.70</td>
<td>21689.70</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>19143.64</td>
<td>19143.64</td>
<td>19143.64</td>
<td>19143.64</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>16862.42</td>
<td>16862.42</td>
<td>16862.42</td>
<td>16862.42</td>
</tr>
</tbody>
</table>
Table 6. Best $\text{SSE}_T$ Values after all Runs for the BTD Dataset using $k$-means

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>$k$-means</th>
<th>GA0-Means</th>
<th>GA1-Means</th>
<th>GA2-Means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>180.91</td>
<td>180.91</td>
<td>180.91</td>
<td>180.91</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>160.23</td>
<td>160.23</td>
<td>160.23</td>
<td>160.23</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>141.46</td>
<td>141.46</td>
<td>141.46</td>
<td>141.46</td>
</tr>
<tr>
<td>Euclidean</td>
<td>7</td>
<td>126.28</td>
<td>126.28</td>
<td>126.28</td>
<td>126.28</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>113.50</td>
<td>113.50</td>
<td>113.50</td>
<td>113.50</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>102.76</td>
<td>102.76</td>
<td>102.76</td>
<td>102.76</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>93.72</td>
<td>93.38</td>
<td>92.84</td>
<td>92.84</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>525.01</td>
<td>525.01</td>
<td>525.01</td>
<td>525.01</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>431.32</td>
<td>431.32</td>
<td>431.32</td>
<td>431.32</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>373.14</td>
<td>373.14</td>
<td>373.14</td>
<td>373.14</td>
</tr>
<tr>
<td>Manhattan</td>
<td>7</td>
<td>333.91</td>
<td>333.91</td>
<td>333.91</td>
<td>332.83</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>299.54</td>
<td>299.54</td>
<td>299.54</td>
<td>299.54</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>274.44</td>
<td>274.44</td>
<td>274.44</td>
<td>273.63</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>252.14</td>
<td>252.14</td>
<td>252.14</td>
<td>251.70</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>105.35</td>
<td>105.35</td>
<td>105.35</td>
<td>105.35</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>94.05</td>
<td>94.05</td>
<td>94.05</td>
<td>94.05</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>82.70</td>
<td>82.70</td>
<td>82.70</td>
<td>82.70</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>7</td>
<td>73.75</td>
<td>73.75</td>
<td>73.75</td>
<td>73.63</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>65.23</td>
<td>65.23</td>
<td>65.23</td>
<td>65.23</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>57.97</td>
<td>57.97</td>
<td>57.97</td>
<td>56.65</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>50.27</td>
<td>50.27</td>
<td>50.27</td>
<td>50.27</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>138.05</td>
<td>138.05</td>
<td>138.05</td>
<td>138.05</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>122.92</td>
<td>122.92</td>
<td>122.92</td>
<td>122.92</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>109.79</td>
<td>109.79</td>
<td>109.79</td>
<td>109.79</td>
</tr>
<tr>
<td>Minkowski</td>
<td>7</td>
<td>99.18</td>
<td>99.18</td>
<td>99.18</td>
<td>98.77</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>88.91</td>
<td>88.91</td>
<td>88.91</td>
<td>88.26</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>79.74</td>
<td>79.06</td>
<td>78.99</td>
<td>78.60</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>69.62</td>
<td>69.62</td>
<td>69.62</td>
<td>69.22</td>
</tr>
</tbody>
</table>
Table 7. Best \( \text{SSE}_T \) Values after all Runs for the BTD Dataset using \( k \)-medoids

<table>
<thead>
<tr>
<th>Metric</th>
<th>( k )</th>
<th>k-medoids</th>
<th>GA0-Medoids</th>
<th>GA1-Medoids</th>
<th>GA2-Medoids</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>4</td>
<td>196.14</td>
<td>196.14</td>
<td>196.14</td>
<td>196.14</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>177.47</td>
<td>177.47</td>
<td>177.47</td>
<td>177.47</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>160.56</td>
<td>160.56</td>
<td>160.56</td>
<td>160.56</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>146.25</td>
<td>146.25</td>
<td>146.25</td>
<td>146.25</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>134.93</td>
<td>134.93</td>
<td>134.93</td>
<td>134.26</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>123.51</td>
<td>123.51</td>
<td>123.51</td>
<td>122.70</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>113.82</td>
<td>113.82</td>
<td>113.82</td>
<td>113.40</td>
</tr>
<tr>
<td>Manhattan</td>
<td>4</td>
<td>568.38</td>
<td>568.38</td>
<td>568.38</td>
<td>568.38</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>500.06</td>
<td>500.06</td>
<td>500.06</td>
<td>500.06</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>446.66</td>
<td>446.66</td>
<td>446.66</td>
<td>446.66</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>400.34</td>
<td>400.34</td>
<td>400.34</td>
<td>400.34</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>364.26</td>
<td>364.26</td>
<td>364.26</td>
<td>364.26</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>332.72</td>
<td>332.72</td>
<td>332.72</td>
<td>330.39</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>305.40</td>
<td>305.40</td>
<td>305.40</td>
<td>303.50</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>4</td>
<td>118.44</td>
<td>118.44</td>
<td>118.44</td>
<td>118.44</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>105.73</td>
<td>105.73</td>
<td>105.73</td>
<td>105.73</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>93.61</td>
<td>93.61</td>
<td>93.61</td>
<td>93.61</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>85.22</td>
<td>85.22</td>
<td>85.22</td>
<td>85.22</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>78.21</td>
<td>78.21</td>
<td>78.21</td>
<td>78.21</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>71.64</td>
<td>71.64</td>
<td>71.64</td>
<td>71.49</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>64.70</td>
<td>64.70</td>
<td>64.70</td>
<td>64.70</td>
</tr>
<tr>
<td>Minkowski</td>
<td>4</td>
<td>151.56</td>
<td>151.56</td>
<td>151.56</td>
<td>151.56</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>136.62</td>
<td>136.62</td>
<td>136.62</td>
<td>136.62</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>123.01</td>
<td>123.01</td>
<td>123.01</td>
<td>123.01</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>111.79</td>
<td>111.79</td>
<td>111.79</td>
<td>111.79</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>102.81</td>
<td>102.81</td>
<td>102.81</td>
<td>102.81</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>95.11</td>
<td>95.11</td>
<td>95.11</td>
<td>95.08</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>87.61</td>
<td>87.61</td>
<td>87.61</td>
<td>87.25</td>
</tr>
</tbody>
</table>

Line plots were made to analyze data collected for the average \( \text{SSE}_T \) per generation for the GAs. Figure 2 describes a typical plot showing the behavior of the
three GAs for the small data sets. Additional line charts for the smaller data sets can be found in appendix A.

![Figure 3. Average SSE_T per Generation for the Three GAs using the GTD Dataset, k-means, and the Chebyshev Metric for k=9](image)

**Analysis**

In Tables 4 and 5, representing the GTD data set, it can be seen that there is no difference in best SSE_T values calculated by all methods for a given parameter set. But what was clearly observed is that GA2 improved more rapidly on average, per generation, than the other two methods, and converged to a lower SSE value. This was also observed for GA1 over GA0. k-means and k-medoids typically took many more runs to produce solutions as good as, or close to, the GAs for all parameter sets, as compared to the number of k-means/k-medoids runs required to reach optimum solutions by the GAs.
In Tables 6 and 7, representing the BTD data set, it can be seen that there is no difference in best \( \text{SSE}_T \) values calculated by all methods for a given parameter set for \( k=4 \) to \( k=8 \). GA2 found better solutions for \( k=9 \) and \( k=10 \) in most cases. Again it was clearly observed that GA2 always found the best \( \text{SSE}_T \) values in fewer generations than the other methods. It was also observed that GA1 found these solutions in fewer generations than GA0. Once again, \( k \)-means and \( k \)-medoids typically took many more runs to produce solutions as good as, or close to, the GAs for all parameter sets, as compared to the number of \( k \)-means/\( k \)-medoids runs required to reach optimum solutions by the GAs.

In the line plots, there was not much difference in the values of the three GAs for \( k=4 \) to \( k=8 \). For \( k=9 \) and \( k=10 \) most plots looked like the one shown in Figure 3. It was repeatedly observed that the three GAs start out with average \( \text{SSE}_T \) values that are close to each other in the first generation. Then there is typically a sharp fall in average \( \text{SSE}_T \) values for the first 15 generations for all GAs. During this phase GA1 typically outperformed GA0 and GA2. From generations 15-100 GA1 showed less progress than the other two GAs. GA2 always ended up showing best average \( \text{SSE}_T \) values for these generations.

The small data sets have relatively few data points and hence are easy to cluster especially when the numbers of centers are low. This is why it was possible to get best known \( \text{SSE}_T \) values in most cases within a few runs of \( k \)-means/\( k \)-medoids or a few generations of the GAs. As the number of centers approached 10, the clustering seemed to get more challenging and the superiority of GA2 could be seen. Since the BTD dataset has four dimensions as compared to the two-dimensional GTD dataset, it was harder to arrive at good solutions for \( k \geq 9 \). For example \( k \)-means took only two runs to arrive at
the best $SSE_T$ value for the GTD dataset using the Euclidean metric for $k=4$ and over 80,000 runs to do the same for the BTD dataset for $k=10$.

**Large Datasets**

*Data*

The data collected for the large data sets for each parameter set of all methods included:

1) Best $SSE_T$ after all runs

2) The average (over all runs) of the average $SSE_T$ per generation for the GAs

Tables 8, 9, 10 and 11 show the best $SSE_T$ data that was collected for the large data sets. 2D bar charts of this data can be found in appendix B. If a method generates a best $SSE_T$ value better than the other methods, it is highlighted in bold.
Table 8. Best $\text{SSE_T}$ Values after All Runs for the TSP-LIB-1060 Dataset using $k$-means

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>$k$-means</th>
<th>GA0-Means</th>
<th>GA1-Means</th>
<th>GA2-Means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>1754891365</td>
<td>1754840260</td>
<td>1754858106</td>
<td>1754840260</td>
</tr>
<tr>
<td>Euclidean</td>
<td>100</td>
<td>114579961</td>
<td>110397077</td>
<td>112908111</td>
<td>110072048</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>69042600</td>
<td>68622949</td>
<td>68459531</td>
<td>68245297</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2859229572</td>
<td>2857120142</td>
<td>2855170695</td>
<td>2854048790</td>
</tr>
<tr>
<td>Manhattan</td>
<td>100</td>
<td>182619925</td>
<td>179310490</td>
<td>180085908</td>
<td>177507634</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>114548544</td>
<td>110186333</td>
<td>110774604</td>
<td>106619843</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1360704030</td>
<td>1360544451</td>
<td>1360544451</td>
<td>1360509992</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>100</td>
<td>90004641</td>
<td>88965597</td>
<td>87863277</td>
<td>87171167</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>56763446</td>
<td>55502277</td>
<td>54894521</td>
<td>54818739</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1545551554</td>
<td>1545551554</td>
<td>1545551554</td>
<td>1545395961</td>
</tr>
<tr>
<td>Minkowski</td>
<td>100</td>
<td>99676911</td>
<td>98641091</td>
<td>98867370</td>
<td>96925467</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>63777711</td>
<td>62350118</td>
<td>59540390</td>
<td>59387919</td>
</tr>
</tbody>
</table>
Table 9. Best $\text{SSE}_T$ Values after All Runs for the TSP-LIB-1060 Dataset using $k$-medoids

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>$k$-medoids</th>
<th>GA0-Medoids</th>
<th>GA1-Medoids</th>
<th>GA2-Medoids</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>10</td>
<td>1815760001</td>
<td>1798495399</td>
<td>1798918825</td>
<td>1790224017</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>138999655</td>
<td>135593537</td>
<td>136992569</td>
<td>132820061</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>85332193</td>
<td>83875953</td>
<td>83016815</td>
<td>82951650</td>
</tr>
<tr>
<td>Manhattan</td>
<td>10</td>
<td>2896041691</td>
<td>2878854232</td>
<td>2879471494</td>
<td>2877521449</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>222062497</td>
<td>214233505</td>
<td>218356450</td>
<td>208187955</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>142447701</td>
<td>133312514</td>
<td>136220934</td>
<td>132991722</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>10</td>
<td>1450764212</td>
<td>1417187059</td>
<td>1427699813</td>
<td>1410293913</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>114392853</td>
<td>108894364</td>
<td>109789751</td>
<td>104370542</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>67447745</td>
<td>66816314</td>
<td>66848798</td>
<td>65695302</td>
</tr>
<tr>
<td>Minkowski</td>
<td>10</td>
<td>1601380503</td>
<td>1591769134</td>
<td>1599470869</td>
<td>1590980856</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>125646350</td>
<td>116877610</td>
<td>122463845</td>
<td>114610472</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>77174284</td>
<td>76132005</td>
<td>75989379</td>
<td>72328140</td>
</tr>
</tbody>
</table>
Table 10. Best $SSE_T$ Values after All Runs for the TSP-LIB-3038 Dataset using $k$-means

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>$k$-means</th>
<th>GA0-Means</th>
<th>GA1-Means</th>
<th>GA2-Means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>560252059</td>
<td>560251201</td>
<td>560251920</td>
<td>560251201</td>
</tr>
<tr>
<td>Euclidean</td>
<td>100</td>
<td>50217668</td>
<td>49967644</td>
<td>50068719</td>
<td>49775353</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>8780259</td>
<td>8777427</td>
<td>8761527</td>
<td>8544511</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>928732398</td>
<td>927839968</td>
<td>927859644</td>
<td>927813058</td>
</tr>
<tr>
<td>Manhattan</td>
<td>100</td>
<td>81124651</td>
<td>81048182</td>
<td>81048182</td>
<td>80928516</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>14000146</td>
<td>13914439</td>
<td>14000146</td>
<td>13684336</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>440913617</td>
<td>440581324</td>
<td>440850819</td>
<td>440581324</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>100</td>
<td>40164984</td>
<td>39726749</td>
<td>39622843</td>
<td>39405006</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>6867796</td>
<td>6844429</td>
<td>6856738</td>
<td>6787559</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>495868718</td>
<td>495866334</td>
<td>495868718</td>
<td>495866334</td>
</tr>
<tr>
<td>Minkowski</td>
<td>100</td>
<td>44482736</td>
<td>44375443</td>
<td>44375443</td>
<td>44004318</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>7787378</td>
<td>7736013</td>
<td>7778594</td>
<td>7676421</td>
</tr>
</tbody>
</table>
Table 11. Best $\text{SSE}_T$ Values after All Runs for the TSP-LIB-3038 Dataset using $k$-medoids

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>$k$-medoids</th>
<th>GA0-Medoids</th>
<th>GA1-Medoids</th>
<th>GA2-Medoids</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>10</td>
<td>563889161</td>
<td>559527644</td>
<td>562489578</td>
<td>558438903</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>55529519</td>
<td>55529519</td>
<td>54981477</td>
<td>54737444</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>10679626</td>
<td>10669312</td>
<td>10659687</td>
<td>10595628</td>
</tr>
<tr>
<td>Manhattan</td>
<td>10</td>
<td>930957802</td>
<td>929204827</td>
<td>930143845</td>
<td>927410957</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>86587445</td>
<td>86587445</td>
<td>86436475</td>
<td>86197605</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>15266295</td>
<td>15254185</td>
<td>15246199</td>
<td>15221359</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>10</td>
<td>442904695</td>
<td>442646087</td>
<td>442787990</td>
<td>440838452</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>43638495</td>
<td>43538434</td>
<td>43638495</td>
<td>43294808</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>8832447</td>
<td>8823944</td>
<td>8823944</td>
<td>8739732</td>
</tr>
<tr>
<td>Minkowski</td>
<td>10</td>
<td>498757784</td>
<td>498184392</td>
<td>498074810</td>
<td>497187133</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>48968234</td>
<td>48875643</td>
<td>48723904</td>
<td>48714335</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>9737885</td>
<td>9704439</td>
<td>9716521</td>
<td>9653757</td>
</tr>
</tbody>
</table>

Line plots were made to analyze data collected for the average $\text{SSE}_T$ per generation for the GAs. Figure 4 is a typical plot showing the behavior of the three GAs for a large data set.
Figure 4. Average $\text{SSE}_T$ per Generation for the Three GAs Using the TSP-LIB-3038 Dataset, $k$-medoids, and the Euclidean Metric for $k=100$

In some cases it was clearly seen the GA2 outperformed the other two GAs to a great extent. Evidence of such behavior is shown in Figure 5.
Figure 5. Average $\text{SSE}_T$ per Generation for the Three GAs using the TSP-LIB-3038 Dataset, $k$-medoids, and the Manhattan Metric for $k=100$

Additional line charts for the large data sets can be found in appendix A.

Analysis

In Tables 8, 9, 10 and 11, it can be seen that GA2 almost always found better solutions than the other methods. It can also be seen that GA0 found better solutions than GA1 for most parameter sets. $k$-means and $k$-medoids typically did not find solutions as good as the GAs for all parameter sets. For the large datasets similar results as Figure 2 were seen for $k=10$. For values of $k > 10$, the superiority of GA2 was clearly evident across all metrics and ROCs.

The large data sets have many data points and hence are difficult to cluster especially when the number of centers is high. This is why these datasets show the true
potential of the GAs. GA2 almost always outperformed the other GAs. This was mainly because of using region-based crossover, replacing unused empty centers with random data points and using $k$-means as a genetic operator. GA0, in most cases, gave second best solutions. This is because GA0 performed bit-wise crossover and mutation after converting the centers to binary bit-strings. The down-side of GA0 was the running time which is discussed later in this chapter. GA1 had the fastest running time but the poorest results. This was mainly because the emphasis was on time rather than quality of solutions. Earlier literature shows the results produced by GA1 on the small datasets and it gave very good results in less time than other GAs because of using the one-step $k$-means operator instead of crossover. GA1 has not been run on the large datasets before and this research proved that the quality of solutions produced by GA1 was poor on large datasets as compared to GA0 and GA2, in most cases. Nevertheless, GA1 gave better results than $k$-means/$k$-medoids.

**Running Time**

$k$-means and $k$-medoids

All the experiments were carried out on an Intel core i7 machine running at 3.16 GHz with 4GB RAM. The most important factor in deciding the running time for $k$-means and $k$-medoids was the choice of dataset and the number of centers ($k$) chosen. The other sets of parameters did affect the running time, but less significantly.

For the small datasets, each $k$-means/medoid run took a few microseconds for all sets of centers and parameters. For the large data sets the running time varied widely
depending on the data set, number of centers chosen, and the metric. Tables 12 and 13 show running times for the large datasets.

Table 12. Approximate Running Time for each $k$-means/$k$-medoid Run for the TSP-LIB-1060 Dataset

<table>
<thead>
<tr>
<th>Metric</th>
<th>$k$</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>10</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>4</td>
</tr>
<tr>
<td>Manhattan</td>
<td>10</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>10</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>4</td>
</tr>
<tr>
<td>Minkowski</td>
<td>10</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>4</td>
</tr>
</tbody>
</table>
Table 13. Approximate Running Time for each \( k \)-means/\( k \)-medoids Run for the TSP-LIB-3038 Dataset

<table>
<thead>
<tr>
<th>Metric</th>
<th>( k )</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Manhattan</td>
<td>100</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Minkowski</td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>33</td>
</tr>
</tbody>
</table>

In the large datasets, \( k \)-medoids marginally took less time than \( k \)-means because convergence was typically reached in no more than 20 iterations across all parameter sets. The runs involving Chebyshev metric typically took longer to compute because the exponent \( p \) used in the metric formula was 1000. On the other hand, the runs using the Manhattan metric took less time than the other metrics because the value of \( p \) was 1. The value of \( p \) was 2 for the Euclidean metric and chosen as 3 for the Minkowski metric respectively.

To illustrate the large amounts of running time taken to run all experiments let us evaluate running time of only \( k \)-means using the Chebyshev metric for the TSP-LIB-3038 dataset for \( k=500 \). A single \( k \)-means run using these parameters took 37 seconds. This
was run 1600 times in order to evaluate $k$-means and compare it to other GAs. This experiment alone took over 16 hours to run.

*Genetic Algorithms*

All the GA methods had common features like initialization, selection, and fitness calculation. But features like mutation, crossover, bit stream conversion, replacement of empty centers, and genetic $k$-means operations were differed for GA0 and GA2. This played a crucial role in varying running times for the GAs. The fitness calculation involved running $k$-means/$k$-medoids and in most cases dominated the running time of the GA.

For example, running GA0-means for 10 generations with a population size of 10 for the TSP-LIB-1060 data set with $k=150$ for the Euclidean metric took 607 seconds. Running GA1 with the same set of parameters took only 100 seconds and GA2 took 278 seconds. Running $k$-means for 100 runs with the same set of parameters took 375 seconds. From this comparison it is evident that GA1 took the least amount of time. This is mainly because crossover was replaced by one-step of $k$-means. GA2 took almost as much time as running $k$-means 100 times. It was not necessary to re-calculate fitness in every generation especially if there is no crossover or mutation taking place.

Replacement of empty centers, using $k$-means as a genetic operator and region based crossover, were overheads of GA2, but these functions marginally increased running time of the GA. GA0 took much more time than GA2 mainly because the centers are reshuffled with every random crossover, so $k$-means/$k$-medians took many more iterations to converge. Conversion from real numbers to binary bit-streams and vice-versa
which needed to be done for each gene of the population in every generation also increased the running time for GA0.

Findings

Tables 14 and 15 show the ratios of the average of the average (over all runs) of the average $SSE_T$ per generation for the GAs for the large datasets.

Table 14. Ratio of Averages of GA0/GA2 and GA1/GA2 for Varied Metrics, Centers and ROCs for the TSP-LIB-1060 Dataset

<table>
<thead>
<tr>
<th>$k$</th>
<th>Metric</th>
<th>GA0/GA2 Means</th>
<th>GA1/GA2 Means</th>
<th>GA0/GA2 Medoids</th>
<th>GA1/GA2 Medoids</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Euclidean</td>
<td>1.000378143</td>
<td>1.000495423</td>
<td>1.005981089</td>
<td>1.014930742</td>
</tr>
<tr>
<td></td>
<td>Manhattan</td>
<td>1.001115032</td>
<td>1.000757373</td>
<td>1.007109589</td>
<td>1.013369941</td>
</tr>
<tr>
<td></td>
<td>Chebyshev</td>
<td>1.000170349</td>
<td>1.001114881</td>
<td>1.01249041</td>
<td>1.026830410</td>
</tr>
<tr>
<td></td>
<td>Minkowski</td>
<td>1.000260470</td>
<td>1.000100256</td>
<td>1.008218273</td>
<td>1.009654326</td>
</tr>
<tr>
<td>100</td>
<td>Euclidean</td>
<td>1.035089376</td>
<td>1.036455117</td>
<td>1.015616627</td>
<td>1.02009708</td>
</tr>
<tr>
<td></td>
<td>Manhattan</td>
<td>1.007511521</td>
<td>1.030327665</td>
<td>1.009315944</td>
<td>1.018741972</td>
</tr>
<tr>
<td></td>
<td>Chebyshev</td>
<td>1.012352897</td>
<td>1.003297601</td>
<td>1.03017768</td>
<td>1.046529561</td>
</tr>
<tr>
<td></td>
<td>Minkowski</td>
<td>1.016807201</td>
<td>1.025093157</td>
<td>1.029096752</td>
<td>1.053722271</td>
</tr>
<tr>
<td>150</td>
<td>Euclidean</td>
<td>1.013376847</td>
<td>1.010324399</td>
<td>1.019688274</td>
<td>1.019013600</td>
</tr>
<tr>
<td></td>
<td>Manhattan</td>
<td>1.009285565</td>
<td>1.026201993</td>
<td>1.025139773</td>
<td>1.034304341</td>
</tr>
<tr>
<td></td>
<td>Chebyshev</td>
<td>1.029742502</td>
<td>1.023408621</td>
<td>1.037398957</td>
<td>1.040298133</td>
</tr>
<tr>
<td></td>
<td>Minkowski</td>
<td>1.041993845</td>
<td>1.034413701</td>
<td>1.013752644</td>
<td>1.003080061</td>
</tr>
</tbody>
</table>
Table 15. Ratio of Averages of GA0/GA2 and GA1/GA2 for Varied Metrics, Centers and ROCs for the TSP-LIB-3038 Dataset

<table>
<thead>
<tr>
<th>( k )</th>
<th>Metric</th>
<th>GA0/GA2 Means</th>
<th>GA1/GA2 Means</th>
<th>GA0/GA2 Medoids</th>
<th>GA1/GA2 Medoids</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Euclidean</td>
<td>1.000061633</td>
<td>1.000127004</td>
<td>1.001282062</td>
<td>1.006047463</td>
</tr>
<tr>
<td></td>
<td>Manhattan</td>
<td>1.000532325</td>
<td>1.000887831</td>
<td>1.005018959</td>
<td>1.006082527</td>
</tr>
<tr>
<td></td>
<td>Chebyshev</td>
<td>1.000020108</td>
<td>1.000656688</td>
<td>1.004929617</td>
<td>1.007982057</td>
</tr>
<tr>
<td></td>
<td>Minkowski</td>
<td>1.000078451</td>
<td>1.000113116</td>
<td>1.002887259</td>
<td>1.001432746</td>
</tr>
<tr>
<td>100</td>
<td>Euclidean</td>
<td>1.002705900</td>
<td>1.006811437</td>
<td>1.012009741</td>
<td>1.005883685</td>
</tr>
<tr>
<td></td>
<td>Manhattan</td>
<td>1.007366658</td>
<td>1.006942475</td>
<td>1.015557108</td>
<td>1.014583853</td>
</tr>
<tr>
<td></td>
<td>Chebyshev</td>
<td>1.007101395</td>
<td>1.018161046</td>
<td>1.013556616</td>
<td>1.013607822</td>
</tr>
<tr>
<td></td>
<td>Minkowski</td>
<td>1.006569237</td>
<td>1.009022603</td>
<td>1.018520211</td>
<td>1.014954022</td>
</tr>
<tr>
<td>500</td>
<td>Euclidean</td>
<td>1.020746628</td>
<td>1.012518283</td>
<td>1.051801952</td>
<td>1.045031418</td>
</tr>
<tr>
<td></td>
<td>Manhattan</td>
<td>1.007167205</td>
<td>1.013144834</td>
<td>1.029465240</td>
<td>1.021863563</td>
</tr>
<tr>
<td></td>
<td>Chebyshev</td>
<td>1.012232678</td>
<td>1.015234882</td>
<td>1.00517342</td>
<td>1.013492604</td>
</tr>
<tr>
<td></td>
<td>Minkowski</td>
<td>1.005841592</td>
<td>1.015141917</td>
<td>1.007651413</td>
<td>1.012808735</td>
</tr>
</tbody>
</table>

From Tables 14 and 15 it is evident that GA2 performed better than all the other methods across all sets of parameters. This is evident from best SSE\(_T\) data and average SSE\(_T\) per generation data. The superior results of GA2 were more pronounced in the larger datasets due to the challenge of searching significantly larger search spaces. The main reasons for GA2’s superior performance were region-based crossover, replacement of empty centers and using \( k \)-means as a genetic operator. This was true since these three strategies are absent from GA0 and GA1. In GA2, the crossover operator exchanged centers that occupy a specific region of space. In spatial data clustering, center selection is spatially decomposable allowing centers in a specific region of space to make better candidates for crossover rather than random centers. This leads to significantly better solutions as compared to other crossover methods that exchange random centers. Moving
chromosome centers/genes to new positions after running \( k \)-means and replacing newly positioned centers that do not serve as centroids, helps keep the search focused on finding good partitions and reducing the possibility of having useless centers. Also, the superior performance of GA2 became more evident as the number of centers was increased across all datasets. This was because region-based crossover yielded sets of centers that converge with fewer iterations under \( k \)-means/\( k \)-medoids.

GA1 ran faster than other methods across all sets of parameters. This was mainly because of using one-step of \( k \)-means/\( k \)-medoids instead of crossover. This approach did not require re-calculating the fitness by re-running \( k \)-means/\( k \)-medoids every time only crossover was performed. Fitness needed to be recalculated only when mutation occurred which was very rare. Fitness calculation dominated running time of the GAs.

GA1 almost always performed better than the other GA methods in the first few generations but then showed little or no evolution after 10 generations. This was because using one step of \( k \)-means for crossover is a good approach for speeding up the GA but not for providing promising solutions over several generations.

GA0 showed better performance than GA1 across most sets of parameters but took more running time than all other methods. This was because centers are reshuffled with every random crossover, so \( k \)-means/\( k \)-medians took many more iterations to converge.

With respect to quality of solution, there was no dependence between GA0 and choice of metric. The same was true for GA1 and GA2. This can be seen from the data in
Tables 14 and 15 and by comparing the “average $SSE_T$ per generation” line plots in appendix A, across different metrics.

There was a slight dependence between the GAs and ROCs in the larger datasets for $k=10$. The superiority of GA0 and GA2 was a little more pronounced when medoids were used instead of means. This can be seen by comparing the “average $SSE_T$ per generation” line plots in appendix A, across different ROCs. For other values of $k$ there was no dependence between a GA and a ROC. This can be seen from the data in Tables 14 and 15 and once again by comparing the “average $SSE_T$ per generation” line plots in appendix A, across different ROCs.

The best $SSE_T$ values presented in this research did not always match the best-known $SSE_T$ values in prior literature because the GAs were not run for as many generations with large population sizes. Determining the best $SSE_T$ values for a given parameter set, was not the goal of this research. The goal of this research was to prove that GA2 performs better than other methods across a range of datasets, metric, ROCs and number of centers. This goal has been accomplished and is evident in the results.
Chapter 5
Conclusions, Implications, Recommendations, and Summary

Conclusions

This research aimed at proving that the region-based genetic algorithm (RBGA) is a promising algorithm for partitional $k$-means clustering across a range of objective functions. (Laszlo & Mukherjee, 2007) proved that the region-based method performed better than other GA methods when centroids were used as the representative object of the cluster and the Euclidean metric was used as distance metric. This research went further in the assessment of the region-based approach by varying the range of objective functions, and successfully proved that it performs better than other methods across all sets of parameters on four benchmark datasets.

This hypothesis can be justified by the experiments conducted, and the results acquired in this research. The RBGA was compared to two other GA methods and the generic $k$-means / $k$-medoids approach. Four benchmark datasets with two- and four-dimensional data were used in the study. The metrics used for clustering were Euclidean, Manhattan, Chebyshev and Minkowski. The representative objects of the cluster that were used were centroids and medoids.
Across all these parameters the RBGA found better or comparable results than the other methods. The superior performance of the RBGA was more pronounced in the large datasets that consisted of larger search spaces and more data points. Superior performance was also seen when the number of centers \( k \) was higher for all datasets.

The criterion used to measure the performance of all methods was the total SSE of all the partitions of the search space. The lower the value of \( \text{SSE}_T \), the better is the performance. The data acquired in this research included best \( \text{SSE}_T \) values for all methods and average (over all runs) of the average \( \text{SSE}_T \) values per generation for the GAs. This data was plotted into bar graphs and line plots and is displayed in the appendixes. From the 80 line plots and 48 bar graphs in the appendixes it is clearly evident that the RBGA found better partitions than other methods across all sets of parameters for all four datasets that were used in the study.

**Implications**

This study implies that using genetic algorithms to solve the center selection problem of \( k \)-means is promising over a range of metric and ROCs. This can be seen from the results in Appendix B which almost always shows that all the GA methods provided better results than simply running \( k \)-means or \( k \)-medoids across all sets of parameters for all data sets. This is mainly because of the selection process of a GA which selects the fittest individuals to breed in the next generation, as opposed to \( k \)-means / \( k \)-medoids which randomly select centers for every run.
This study also reinforces the concept stated by (Laszlo & Mukherjee, 2007), that the center selection problem of k-means is spatially decomposable to a large extent. The RBGA makes use of spatial decomposition of the search space by creating regions and performing crossover between chromosome centers in a specific region. This concept has led to significant improvement in the partitions produced.

This spatial decomposition theory could be applied to other steps of a genetic algorithm like initialization, mutation, selection and fitness calculation to produce better results. This concept could also be directly applied to the k-means algorithm so that the initial set of centers could be spatially decomposed into regions rather than randomly selected.

**Recommendations**

This study used four datasets that had data points up to four dimensions. To assess the RBGA more thoroughly datasets with higher dimensions could be used. It would be interesting to see the nature of the GA methods as higher dimensions are used.

This study was limited to four distance metric and two ROCs. Other distance metrics and ROCs could be used to assess the GAs. Some recommendations for metrics are the Pearson correlation metric, and the Spearman correlation metric. Some recommendations for ROCs are medians, the Chebyshev center, and geometric mean.
This study proves that spatially decomposing the search space into half-spaces significantly improves GA performance. Nevertheless, such half-spaces might not be the ideal region within which crossover is performed. Different methods of spatial decomposition should be tried and tested.

**Summary**

Data clustering deals with separating data points into clusters or partitions. Clustering has many useful applications in economics, science and engineering. Broadly speaking, there are two types of data clustering techniques, partitional and hierarchical. Out of the various partitional clustering algorithms, the k-means algorithm is most popular, simple and efficient one. One problem with the k-means algorithm is that the quality of partitions produced is highly dependent on the initial selection of centers. This problem has been tackled using genetic algorithms (GA). The region-based genetic algorithm (RBGA) has proven to be an effective GA technique when the centroid was used as the representative object of a cluster and the Euclidean distance was used as the distance metric. This research aimed at assessing the region-based crossover genetic algorithm with a variety of representative objects of clusters and distance metrics.

In order to perform this assessment it was required to select other GA methods, data sets, distance metrics and representative objects of the clusters (ROC). After careful literature review the GA methods chosen were GA with random crossover (Babu & Murthy, 1993), and GA with no crossover (Krishna & Murthy, 1999). The ROCs selected were centroid and medoid, and the distance metrics chosen were Euclidean, Manhattan,
Chebyshev and Minkowski. The four benchmark datasets that were chosen were GTD, BTD, TSP-LIB-1060 and TSP-LIB-3038. The main measurable criterion to determine the quality of the solution was the total sum of the squared error ($SSE_T$) of the partitions of the search space.

Code was written in C++ to assess all these GAs across the above-mentioned metrics, ROCs and datasets. The code was run on a PC and took months of running time to produce all the required output data. The code prompted the user for input parameters. The output of the code consisted of best $SSE_T$, average (over all runs) of the average $SSE_T$ values per generation for the GAs. This data was plotted on bar graphs and line plots. Bar graphs were used to depict best $SSE_T$ values and line plots were used to compare average $SSE_T$ values between GA methods.

The results proved that the RBGA performed better than all other methods across all sets of parameters on all datasets. This conclusion could be drawn because the RBGA found comparable or best partitions (lowest $SSE_T$ values) than other methods in all cases. This research also revealed some information about the metrics, ROCs and methods that are not directly related to the goal of the research. The results did show that there was no clear dependence between a GA method and a metric or a GA method and an ROC. It also showed that GA1 ran the fastest while GA0 ran the slowest. The GAs ran fastest when the Manhattan metric was used. The number of iterations taken for convergence was much lesser in the case of medoids rather than means.

The two main concepts that drove the RBGA to better results were intensification and diversification. The crossover operations intensified the search. The premise of the region-based crossover was that it spatially decomposed the search space into regions and
performed crossover between centers of individuals that lied in a region of space, rather than performing crossover between random centers. Two diversification concepts that led to superior results by this GA were using $k$-means as a genetic operator, which moved genes (centers) to new positions after applying $k$-means, and replacement of unused centers with random data points. In addition to these concepts, the selection operator helped intensification while mutation further led to diversification.

This study implies that in the field of partitional data clustering, spatially decomposing a search space into regions could be significantly beneficial. The RBGA spatially decomposes the search space into half-spaces in order to perform region-based crossover. This study proved that this concept gave significantly superior results under a variety of objective functions, making it a promising technique.

Though the region based approach was used only for crossover, this concept could be extended to other parts of a GA like mutation, selection, initialization and fitness calculation. Spatial decomposition techniques could also be used on other partitional clustering algorithms making them more efficient.
Appendix A

Plots Showing Average (over all runs) of Avg. \(SSE_T\) per Generation for GAs

The following plots in this appendix compare the three GA methods by showing the average, over all runs, of the average \(SSE_T\) per generation. The dataset, number of centers, and metric used are stated below each plot. For the small datasets, the GAs were run 10 times, for 100 generations with a population size of 100. For the large datasets, the GAs were run 4 times, with a population size of 20 for 100 generations for \(k=10\) and for 20 generations for \(k>10\). The mutation probability of 0.001 and crossover probability of 0.8 was used for all parameter sets.

Dataset = GTD, \(k = 9\), Metric = Euclidean
Dataset = GTD, $k = 9$, Metric = Manhattan

Dataset = GTD, $k = 9$, Metric = Chebyshev
Dataset = GTD, $k = 9$, Metric = Minkowski

Dataset = GTD, $k = 9$, Metric = Euclidean
Dataset = GTD, $k = 9$, Metric = Manhattan

Dataset = GTD, $k = 9$, Metric = Chebyshev
Dataset = GTD, $k = 9$, Metric = Minkowski

Dataset = GTD, $k = 10$, Metric = Euclidean
Dataset = GTD, $k = 10$, Metric = Manhattan

Dataset = GTD, $k = 10$, Metric = Chebyshev
Dataset = GTD, $k = 10$, Metric = Minkowski

Dataset = GTD, $k = 10$, Metric = Euclidean
Dataset = GTD, $k = 10$, Metric = Manhattan

Dataset = GTD, $k = 10$, Metric = Chebyshev
Dataset = GTD, $k = 10$, Metric = Minkowski

Dataset = BTD, $k = 9$, Metric = Euclidean
Dataset = BTD, $k = 9$, Metric = Manhattan

Dataset = BTD, $k = 9$, Metric = Chebyshev
Dataset = BTD, $k = 9$, Metric = Minkowski

Dataset = BTD, $k = 9$, Metric = Euclidean
Dataset = BTD, $k = 9$, Metric = Manhattan

Dataset = BTD, $k = 9$, Metric = Chebyshev
Dataset = BTD, $k = 9$, Metric = Minkowski

Dataset = BTD, $k = 10$, Metric = Euclidean
Dataset = BTD, $k = 10$, Metric = Manhattan

Dataset = BTD, $k = 10$, Metric = Chebyshev
Dataset = BTD, $k = 10$, Metric = Minkowski

Dataset = BTD, $k = 10$, Metric = Euclidean
Dataset = BTD, $k = 10$, Metric = Manhattan

Dataset = BTD, $k = 10$, Metric = Chebyshev
Dataset = BTD, \( k = 10 \), Metric = Minkowski

Dataset = TSP-LIB-1060, \( k = 10 \), Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 10$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 10$, Metric = Chebyshev
Dataset = TSP-LIB-1060, \( k = 10 \), Metric = Minkowski

Dataset = TSP-LIB-1060, \( k = 10 \), Metric = Euclidean
Dataset = TSP-LIB-1060 $k = 10$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 10$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 10$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-1060 $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 150$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 150$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 150$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 150$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 150$, Metric = Euclidean
Dataset = TSP-LIB-1060 $k = 150$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 150$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 150$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 10$, Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 10$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 10$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 10$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 10$, Metric = Euclidean
Dataset = TSP-LIB-3038 \( k = 10 \), Metric = Manhattan

Dataset = TSP-LIB-3038, \( k = 10 \), Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 10$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 500$, Metric = Euclidean
Dataset = TSP-LIB-3038, \( k = 500 \), Metric = Manhattan

Dataset = TSP-LIB-3038, \( k = 500 \), Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 500$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 500$, Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 500$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 500$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 500$, Metric = Minkowski
Appendix B

Charts Showing Best $SSE_T$ Values of Large Datasets across all Parameter Sets

The following plots in this appendix compare the best $SSE_T$ values of the three GA methods and $k$-means/$k$-medoids after all runs. The dataset, number of centers, and metric used are stated below each plot. For the large datasets, the GAs were run 4 times, with a population size of 20 for 100 generations for $k=10$ and for 20 generations for $k>10$. The mutation probability of 0.001 and crossover probability of 0.8 was used for all parameter sets. In addition, for comparison, $k$-means and $k$-medoids were separately run 8,000 times for $k=10$, and 1,600 times for $k>10$.

Dataset = TSP-LIB-1060, $k = 10$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 10$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 10$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 10$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 10$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 10$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 10$, Metric = Chebyshev
1. Dataset = TSP-LIB-1060, $k = 10$, Metric = Minkowski

2. Dataset = TSP-LIB-1060, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 150$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 150$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 150$, Metric = Chebyshev
Dataset = TSP-LIB-1060, $k = 150$, Metric = Minkowski

Dataset = TSP-LIB-1060, $k = 150$, Metric = Euclidean
Dataset = TSP-LIB-1060, $k = 150$, Metric = Manhattan

Dataset = TSP-LIB-1060, $k = 150$, Metric = Chebyshev
Dataset = TSP-LIB-1060, \( k = 150 \), Metric = Minkowski

Dataset = TSP-LIB-3038, \( k = 10 \), Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 10$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 10$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 10$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 10$, Metric = Euclidean
Objectives and Results

1. Dataset = TSP-LIB-3038, $k = 10$, Metric = Manhattan

2. Dataset = TSP-LIB-3038, $k = 10$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 10$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 100$, Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 100$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 100$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 100$, Metric = Minkowski

Dataset = TSP-LIB-3038, $k = 500$, Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 500$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 500$, Metric = Chebyshev
Dataset = TSP-LIB-3038, \( k = 500 \), Metric = Minkowski

Dataset = TSP-LIB-3038, \( k = 500 \), Metric = Euclidean
Dataset = TSP-LIB-3038, $k = 500$, Metric = Manhattan

Dataset = TSP-LIB-3038, $k = 500$, Metric = Chebyshev
Dataset = TSP-LIB-3038, $k = 500$, Metric = Minkowski
References


