7CCE4EEP

Clustering Service Area of Different Flying Base Station Located at the same Depot

Final Project Report

Author: Juan Sigman

Supervisor: Dr. Vasilis Friderikos

Student ID: 1903592

April 11, 2023

Abstract

The use of drones as flying base stations to provide connectivity in wireless networks has been explored in recent years. However, to maximise efficiency, a careful planning of drone trajectories is of paramount importance. In this project, given an arbitrary set of nodes for drones to visit, I aim to develop a clustering algorithm to help decide what nodes each drone visits. This algorithm was then validated with an existing clustering algorithm, Kmeans++, to show the potential of the new program.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 40 pages.

Juan Sigman

April 11, 2023

Acknowledgements

I would like to thank my supervisor, Dr Vasilis Friderikos, for all of his aid towards the completion of this project. I certainly could not have achieved what I did without him.

A Note on Figures

Every unreferenced figure in this paper was created by the author, either in MATLAB or in PowerPoint.

Contents

Executive Summary

In recent years, the use of UAVs or drones to provide cellular connectivity has attracted increasing attention and their potential for completing terrestrial networks has been investigated extensively in the literature. Due to their agility and flexibility, UAV base stations (BS) can handle temporary spikes in data demands during short-term events. However, the limited flying time of UAVs make it very difficult to supply all data when it's necessary [1]. .

Because of this, careful planning of the drones' trajectory is of paramount importance. If there are several drones that have to visit many points in their close proximity, not only the path for each drone has to be optimised to minimise flying time, but also the choice of what drones visit which points must be carefully thought out.

This project will focus on the latter problem. Given an arbitrary starting position with a home depot and a set of points for the drones to visit, I will be developing an algorithm to group up the points and choose what drone visits which group. Once that has been done, I will also give the optimized path for each drone. However, as this is not the focus of the project, I will not be developing my own algorithm for this, but rather use an existing one such as 2-opt [2].

This problem is similar to an already existing one called the Vehicle Routing Problem (VRP) [3]. Generally there are two ways to solve this problem: 1–step solutions or 2–step solutions. 1–step solutions do both the clustering and path optimization in one go, while 2–step solutions first cluster the nodes and then perform a path optimization for each cluster. For this project, I will be focusing on 2–step solvers.

Lastly, I will compare the performance of the designed algorithm with already existing ones such as K–Means++ to provide a final evaluation.

Chapter 1

Introduction

Mobile Data

The first mobile data services became available in 1991 with the development of the second generation (2G) of mobile phone technology [4]. Since then, mobile data demand has been rapidly increasing to staggering amounts. With the development of smartphones, tablets and other mobile devices, people are using more data than ever before. According to a report by Cisco, global mobile data traffic grew by 63% in 2020, reaching 77.5 exabytes per month, and is expected to continue increasing in the coming years. [5]. Figure 1.1 below shows the total data uploaded and downloaded between the years 2011 and 2021.

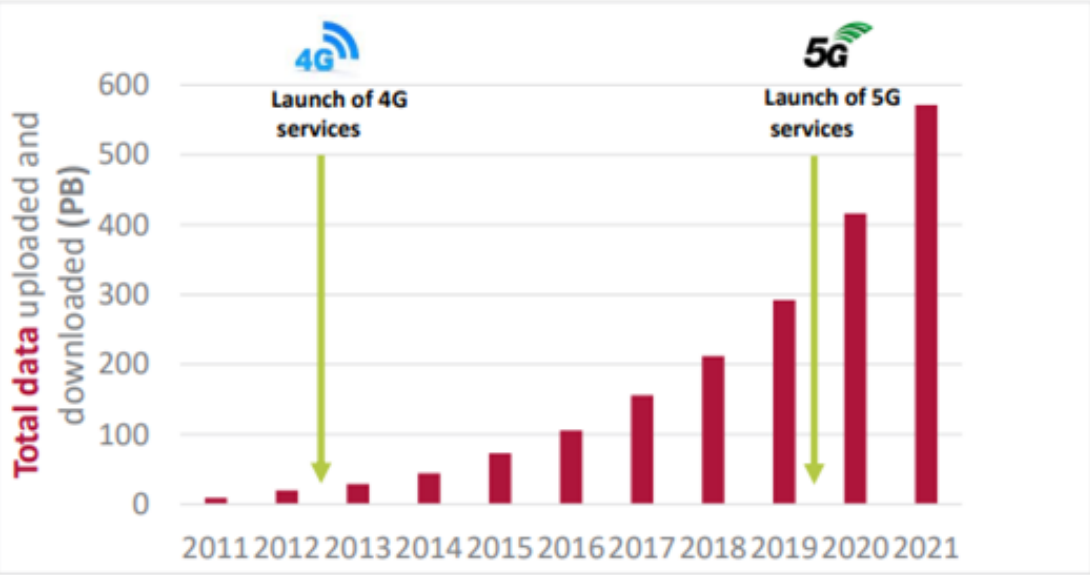


Figure 1.1: Total Data Uploaded and Downloaded between 2011 and 2021 [6]

This rise in mobile data traffic is driven by various factors, including the rise of social

media, streaming services and mobile gaming. Mobile video consumption is also an important contributor to this increase. According to a report by Ericsson, video accounts for an estimated 82% of all mobile data traffic in 2020. This is credited to the increasing quality and availability of video content, as well as the adoption of high–speed mobile networks such as 5G[7].

Spikes in Mobile Data Demand

One issue with this increase in data demand is that it is not uniform in time or space, as there exist spikes in mobile data usage. This refers to a sudden and significant increase in data traffic over a mobile network, which can be caused by a variety of factors, including events, emergencies, or even popular social media posts. However, it is inefficient to place a cell tower in close proximity, as once the event causing the spike in traffic ends, the demand rapidly falls to an ordinary amount. For example, according to a report by Ericsson, the opening ceremony at the 2016 Summer Olympic Games in Rio de Janeiro created a massive spike in mobile data usage, with data traffic reaching 5.5 terabytes, a 50% increase compared to the previous peak data usage[8].

Another example ocurred on Wednesday, October 19 2022, when Virgin Media O2 reported a record network traffic spike. This happened because that night there were five live Premier League matches, as well as users pre–downloading the new–awaited game Call Of Duty: Modern Warfare II, which was due to release the next day. At 9:20pm on that day, network traffic peaked at a level 40% higher than an average weeknight. Furthermore, the spike was 6.6% higher than the previous record, which happened on December 28 of the previous year [9].

Overall, spikes in mobile data usage can have significant implications for both mobile operators and consumers. Therefore, it is important to address these issues. One proposed solution is to use flying base stations to provide this data.

Flying Base Stations

Flying base stations, or drone base stations, are unmanned aerial vehicles (UAVs) that can be used to provide wireless connectivity to areas where it might not be efficient to use a traditional fixed base station. These UAVs can be equipped with several different types of wireless communication technologies, such as cellular, Wi–Fi or satellite.

One of the advantages of flying base stations is their ability to quickly deploy and provide temporary wireless coverage during a wide variety of situations such as emergencies or other events that led to a spike in mobile data traffic. For example, after Hurricane Maria in Puerto

Rico, Alphabet's Project London used high–altitute balloons to provide wireless connectivity to the island[10].

As was mentioned, another use of flying base stations is to handle spikes in mobile data traffic. For example, in Japan, NTT DOCOMO has been testing the use of drone base stations for monitoring traffic congestion and providing wireless coverage to outdoor events[11].

Overall, the use of flying base stations has the potential to provide wireless connectivity to areas that are difficult to reach with traditional fixed base stations, or situations in which it might be inefficient to do so.

Chapter 2

Background

2.1 Clustering Algorithms

Clustering is a fundamental problem in machine learning which aims to group similar data points together into clusters. Clustering has a wide range of applications in many fields, including image processing, bioinformatics or network analysis. In recent years, there have been significant advancements in clustering algorithms, resulting in more efficient and accurate solutions. In this section, I will survey some of the recent advances in clustering algorithms and their implications.

One of the most popular clustering algorithms is K–means clustering, which partitions the data into K clusters. K–means clustering is simple, efficient, and can handle large datasets, but has limitations such as the need for an intiial guess of the cluster centers and the sensitivity to the choice of K. Several variants of K–means clustering have been proposed to address these limitations. For example, K–means++ slightly changes how the initial cluster centroids are chosen, increasing the accuracy of the final solution compared to nominal K–means. Another example is fuzzy clustering, which allows each data point to belong nto multiple clusters with different degrees of membership [12], and spectral clustering uses the eigenvectrs of the similarity matrix to perform clustering [13].

Another widely used clustering technique is hierarchical clustering, which creates a hierarchical tree of clusters. Within this technique, two main methods have been proposed and can be read on the paper *Data Clustering: A Review*, by A.K. Jain, M.N. Murty and P.J. Flynn [14]. Firstly, agglomerative hierarchical clustering starts with each data point in its own cluster and iteratively merges the closest pair of clusters until a single cluster is formed. On the

other hand, divisive clustering does the opposite; all data points begin in one cluster, and the algorithm recursively splits them into smaller clusters. Hierarchical clustering is flexible and can produce clusters at different levels of granularity, but it is computationally expensive and may produce suboptimal results.

Density–based algorithms are effective for discovering clusters of arbitrary shapes and sizes in noisy data. In the papers *A density–based algorithm for discovering clusters in large spatial databases with noise*, by Ester et al [15] and *OPTICS: Ordering Points To Identify the Clustering Structure*, by Akerst et al [16], two such algorithms are proposed. Firstly, DBSCAN defines a neighborhood around each data point and identifies clusters based on their density, which is determined by the number of neighboring points within a given radius. The second paper proposes the OPTICS algorithm, which constructs a reachability graph that connects data points based on their relative density. The reachability between two points measures how easily one point can be reached from the other point, taking into account the local density of points along the path between them. Advantages of these algorithms include their ability to identify clusters of varying densities and do not require the number of clusters to be specified in advance. However, they can be computationally expensive and sensitive to the choice of the minimum density parameter.

Lastly, in recent years, deep learning–based algorithms have been proposed that use neural networks to perform clustering. Paper such as *An improved OPTICS clustering algorithm for discovering clusters with uneven densities*, by Tang et al [17], or *Adaptive partitioning by local density-peaks: An efficient density-based clustering algorithm for analyzing molecular dynamics trajectories* by Yang et al [18] show two such algorithms. These learn a low–dimensional representation of the data and use it to perform clustering. Deep clustering algorithms can handle complex data structures and can automatically discover useful features from the data. However, they are computationally expensive and may require a large amount of labeled data to train.

2.2 Clustering for Flying Base Stations

In recent years, a lot of work has been done to develop clustering algorithms with the objective of optimising flying base stations' placement and movement to maximise the coverage and capacity of the wireless network. In the paper *A Clustering-Driven Approach to Predict the Traffic Load of Mobile Networks for the Analysis of Base Stations Deployment*, by Mahdy et al [19], the authors propse a clustering algorithm based on K–means and K–medoids for flying base

stations placement in 5G networks. The algorithm aims to minimise the distance between base stations in the same cluster, while maximising the distance between ones in different clusters. The authors then showed that the proposed algorithm outperforms other clustering algorithms in terms of coverage and capacity of the wireless network.

Furthermore, in the paper *Aerial node placement in wireless sensor networks using Fuzzy K-means clustering* [20], the authors prtopose a fuzzy clustering algorithm which considers the residual energy of the base stations and the coverage overlap between them to assign them to clusters. In the paper it was showed that the proposed algorithm can prolong the network lifetime and improve the coverage and capacity of the wireless network.

Furthermore, a new method called geographical division clustering was proposed in the paper *A geographical division clustering algorithm for multiple flying base stations* [21]. This algorithm divides equally the plane into k regions using linear line segments that intersect at the location of the base depot. The segments are rotated by an angle θ, and for each rotation a new cluster is created and the overall route length is calculated. Numerical investigations showed that the proposed algoriothm has a maximum gain of up to 31.5% when compared to classical clustering algorithms.

All in all, there have been many contemporary developments in the field of clustering, and more specifically clustering towards flying base stations in wireless networks. Many algorithms, both new ones and improvements on existing ones, have been proposed to maximise drone efficiency. I will be contributing to the field by proposing a new flavor on the Kmeans++ algorithm to attempt to minimise the total distance traversed by all drones, and therefore minimise the energy consumed.

Chapter 3

Objectives

As was mentioned in the introduction, the final objective of the project is to design a clustering algorithm that can be applied to flying base stations to minimise the total displacement by each drone, therefore minimising energy consumed. In order to do this, a three–step plan was designed.

Step I – Designing a Theoretical Algorithm

The first step towards the completion of the final product was to design a theoretical algorithm which could then be implemented. This included reading contemporary papers and articles on clustering and path optimisation for flying base stations as well as nominal clustering techniques, and try to apply the knowledge to a new algorithm.

Step II – Implementing the Algorithm with Two Clusters and Ten Nodes

Once the theoretical framework was designed, it would be implemented in MATLAB, but only with two clusters and ten nodes. This is because it's simpler to implement a clustering algorithm with only two clusters, and less computationally expensive to have only ten nodes. This simple program would allow to evaluate the algorithm before a full program with k clusters and n nodes was done.

Step III – Implementing the Full Algorithm

After the initial, simple algorithm was completed, evaluated and validated, it could be expanded to cluster n nodes into k clusters. Then, the full algorithm can be evaluated.

Chapter 4

Desinging a Theoretical Algorithm

The first step towards the completion of the project was to design a theoretical algorithm that could then be implemented. To do this, a survey of different clustering algorithms was done. A simple, yet effective algorithm that could be expanded on is K–Means++. This is a two–phase iterative algorithm that minimises the sum of point–to–centroid distances over all clusters. There are generally three steps for this algorithm [22]:

1. **Step 1 – Find Initial Cluster Centers**: k nodes are chosen to be the initial cluster centers.

2. **Step 2 – First Pass – Batch Update**: The distance from every node to each cluster center is calculated, and nodes are assigned to the clusters to which the distance is minimised. Then, new cluster centers are calculated by taking the average x and y coordinates of all nodes. This is repeated until the cluster centers stop changing.

3. **Step 3 – Second Pass – Online Update**: For each node, change it to a different cluster and calculate the new total sum of distances from each node to its cluster center. Do this for each cluster and, if the new total distance is smaller than the previous one, change the node's cluster to the one that gives the lower distance. Then, repeat the process for all nodes.

The designed algorithm is heavily based on the K–Means++ one, but with two key differences to be explained below.

4.1 New Second Pass

While the goal of the project is to cluster nodes, the grander objective is to minimise the total distance traversed by the drones. Therefore, in the designed algorithm, the second pass will not use as a metric the total distance from each node to its cluster center. Rather, the program will calculate all optimised paths for each cluster, and use the total distance for all clusters to decide if a node will change clusters. For example, consider the following simple situation after the first pass of the K–Means++ algorithm:

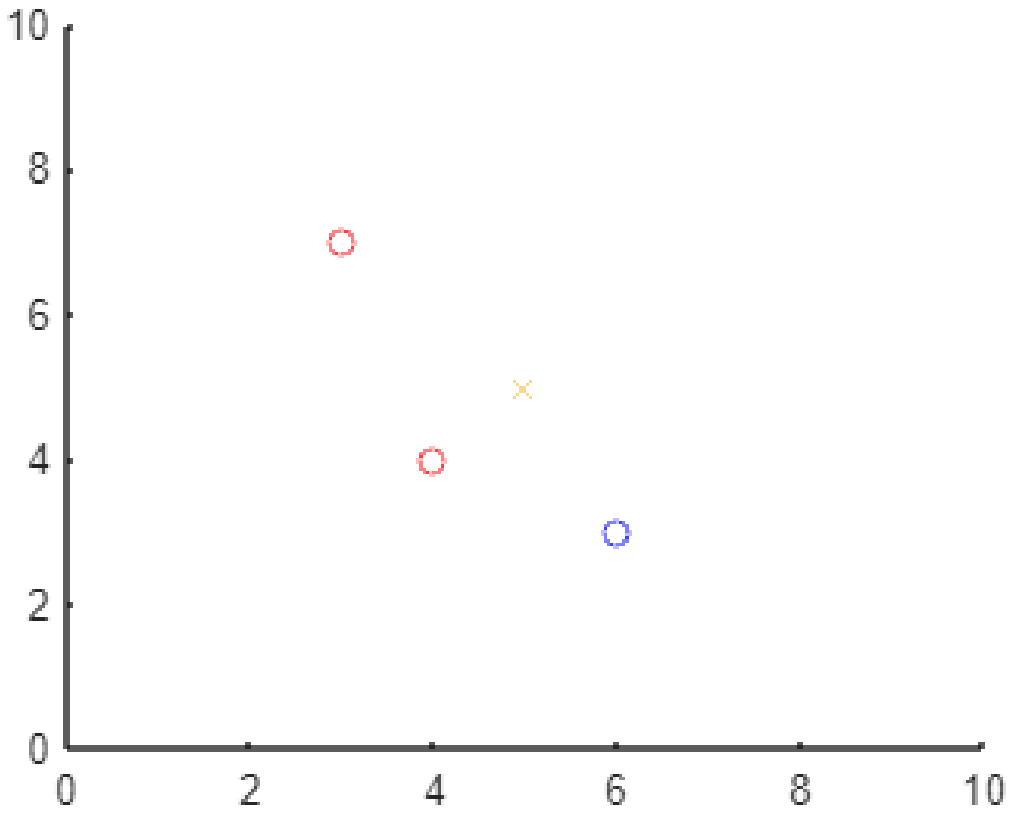


Figure 4.1: Example Clustering After 1st pass of K–Means++

Here, there are three nodes at (3,7), (6,3) and (4,4) and a depot at (5,5). Assume that after the first pass, the algorithm clustered the nodes as can be seen above. Now take for example the point (4,4) for the second pass. Firstly, the total distance of an optimised path has to be calculated. As there are one and two nodes per cluster, this can easily be done without requiring any optimisation algorithnm:

$$
\begin{aligned}
d_1 &= \left\| \begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 7 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} 4 \\ 4 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 4 \\ 4 \end{bmatrix} - \begin{bmatrix} 5 \\ 5 \end{bmatrix} \right\| = 4.48 \\
d_2 &= \left\| \begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 6 \\ 3 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 6 \\ 3 \end{bmatrix} - \begin{bmatrix} 5 \\ 5 \end{bmatrix} \right\| = 7.4 \\
d_{tot} &= 4.48 + 7.4 = 11.88
\end{aligned} \tag{4.1}
$$

Now, change the node such that it is in a different cluster. This can be seen in Fig.4.2 below:

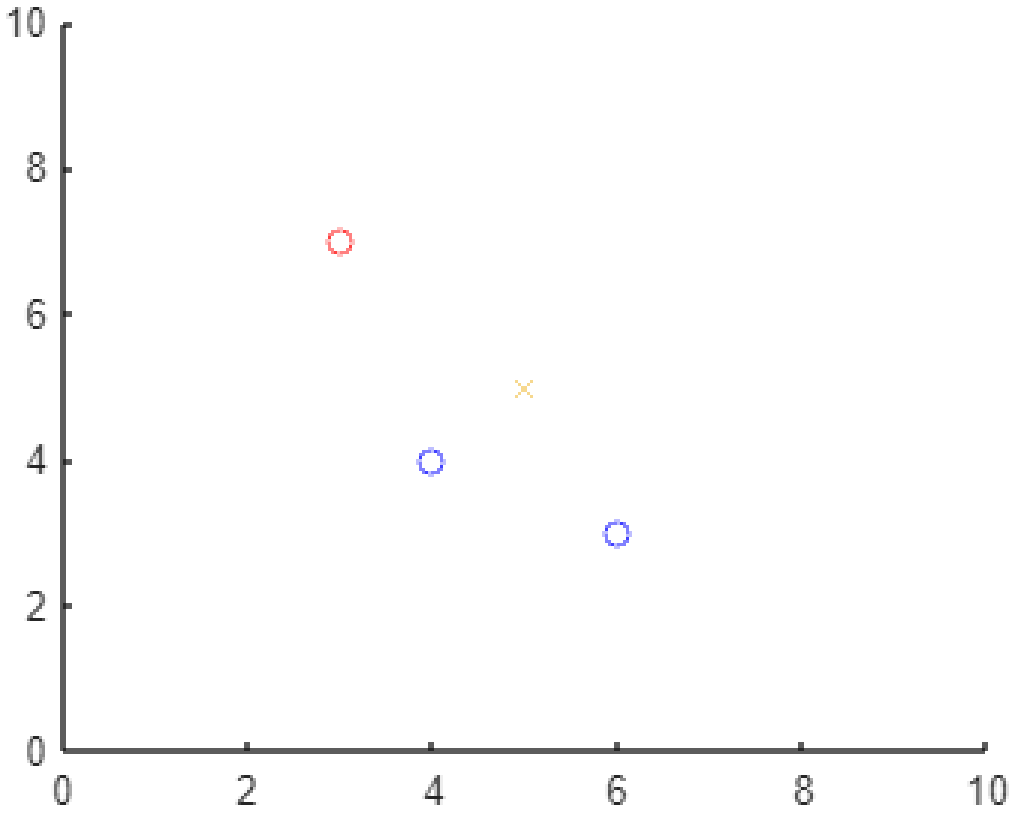


Figure 4.2: Example of the Second Pass

Now, recalculate the total path distance for each cluster:

$$
\begin{aligned}
d_1 &= \left\| \begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 7 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 3 \\ 7 \end{bmatrix} - \begin{bmatrix} 5 \\ 5 \end{bmatrix} \right\| = 5.64 \\
d_2 &= \left\| \begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 6 \\ 3 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 6 \\ 3 \end{bmatrix} - \begin{bmatrix} 4 \\ 4 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 4 \\ 4 \end{bmatrix} - \begin{bmatrix} 5 \\ 5 \end{bmatrix} \right\| = 5.89 \\
d_{tot} &= 5.64 + 5.89 = 11.53
\end{aligned}
\tag{4.2}
$$

Then, as the new total path distance for all clusters is less than it was before, the node (4,4) would be assigned to the second (blue) cluster. This would then be repeated for all other nodes. If there are more than two clusters, this has to be repeated for all clusters and then choose the cluster that gives the smallest total distance.

4.2 New Third Pass

One issue with the new second pass is that the order in which points are analysed matters. More importantly, if a node that is close to the depot is one of the first to be analysed in the second pass, it was found that the final result was inefficient, giving a final clustering that could look like Fig.4.3 below.

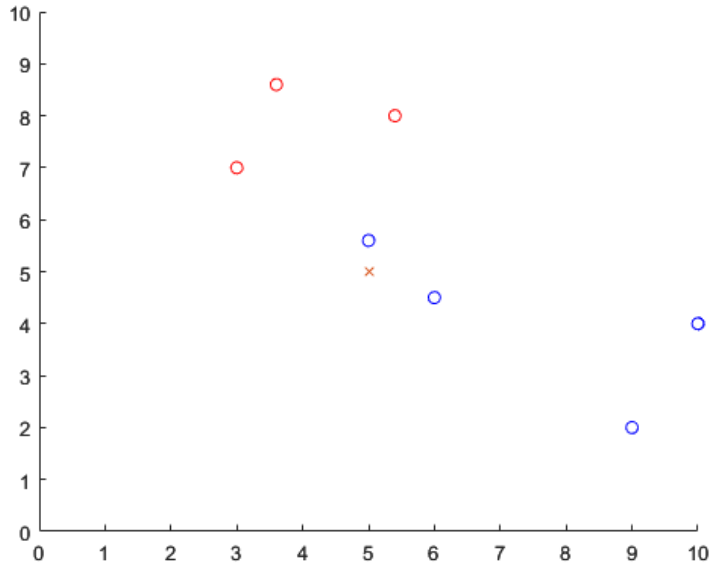


Figure 4.3: Sample Clustering After the Second Pass

As can be seen, the node at (5,5.6) should be in the first (red) cluster, but as the second pass

started with that node, it is incorrectly clustered. Therefore, a third pass is added to correct this. The third pass is identical to the first one, it calculates the mean x and y coordinates of all nodes in a cluster to find the cluster center and then it assigns nodes to clusters based on the cluster center that is closest to the node. The process is then repeated until the cluster centers stop changing. This would leave correctly clustered nodes unchanged, and correct nodes such as the one seen in Fig.4.3.

4.3 Implementation

With the algorithm already designed, it is almost ready to be implemented. However, there's still one key aspect that has to be decided, namely the path optimisation technique to be used. To do this, several algorithms were surveyed. As the goal of the project is not to give an optimised path, but rather an optimised clustering technique, a balance between simplicity and effectiveness is sought after. A summary of some of the explored algorithms can be seen below:

- **Brute–Force**: The first algorithm to consider is the simplest; calculating every possible path and finding the optimal one. While this algorithm will definetly provide the best results, it is very computationally expensive, specially as the number of nodes grows.
- **Nearest Neighbor Heuristics**: Another simple, yet computationally cheaper algorithm is Nearest Neighbor Heuristics (NNH). This algorithm simply selects the next node to visit based on its proximity with the current node, always picking the one with the shortest distance. While simple and effective, it's an inefficient algorithm that can many times lead to a suboptimal solution
- **2Opt**: Probably one of the most well–known algorithms to solve the TSP problem is the 2Opt algorithm. This is an expansion on NNH, where the path is decided using NNH, but the path is improved on by removing everytime the path crosses itself.

After the research, a final decision was arrived to regarding what path optimisation algorithm was to be used. For the first objective, that is having two clusters and ten nodes, the Brute–Force algorithm will be used. This is because having only ten nodes will not be too demanding for the computer, even if brute–force is used. Furthremore, as this algorithm will definetly give the optimal path, using it will allow to truly evaluate the designed clustering algorithm.

For the final program with k clusters and n nodes, 2Opt will be used. This is because it is a simple, yet incredible powerful algorithm that will always find a local minimum.

4.4 Attempting to Find a Global Minimum

Lastly, since the algorothm is stochastic in nature (when selecting the initial cluster centers), it is not guaranteed that it will arrive to a global minimum. To try to improve on this, the program will run the whole algorithm five times and keep track of the results. Finally, the program will return the iteration with the minimum total path distance for all clusters. While this does not guarantee a global minimum, it increases the chances of finding one.

Chapter 5

Objective 1 – Two Clusters and Brute–Force Path Optimisation

5.1 Code Implementation

5.1.1 Step 1 – Find the Initial Cluster Centers

As this program only considers two clusters, the first step of the algorithm is quite simple. To find the first cluster center, simply pick a node at random by using the `randi` function in MATLAB. To find the second center, each node's m distance to the first center, $d(x_m, c_1)$ is calculated. Then, the second center is chosen stochastically with probability,

$$P(c_2 = x_m) = \frac{d^2(x_m, c_1)}{\sum_{j=1}^{n} d^2(x_j, c_j)} \tag{5.1}$$

To implement this on MATLAB, a probability distribution can be calculated and transformed into a cummulative probability function with the `cumsum` function. Lastly, a random number between 0 and 1 can be generated, and depending on what the number is with respect to the cummulative probability function, the second center can be found. Below is a snippet of the MATLAB code finding the second cluster center:

```
s_distance = (x-c1(1)).^2-(y-c1(2)).^2;
prob = s_distance./sum(s_distance);
index_c2 = find(rand<cumsum(prob),1,'first');
c2 = [x(index_c2),y(index_c2)];
```

Where `c1` is a 1×2 vector with the x and y values of the first cluster center.

5.1.2 Step 2 – First Clustering Update

The first update is also simple to implement. To do so, a for loop with length equal to the number of nodes was added. In each iteration, the distance between a node and each cluster center is calculated. Then, the node is assigned to the cluster that has as center the closest center to the node. This can easily be done with an `if` statement as can be seen below:

```
if (d1<d2)
            k1x = [k1x x(i)];
            k1y = [k1y y(i)];
            idx = [idx 1];
        else
            k2x = [k2x x(i)];
            k2y = [k2y y(i)];
            idx = [idx 2];
        end
```

Where `d1` and `d2` are the distances between a given node and the the cluster centers, `k1x`, `k1y`, `k2x` and `k2y` are the x and y coordinates of the nodes in each cluster and `idx` is a 1×10 vector with the cluster each node belongs in.

The `for` loop is embedded in a `while` loop that calculates the cluster centers after every completion of the `for` loop. If in any iteration the cluster centers are the same as in the previous one, the `while` loop is exited and the function returns each cluster's centers and nodes. This can be seen below:

```
if (~isequal(m1,m1_old) && ~isequal(m2,m2_old))
        k1x = [];
        k2x = [];
        k1y = [];
        k2y = [];
        idx = [];
    else
        cond = 1;
    end
```

Where `cond` is a predifined variable that equals to 0 and is just used as an exit condition for the loop. `m1`, `m2`, `m1_old` and `m2_old` are the current and previous iteration cluster means (or centers) respectively.

5.1.3 Step 3 – Second Clustering Update

To implement the second update, a `for` loop with length equal to the number of nodes is created. Then, the first step is to calculate the total path distance as such,

```
[path_1,dist_1] = PathOpt(k1x_c,k1y_c,x_o,y_o);
[path_2,dist_2] = PathOpt(k2x_c,k2y_c,x_o,y_o);
```

Where `PathOpt(kx,ky,x_o,y_o)` is a Brute–Force path optimisation algorithm that will be explained later. Once the total distance is calculated, the specific node that is analysed in the iteration of the `for` loop is changed clusters. This can be done by simply changing `idx` and recalculating the clusters from that:

```
if idx(i) == 1
    idx(i) = 2;
else
    idx(i) = 1;
end

k1x_c = x(find(idx==1));
k2x_c = x(find(idx==2));
k1y_c = y(find(idx==1));
k2y_c = y(find(idx==2));
```

Lastly, the total path distance is calculated. Then, if the new distance is smaller than the previous one, the node is maintained in the new cluster. Otherwise, it's returned to its orignial cluster:

```
if (tot_dist_new > tot_dist)
    if idx(i) == 1
        idx(i) = 2;
    else
        idx(i) = 1;
```

```
        end
    end
```

5.2 Third Pass

The third pass is identical to the first pass, with one difference. As the initial cluster centers are not given, first the mean x and y coordinates of every cluster's node is calculated to find the mean and use that as an initial cluster center.

5.3 Path Optimisation

To implement path optimisation, every possible path has to be calculated. This can be done by using the MATLAB function `perms`, finding every possible permutation of the matrix with the data points. Then, every individual path is analysed and the total distance is calculated. If the total distance is lower than the previous best distance, then the program keeps the current path.

The biggest complication in this function is to keep track of what x–value goes with what y–value. As the `perms` function permutates the whole matrix and not an individual column, a dictionary was used to keep track of the pair of coordinates. Then, only the first column of the matrix of data points was permutated, and the y–values were obtained from the dictionary:

```
combi = perms(matrix(:,1));
d = dictionary(x',y');
sum = 1000000;


for i = 1:factorial(length(x))
    x_path = combi(i,:);
    path_1 = [];
    for j = 1:length(x)
        path_1 = [path_1; x_path(j) d(x_path(j))];
end
```

Where `matrix` is a $n \times 2$ matrix with the x and y coordinates of the nodes. `sum` is an arbitrarily large number to use as the initial total distance. As we require a Hamiltonian cycle

starting on the depot, the origin is not included in this matrix, but is added after `path_1` is calculated.

To find the total path distance, the partial sums per segment can be obtained and summed using a `for` loop. Then, the program can check if the distance is lower than the previous one with a simple `if` statement.

5.4 Final Program

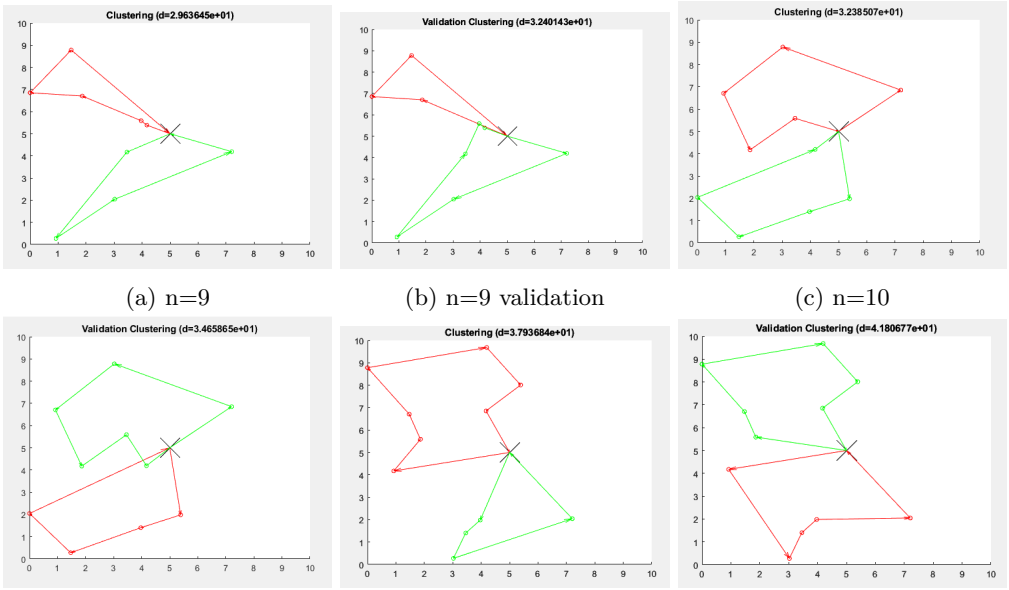
With the algorithm coded, the results can be plotted. Different clusters can be plotted with different colors, and the path within a cluster using the `quiver` function. Below is a sample clustering with 10 nodes:

Chapter 6

Objective 1 – Results

6.1 Final Clustering

When run, the program outputs `idx`, a vector containing the cluster each node is assigned to. This can be plotted along with the brute path optimisation. Fig. 6.1 shows the final clustering and path, along with validation, for $n = 9, 10, 11$.



(a) n=9 (b) n=9 validation (c) n=10

(d) n=10 validation (e) n=11 (f) n=11 validation

Figure 6.1: Final Clustering with 9, 10 and 11 Nodes

Before numerically analysing the results, it can already be seen that the progran's clusters show better results than MATLAB's, as different clusters never intersect. However, to prove this, numerical analysis has to be done.

6.2 Computing Energy Consumed

In ordert to compute the consumed energy for the UAVs, some assumptions were made. Firstly, it was assummed that all velocities for the UAVs are identical. Furthermore, any aspect regarding accelerateion and deceleration was not considered. Lastly, only the straight–and–level flight (SLF) energy consumption was calculated. This is done for simplicity, as, even though the exact energy consumption will not be found, the obtained value can still be used to compare different algorithms.

With these assumptions, the SLF version of the energy consumption model in *Energy-efficient UAV communication with trajectory optimization*, by Zeng Y and Zhang R [23] is used, where the total energy consumptuion is Joules can be expressed as seen below,

$$E(V) = T\left(c_1 V^3 + \frac{c_2}{V}\right) \tag{6.1}$$

Where T is the total flying time in seconds, V is the velocity of the UAVs and c_1 and c_2 are constant values that are to be found.

In addition, it was also assumed that the flying base stastions spend an equal amount of time on each node, and therefore the hovering time is constant. Based on this, the total travel time is the total tour length L divided by the velocity V. Therefore, the total travel time can be written as,

$$T = \frac{L}{V} + \sum_{i=1}^{N} T_{serv}[i] \tag{6.2}$$

Using the UAV model Penguin C UAS and following the reasoning in the paper *A geographical division clustering algorithm for multiple flying base stations* [21], the following parameters will be usued:

$$\begin{aligned} V &= 14.90\frac{m}{s} \\ [c_1, c_2] &= [0.015, 2226.5] \\ T_{serv} &= 10s \end{aligned} \tag{6.3}$$

6.3 Results

Several results were obtained from the program. Firstly, the program was run 50 times with the same parameters, with the total path distance being calculated each time. This was then compared with MATLAB's built in `kmeans` function. The full data can be seen in the appendix.

Fig.6.2 shows the total path distance for 8,9,10 and 11 nodes for the designed program and MATLAB's `kmeans`:

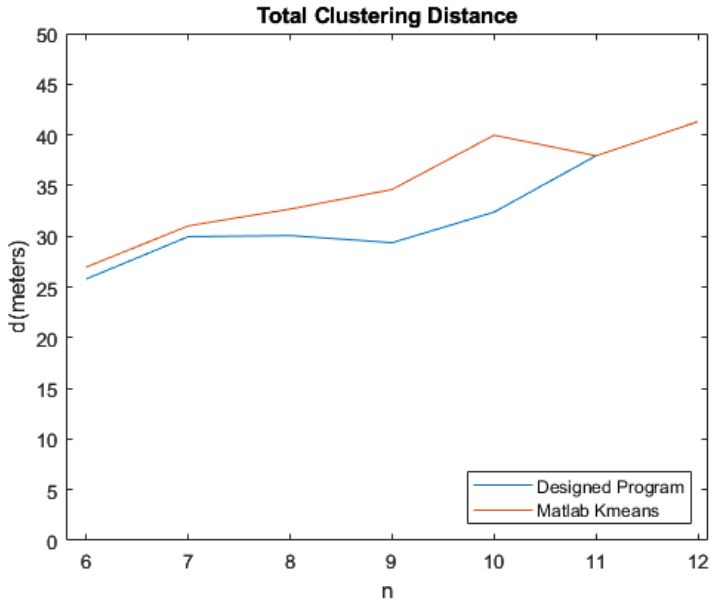


Figure 6.2: Sample Clustering After the Second Pass

As can be seen, the designed program consistently gives better results in $6 \leq n \leq 10$, and for $n = 10, 11$ the total distance is identical. Furthermore, the total energy consumed by all drones can be obtained using equations 7.1 and 7.2 and the parameters in 7.3. As the energy values are quite large and the difference between the results from the designed program and MATLAB's built–in function are smaller in comparison, the results are easier to visualize by plotting the difference in given energies. This can be seen in Fig.6.3 below:

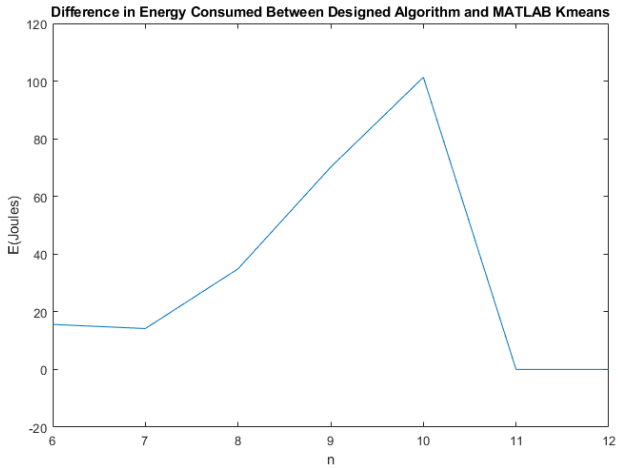


Figure 6.3: Sample Clustering After the Second Pass

As can be seen, the designed program outputs paths that consume less energy when com-

pared to MATLAB's `kmeans`, with the difference ranging between 0 for $n = 11, 12$ and reaching to $100J$. A more detailed analysis of the results can be seen in the Results and Discussion section of the report.

6.4 Time Complexity

While the algorithm can give good results, it is very computationally expensive. To find the time complexity of the program, it was run several times with a different number of nodes as a parameter, and the time taken to run was recorded. Fig.6.4 shows these results.

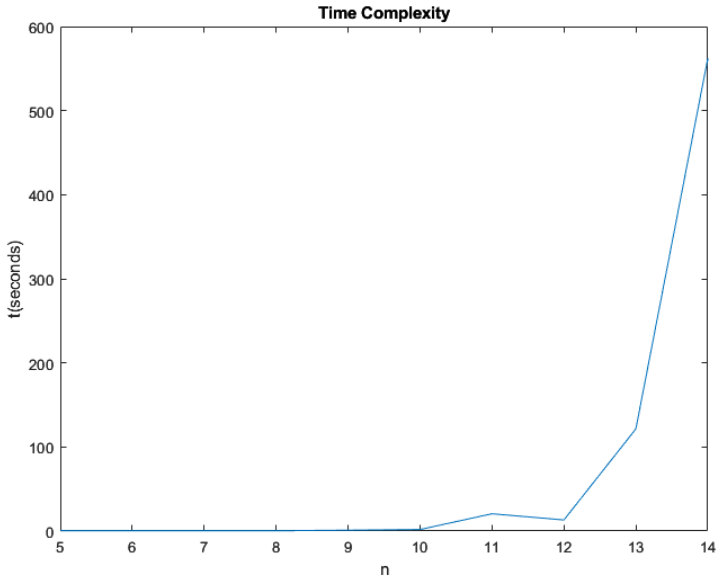


Figure 6.4: Program Runtime for Different ns

As can be seen, the time taken to run the program quickly spirals out of control. This is due to the chosen path optimisation algorithm, which runs $n!$ times. Therefore, the time complexity of the program is of $\mathcal{O}(n!)$, which is very inefficient.

Chapter 7

Objective 2 – k Clusters and 2Opt Path Optimisation

7.1 Code Implementation

Step 1 – Find the Initial Cluster Centers

As this program now considers more than two clusters, the first step is slightly changed. Finding the first two centers is identical as before, but to find any subsequent centers, the distance from each node to each center has to be calculated. Then, for each node, only the smaller of the distances to each center is considered, and the new center is chosen with the same probability as before. This is simple to implement, it just requires a new `for` loop from 3 to k that finds the minimum distance from each point to a center. The code snippet for this can be seen below:

```
for j = 1:length(x)
        dist_2_center = zeros(1,length(c));
        for m = 1:length(c)
            dist_2_center(m) = (x(j)-c(1,m))^2+(y(j)-c(2,m))^2;
        end
        final_s_dist(j) = min(dist_2_center);
end
```

Where `c` is the current number of cluster centers the program already found. After finding all of the distances, the same method for finding the next cluster center as before is used. This is then repeated until k centers have been obtained.

Step 2 – First Clustering Update

This step of the algorithm is identical to the simpler program with only two clusters. This time, `c` is a $1 \times k$ vector representing the centers for each cluster.

Step 3 – Second Clustering Update

Implementing the second step is more complicated with k clusters. This is mainly due to the fact that one cannot create the vectors $k1x$, $k1y$, $k2x$ and kyx, as the number of clusters is unknown. The solution to this is to create two variables kx and ky that will contain every point for each cluster. Since the clusters are not necessarily the same size, using a matrix will give an error. Therefore, kx and ky are cell type variables.

With that small difference, the rest of the function is very similar. In fact, it's identical (except in the choice of path optimisation algorithm) until the end, where the program decides what cluster the node will be in. To do this, every path distance is computed and stored in a vector, and using the built–in function `min`, the program can get the index of the smallest of these distances. As the vector will be ordered, that is to say, entry k corresponds to the total path length of cluster k, the index of the minimum value in the vector will also be the cluster that the node should be in. This is showed below.

```
[~,cluster] = min(new_sum_of_path);
idx(i) = cluster;
```

Cost

One change that was made to this program is the option to change the cost of the total distances in the second step. That is to say, either the linear distances, the squared distances or the cubed distances could be used as a metric to decide if nodes change clusters. This will be further discussed in the results section, where an optimal cost will be obtained.

Step 4 – Third Clustering Update

As the third pass in the algorithm is the same as the first pass, this too is identical to its counterpart in the simpler algorithm with only two clusters. Once again, the only difference between this pass and the first one is that the cluster centers have to be calculated before starting.

Path Optimisation

To implement 2Opt, the first step is to design a Nearest Neighbor Heuristics program. To do this, the function calculates the distance from the starting nodes to all other nodes, and simply picks the node with the shortest distance. Then, this is repeated until an initial path is found.

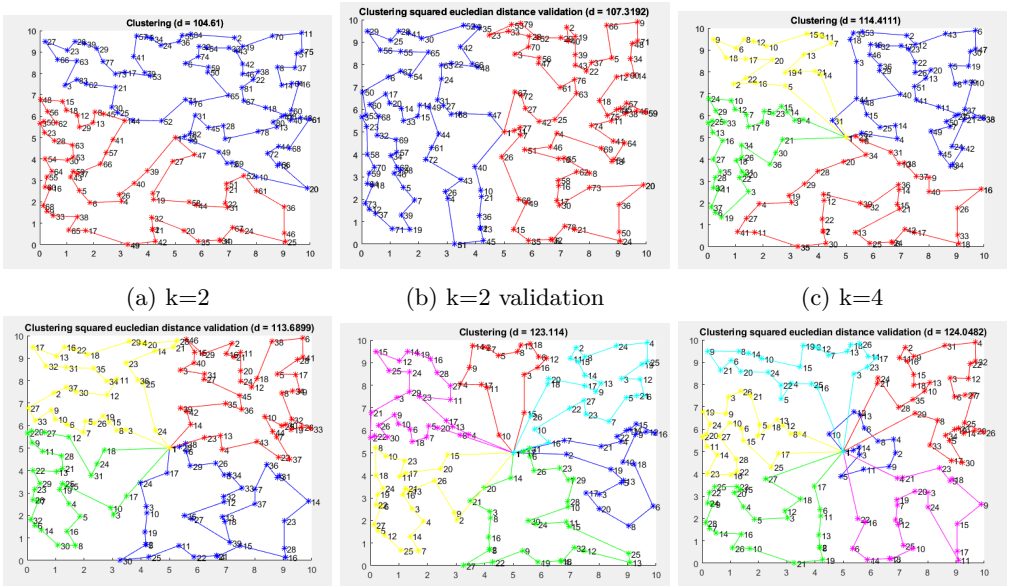
Then, the program iterates through all nodes with two nested for loops, swapping the nodes. If the new total path distance is smaller, then the path is left with the swap. On the other hand, if it's bigger, the swap is reverted.

Chapter 8

Objective 1 – Results

8.1 Final Clustering

Once again, the program can be run with different n and k to visualize the final clustering. Fig. 8.1 shows the final clustering and validation for $n = 150$ and $k = 2, 4, 6$.



(a) k=2 (b) k=2 validation (c) k=4

(d) k=4 validation (e) k=6 (f) k=6 validation

Figure 8.1: Final Clustering with 150 nodes and 2, 4 and 6 Clusters

As can be seen in the figures, the program's clustering seems to be more efficient than the validation, as there are less intersections. This can clearly be seen when there's six clusters. In the validation clustering, the dark blue cluster intersects the cyan, red and purple clusters, while there's less crossover in figure e). However, take for example the green cluster. It seems

that the first three nodes on the path should be in the dark blue cluster. However, they are incorrectly (or so it seems) placed in the green one. This could be caused by a variety of reasons to be discussed in the evaluation. With that being said, numerical analysis is also required to fully evaluate the clustering algorithm.

8.2 Computing Energy Consumed

To compute the consumed energy for the UAVs, the same equations as in Chapter 7 will be used with the same parameters.

8.3 Results

Once again, several results were obtained for this program. Firstly, the program was run 100 times with $n = 100$ and varying k and the cost. Then, the total path distance for all clusters was obtained. Furthermore, the clustering was also done using MATLAB's `kmeans` funtion with all four parameters for the metric to be used[1]:

- sqeuclidewan: Squared Eucledian Distance
- cityblock: Sum of absolute differences
- cosine: One minus the coside n the included angle between points
- correlation: One minus the sample correlation between points

Table 8.1 shows the average total distance for different number of clusters compared to MATLAB's `kmeans` function with only 'sqeuclidean' parameter. This is because this was the parameter that gave the best results.

[1] In theory there are five parameters, but 'hamming' is only suitable for binary data

k	Cost	Average Program's Distance (m)	Average Validation Distance (m)
2	Linear	85.31589937	84.68131802
2	Quadratic	85.34694402	84.83545254
2	Cubic	85.20006544	84.64385021
4	Linear	91.45593275	90.14812352
4	Quadratic	91.74610619	90.02385067
4	Cubic	91.5883991	90.3675898
6	Linear	102.0441633	103.6676892
6	Quadratic	101.2165296	103.9276587
6	Cubic	101.5390035	103.5219651

Table 8.1: Average Program Distance Validated With MATLAB's KMEANS Squared Euclidean

Several observations can be drawn from the table above. Firstly, it can be seen that the choice of cost doesn't substantially affect the final result. For each k, the three total distances are very similar and the cost that leads to the best distance varies for all three ks. Furthermore, it can also be seen that the results are not as promising as with the previous program. In fact, for smaller number of clusters, MATLAB's function provides better results, while for $k = 6$, the designed program is more efficient. This can be due to several reasons which will be discussed in the evaluation. For easier visualization, the data can also be plotted and seen in Fig.8.2.

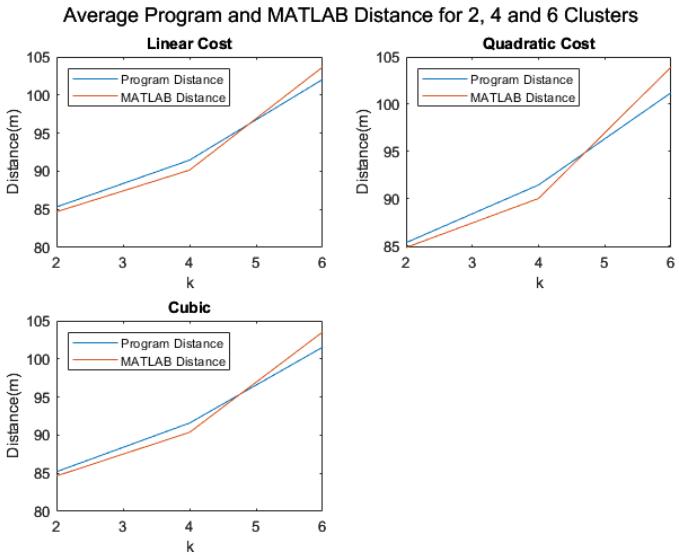


Figure 8.2: Total Path Distance For Different ks and Costs With Validation

The figure above shows what was discussed earlier. Firstly, the choice of cost is not of upmost importance, as the results are similar regardless of the cost used. Furthermore, the total distances are very similar, with MATLAB's `kmeans` taking the edge for $k = 2, 4$ and the designed program showing better results for $k = 6$.

Once again, the energy consumed can be derived using the equations presented earlier.

k	Cost	Average Total Energy Consumed (J)	Validation Average Total Energy Consumed (J)
2	Linear	200188.4984	200180.021
2	Quadratic	200188.9131	200182.0801
2	Cubic	200186.951	200179.5205
4	Linear	200270.523	200253.052
4	Quadratic	200274.3994	200251.3918
4	Cubic	200272.2926	200255.9838
6	Linear	200411.9709	200433.6595
6	Quadratic	200400.9145	200437.1325
6	Cubic	200405.2225	200431.7128

Table 8.2: Average Program Total Enegy Consumed Validated With MATLAB's KMEANS Squared Euclidean

The above table shows similar results as Table 8.1. The total energy for all drones is similar for the designed program and MATLAB's `kmeans`, with MATLAB giving better results for $k = 2, 4$ and the program giving better results for $k = 6$.

8.4 Time Complexity

Once again, the time complexity was evaluated for the program. To do this, a Monte Carlo simulation was setup, with the program running for $5 \leq n \leq 150$. Furthermore, the program was run 5 times for each n, and the average time to run was obtained. Fig.8.3 shows the total running time for each n.

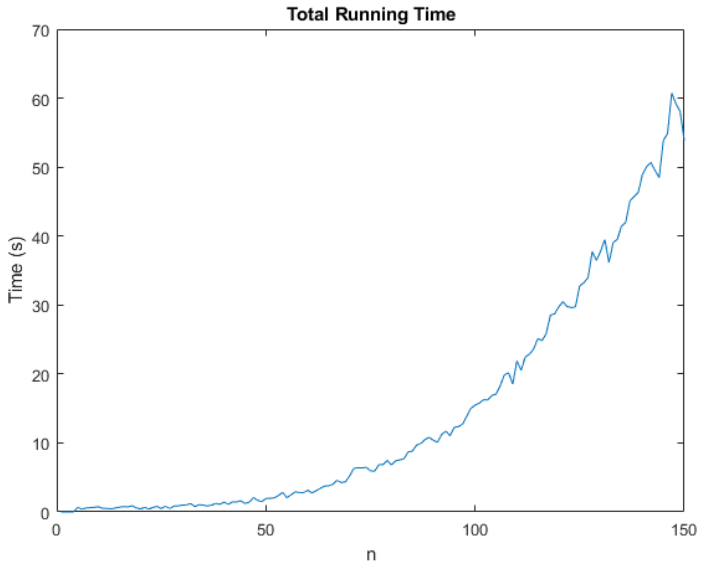


Figure 8.3: Total Runtime

As is expected, the runtime increases with the number of nodes, but it doesn't do so in a factorial way as earlier, but rather quadratically, with a time complexity of $\mathcal{O}(n^2)$

Chapter 9

Discussion and Evaluation

9.1 Program 1

As was shown above, the theoretical framework using a Brute–Force path optimisation algorithm showed very promising results. Both the total path distance and total energy consumed provided better or equal results to MATLAB's `kmeans` function. However, the program cannot be run with $n > 14$, as the computer quickly runs out of RAM memory. It would be interesting to analyse the program with a stronger computer to analyse how it behaves with a bigger number of nodes. Furthermore, the same framework using the Brute–Force path optimisation algorithm can be applied for $k > 2$ to analyse how the algorithm behaves when creating multiple clusters.

9.2 Program 2

The full program with $n > 15$ and $k > 2$ also shows promising results, albeit worse than the first program. This could be due to a variety of reasons to be discussed below.

Firstly, while 2Opt finds a local minimum in a very efficient time, the path optimisation is not optimal. While it can be argued that the difference between a local and global optimum is minimal, as this is done in every iteration of the second pass in the clustering, these small errors can add up and lead to noticeable ones. Therefore, different path optimisation algorithms can be explored to decide which one gives the best results for the objective at hand.

Furthermore, as was mentioned earlier, the order in which nodes are analysed in the second pass of the algorithm can lead to issues in the final results. If the nodes closer to the depot

are analysed first, they can be placed in the wrong cluster and never corrected. There are two possible solutions to attempt to solve this issue. Firstly, the second pass can be run twice with the objective that the second pass corrects the wrongly assigned nodes in the first pass. Another option is to first sort the nodes according to their euclidean distance to the home depot. Then, the nodes are analysed in order of decreasing distanc, starting with the nodes that are further away from the depot and ending with the ones that are closer. This would also solve the issue of having nodes close to the center incorrectly placed.

Chapter 10

Professionalism and Responsibility

10.1 Sustainability Goals

The Sustainability Development Goals are an urgent call of action by all countries, both developed and developing. They consist of a list of 17 goals with aiming to transform our world. These goals have been recognized by all United Member States in 2015 and are a priority for lots of businesses around the world when planning for future projects [24]. While this project was developed individually and in a private manner, it still directly affects three of these goals:

Goal 11 – Sustainable Cities and Communities: By using UAVs as flying base stations, the use of geographical antennas is void, leading to more sustainable cities and communities.

Goal 12 – Responsible Consumption and Production: Related to the previous goal, optimising the path of each drone results in lowering the total energy consumption.

Goal 13 – Climate Action: Lastly, reducing the energy consumption is a step towards reducing climate change, as energy waste is minimised.

Moreover, the project indirectly affects several of the sustainability goals, including:

Goal 3 – Good Health and Well–Being: While few studies have been conducted in the area, it is thought that having electromagnetic waves constantly passing through one's body is detrimental towards the health of the individual. By using UAVs as base stations, the waves will not be constantly present, but rather only when there is a spike in mobile data.

Goal 4 – Quality Education: Drones can be used to provide data to educational events such as guest lectures or university fairs.

10.1.1 Ethical Concerns

While there are no direct ethical concerns related to the project, the use of drones have created several ethical conundrums. Because of this, many countries, specially in the Middle East or North Africa have completely banned the use of drones, or greatly restricted them. Other countries with more lax drone laws, such as the UK, still require a license to operate a UAV, and there are many restrictions related to where and when one can fly them. Furthermore, there are several conditions that the drone needs to meet in order to be considered valid to fly in a public air space. Because of this, the use of drones as flying base stations has not been adopted globally and still is only used in the event of emergencies or specially strong spikes in mobile data demand. However, with the advancement of drone technologies and new or corrected legislation, the future possibilities of this field are limitless.

Chapter 11

Conclusion and Future Work

11.1 Future Improvements

Change the Path Optimisation Method

As was mentioned above, one possible cause for the diminishing quality of results of the full program is the path optimisation algorithm. Therefore, a possible future improvement for the proposed algorithm is to change the path optimisation method to an algorithm that better works with the constraints of the program.

Improvement on the Second Pass

Furthermore, the order in which nodes are evaluated for the second pass of the algorithm can also lead to inefficiencies. If the nodes closer to the depot are analysed first, this can lead to the nodes being placed in wrong clusters. Therefore, another improvement could be to change the order in which these nodes are analysed such that the algorithm begins with nodes further away from the depot, and end with the nodes that are closer.

Machine Learning

During the research phase of the project, many clustering algorithms that used machine learning were read on. This could be implemented in the program with the use of a transformer [25]. This would be an elegant solution that would also correct the issues with the path optimisation. While it's computationally expensive to train the model, once trained it could potentially provide better, cheaper results.

11.2 Conclusion

All in all, the project was a success. An theoretical framework was proposed and proven to improve on the nomial Kmeans++ algorithm. While the final program didn't provide the expected results, the framework can be improved upon such that eventually it provides the desired results. It has been proven that the use of path optimisation algorithms within clustering algorithms can improve the final clustering such that the total path distance for all clusters is lowered.

From reading about clustering algorithms all the way to designing my own program, this project has given me a lot of knowledge and tools that I will most certainly use in the future. This experience has been a please and I cannot wait to see the further developments in this field of engineering.

References

[1] B. Li, Z. Fei, and Y. Zhang, "Uav communications for 5g and beyond: Recent advances and future trends," *IEEE Internet of Things*, 2019.

[2] "2–opt." Available at https://en.wikipedia.org/wiki/2-opt. Accessed: 2022-10-21.

[3] "Vehicle routing problem." Available at https://developers.google.com/optimization/routing/vrp#:~:text=In\%20the\%20Vehicle\%20Routing\%20Problem,with\%20the\%20least\%20total\%20distance. Accessed: 2022-10-16.

[4] "Mobile broadband." Available at https://en.wikipedia.org/wiki/Mobile_broadband. Accessed: 2022-10-12.

[5] "Cisco annual internet report (2018-2023)." Available ar https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, 2020.

[6] Ofcom, "Mobile networks and spectrum, meeting future demand for mobile data." Available at https://www.ofcom.org.uk/__data/assets/pdf_file/0017/232082/mobile-spectrum-demand-discussion-paper.pdf, 2022.

[7] Ericsson, "Ericsson mobility report, november 2022." Available at https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/november-2022, 2022.

[8] Ericsson, "Enhancing the event experience." Available at https://www.ericsson.com/en/reports-and-papers/mobility-report/articles/enhancing-event-experience.

[9] M. Landi, "Virgin media o2 reports record network traffic spike," *Evening Standard*, 2022.

[10] "Google to use balloons for puerto rico phone service." Available at https://www.aljazeera.com/news/2017/10/7/

google-to-use-balloons-forpuerto-rico-phone-service#:~:text=Alphabet\
%20Inc\%20is\%20sending\%20high,island\%20devastated\%20by\%20Hurricane\
%20Maria.&text=Alphabet\%20Inc\%2C\%20the\%20company\%20that,by\
%20Hurricane\%20Maria\%20last\%20month., 2017.

[11] press, "Zephyr high altitude platform station (haps) achieves connectivity in trial conducted by airbus and ntt docomo." Available at https://www.suasnews.com/2021/11/zephyr-high-altitude-platform-station-haps-achieves-connectivity-in-trial-conducted-by-airbus 2021.

[12] "Fuzzy clustering."

[13] D. Huang, C.-D. Wang, J.-S. Wu, J.-H. Lai, and C.-K. Kwoh, "Zephyr high altitude platform station (haps) achieves connectivity in trial conducted by airbus and ntt docomo," 2020.

[14] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," 1999.

[15] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise."

[16] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," 1999.

[17] C. Tang, H. Wang, Z. Wang, X. Zeng, H. Yan, and Y. Xiao, "An improved optics clustering algorithm for discovering clusters with uneven densities," 2021.

[18] S. Liu, L. Zhu, F. K. Sheong, W. Wang, and X. Huang, "Adaptive partitioning by local density-peaks: An efficient density-based clustering algorithm for analyzing molecular dynamics trajectories," 2017.

[19] B. Mahdy, H. Abbas, H. S. Hassanein, A. Noureldin, and H. Abou-zeid, "A clustering-driven approach to predict the traffic load of mobile networks for the analysis of base stations deployment," 2020.

[20] A. Talgini, V. Shakarami, F. Sheikholeslam, and A. Chatraei, "Aerial node placement in wireless sensor networks using fuzzy k-means clustering," 2014.

[21] J. Lee, X. Lyu, and V. Friderikos, "A geographical division clustering algorithm for multiple flying base stations," 2020.

[22] "kmeans." Available at https://uk.mathworks.com/help/stats/kmeans.html.

[23] Y. Zeng and R. Zhang, "Energy-efficient uav communication with trajectory optimization."

[24] "The 17 goals." Available at https://sdgs.un.org/goals.

[25] "Transformer (machine learning model)." Available at https://en.wikipedia.org/wiki/Transformer_(machine_learning_model).

Appendix A

Source Code

A.1 Program 1

```
function [our_distance,dist_v,time] = Final(n)
format long
close all

n=10;

idx_iter = [];
our_distance = [];
path_1_iter = {};
path_2_iter = {};

x_o = 5;
y_o = 5;

%Step 1 - Creating Values
rng('default')
rng(1)
[x,y] = CreateVals(n,10);

for p = 1:5
```

```
21      %Step 2 - Find Cluster Centers
22      rng shuffle
23      [c1,c2] = CreateCenters(x,y);
24
25      %Step 3 - Clustering-First Pass
26      [k1x,k1y,k2x,k2y,m1,m2,idx] = Clustering1(x,y,c1,c2);
27
28      %Step 4 - Clustering-Second Pass
29      idx = Clustering2(idx,x_o,y_o,x,y);
30
31      %Step 5 - Clustering-Third Pass
32      idx = Clustering3(idx,x,y);
33
34      %Getting results to take the best iteration
35      k1x = x(find(idx==1));
36      k2x = x(find(idx==2));
37      k1y = y(find(idx==1));
38      k2y = y(find(idx==2));
39      [m_path_1,dist_1] = PathOpt(k1x,k1y,x_o,y_o);
40      [m_path_2,dist_2] = PathOpt(k2x,k2y,x_o,y_o);
41      dist = dist_1+dist_2;
42      our_distance = [our_distance dist];
43      idx_iter = [idx_iter idx];
44      path_1_iter{p} = m_path_1;
45      path_2_iter{p} = m_path_2;
46  end
47      %Finding best iteration
48      [our_distance,best_pass] = min(our_distance);
49
50      final_idx = idx_iter((best_pass*n)-(n-1):(best_pass*n));
51      m_path_1 = path_1_iter{best_pass};
52      m_path_2 = path_2_iter{best_pass};
53
```

```
54      k1x = x(find(final_idx==1));
55      k2x = x(find(final_idx==2));
56      k1y = y(find(final_idx==1));
57      k2y = y(find(final_idx==2));
58
59
60
61      %Validation
62      idx_v = kmeans([x' y'],2);
63      k1x_v = x(find(idx_v==1));
64      k2x_v = x(find(idx_v==2));
65      k1y_v = y(find(idx_v==1));
66      k2y_v = y(find(idx_v==2));
67
68      [path_v_1,dist_v_1] = PathOpt(k1x_v,k1y_v,x_o,y_o);
69      [path_v_2,dist_v_2] = PathOpt(k2x_v,k2y_v,x_o,y_o);
70      dist_v = dist_v_1 + dist_v_2;
71
72      %Plotting the results
73      figure()
74      plot(k1x,k1y,k2x,k2y,x_o,y_o,our_distance,'Clustering (d=%d)',
            m_path_1,m_path_2);
75      figure()
76      plot(k1x_v,k1y_v,k2x_v,k2y_v,x_o,y_o,dist_v,'Validation
            Clustering (d=%d)',path_v_1,path_v_2);
77
78
79  %Program Functions
80  %Function 1 - Creating Values
81  function [x,y] = CreateVals(n_values,range)
82      x = rand(1,n_values)*range;
83      y = rand(1,n_values)*range;
84  end
```

```
85
86 %Function 2 - Find cluster centers
87 function [c1,c2] = CreateCenters(x,y)
88 %Select a random point as the first cluster
89     init_c = randi(length(x));
90     c1 = [x(init_c),y(init_c)];
91
92     %For each point, find it's squared distance to c1 and create a
93     %probability distribution with those numbers. Find second center
94     %randomly with that given distribution
95     s_distance = (x-c1(1)).^2-(y-c1(2)).^2;
96     prob = s_distance./sum(s_distance);
97     index_c2 = find(rand<cumsum(prob),1,'first');
98     c2 = [x(index_c2),y(index_c2)];
99 end
100
101 %Function 3 - Clustering (FIRST UPDATE)
102 function [k1x,k1y,k2x,k2y,m1,m2,idx] = Clustering1(x,y,c1,c2)
103     cond = 0;
104     k1x = []; %x-vals in cluster 1
105     k2x = []; %x-vals in cluster 2
106     k1y = []; %y-vals in cluster 1
107     k2y = []; %y-vals in cluster 2
108     idx = []; %What cluster each entry is in
109
110     m1 = c1;
111     m2 = c2;
112
113     while (cond == 0)
114         m1_old = m1;
115         m2_old = m2;
116
117         %For every point, find the distance to each cluster center
```

```
            and
        %assign it to the cluster with min distance
        for i = 1:length(x)
            point = [x(i) y(i)];
            d1 = Distance2Points(point,m1_old);
            d2 = Distance2Points(point,m2_old);

             if (d1<d2)
                k1x = [k1x x(i)];
                k1y = [k1y y(i)];
                idx = [idx 1];
            else
                k2x = [k2x x(i)];
                k2y = [k2y y(i)];
                idx = [idx 2];
            end
        end
        %Find new cluster centers
        m1(1) = mean(k1x);
        m1(2) = mean(k1y);
        m2(1) = mean(k2x);
        m2(2) = mean(k2y);
        %Check if cluster centers keep changing
        if (~isequal(m1,m1_old) && ~isequal(m2,m2_old))
            k1x = [];
            k2x = [];
            k1y = [];
            k2y = [];
            idx = [];
        else
            cond = 1;
        end
    end
```

```
150 end
151
152 %Function 4 - Clustering (SECOND PASS)
153 function [idx] = Clustering2(idx,x_o,y_o,x,y)
154     for i = 1:length(x)
155         k1x_c = x(find(idx==1));
156         k2x_c = x(find(idx==2));
157         k1y_c = y(find(idx==1));
158         k2y_c = y(find(idx==2));
159
160         [path_1,dist_1] = PathOpt(k1x_c,k1y_c,x_o,y_o);
161         [path_2,dist_2] = PathOpt(k2x_c,k2y_c,x_o,y_o);
162
163         tot_dist = dist_1 + dist_2;
164
165         if idx(i) == 1
166             idx(i) = 2;
167         else
168             idx(i) = 1;
169         end
170
171         k1x_c = x(find(idx==1));
172         k2x_c = x(find(idx==2));
173         k1y_c = y(find(idx==1));
174         k2y_c = y(find(idx==2));
175
176         [path_1_new,dist_1_new] = PathOpt(k1x_c,k1y_c,x_o,y_o);
177         [path_2_new,dist_2_new] = PathOpt(k2x_c,k2y_c,x_o,y_o);
178
179         tot_dist_new = dist_1_new + dist_2_new;
180
181         if (tot_dist_new > tot_dist)
182             if idx(i) == 1
```

```
183                     idx(i) = 2;
184                 else
185                     idx(i) = 1;
186                 end
187             end
188         end
189 end
190
191 %Function 4 - Clustering (THIRD PASS)
192 %Clustering (THIRD UPDATE)
193 function [idx] = Clustering3(idx,x,y)
194     %Find cluster centers
195     m1(1) = mean(x(idx==1));
196     m1(2) = mean(y(idx==1));
197     m2(1) = mean(x(idx==2));
198     m2(2) = mean(y(idx==2));
199     cond = 0;
200     %Perform first update again
201     [~,~,~,~,~,idx] = Clustering1(x,y,m1,m2);
202 end
203
204 %Function 5 - Plotting
205     function plot(k1x,k1y,k2x,k2y,x_o,y_o,dist,string,path_1,path_2)
206     scatter(k1x,k1y,20,'red')
207     hold on
208     scatter(k2x,k2y,20,'green')
209     hold on
210     scatter(k1x,k1y,20,'red')
211     hold on
212     scatter(k2x,k2y,20,'green')
213     hold on
214     scatter(x_o,y_o,1000,'x','black')
215     hold on
```

```
216     for q = 1:length(path_1)-1
217         quiver(path_1(q,1),path_1(q,2),(path_1(q+1,1)-path_1(q,1)),(
                path_1(q+1,2)-path_1(q,2)),0,'red')
218     end
219     for j = 1:length(path_2)-1
220         quiver(path_2(j,1),path_2(j,2),(path_2(j+1,1)-path_2(j,1)),(
                path_2(j+1,2)-path_2(j,2)),0,'green')
221     end
222     xlim([0  10])
223     ylim([0  10])
224     str = sprintf(string,dist);
225     title(str)
226     hold off
227 end
228
229
230
231 %Function 6 - Path optimisation
232 function [min_path,tot_dist] = PathOpt(x,y,x_o,y_o)
233     matrix = [x' y'];
234     x_path = [x_o;x';x_o];
235     y_path = [y_o;y';y_o];
236
237     combi = perms(matrix(:,1));
238     d = dictionary(x',y');
239     sum = 1000000;
240
241     for i = 1:factorial(length(x))
242         x_path = combi(i,:);
243         path_1 = [];
244         for j = 1:length(x)
245             path_1 = [path_1; x_path(j) d(x_path(j))];
246         end
```

```
247
248         path = [x_o  y_o;path_1;x_o  y_o];
249
250         new_sum = 0;
251
252         for j = 1:(length(x)+1)
253             partial_sum = sqrt((path(j+1,1)-path(j,1))^2 + (path(j
                +1,2)-path(j,2))^2);
254             new_sum = new_sum+partial_sum;
255         end
256
257         if new_sum < sum
258                 sum = new_sum;
259                 min_path = path;
260         end
261         tot_dist = new_sum;
262     end
263 end
264
265
266 %AUXILIARY FUNCTIONS
267 %Function 1 - Finding the distance between 2 points
268 function [d] = Distance2Points(a,b)
269     d = sqrt((a(1)-b(1)).^2 + (a(2)-b(2)).^2);
270 end
271
272 end
```

A.2 Program 2

```
1 function [total_dist,total_dist_v_sqe,total_dist_v_cb,
    total_dist_v_cos,total_dist_v_cor,t] = Final(n_values,rngseed)
2 close all
3 %Creating Values
```

```
4
5  rng('default')
6  rng(rngseed)
7
8  [x,y] = CreateVals(n_values,10);
9
10 x_o = 5;
11 y_o = 5;
12
13 %{
14 %Only here when calculating time complexity
15 total_dist_v_sqe = 0;
16 total_dist_v_cb= 0;
17 total_dist_v_cos = 0;
18 total_dist_v_cor = 0;
19 %}
20
21 k = 6;
22 cost = 3;
23
24 idx = {};
25
26 for q = 1:5
27     %Find Centers
28     rng shuffle
29     c = CreateCenters(x,y,k);
30
31     %Clustering - First pass
32     [m,idx{q}] = Clustering1(x,y,c);
33
34     %Clustering - Second pass
35     idx{q} = Clustering2(idx{q},x_o,y_o,x,y);
36
```

```
37      %Clustering - Third pass
38      idx{q} = Clustering3(idx{q},x,y);
39
40      %Path optimisation for all clusters
41      nodes = {};
42      D = {};
43      path = {};
44      dist = [];
45      for i = 1:length(c)
46          nodes{i} = [x_o y_o;x(idx{q}==i).' y(idx{q}==i).'];
47          D{i} = MatrixDist(nodes{i});
48          path{i} = NNH(1,D{i});
49          path{i} = twoopt(path{i},D{i});
50          dist(i) = distTSP(path{i},D{i});
51      end
52
53      total_dist(q) = sum(dist);
54      total_dist_s(q) = sumsqr(dist);
55
56  end
57
58  [total_dist,iter] = min(total_dist);
59  idx = idx{iter};
60
61  %Path optimisation for all clusters
62  nodes_t = {};
63  D_f = {};
64  path_f = {};
65  dist_f = [];
66  for i = 1:length(c)
67      nodes_t{i} = [x_o y_o;x(idx==i).' y(idx==i).'];
68      D_f{i} = MatrixDist(nodes_t{i});
69      path_f{i} = NNH(1,D_f{i});
```

```
70      path_f{i} = twoopt(path_f{i},D_f{i});
71      dist_f(i) = distTSP(path_f{i},D_f{i});
72  end
73
74  total_dist = sum(dist_f)
75  total_dist_s = sumsqr(dist_f);
76
77  %Plotting
78  C = {'b','r','g','y','m','c','k'};
79  C2 = {'b*','r*','g*','y*','m*','c*','k*'};
80  figure(1)
81  for i = 1:length(c)
82      PrintSol(path{i},nodes{i},C2{i},C{i});
83      hold on
84  end
85  title(['Clustering (d = ',num2str(total_dist),')'])
86
87
88
89  %MUTE ALL VALIDATIONS WHEN RUNNING TIME COMPLEXITY
90  %VALIDATION - sqeuclidean
91  idx_v_sqe = kmeans([x.' y.'],k,'Distance','sqeuclidean');
92  idx_v_sqe = idx_v_sqe.';
93
94  nodes_v_sqe = {};
95  D_v_sqe = {};
96  path_v_sqe = {};
97  dist_v_sqe = [];
98  for i = 1:length(c)
99      nodes_v_sqe{i} = [x_o y_o;x(idx_v_sqe==i).' y(idx_v_sqe==i).'];
100     D_v_sqe{i} = MatrixDist(nodes_v_sqe{i});
101     path_v_sqe{i} = NNH(1,D_v_sqe{i});
102     path_v_sqe{i} = twoopt(path_v_sqe{i},D_v_sqe{i});
```

```
103     dist_v_sqe(i) = distTSP(path_v_sqe{i},D_v_sqe{i});
104 end
105
106 total_dist_v_sqe = sum(dist_v_sqe)
107 total_dist_v_s_sqe = sumsqr(dist_v_sqe);
108
109 figure(2)
110 for i = 1:length(c)
111     PrintSol(path_v_sqe{i},nodes_v_sqe{i},C2{i},C{i});
112     hold on
113 end
114 title(['Clustering squared eucledian distance validation (d = ',
        num2str(total_dist_v_sqe),')'])
115
116 %VALIDATION - cityblock
117 idx_v_cb = kmeans([x.' y.'],k,'Distance','cityblock');
118 idx_v_cb = idx_v_cb.';
119
120 nodes_v_cb = {};
121 D_v_cb = {};
122 path_v_cb = {};
123 dist_v_cb = [];
124 for i = 1:length(c)
125     nodes_v_cb{i} = [x_o y_o;x(idx_v_cb==i).' y(idx_v_cb==i).'];
126     D_v_cb{i} = MatrixDist(nodes_v_cb{i});
127     path_v_cb{i} = NNH(1,D_v_cb{i});
128     path_v_cb{i} = twoopt(path_v_cb{i},D_v_cb{i});
129     dist_v_cb(i) = distTSP(path_v_cb{i},D_v_cb{i});
130 end
131
132 total_dist_v_cb = sum(dist_v_cb)
133 total_dist_v_s_cb = sumsqr(dist_v_cb);
134
```

```
135 figure(3)
136 for i = 1:length(c)
137     PrintSol(path_v_cb{i},nodes_v_cb{i},C2{i},C{i});
138     hold on
139 end
140 title(['Clustering cityblock distance validation (d = ',num2str(
        total_dist_v_cb),')'])
141
142 %VALIDATION - cosine
143 idx_v_cos = kmeans([x.' y.'],k,'Distance','cosine');
144 idx_v_cos = idx_v_cos.';
145
146 nodes_v_cos = {};
147 D_v_cos = {};
148 path_v_cos = {};
149 dist_v_cos = [];
150 for i = 1:length(c)
151     nodes_v_cos{i} = [x_o y_o;x(idx_v_cos==i).' y(idx_v_cos==i).'];
152     D_v_cos{i} = MatrixDist(nodes_v_cos{i});
153     path_v_cos{i} = NNH(1,D_v_cos{i});
154     path_v_cos{i} = twoopt(path_v_cos{i},D_v_cos{i});
155     dist_v_cos(i) = distTSP(path_v_cos{i},D_v_cos{i});
156 end
157
158 total_dist_v_cos = sum(dist_v_cos)
159 total_dist_v_s_cos = sumsqr(dist_v_cos);
160
161 figure(4)
162 for i = 1:length(c)
163     PrintSol(path_v_cos{i},nodes_v_cos{i},C2{i},C{i});
164     hold on
165 end
166 title(['Clustering cosine distance validation (d = ',num2str(
```

```
    total_dist_v_cos),')'])

%VALIDATION - correlation
idx_v_cor = kmeans([x.' y.'],k,'Distance','correlation');
idx_v_cor = idx_v_cor.';

nodes_v_cor = {};
D_v_cor = {};
path_v_cor = {};
dist_v_cor = [];
for i = 1:length(c)
    nodes_v_cor{i} = [x_o y_o;x(idx_v_cor==i).' y(idx_v_cor==i).'];
    D_v_cor{i} = MatrixDist(nodes_v_cor{i});
    path_v_cor{i} = NNH(1,D_v_cor{i});
    path_v_cor{i} = twoopt(path_v_cor{i},D_v_cor{i});
    dist_v_cor(i) = distTSP(path_v_cor{i},D_v_cor{i});
end

total_dist_v_cor = sum(dist_v_cor)
total_dist_v_s_cor = sumsqr(dist_v_cor);

figure(5)
for i = 1:length(c)
    PrintSol(path_v_cor{i},nodes_v_cor{i},C2{i},C{i});
    hold on
end
title(['Clustering correlation distance validation (d = ',num2str(
    total_dist_v_cor),')'])


%PROGRAM FUNCTIONS
%Fuction 1 - Creating Values
function [x,y] = CreateVals(n_values,range)
```

```
198     x = rand(1,n_values)*range;
199     y = rand(1,n_values)*range;
200 end
201
202 %Function 2 - Find cluster centers
203 function [c] = CreateCenters(x,y,k)
204 %Select a random point as the first cluster
205     init_c = randi(length(x));
206     c(:,1) = [x(init_c);y(init_c)];
207
208     %For each point, find it's squared distance to c1 and create a
209     %probability distribution with those numbers. Find second center
210     %randomly with that given distribution
211     s_distance = (x-c(1,1)).^2+(y-c(2,1)).^2;
212     prob = s_distance./sum(s_distance);
213     index_c2 = find(rand<cumsum(prob),1,'first');
214     c(:,2) = [x(index_c2);y(index_c2)];
215
216     %To find the rest of the centers, do the same thing but only use
            the
217     %minimum distance from each point to a cluster center
218     for i=3:k
219         %Find the distance to each cluster center
220         for j = 1:length(x)
221             dist_2_center = zeros(1,length(c));
222             for m = 1:length(c)
223                 dist_2_center(m) = (x(j)-c(1,m))^2+(y(j)-c(2,m))^2;
224             end
225             final_s_dist(j) = min(dist_2_center);
226         end
227         prob = final_s_dist./sum(final_s_dist);
228         index = find(rand<cumsum(prob),1,'first');
229         c(:,i) = [x(index);y(index)];
```

```
230     end
231 end
232
233 %Function 3 - Clustering (FIRST UPDATE)
234 function [m,idx] = Clustering1(x,y,c)
235     idx = [];
236     k = length(c);
237     cond = 0;
238     m = c;
239
240     flag = 0;
241
242     while (cond == 0 && flag < 10000)
243         flag = flag + 1;
244         m_old = m;
245         %For every point, find the distance to each cluster center
              and
246         %assign it to the cluster with minimum distance
247
248         for i = 1:length(x)
249             point = [x(i) y(i)];
250             for j = 1:k
251                 d(j) = Distance2Points(point,c(:,j));
252             end
253             [m,cluster] = min(d);
254             idx = [idx cluster];
255         end
256
257         %Find new cluster centers
258         for i = 1:k
259             m(1,i) = mean(x(idx==i));
260             m(2,i) = mean(y(idx==i));
261         end
```

```
262
263         %Check if cluster centers keep changing
264         if (m == m_old)
265             idx = [];
266         else
267             cond = 1;
268         end
269
270     end
271 end
272
273 %Function 4 - Clustering (SECOND UPDATE)
274 function [idx] = Clustering2(idx,x_o,y_o,x,y)
275     kx = {};
276     ky = {};
277     k = max(idx);
278
279         for i = 1:length(x)
280             for p = 1:k
281                 kx{p} = x(idx==p);
282                 ky{p} = y(idx==p);
283             end
284
285             for j = 1:k
286                 nodes = [x_o y_o;kx{j}.' ky{j}.'];
287                 D = MatrixDist(nodes);
288                 path = NNH(1,D);
289                 path = twoopt(path,D);
290                 dist(j) = distTSP(path,D)^cost;
291             end
292             old_sum_of_path = sum(dist);
293             idx_new = idx;
294             for j = 1:k
```

```
295                     if j ~= idx(i)
296                         idx_new(j) = j;
297                         for n = 1:k
298                             kx{n} = x(idx_new==n);
299                             ky{n} = y(idx_new==n);
300                             nodes = [x_o y_o;kx{n}.' ky{n}.'];
301                             D = MatrixDist(nodes);
302                             path = NNH(1,D);
303                             path = twoopt(path,D);
304                             new_dist(n) = distTSP(path,D)^cost;
305                         end
306                         new_sum_of_path(j) = sum(new_dist);
307                     else
308                         new_sum_of_path(j) = old_sum_of_path;
309                     end
310                 end
311                 [~,cluster] = min(new_sum_of_path);
312                 idx(i) = cluster;
313             end
314 end
315
316 %Clustering (THIRD UPDATE)
317 function [idx] = Clustering3(idx,x,y)
318     k = max(idx);
319     %Find cluster centers
320     for i = 1:k
321             m(1,i) = mean(x(idx==i));
322             m(2,i) = mean(y(idx==i));
323     end
324     cond = 0;
325     %Perform first update again
326     [m,idx] = Clustering1(x,y,m);
327 end
```

```
328
329 %AUXILIARY FUNCITONS
330 %Funciton 1 - Finding the distance between 2 points
331 function [d] = Distance2Points(a,b)
332     d = sqrt((a(1)-b(1)).^2 + (a(2)-b(2)).^2);
333 end
334 end
```

A.3 2Opt

A.3.1 MatrixDist

```
1  %Creating a Matrix of Distances
2  function D = MatrixDist(x)
3  cordx = x(:,1);
4  cordy = x(:,end);
5  for i=1:length(cordx)
6      for j=1:length(cordy)
7          x1 = cordx(i);
8          x2 = cordx(j);
9          y1 = cordy(i);
10         y2 = cordy(j);
11         %Calculate distances between points
12         distEucledian = sqrt((x2-x1)^2+(y2-y1)^2);
13         D(i,j) = distEucledian;
14     end
15 end
```

A.3.2 NNH

```
1  function y = NNH(x0,D)
2  %Nearest neighbor Heuristics
3  %xo = starting point
4  %D = Distance matrix
5  %Returns y: path
```

```
6  n = length(D);
7  y = x0;
8  D(:,x0) = inf; % Change the distance matrix to infinity so it never
       visits the point again
9  for i = 1: n-1 %For the rest of the points
10     [m x0] = min(D(x0,:)); %Look for the point with the smallest
           distance
11     y = [y x0];
12     D(:,x0) = inf;
13 end
```

A.3.3 twoopt

```
1  function y=twoopt(y,D)
2  n = length(y);
3  flag = 1; %For when there's no more intersections
4  while flag == 1
5      s = distTSP(y,D);
6      flag = 0;
7      for i=1:n-2
8          for k = i+2:n
9              yT = swap(y,i,k);
10             sT = distTSP(yT,D);
11             if sT < s %If the solution is improved
12                 y = yT;
13                 s = sT;
14                 flag = 1;
15             end
16         end
17     end
18 end
```

A.3.4 distTSP

```
1  function dist = distTSP(y,D)
```

```
%Given a TSP solution, find the displacement
%y = TSP solution, D = Distance matrix

n = length(D);
dist = 0;
for i=1:n-1
    dist = dist+D(y(i),y(i+1));
end
dist = dist + D(y(n),y(1));
```

A.3.5 PrintSol

```
function PrintSol(y,p,c,c2)
hold on
n = length(y);
for i=1:n
    plot(p(i,1),p(i,2),c)
    cordx(i)=p(y(i),1);
    cordy(i)=p(y(i),2);
    txt=sprintf(' %d',y(i));
    text(cordx(i),cordy(i),txt)
end
cordx(n+1) = p(y(1),1);
cordy(n+1) = p(y(1),2);
plot(cordx,cordy,'color',c2)
end
```

A.3.6 swap

```
function y = swap(y,i,k)
j=i+1; l=k+1;
y1 = y(1:i); y2 = y(k:-1:j); y3 = y(l:end);
y = [y1 y2 y3];
```

Appendix B

Code To Gather Results

B.1 Objective 1

B.1.1 runx50 – Program 1

```
function yourLoop
result = zeros(50, 2);
tic
for k = 1:50
    counter = k
    [my_dist,their_dist] = Final();
    result(k,1) = my_dist
    result(k,2) = their_dist
end
t = toc/100
writematrix(result,'n10.xlsx','Sheet',1)
end
```

B.1.2 Time Complexity – Program 1

```
t = zeros(20,1);

    for n = 5:14
    counter = n
    [~,~,time] = Final(n);
```

```
    t(n) = time;
    end

writematrix(t,'time_complexity_MonteCarlo,2K,linear_cost.xlsx','
    Sheet',1)
```

B.2 Objective 2

B.2.1 runx50 – Program 1

```
function yourLoop
result = zeros(50, 5);
tic
for k = 1:50
    counter = k
    [my_dist,their_dist_sqe,their_dist_cb,their_dist_cos,
        their_dist_cor,~] = Final(100);
    result(k,1) = my_dist
    result(k,2) = their_dist_sqe
    result(k,3) = their_dist_cb
    result(k,4) = their_dist_cos
    result(k,5) = their_dist_cor
end
t = toc/100
writematrix(result,'100nodes,6K,cubed_cost.xlsx','Sheet',1)
end
```

B.2.2 Time Complexity – Program 1

```
t = zeros(150,1 );

for j = 1:8
    for n = 5:150
    counter = n
    [~,~,~,~,~,time] = Final(n,j);
```

```
7         t(n,j) = time;
8         end
9   end
10
11  writematrix(t,'time_complexity_MonteCarlo,2K,linear_cost.xlsx','
        Sheet',1)
```