Table of Contents

[1 Abstract 3](#_Toc40106908)

[2 Introduction 3](#_Toc40106909)

[2.1 Aims and Objectives 3](#_Toc40106910)

[3 Literature Review 3](#_Toc40106911)

[3.1 Drone Delivery 4](#_Toc40106912)

[3.2 Problem Specification 5](#_Toc40106913)

[3.3 The Travelling Salesman Problem (TSP) 7](#_Toc40106914)

[3.3.1 Brute Force 8](#_Toc40106915)

[3.3.2 A\* Search 8](#_Toc40106916)

[3.3.3 Hill Climbing 8](#_Toc40106917)

[3.3.4 Convex Hull & Cheapest Insertion 9](#_Toc40106918)

[3.3.5 Genetic Algorithm 10](#_Toc40106919)

[3.4 Clustering 11](#_Toc40106920)

[3.4.1 K-means 12](#_Toc40106921)

[3.4.2 Affinity Propagation 12](#_Toc40106922)

[3.5 Scheduling 13](#_Toc40106923)

[3.6 Summary 14](#_Toc40106924)

[3.7 Conclusion 14](#_Toc40106925)

[4 Design and Methodology 15](#_Toc40106926)

[4.1 Requirements 15](#_Toc40106927)

[4.1.1 Functional Requirements 15](#_Toc40106928)

[4.1.2 Non-Functional Requirements 16](#_Toc40106929)

[4.2 Key Components 17](#_Toc40106930)

[4.2.1 Graphic User Interface (GUI) 17](#_Toc40106931)

[4.2.2 Clustering 18](#_Toc40106932)

[4.2.3 Weather Data 18](#_Toc40106933)

[4.2.4 Path Finding 18](#_Toc40106934)

[4.2.5 Time of each leg 18](#_Toc40106935)

[4.2.6 Greedy Best First 19](#_Toc40106936)

[4.2.7 Genetic Algorithm 20](#_Toc40106937)

[4.2.8 Database 20](#_Toc40106938)

[4.2.9 Output 21](#_Toc40106939)

[4.3 Development Methodology 22](#_Toc40106940)

[5 Implementation 23](#_Toc40106941)

[5.1 GUI 23](#_Toc40106942)

[5.2 Clustering 23](#_Toc40106943)

[5.2.1 Affinity Propagation 24](#_Toc40106944)

[5.2.2 kMeans 24](#_Toc40106945)

[5.3 Wind Data 24](#_Toc40106946)

[5.4 Generating Routes 24](#_Toc40106947)

[5.4.1 Haversine + Azimuth + E6B 24](#_Toc40106948)

[5.4.2 Genetic Algorithm 25](#_Toc40106949)

[5.4.3 Greedy Best First 25](#_Toc40106950)

[5.5 Database 26](#_Toc40106951)

[5.6 Output 26](#_Toc40106952)

[5.6.1 Folium 26](#_Toc40106953)

[5.6.2 BeautifulSoup 26](#_Toc40106954)

[5.7 Challenges 26](#_Toc40106955)

[5.7.1 Impossible Clusters 26](#_Toc40106956)

[5.7.2 Weather API 27](#_Toc40106957)

[5.7.3 Limitations of Folium 27](#_Toc40106958)

[5.8 Implementation Methodology 27](#_Toc40106959)

[6 Testing 27](#_Toc40106960)

[6.1 Acceptance Testing 28](#_Toc40106961)

[6.1.1 Must Have 28](#_Toc40106962)

[6.1.2 Should Have 28](#_Toc40106963)

[6.1.3 Could Have 28](#_Toc40106964)

[6.2 Optimising Genetic Algorithm 29](#_Toc40106965)

[6.3 Variation in GA results 29](#_Toc40106966)

[6.4 Genetic Algorithm vs Greedy Best First 29](#_Toc40106967)

[6.4.1 Route length, time and run times 29](#_Toc40106968)

[6.5 Unit Testing 30](#_Toc40106969)

[7 Evaluation 31](#_Toc40106970)

[7.1 Overview 31](#_Toc40106971)

[7.2 Design 31](#_Toc40106972)

[7.3 Implementation 31](#_Toc40106973)

[7.4 Test Results 32](#_Toc40106974)

[7.4.1 Optimising Genetic Algorithm 32](#_Toc40106975)

[7.4.2 Variation in GA results 32](#_Toc40106976)

[7.4.3 Genetic Algorithm vs Greedy Best First 33](#_Toc40106977)

[8 Conclusion 35](#_Toc40106978)

[8.1 Overview 35](#_Toc40106979)

[8.2 Findings 35](#_Toc40106980)

[8.3 Future Work 35](#_Toc40106981)

[9 Appendices 36](#_Toc40106982)

[9.1 Code 36](#_Toc40106983)

[9.2 Project Log 36](#_Toc40106984)

[9.3 Test results 41](#_Toc40106985)

[9.3.1 Finding optimal population size and no. generations 41](#_Toc40106986)

[9.3.2 Multiple runs on same data 43](#_Toc40106987)

[9.4 ‘Aberdeen’ dataset 47](#_Toc40106988)

[9.5 Poster 47](#_Toc40106989)

[9.6 Ethics Form 48](#_Toc40106990)

[9.7 Project Proposal 52](#_Toc40106991)

[9.8 Requirements Analysis 56](#_Toc40106992)

# Abstract

# Introduction

Drones are an emerging technology for delivery. Many companies are developing systems to utilise them, such as Amazon with their “Prime Air” platform, and DHL with their “Parcelcopter”. They hold many benefits over traditional road-based delivery methods, such as lower cost, faster delivery, and lower environmental impact. However, there are some obstacles to overcome before they see widespread use, such as legal issues and low flight range.

As with any delivery method, it is important that drones do not waste time. Their routes need to be optimised to speed up delivery for customers and companies alike. Because of this, it is vital that a schedule is created and maintained that details where each drone will be going and when.

## Aims and Objectives

The aim of this project is to create a drone delivery scheduler. The following objectives must be completed to succeed at the task:

- Implement a simple Graphic User Interface (GUI)

- Develop a system to generate routes between customers

- Present the results graphically

- Allow comparison of different algorithms

It is worth noting that the objectives listed here differ from the ones outlined in the project proposal. As I conducted research during the literature review and design stages of the project, I realised that the original objectives did not properly convey what I wanted to create. As such, they have been updated to outline what the solution I have implemented has set out to do.

# Literature Review

This section explores literature related to drone delivery scheduling. It will cover the background of drones and difficulties in implementing a scheduler. Finally, a conclusion will be drawn from the findings, as well as suggested techniques to solve the problem.

## Drone Delivery

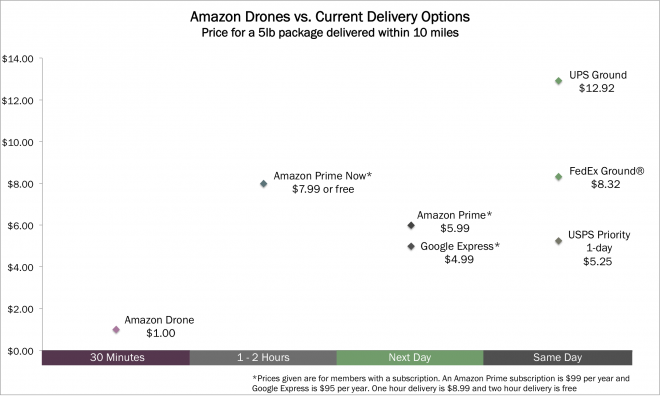
A drone is a small, unmanned flying vehicle. In recent years many companies have unveiled plans to begin delivering packages via drone. In 2013 Amazon announced its ‘Prime Air’ service that will deliver packages to customers (Dorling et al., 2017). DHL has announced its ‘Parcelcopter’ project, which has successfully delivered medicine to the island of Juist in the North Sea (Dorling et al., 2017). Many more companies are doing the same, such as Google and Swiss Post (Dorling et al., 2017). This sudden upsurge in use has been brought on by advancements in technology used in the construction of drones. Improvements in battery technology allow drones to fly faster and further than ever before (Dorling et al., 2017).

There is a great demand from customers for a faster, more reliable option for delivery. A study conducted on over 4700 people from China, Germany, and the USA showed that 23% of customers are willing to pay extra for the benefit of same-day delivery (McKinsey & Company, 2016).

**Figure 1 – Amazon Prime Air drone (Amazon)**

There are several advantages to using drones for last-mile delivery. The primary advantage of drones versus truck is the speed and timing accuracy as drones are not affected by traffic or road layout of a city. The lack of these constraints enables them to offer fast delivery and tell the customer to the minute when the item will arrive (Lee, 2016).

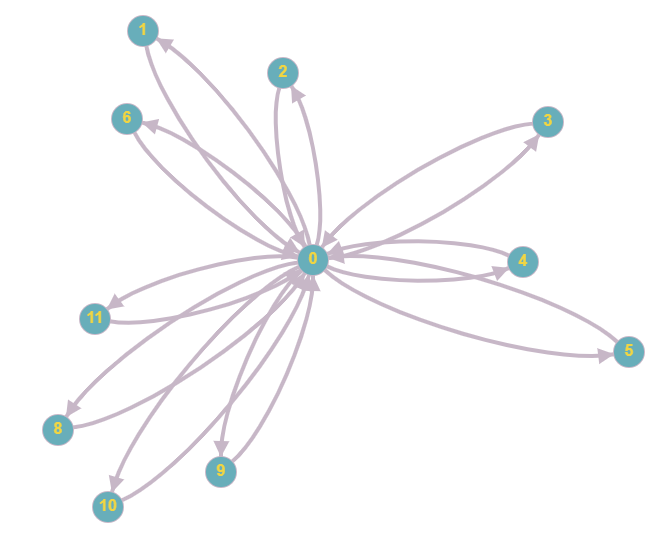
For the company deploying the delivery solution, they will likely save money. A study performed by ARK Invest suggests that Amazons drone delivery service could be charged at just $1 per delivery and still be profitable. (Keeney, 2015)(Figure 2).

Research suggests that delivery by drones will be environmentally beneficial. As drones are battery powered, they do not directly produce any diesel pollution. Research shows that carbon dioxide emissions produced by drones are lower than that of trucks when used for locations close to the depot, or small numbers of recipients, or both. (Goodchild and Toy, 2018). The speed, cost, and environmental benefits are the critical advantages of delivery via drone. They are the driving force behind the recent upsurge in use.

**Figure 2 – Comparison of delivery cost and time across several mediums and companies (Keeney, 2015)**

## Problem Specification

We are going to assume that a company has set up a depot. They will use this as the base for their drones. They will also use it as a charging station while they are not in use.

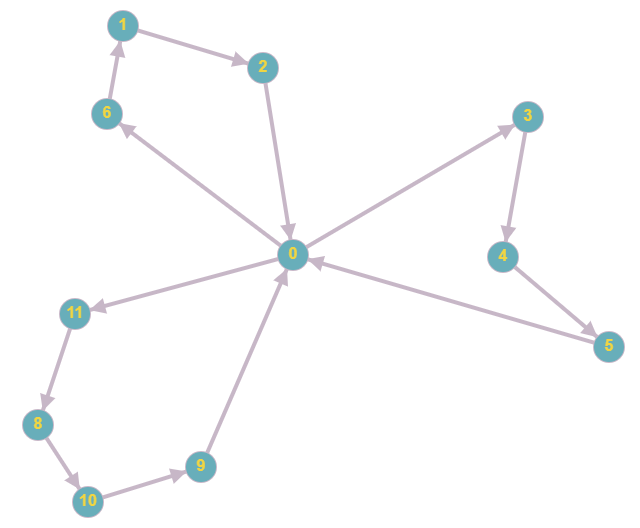
Amazon announced on 5th June 2019 that they expect their drones to be able to fly up to 15 miles and deliver packages under five pounds (Wilke, 2019). Jeff Bezos has stated that 86% of items delivered by Amazon weigh five pounds or less (Guglielmo, 2013). This statistic shows us that Amazon is aiming to deliver one item per drone per trip. Because of this limitation, the current scheduling task is simple. If a drone can only carry one item at a time, a fair solution would be to send items in chronological order. The orders would enter a queue and wait their turn for a drone to be available. If drones were able to carry more weight while retaining a 15-mile range, we can look to solve a more complex problem. The Alta 8 from FreeFly Systems can carry up to 18kg (Freefly Systems, 2019). It is reasonable to assume that technology will continue to improve to the stage where companies will use one drone to carry several items at once.

**Figure 3 – Delivery with one item per flight**

If drones can carry multiple items per flight, it allows the company to deliver to many more customers concurrently, without having to scale up the number of drones they own. With this model, we need to calculate optimal routes and group customer locations so that one drone can serve many people. While we would still have to consider when an order is placed, it would no longer be the only parameter for our scheduling. Someone may move from position one in the queue to position five if we find that a more efficient route is to send a single drone to four other people first. With this system, we can still provide accurate delivery estimations while also improving the efficiency of our system. These assumptions of the future of drone tech and their use within industry turn this problem from simple delivery time scheduling to a travelling salesman problem.

In summary:

* Current delivery services are aiming for one item per drone per flight
* Scheduling delivery is a matter of maintaining a first-in-first-out system which dispatches items in the same order that customers place them
* We are looking to the future where multi-item deliveries will likely be possible
* Routes will be created to allow one drone to serve multiple customers per flight
* This system will allow us to handle more customers simultaneously without increasing the number of drones we need



**Figure 4 – Delivery with multiple items per flight**

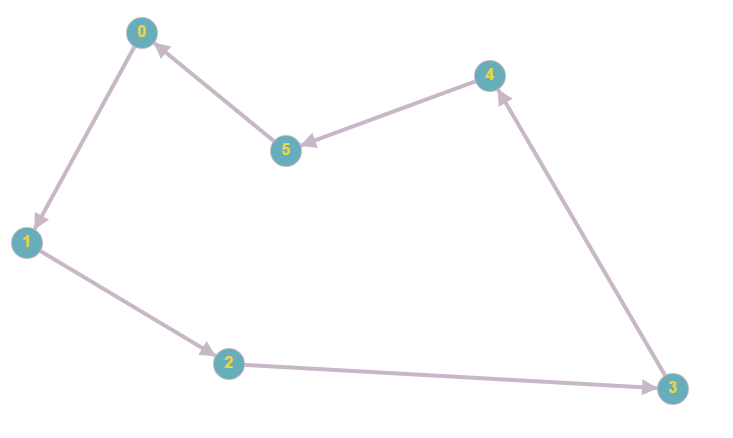
## The Travelling Salesman Problem (TSP)

The travelling salesman problem is an old one, and variations appear as early as 1759 by Euler (Larrañaga et al., 1999). In the 1930s, mathematicians in Vienna and Harvard studied the problem(math.utwaterloo.ca). The problem describes a salesman who must visit multiple cities. He only wants to visit each city once and wants to finish where he starts. A perfect solution to the problem finds the shortest route for the salesman to take to complete their journey (Saiyed, 2012). The problem of drone scheduling is an example of a TSP. It is scaled down, so instead of a salesman travelling between cities, we have a drone travelling between people’s homes.

Initially the problem seems trivial. It is simple to understand, and the method of solving it is not complicated. All we must do is find every route and pick the shortest one.

If we name the number of homes ‘n’, the number of possible routes is the factorial of ‘n’ (Saiyed, 2012). If the drone must visit five homes, there are 120 possible routes. If we increase this to 10 homes, there are 362,800 possible routes, and 15 homes give us 1.3e12 possible routes. This exponential growth is where the difficulty lies in solving the TSP.

If we were able to find and evaluate 1 million routes per second, it would take over 15 days to find the solution for a 15-point route. We may have thousands of orders a day, and the amount of processing time and power to brute force the best route is unrealistic. For this reason, we need to find alternative methods to solve the TSP. The problem is NP-hard, which means that there are no known techniques to solve it in polynomial time. (Bryant, 2000)



**Figure 5 –Typical travelling salesman route with 6 locations**

### Brute Force

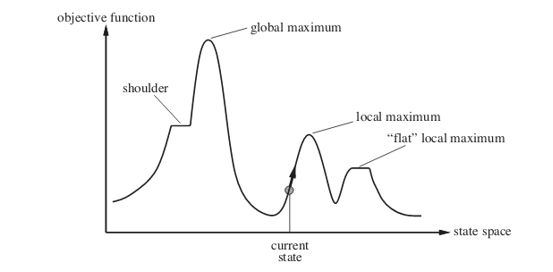
The simplest method of solving the problem is brute force. The method is to run through every possible route and pick the shortest one. This method is possible for a small number of destinations; however, this is an unrealistic method as the number of possible routes increases so dramatically for each extra item that must be delivered. For drone delivery, this is not a feasible method to use.

### A\* Search

An informed search is a type of search that contains an array of knowledge about the search space, such as cost so far and distance from the target. There is a range of algorithms that fall under informed searches, but we will focus on A\*. A\* has several benefits but it mainly sees use due to being an optimal and complete algorithm. What this means is that on any given search space, if there is a solution, A\* is guaranteed to find the best one (Nosrati, Karimi, Hasanvand, 2012).

There are two critical issues with using this type of search on this domain. Primarily, we do not know what the target is for the search to find, other than to have visited every location and have the shortest route possible. If we knew this target, there would be no need to perform a search at all. A\* searches are useful for finding the route to a target and showing how to get there. In our case, we do not care about how to get there; we only want the final route. The second issue we have is hardware limitations. As A\* will only end when the best solution is found, the time taken to complete the search can be extremely long (Nosrati, Karimi, Hasanvand, 2012). These limitations add up and make A\* an unsuitable method of solving our problem.

### Hill Climbing

Hill Climbing is a basic local search algorithm where it looks to each of its neighbouring states and selects the one that seems the best. In this domain, it would look to every location it has not visited and select the closest one. The search is complete and returns the route that it has found once there are no better choices immediately surrounding it. The issue with this type of search is that it can become stuck in a local maximum, where there are no better places for the search to go locally, but there are elsewhere within the domain. (Saiyed, 2012)

**Figure 6 – Hill Climbing algorithm (Geeksforgeeks)**

### Convex Hull & Cheapest Insertion

These are a combination of techniques that are viable to solve the TSP. Figure 7 shows how these methods can be used together to create a route for our drone.

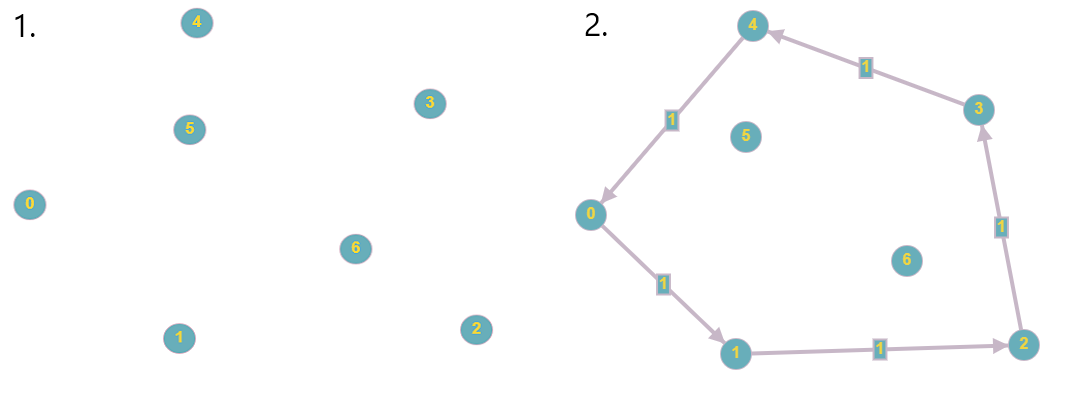
The convex hull algorithm is used to create an outside boundary that all locations lie within. The process is as follows:

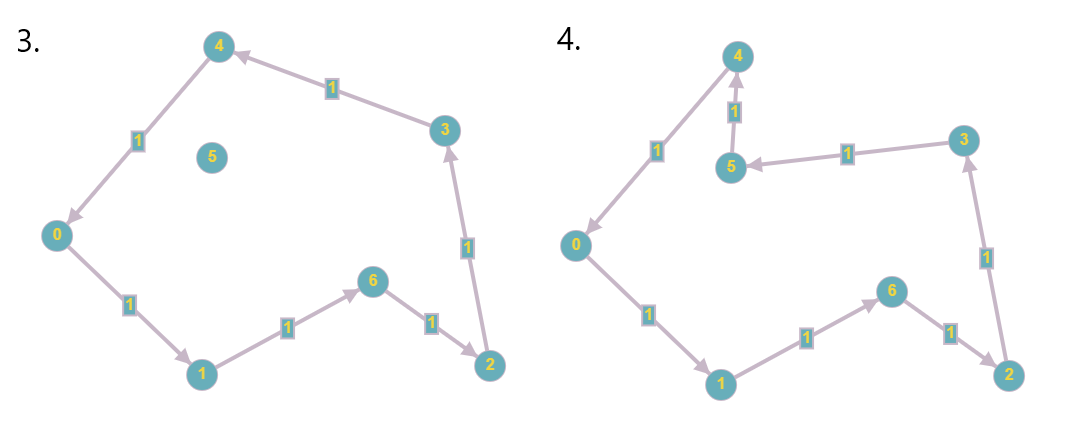
* Select an extreme point such as leftmost
* Select an angle away from every other point
* Beginning from this angle, choose a direction to turn to look for another point
* The first one that is found becomes the next point
* Repeat until we return to the starting point

From here, we use a method called cheapest insertion to visit all the locations that lie within this boundary. It works on these steps:

* Find every location and every way to get to this location
* Calculate a penalty for travelling. This is the difference between the distance travelled for the new route, and distance travelled for the old route
* Select the route with the lowest penalty
* Repeat until every point has been visited

(Goetschalckx,2011)





**Figure 7 – Route creation using convex hull and cheapest insertion**

### Genetic Algorithm

Bremermann et al first proposed genetic algorithms in 1965 (Larrañaga et al., 1999). They are intended to simulate evolution as it occurs in nature. They mimic natural selection by selecting only the best individuals to go on to produce more individuals in the next generation. Each individual has a set of characteristics, and they pass this on to their offspring, so the algorithm maintains healthy genes through generations.

The search begins with a randomly generated set of individuals, or in the case of TSP, routes. These individuals are characterised by a set of parameters, and should all be different. From here, a fitness score is calculated for each individual. This score determines how good the solution is compared to others. Here our fitness score would be the length of the route.

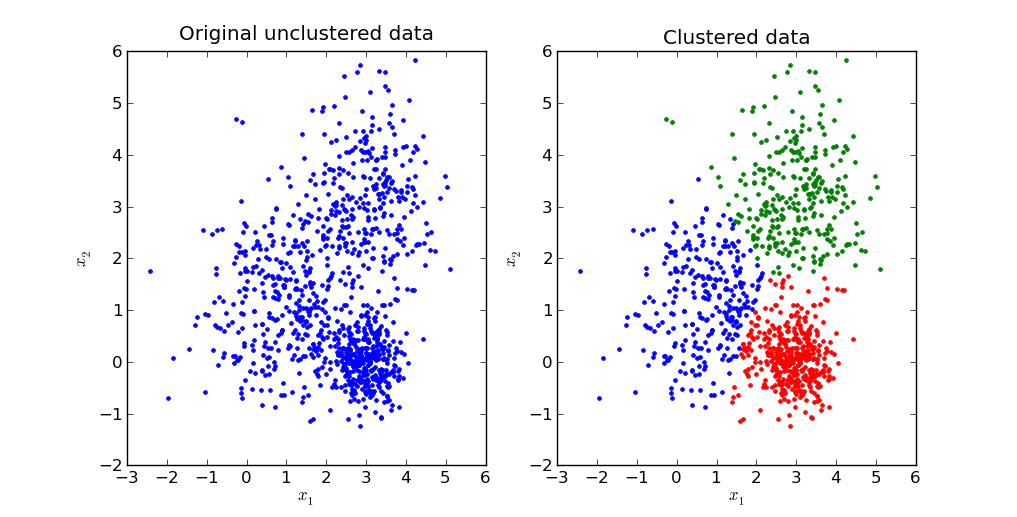
Next, selection occurs. Selection is the method of passing on genes to the next generation. Two pairs of individuals are selected based on their fitness score and move forward.

**Figure 8 – Genetic Algorithm flow chart (Apache ignite)**

A process called crossover then occurs. For each pair selected to produce offspring, a crossover point is selected. The crossover point is a random point somewhere in the genes. Offspring are generated by exchanging genes within this crossover point, and the offspring are added to the population. When offspring are formed, there is a low probability that mutation will occur, meaning that specific properties of the new individual change in some way. For our domain, this may be a pair of customer locations swapping position in the route.

The process continues until the population has converged. Conversion here means that the offspring being created are not significantly different from the generation that created them. The size of the population does not grow. Once new generations have been formed, individuals that have the lowest score are removed from the search space. (Larrañaga et al., 1999) (Bryant, 2000)

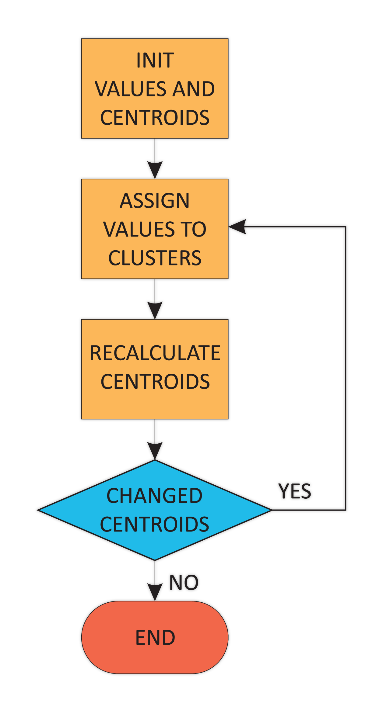
## Clustering

Because of both the difficulty of the Travelling salesman Problem, and the limitations of drone technology, it is useful to break down our problem into smaller groups, or clusters. If we take the above example of 15 cities, and we split this into three sets of five, our computation time drops dramatically. Assuming the same processing power, this would take 0.00036 seconds to find and evaluate the best routes. While one million routes per second is an unrealistic number, it illustrates the potential benefits to computing and time resources needed. 

**Figure 9 – Clustering data into 3 clusters (mubaris)**

Clustering allows us to model a delivery system with multiple drones delivering simultaneously, which is more realistic than just creating one route for one drone to carry out. If we assume that drones will be able to carry more weight but fly the same distance as they currently can, we still need to create clusters. We do not want drones to be flying randomly from one edge of their range to the other, but instead to deliver to a few tightly grouped locations and return to the depot. These two factors show the need for clustering on our problem.

### K-means

The k-means technique takes a parameter of k, and randomly selects that many locations to begin. These locations are set as initial centroids, or exemplars. From here, the algorithm assigns each location to a cluster depending on which centroid is nearest. It then recalculates the centroid by taking the mean of all the locations per cluster. Finally, it reassigns locations to their nearest centroid again. This process repeats until no locations change cluster. The model can be adapted slightly to assign the closest location to the mean as the centroid. This adjustment is known as k-mediod. (Bruggeman et al., 2010)

**Figure 10 – K-means clustering (oreilly)**

While this technique sees wide use, it has several issues (Google Developers, 2019). Primarily we need to define the number of clusters ourselves. To do this requires additional analysis before deciding what value to use for the final model, testing a few different values of k, and comparing against some metric. Additionally, k-means can give poor results if a poor spread of initial locations is selected. Because the locations are randomly selected, they may end up all in proximity. The solution is to run the algorithm several times until it finds a good result; however, this costs time and processing resources. (Google Developers, 2019).

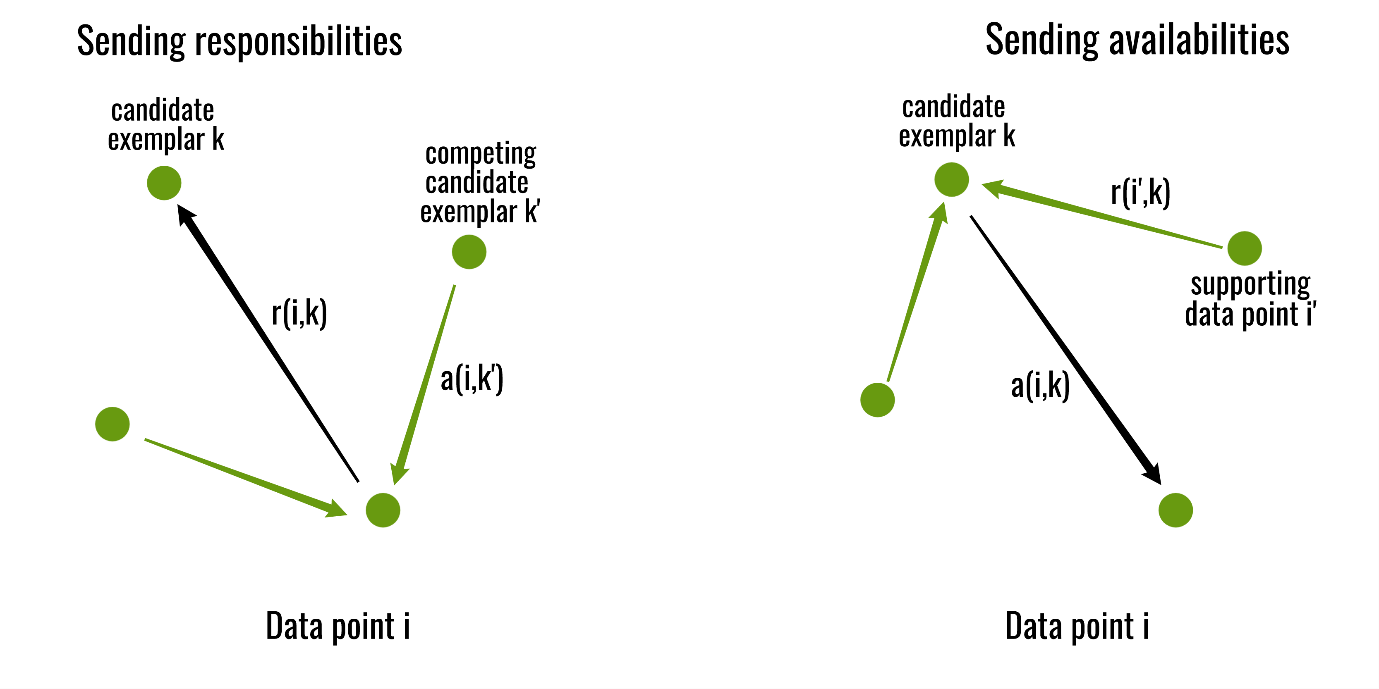
### Affinity Propagation

Affinity Propagation solves these key issues that k-means faces. It tackles the issue of determining how many clusters to have; it defines its own, with no input from the user. Because of this, it does not fall victim to a poor initial selection of locations.

The process works by alternating between two steps:

* Responsibility. This step shows how well fitted a location is to be the exemplar for another relative to all other locations. Low similarity means a low chance of two points becoming part of the same cluster. Each location sends this similarity to each other location.
* Availability. This step shows how appropriate it would be for a location to pick another location as its exemplar, considering the responsibility score it received from each other location. This score is sent back as a reply from each location to each other location.

The process iterates until there is an agreement between all locations for where is the best exemplar for where, and no further changes are needed. The process iterates until there is an agreement for which nodes are the best exemplars. (Tan et al., 2016).



**Figure 11 – Affinity propagation steps (Geeksforgeeks)**

Lizhuang Tan et al performed a comparative study of k-means and affinity propagation for clustering with a travelling salesman problem. They surmised that both algorithms showed an improvement in computational cost than when solving the same problem without clustering. Furthermore, they concluded that they prefer affinity propagation because of the sensitivity of k-means to poor initial centroids, as well as a requirement for a pre-set number of clusters.

## Scheduling

A schedule is the list of destinations for our system to carry out. It outlines which drone needs to go to which customer, and in what order. Figure 3 shows a sample domain where drones can only handle one item per flight. Here the system assigns drones to customers using a first-in-first-out system.

Point zero represents the depot, and each other point represents a customer location. The lines show the flight paths of drones.

Assuming we have three drones, our schedule may be:

**Drone 1**: 0 – 1 – 0 – 2 – 0 – 3 – 0 – 4 – 0

**Drone 2**: 0 – 5 – 0 – 6 – 0 – 8 – 0

**Drone 3**: 0 – 9– 0 – 10 – 0 – 11 – 0

Figure 4 shows the same customers and depot, but with drones allowed to carry multiple items per flight. In this example, we can break the problem down. We apply a clustering technique to group nearby customers together and then generate a route using one of the TSP solving algorithms.

Here our schedule may look like:

**Drone 1**: 0 – 3 – 4 – 5 – 0

**Drone 2**: 0 – 11 – 8 – 10 – 9 – 0

**Drone 3**: 0 – 6 – 1 – 2 – 0

In our first schedule, the drones make 20 trips between points. When they carry many items, only 13 trips are needed. The elimination of unnecessary trips back to the depot will save time and money.

## Summary

In summary, drones are becoming an increasingly prominent method of delivering items to customers. Due to advances in technology, it is becoming more viable to use them. Several large multinational companies have expressed interest in using the technology. There are several issues to be resolved before this becomes a reality, one of which is fast and efficient delivery scheduling. Efficient delivery benefits both the company, as they save money, and the customer, as they save time. This project will aim to solve the scheduling problem.

## Conclusion

Based on findings in the literature above, we can select several techniques to create a solution for drone delivery scheduling. Some constraints of drones will be relaxed, and we will assume they can carry more weight and, thus, multiple items at once. Research suggests that breaking customer locations into clusters is essential so that the solution is likely to be relevant in the near future. For this, we will use the affinity propagation technique outlined in section 2.2.2, as our research shows that this method is more fitting to our domain than k-Means or k-Mediods. Once the problem is broken into clusters, we are left with multiple travelling salesman problems to solve. We will use a genetic algorithm to solve these. The genetic algorithm has been chosen as it is a stimulating technique to learn and has seen much application on solving travelling salesman problems.

# Design and Methodology

This chapter will discuss the requirements of the project. The requirements are a direct result of the aims and objectives, and are steps required to complete these.

Key pieces of code that have been implemented are then explored for details of their design and intended operation. Care has been taken to ensure the software is broken down into many distinct sections, this ensures that separation of concerns occurs, which will allow easier debugging and testing.

Finally, the methodology used throughout the development of this project is explored and justified.

## Requirements

The requirements have been split into categories depending on importance using the MoSCoW system. Each requirement is classed as either “Must”, “Should”, “Could”, “Won’t”. For this project, “Won’t” has not been detailed.

“Must” requirements are those that the project cannot go without.

“Should” are important requirements, but not completely essential to the project.

“Could” are additional features that would be nice to have, but are a bonus, and their omission does not affect the project in any way.

### Functional Requirements

The functional requirements are functions of the implemented system. These detail tasks that the completed system will complete.

#### Must

- A GUI – It is crucial to make the application usable to anyone that it has a simple GUI. This GUI has several characteristics that must be implemented to complete this task:

- A real map of a city. It allows the user to test out and see real-world results of their use.

- Input orders from the GUI. It is no good having a GUI if the user still must input data using some other means

- Controllable parameters. One of the main aims of the project was to allow comparison of parameters. It will help users to see how adding or taking away drones for example will affect the results.

- Division of customers into smaller groups. As discussed in the Literature Review, it is vital that customers are broken down into manageable groups. Without this feature, the software will likely take far too long to complete path-finding, will be less likely to produce a good route, and also will not create a realistic scenario of multiple drones being able to fly at once.

- Sensible routes must be created. This is perhaps the main function of the application, and without it there is little to show. Routes must be created among customer groups that are sensible and follow rules in an attempt to optimise them.

#### Should

- Multiple path finding algorithms implemented. This falls under the aim of comparing input parameters. It will be interesting for users to see and compare different algorithms, however providing there are other parameters that can be controlled, and one path-finding algorithm is included, this is not vital functionality.

- Consider weather conditions. One of the big drawbacks of drones is that they are greatly affected by the weather. To simulate a real-world system more accurately, this should be taken into consideration. Weather data such as wind speed, direction and precipitation may all be considered.

- Interactive user input. In a real-world system, the application would be running constantly and waiting for new orders to come in. If a new order is placed, the system should not shut down while it deals with this, it should be able to change an existing route or create a new one to accommodate the new customer.

#### Could

- A display of the current state of routes as they are created. This would be an interesting feature to allow the user to see how the path-finding algorithm selected works.

- When running, show the location of each drone as they move through their route. This would be a bonus feature where the system is running in real time and updates with progress of drones along their routes.

### Non-Functional Requirements

Non-Functional Requirements are those that do not detail functionality of the solution. They outline things such as hardware and software environments and how the solution should be delivered.

#### Must

- Use python for development. Python has been selected due to familiarity from previous projects. Python is known for being useful for implementing AI solutions, which this project aims to do. Additionally, there are a vast range of libraries to aid in development.

#### Should

- Completed product should ship as a .exe file. This file type will run in windows, which most users around the world will be running. With a little configuration this type of file can also run on Linux and MacOS. It allows users to simple double click the file and have the program run, without worrying about installing python themselves.

- Use an appropriate API for weather data if this function is implemented. An API should be selected that allows current or at least hourly weather data to be gathered either for free or for a low cost.

- Have a fast runtime. As this software is designed to emulate a real-world system, it is vital that the time taken to create a schedule is as fast as possible. If the system takes several hours to produce a solution, it would not be viable for real world use.

#### Could

- Use scikit-learn library for clustering algorithms. The library provides a range of algorithms, such as both that “kMeans” and “affinity propagation”, which are detailed in the literature review. Developing a clustering algorithm ourselves is possible, however it is not the aim of this project to do.

- Use pyeasyga for the genetic algorithm. This is another free library for python. This one implements the Genetic Algorithm. As with clustering, the task that this project is aiming to solve is not whether this algorithm can be implemented, thus using a library is a good time-saving method.

- Could use PyQt to produce a GUI. PyQt is a powerful tool for creating attractive GUIs. It will allow a clean and simple user interface to be developed quickly.

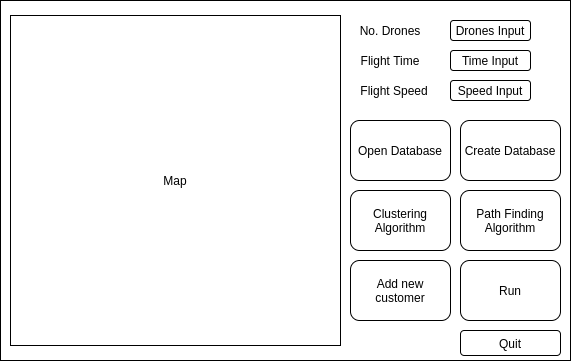
## Key Components

### Graphic User Interface (GUI)

The graphic user interface is a key part of this project. It serves to both allow input and show resulting output. A simple layout was desired so that anyone would be able to pick up and use the app without having to read instructions. Ideally, the GUI will be system independent, with its graphics being taken from the system defaults. This will ensure that the application is as cross-platform as possible.

Clearly labelled text inputs, buttons and drop-down boxes should be used where appropriate to create a clean interface.

The final GUI design is shown in Figure 12.



**Figure 12 – Graphic User Interface Design**

### Clustering

Clustering is the second main component of the project. The clustering algorithm is responsible for breaking down the customers into manageable groups. For this task there are a range of algorithms available. The solution that is implemented should be able to group customers by their proximity to one another in a reasonable timeframe. A range of algorithms could be implemented to allow comparison between their effectiveness.

### Weather Data

I would like to account for the weather with my solution. For this I will select a suitable API that will give me various pieces of data. The key two factors I wish to include are wind speed and wind direction. This will allow me to create a more realistic solution. We will also be able to use the data to improve our search algorithms to use the weather to their advantage wherever possible.

### Path Finding

Path finding is the final major step in our project. Once the data has been broken into clusters, we can attempt to find a route through the customers in each cluster. Weather data is considered while creating routes. The effect of wind speed and bearing on flight time is used to optimise the routes and decrease the total time taken.

### Time of each leg

To calculate the time of each leg of the routes, we must use several calculations.

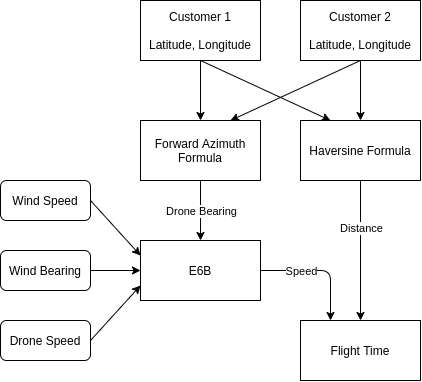
First, we need to find the distance between two sets of coordinates. For this we will use the Haversine formula, which calculates the great-circle distance between two latitude longitude pairs. The great circle is the distance accounting for the curvature of the earth. Use of this formula allows greater accuracy than doing a simple calculation using Pythagoras.

The next piece of data needed is the bearing between the two locations. There is a formula which gives the us the bearing known as the forward azimuth. As we travel along a great-circle arc, the angle we are travelling at changes slightly. The forward azimuth is the bearing required when we account for the great-circle arc.

We can use the data gathered to conduct two further calculations. The calculations are the same that are performed by pilots using device called an E6B. This device is used to calculate the required flight bearing and corrected speed accounting for wind.

Using the drone speed and desired bearing and the wind speed and bearing we can calculate the corrected bearing. Once we have this angle, we can use it in the next stage to calculate the actual ground speed of the drone when accounting for the wind.

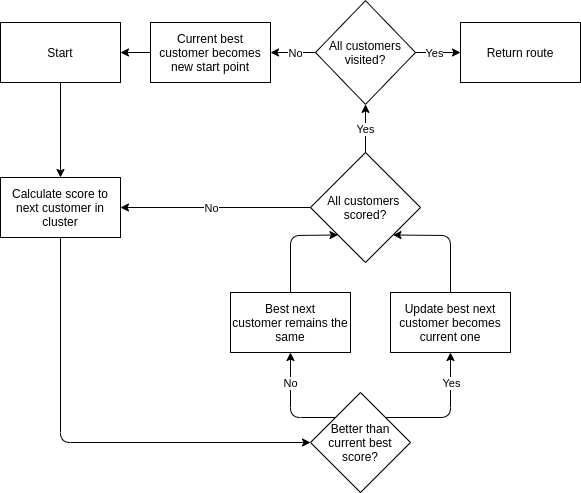
Finally, we will use the great-circle length and E6B corrected flight speed to calculate the time of each leg of the journey. This series of calculations is crucial to optimising the path-finding algorithms in the next stage.



**Figure 13 – Calculating the flight time between two customers**

### Greedy Best First

The basic flow of the algorithm is to calculate the time taken to get from the current location to each other location that is unvisited and pick the one with the smallest corresponding time. The process repeats until there are no more unvisited locations, at which point it returns to the depot. Figure 14 outlines the architecture of the algorithm. This should be implemented and available as an option through the GUI.

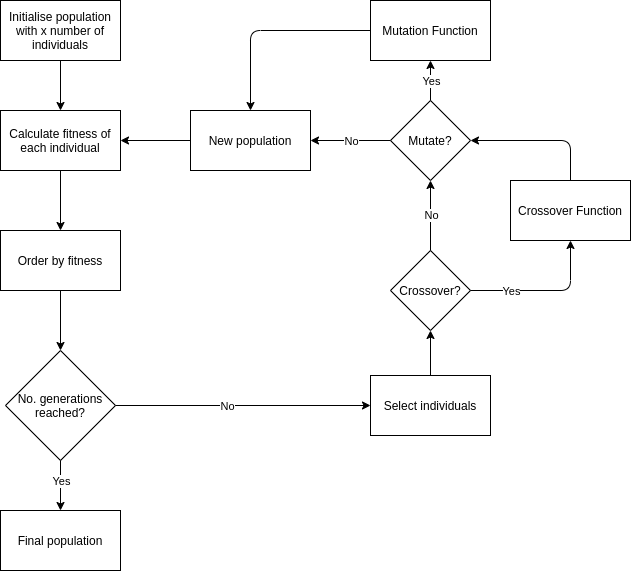


**Figure 14 – Architecture of the Greedy Best First algorithm**

### Genetic Algorithm

The second algorithm to be implemented is the genetic algorithm. It is more complex than greedy best first and a third-party library may be required to speed along development. Figure 15 outlines how the algorithm manipulates the data to attempt to improve the solution.

As with greedy best first, this should be available as an option in the GUI.



**Figure 15 – Architecture of the Genetic Algorithm**

### Database

The database should be simple. One table will be required which will outline customers alongside their coordinates and details of their order, such as item name and order time.

For the purposes of this application, there is no need to store in-depth information about users, nor have a separate table for users as I will not be implementing a login in system of any kind. The scope of the project is just to simulate a real system, thus more data than outlined is not necessary. A sample table layout is shown in figure 16.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Latitude** | **Longitude** | **Item\_name** | **Timestamp** |
| 1 | 1.234 | 5.678 | Mobile Phone | 05/05/20 - 15:28 |

**Figure 16 – Database table design**

### Output

The final key component of the program is the output. I intend to have two forms of output:

1 – The map shown on the GUI in Figure 12. This will be a real map and once the program is run, there should be clearly labelled customers, clusters and routes marked on.

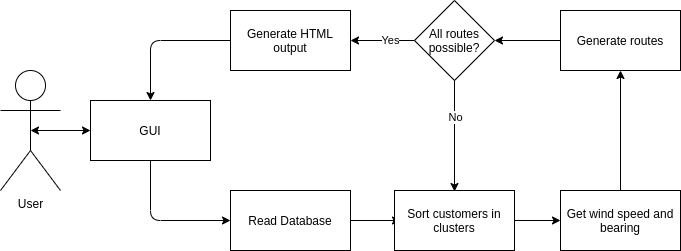
2 – A separate file outlining some of the key parameters used in the program such as inputs, search algorithm and clustering algorithm. Here there should also be the actual schedule that outlines which drone will go to which customer.

Architecture

## System Architecture

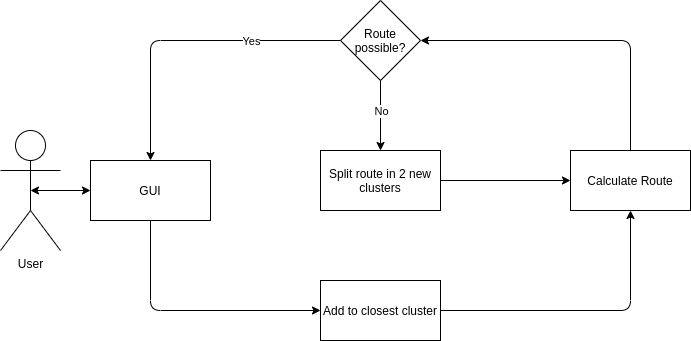
These components are brought together to create a simple program which can take user input of customer locations, parameters of number of drones and flight time/speed, as well as which path finding algorithm they would like. Once the program is run, it follows a simple workflow to solve the problem and display the resulting clusters and routes to the user.

The architecture of the main program functionality is shown in Figure 17.



**Figure 17 – Main program loop**

Once the applications main function has run, the user may need to input a new customer. To save processing power and time, the new customer is appended to the closest existing cluster, and a new route is generated for this cluster only. The architecture of this function is detailed in Figure 18.



**Figure 18 – Adding a new user after clustering and path finding**

## Development Methodology

The project development has followed the agile methodology. When followed, agile allows the software to be iteratively designed, tested, and evaluated. The requirements are created to produce the requirements specification. A design is created based on these requirements and an implementation is created based on the design. This implementation is then tested and evaluated against the requirements. If there are bugs or improvements to be made, the process begins again.

Throughout development, unit tests are written to ensure that every component works correctly on its own. An isolated test is written before the component is integrated with others. Once complete, these are commented out to not use unnecessary resources when developing other parts of the project, however they can always be revisited later for further testing. Once unit testing is complete, the agile process begins back at the design stage for any modifications or further functions.

# Implementation

Several freely available software libraries were used. Libraries were used to aid development speed and to ensure certain algorithms were correctly implemented.

Care has been taken to follow the design outlined in Section 4. Some changes were made throughout the implementation stage due to the agile methodology I used, however most features remained as specified there.

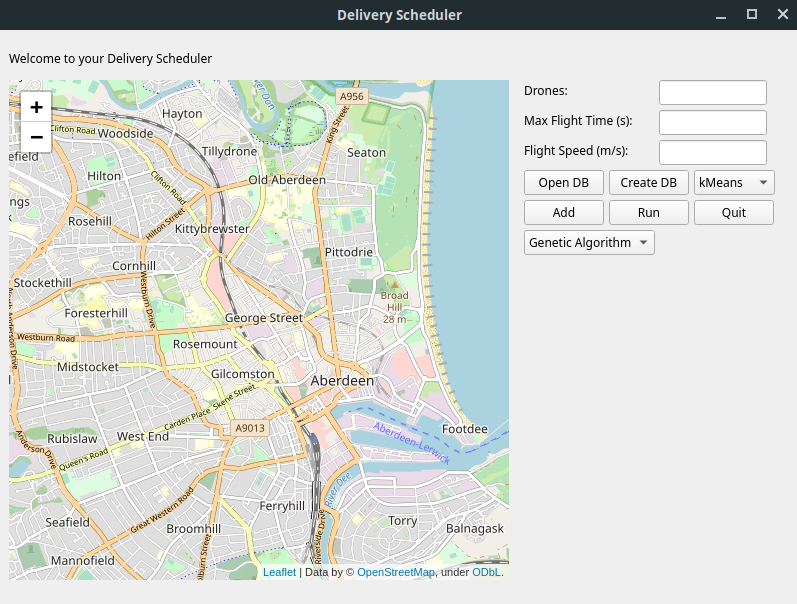
## GUI

The GUI was implemented using the PyQt5 library created by Riverbank Computing Limited: <https://pypi.org/project/PyQt5/>

The is library was selected due to its ease of use and extensive documentation. It allows complete control over the layout of the GUI and has many ‘widgets’ available with just a few lines of code, such as buttons, drop-down boxes and embedded HTML.

Alternative libraries were explored; however, they were universally found not to provide the same level of control over the design that PyQt5 offers.

Figure 19 shows the GUI that has been implemented. It follows the intended design, with the only changes being to the size of buttons on the right-hand side of the interface.



**Figure 19 – The implemented GUI**

## Clustering

Two clustering algorithms have been implemented and are available to select through the GUI from a drop-down box.

### Affinity Propagation

Early on during the design and implementation of the project I believed that this algorithm would be the most appropriate for clustering. It is different from other algorithms in that it does not take a user input of the desired number of clusters. The algorithm creates however many clusters are necessary depending on some other parameters such as how similar they should be to one another. During testing it was realised that this algorithm wasn’t appropriate for the project, and as the option to select it has since been removed.

The algorithm was implemented with help from the scikit-learn library: <https://scikit-learn.org/stable/about.html>

This library contains a range of algorithms for various machine learning tasks and is freely available.

### kMeans

kMeans was the second clustering algorithm implemented. It has been discovered to be more practical for this project, as users are likely to know how many drones they have/wish to have. The number of drones is taken as an input from the GUI. This number of clusters are then created, with customers being grouped by proximity to one another.

The kMeans algorithm is used in the first stage of the program’s main functionality. It also is used when adding a new customer after the program has been run.

As with affinity propagation, the algorithm was implemented with help from the scikit-learn library.

## Wind Data

In order to account for wind I needed an API that would give both wind speed and direction. For this I have used the darksky API, which offers up to 1000 free API calls per 24 hours:

<https://darksky.net/dev>

There is a wealth of data returned by the API, and as an input it only needs a latitude and longitude pair.

I am only using the current wind speed and bearing at the time of the call. However, in future expansion of the project we could easily use more, such as precipitation levels and predicted weather for the coming hours and days. This would allow us to predict ‘no-fly’ times and adjust delivery times for customers accordingly.

## Generating Routes

### Haversine + Azimuth + E6B

As outlined in the Design section, this is a multi-section process. Figure 13 outlines how the data should be manipulated to complete the task.

The Haversine formula was implemented using a python library. As an input all we need to give is the latitude and longitude of the start and end point, alongside the desired unit we want the answer returned in. The library is available here:

<https://pypi.org/project/haversine/>

For calculating the forward azimuth, I was unable to find any suitable libraries, so implemented this myself. The built-in maths module for Python gave us the functionality required. Care was taken during testing to ensure that the results given by the solution implemented are accurate.

The E6B was also implemented entirely by me, with assistance from the maths module. As with the forward azimuth calculation, extensive testing was conducted to ensure the algorithms had been coded correctly.

### Genetic Algorithm

For the genetic algorithm, the pyeasyga library was used to aid development:

<https://pyeasyga.readthedocs.io/en/latest/readme.html>

The library provides a class that defines the flow of the algorithm. From this I was able to quickly create a working algorithm by defining how the code should be manipulated at each stage, without spending additional time testing that everything was done in the correct order. See diagram for flow chart of the algorithm.

- Creation of individuals is done by taking the latitude and longitude pairs of customers in each cluster and shuffling them to produce a random route.

- Crossover is done by randomly selecting a position and swapping data from either side of two individuals to create two new children.

- Mutation randomly selects two customers in a route and swaps their position in the route. For example, if the parent route goes 1 – 2 – 3 – 4 – 5 and customer 2 and 4 are selected, the resulting child route would be 1 – 4 – 3 – 2 – 5.

- Selection is done by taking a random sample of 10% of the population. The individuals in the sample are ranked according to the fitness function and the best individual is selected.

- The fitness function calculates the total length of the route and from there the total time taken for a drone to complete the route. The time is the score of each route, rather than distance. This is the case since wind is taken into consideration. Drones have a set amount of time they can fly; however, the distance is in part dictated by external factors such as the wind speed and bearing relative to the drone.

### Greedy Best First

I searched for a library to aid with implementing the greedy best first search algorithm, however, was not able to find one. As the algorithm is reasonably simple, I opted to implement it myself from scratch. This was a reasonably quick process, and additional testing was done to ensure that the algorithm is working correctly.

The algorithm uses a heuristic to decide which is the next closest customer the drone can visit. As with the genetic algorithm, the heuristic is based on flight time of the drone, which considers the wind speed and bearing.

## Database

SQLite was selected as the database used to store details of customer locations and item names. I opted to use SQLite over a noSQL database type due to the simple schema the data needed to adhere to. One table within a database is all that was required and is unlikely to change in future versions. SQLite unlike most other SQL databases does not have a seperate server process, it reads and writes directly to ordinary files. It is also largely cross-platform and completely free to use.

Python ships with library for reading and modifying sqlite databases, this was used for the implementation of this project.

## Output

### Folium

The drawing of the map is done with Folium. Folium allows data to be drawn on a real map. In our case we can have latitude longitude pairs with associated item names. This data can be used to place markers on the map at those exact locations. Once routes have been calculated we are able to draw lines on the map to represent the path of each drone. The map is saved as a regular HTML file.

### BeautifulSoup

BeautifulSoup is used to edit HTML files. With BeautifulSoup we are able to read the HTML file generated by Folium and modify the body of the file to include data from elsewhere in the application, such as the number of drones and the number of routes required.

## Challenges

### Impossible Clusters

There are instances where the path finding algorithms are unable to find a possible route through the customers in a cluster. This can be caused by the cluster simply being too large, or by strong winds slowing the progress of each drone.

To solve the problem, I opted to use kMeans to split any impossible clusters in two. Once the two smaller clusters are created, the selected path finding algorithm again attempts to create routes. This is an iterative process which will continue until the route is direct from the depot to the customer. If it is still not possible the process ends, and the user is informed via the HTML output file.

There is still room for improvement with this function, as the smaller clusters are all maintained. If for example just one customer is too far away for delivery, and they were a part of a cluster of 15 others originally, the original cluster would be broken down into up to 6 smaller clusters. There is no attempt to combine these smaller clusters again, even if it is just one customer who is causing the route to be too long. This could lead to wasted time as routes are generated where drones serve just one or two customers and have plenty of battery life left.

### Weather API

Originally I was using the Openweathermap API for collecting weather data (<https://openweathermap.org/api>). Unfortunately, there were occasions where the API would not return either the wind speed or bearing. After troubleshooting I could not find a solution and so decided to switch API to Darksky.

This swap required some modifications to the code due to the data returned being structured differently. It also will lead to problems down the line, as Darksky have recently announced that they have joined Apple. The result of this is that their API will likely cease to exist after 2021 (<https://blog.darksky.net/>). This does give some time to find a suitable alternative, but changes will have to be made to the code in future.

### Limitations of Folium

When I began the implementation I had intentions of allowing users to easily add customers by clicking a location on the map, and then an add button from the pop-up which would bring them to the add dialog box and have the latitude and longitude details already filled in.

Unfortunately, Folium does not include such functionality out of the box. In future iterations I will work to add this feature, however it would require custom Javascript to be written and added to the HTML file. As I am relatively new to Folium and its parent library leaflet.js, I did not want to spend time familiarising myself with their code and how I would implement what I wanted. Because of this, I opted not to implement this feature, as it is not completely essential.

## Implementation Methodology

As described in the design methodology section, the agile methodology has been used for this project. Using the requirements specification, an initial design was created for the application. This bare-bones version was implemented to cover just the “must have” section of the requirement specification.

The process then iterated to include more of the “should have” functionality, as well as refining and optimising the “must have” functions. As this process iterated, the design was modified depending on obstacles discovered during implementation.

Throughout development the application has been tested. Each new function added to the program has been tested extensively to ensure its correctness.

The application was managed using a git repository. Each commit to the repository adds or improves functionality of the application and is given a description of the change along with a timestamp. See section 9.2 for the full git project log.

# Testing

The purpose of the testing is to allow an accurate and detailed evaluation of the application. It should ensure that all the different functions of the program are working correctly and as expected. Finally, tests should be conducted to allow us to compare the suitability of the two implemented path finding algorithms.

Various methods of testing have been used to ensure that these targets are all met to allow us to evaluate effectively.

The tests were performed on the “aberdeen” dataset (section 9.4). The drone maximum flight time and speed have been set at 900s and 15m/s respectively unless otherwise stated.

Testing was performed on a machine running Linux Mint version 19.1, using an Intel i5 @ 3.30GHz and 16GB DDR3 RAM @ 1600 MT/s. The same machine has been used throughout to ensure we can fairly compare all our test results.

This section will detail the testing that has been performed and show the results gathered.

## Acceptance Testing

Acceptance testing is a crucial part of the project. This testing pertains to the requirements specification. It is a simple check to ensure that crucial functionality has been completed.

### Must Have

All the “must have” functions have been completed. These are crucial to the application and it could not function without them. Because of this, I set out to complete these early on in development.

### Should Have

Of the “should have” requirements, four of six have been completed fully, and one of the two incomplete requirements is partially complete. These are important features but non-essential.

The partially complete requirement is the interactive user input. I wanted to have the application running and allow users to be able to add, remove and edit an order while the system was running. It has partially been completed as although the program does not have a state where it is running and updating constantly, when clusters and routes have been found and you add a new customer, everything doesn’t start again. The program can append the new customer to the closest cluster and find a route for just the updated cluster.

The completely incomplete requirement is that the software is not delivered as a .exe file. I wanted to be able to distribute the solution as a .exe file so that users would not have to worry about installing python and maintaining an architecture. As it stands, there are 11 python files which are all required to stay together in the same directory to use the application. Additionally, end users would currently have to install the third-party libraries themselves.

This requirement was not met as the third-party library I have attempted to use to create the .exe file does not appear to create a valid file on my machine. I suspect this is because I am using Linux as my operating system, which does not natively support the .exe file type. With future development I will look to solve this issue.

### Could Have

Both requirements pertain to providing a more interesting graphical output for the user. Ideally, I wanted to show how the routes changed as they were generated, as I felt this would be interesting for the user to see. Additionally, I wanted to show where drones were along their routes, which would rely on the partially met ‘should have’ requirement being met.

Because of time restraints and the non-essential nature of these requirements, they have not been met.

## Optimising Genetic Algorithm

Tests were conducted of the effect changing parameters of the genetic algorithm on performance of the algorithm. I wanted to find a set of parameters that give good results for the problem.

For this I ran a set of tests with varied population size and number of generations to find a good balance of run time and results. This testing was performed on the same data set split into various amounts of clusters. The number of generations and population sizes tested are every combination of 50, 150 and 300 of each, and the number of clusters tested on were 3, 5 and 7.

The main code of the software was modified to automate this testing. A loop was added to iterate through the different generation number settings. For each iteration of this, the population sizes were iterated through. This process of automation ensured the tests were completed in as timely a manner as possible.

The resulting data can be found in section 9.3.1.

## Variation in GA results

As there is variation in the genetic algorithm results, I have conducted tests on the data, again with 3,5 and 7 clusters. I ran the algorithm on each test 5 times to see if doing so would be a method of providing a better result.

## Genetic Algorithm vs Greedy Best First

In this section I will detail the tests conducted to allow comparison of the genetic algorithm and the greedy best first search algorithm. For these tests, the genetic algorithm had a population size of 150 and ran for 150 generations.

### Route length, time and run times

I ran a series of tests on the dataset using both greedy best first and genetic algorithm. The purpose of this test was to compare how well the two algorithms can produce routes depending on the route length. The number of routes that tests were performed on range for 1 to 15.

While these tests were being performed, I recorded the time taken to complete all routes for each iteration of the program.

Again, the process was automated as far as possible to save time and ensure accuracy.

## Unit Testing

Where appropriate, unit tests have been written ensure that functionality is as expected. These are done as small add-ons outside of the class in each file. The tests are written to take an input and print the result to the console. When not in use the tests are commented out, to ensure time is not wasted when we are using another part of the program.

They are designed to allow a range of inputs to easily be tested on to ensure each function works properly, testing normal, extreme, and exceptional data.

Results are evaluated as far as possible, however with some functions such as the genetic algorithm and clustering algorithms, it is not reasonable to produce exact expected results. In these cases, results are evaluated to ensure that a complete route is formed, or that the correct number of clusters are created. These functions also have the advantage of coming from trusted third-party libraries, which are effectively crowd-tested. If any other user noticed errors in their testing, they could report to the author and the issue should be fixed.

# Evaluation

## Overview

The project and resulting software application can be considered a success. All the essential functionality has been implemented, with only a few non-essential requirements not fulfilled. The design of the application is sensible and has produced an intuitive GUI. The program can take input of customers and their locations and produce realistic and well-optimised routes for drones to follow to deliver to them.

With that said, it can by no means be considered a finished product.

## Design

The design is good overall. It is highly modular which allows different functions to be modified and tested independently of each other. This means that if something does go wrong, we can easily isolate and fix the problem without interfering with other bits of code. The workflow from the user giving some input, to the user receiving the results back to the GUI is sensible and combines most of the main functionality of the program into a single loop.

However, the design was lacking in areas. Particularly the output file is very bare bones. It was added late on in the projects lifetime which has prevented any styling being added to it. Any actual user of the program would surely expect to see something much prettier.

Additionally, there was no thought given to optimising the schedule as shown in the html output file. Currently if there are more routes than drones, the drones are just systematically assigned the next route in the output queue. This was a massive oversight, as for most of the project I was focussed on creating good routes and clusters. I did not consider that once I had these routes, they had to be assigned in an optimised fashion to the drones. This would have to be implemented in future iterations of the project.

All things considered, the design that was produced is solid. However more time should have been spent on the design process to ensure that no features were missed.

## Implementation

The implementation closely follows the design and during the agile process has also informed the design. There were many challenges and some failures during the implementation, however all the crucial functional and non-functional requirements have been achieved.

The implementation successfully provides a simple GUI for a user to interact with. From the GUI they can control all functionality of the program, including selecting different path finding algorithms and parameters for their drones. It allows users to manage multiple databases of customers, which could be used for modelling orders due on different days for example.

The way the path-finding algorithms have been coded allow the optimisation functions to be modified later. One example of a potentially desirable feature is optimising so people who ordered first are placed near the front of the flight path. Within the Genetic Algorithm this feature was implemented, however has been commented out as I deemed in non-essential for the time being. It is a simple process to add more methods of evaluating routes and have them work in harmony.

Given extra time there would be further improvements. The readability of the code is poor in places. This in part comes from rushing into the implementation without properly planning and designing beforehand. Time should be taken to go back and refactor the code to ensure it is more easily maintained in future.

There are some pieces of functionality that would be nice to have but due to time constraints were not implemented. For a finished product, it would ideally be running round the clock. For a large company such as Amazon, they likely would prefer a system which tracks where drones are along their routes. This would work alongside a more realistic queue where customers are added when their order is placed, and once their item has been received, they are removed. It would allow more accurate time predictions, giving an actual time rather than the number of seconds after the drone leaves the depot.

## Test Results

### Optimising Genetic Algorithm

Through the tests conducted on the genetic algorithm, I can draw several conclusions:

1 – There is no clear correlation between quality of route and the settings of population size and number of generations.

2 – Time taken to create routes is directly linked to the population size and number of generations

3 – A combination of population size 50 and number of generations 150 consistently produces the best or near best results of all the test data

4 – Often, population size 50 and number of generations 50 was able to produce the best results, and always with the fasted time.

I had originally predicted that the quality of route produced would increase the more generations and population size, however this is not the case. This is a result of the crossover and mutation functions within the genetic algorithm. The more generations there are, the more chance there is that a good solution will change for the worse. Limiting these parameters allows a good solution to be found in a reasonable time frame.

Alternatively, a convergence function could be developed in future iterations of the algorithm. This would ensure that if several of the individuals in the population were the same, the algorithm would cease. This ensures that if several individuals have consensus on a good route, the route is retained. It also would also have the added benefit of potentially reducing runtime.

### Variation in GA results

Due to the nature of the genetic algorithm, it is possible to obtain a different result each time the algorithm is used. This is particularly prominent when the number of possible routes is very high. Because of this, another optimisation technique is to run the algorithm several times with the same data, and then select the best route.

This testing was carried out on the dataset with 3,5 and 7 clusters. The results can be found in section 9.3.2. The results show that in most test cases, the genetic algorithm produced a range of results. Many of these are insignificant, with only a few seconds difference. However as seen in section 9.3.2.4 route 3, the algorithm produced a minimum route time of 581.6s and a maximum of 716.2s, a difference of over 2 minutes.

When we imagine that the system might be running around the clock for years and using hundreds of drones, running the genetic algorithm several times is clearly a worthwhile endeavour.

### Genetic Algorithm vs Greedy Best First

Now that we have good settings for the genetic algorithm, we can begin comparison with the greedy best first algorithm.

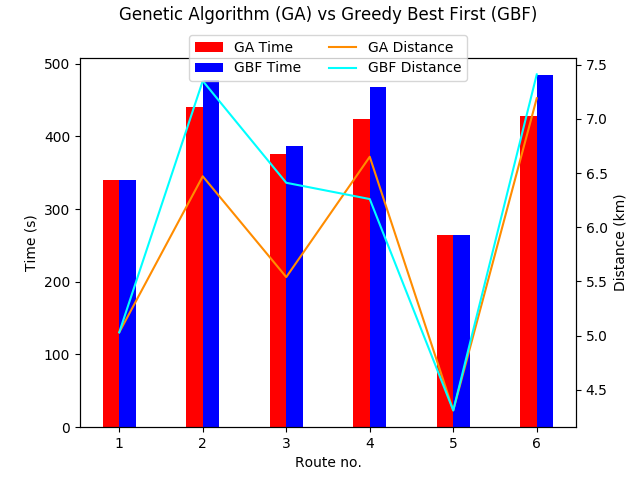
#### Route length and time

The data was split into a range of clusters from 1 to 15. For each of these splits, route lengths and times of both algorithms have been collected. What the testing shows is that the genetic algorithm is generally better at solving the problem.

Most of the time, the genetic algorithm produces a route that is shorter both by time and distance than the greedy best first algorithm. Between the tests of 2 and 7 clusters, the genetic algorithm produced an equal or shorter route by time in every case but 1.

An example of the results is shown in Figure 20. The left y axis shows the route lengths in seconds, while the right shows the distances in kilometres. The graph shows us that in four of six routes, the genetic algorithm produces a shorter route by time, with the remaining two being equal.

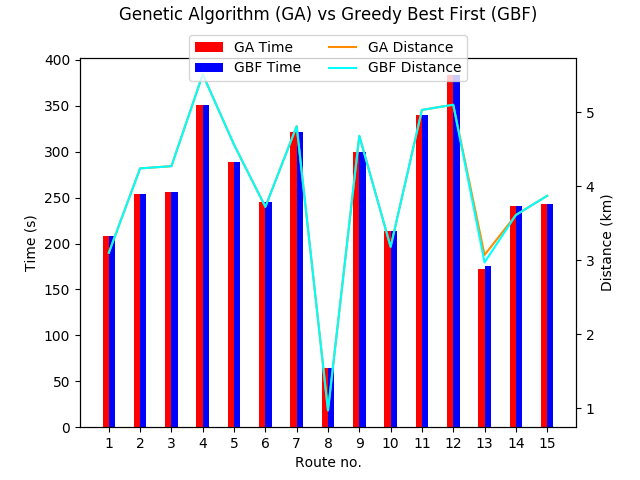
Interestingly there are several cases where the genetic algorithm was able to produce a route that was shorter in time, but longer in distance. This is advantage is caused by the genetic algorithms ability to explore more solutions than greedy best first. The genetic algorithm can explore many more potential routes, allowing it to find a route that uses the wind to its advantage for longer. This is again shown in figure 20, where, in route 4 the flight distance in is higher but the flight time is lower for the genetic algorithm.



**Figure 20 – Comparison of 6 routes created by the search algorithms**

The advantage of the genetic algorithm drops the smaller the routes become. When testing with the data split into 15 clusters, the genetic algorithm only managed to improve one route over greedy best first. This data is shown in Figure 21. When compared to figure 20, we can see that there is a clear decrease in the advantage of the genetic algorithm.

As the number of clusters increases, the number of customers per cluster decreases. In many cases there are only 1 or 2 customers per cluster, which vastly increases the likelihood of greedy best first finding an optimal route.



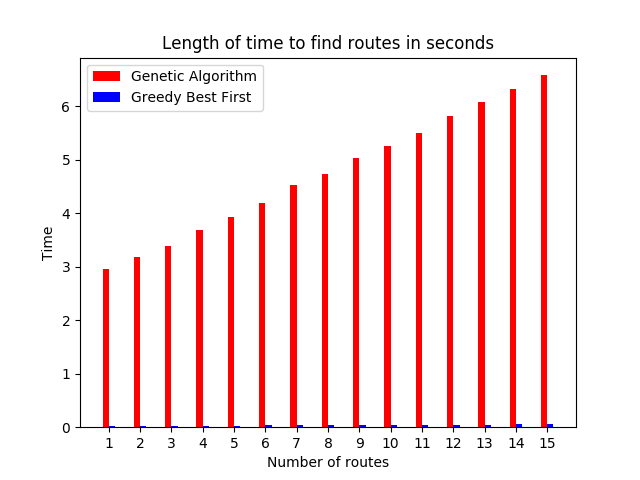
**Figure 21 – Comparison of 15 routes created by the search algorithms**

#### Performance

When testing for runtime of the two algorithms, greedy best first takes the lead. Figure 22 shows the results of the tests detailed above. The genetic algorithm takes around 3 seconds to find one route and requires roughly 0.25s additional seconds per additional route created.

Greedy best first has a much faster runtime. Starting at just 0.02s for 1 cluster and having a maximum of 0.06s for 15 clusters, it shows a massive advantage in this respect over the genetic algorithm.

The faster runtime is a result of the simplicity of greedy best first. It will only calculate the route once and stops immediately when this is complete. The genetic algorithm with the settings used for these tests must perform the same calculation 7500 times, along with additional operations such as swapping route positions in the population, crossover, and mutation.



**Figure 22 – Comparison of performance of the Genetic Algorithm and Greedy Best First search algorithms**

# Conclusion

## Overview

The following objectives were outlined in section 2.1:

- Implement a simple Graphic User Interface (GUI)

- Develop a system to generate routes between customers

- Present the results graphically

- Allow comparison of different algorithms

Research was carried out on a range of literature, and a technical specification was created to fulfil the objectives. From this specification an initial design was created, which then lead to the implementation. Following several iterations of the agile development methodology, all critical requirements and most important requirements have been met.

The main aim of the project was to create a drone delivery scheduler. Through completion of the requirements generated from this aim, a scheduler application has been created. The application produced allows users to graphically input customers and locations, control some parameters and be graphically shown the results. Extensive testing has been carried out to compare the path finding algorithms. The output is also displayed graphically to the user.

While there is extensive work that can be done in future to expand the project, the original aim and objectives have been fulfilled. Therefor the project can be considered a success.

## Findings

From the testing, I can conclude that for longer routes, the genetic algorithm is generally the better choice. It is usually able to produce a shorter route than the greedy best first algorithm. While it does usually produce shorter routes, it does some with an extended run-time, which becomes more prominent the more clusters there are. Additionally, the advantage shown by the genetic algorithm is lessened when working with smaller clusters of 1-3 customers.

The kMeans clustering algorithm has been shown to be an effective tool to aid in solving the problem. Without using such an algorithm, producing routes that are possible would be an insurmountable task.

## Future Work

There are many improvements that can be made to the software. Currently the software is quite locked down, with users being unable to select what city they would like to test on. There is no option to choose where the depot is, nor to have more than one depot. All these features could be implemented which would make the application relevant to many more users and businesses.

The user interface also requires further development. The design of it is very basic and uses simple buttons and drop-down buttons. Adding a new customer is clunky and somewhat unintuitive. To improve this, when the map is pressed there should be an ‘Add at this location’ button available.

There are optimisations to the search algorithms that should be addressed, such as adding a convergence function to improve the performance of the genetic algorithm. This would help to close the gap in performance between the two search algorithms.

The schedule that is produced and output is not optimised in any way. This was a massive oversight and is one of the main failures of the project. It will have to urgently be addressed in future iterations, potentially utilising another genetic algorithm to produce an efficient schedule.

The application has been an exciting endeavour and will be expanded in future. The results produced are accurate and based on real-time weather data for Aberdeen. As drones become more prominent as a delivery method, an expanded version of the application could become a useful tool for many businesses.

# Appendices

## Code

The code is listed on Github, however is on a private repository. Therefor I have included all the code files alongside this submission on moodle in the folder labelled “FULL CODE LISTING.zip”. To run the code, the file “qtGUI.py” must be run with Python v3.

## Project Log

The project has been completed using Github for version control and logging. For each completed piece of work, a commit is added to the repository. These commits contain a short comment and a timestamp.

Comment: Initial commit

Time: 2019-10-02 12:33:14Z

Comment: Initial setup. Project Proposal + Ethics + changelog files

Time: 2019-10-02 12:35:21Z

Comment: Lit review Intro & research on search algorithm

Time: 2019-10-18 10:27:39Z

Comment: Initial testing. Using easyga library + scikit for clustering

Time: 2019-11-15 14:25:39Z

Comment: Formatting for affinitypropagation.py

Time: 2019-11-15 14:31:09Z

Comment: GA + AP, gplot, GUI testing

Time: 2020-01-03 10:30:38Z

Comment: Testing HTML map

Time: 2020-01-03 13:44:00Z

Comment: HTML rendering working within qt app. Works with GA + AP

Time: 2020-01-03 15:21:09Z

Comment: Modified AP + GA + GUI code to use classes for a neater main file

Time: 2020-01-04 12:55:19Z

Comment: Add dialog created. Accepts just coords for now. Returns to main form

Time: 2020-01-04 13:55:10Z

Comment: Tidied up directory. Added initial DB setup

Time: 2020-01-04 14:57:41Z

Comment: Add to DB function works from GUI

Time: 2020-01-04 15:16:34Z

Comment: DB works with GA + AP

Time: 2020-01-04 15:48:55Z

Comment: Folium made a class and working with DB

Time: 2020-01-04 16:34:12Z

Comment: Combined all functionality. Main GUI updates with GA+AP applied to HTML

Time: 2020-01-05 13:23:21Z

Comment: Tidied up some code. Added markers at customer locations

Time: 2020-01-05 16:01:02Z

Comment: Modified fitness function to account for time of order

Time: 2020-01-06 13:14:24Z

Comment: Modified main GUI to work with new fitness function

Time: 2020-01-06 14:29:50Z

Comment: Tidied up GUI. Added run, quit, open and save DB

Time: 2020-01-06 16:35:58Z

Comment: Tidied GUI code. Added latlngmarkers to map for finding coords

Time: 2020-01-07 12:56:22Z

Comment: Added some error handling to prevent db operations before opening a db

Time: 2020-01-07 14:07:49Z

Comment: Fixed bug where the last location wasn't routed to

Time: 2020-01-07 16:04:34Z

Comment: Fixed real route length finder only giving 1 leg of journey

Time: 2020-01-07 16:23:58Z

Comment: Removed time as a consideration in clustering

Time: 2020-01-07 16:43:01Z

Comment: Added wind direction into fitness function

Time: 2020-02-10 14:01:02Z

Comment: Real wind direction added from API. Config file creates key

Time: 2020-02-10 16:11:44Z

Comment: Properly account for affect of wind on the drones

Time: 2020-02-21 15:58:38Z

Comment: Edited for readibility and useability

Time: 2020-02-24 13:14:29Z

Comment: E6B added to main program. Fitness now uses time from d=vt

Time: 2020-02-24 16:27:16Z

Comment: Coded own bearing finder to fix on laptop. Implemented to GUI

Time: 2020-02-24 18:08:00Z

Comment: Commented line to finalise laptop fix

Time: 2020-02-24 18:11:09Z

Comment: Updated API to use DarkSky. OpenWeatherMap failing to return wind degree

Time: 2020-02-25 12:00:55Z

Comment: Different colours for each route added

Time: 2020-02-25 12:08:47Z

Comment: Formatted console output. Removed prints causing long run time

Time: 2020-02-25 12:29:56Z

Comment: Laptop temp

Time: 2020-02-25 16:29:34Z

Comment: Playing with parameters

Time: 2020-02-25 16:43:39Z

Comment: Merge branch 'implementation' of https://github.com/DessyT/Drone-Delivery-Scheduling into implementation

Time: 2020-02-26 10:54:44Z

Comment: Fixed parameters

Time: 2020-02-26 10:56:54Z

Comment: Added kmeans class

Time: 2020-02-27 13:22:18Z

Comment: Merge branch 'implementation' of https://github.com/DessyT/Drone-Delivery-Scheduling into implementation

Time: 2020-02-27 13:22:28Z

Comment: kmeans running through gui

Time: 2020-02-27 13:29:13Z

Comment: kmeans input validation, error handling

Time: 2020-02-27 14:01:40Z

Comment: User can choose kmeans or AP through GUI

Time: 2020-02-27 14:15:05Z

Comment: Tidied error handling

Time: 2020-02-27 14:34:23Z

Comment: Tidied up UI

Time: 2020-02-28 15:54:37Z

Comment: Semi working recursion. Skips depot

Time: 2020-02-28 17:31:17Z

Comment: Merge branch 'implementation' of https://github.com/DessyT/Drone-Delivery-Scheduling into implementation

Time: 2020-03-01 20:43:31Z

Comment: Fixed merge markers

Time: 2020-03-01 20:46:35Z

Comment: Reverted to pre recursion state

Time: 2020-03-03 14:06:26Z

Comment: Implemented greedy best first search class

Time: 2020-03-05 13:47:42Z

Comment: Added code to main GUI

Time: 2020-03-05 14:34:22Z

Comment: Added dropdown box to choose which search we want

Time: 2020-03-05 14:39:53Z

Comment: Optimised parameters for GA. Added more locations

Time: 2020-03-05 14:55:13Z

Comment: Implemented max range of routes being verified

Time: 2020-03-11 17:31:30Z

Comment: Tidied up and allowed for user input max flight range

Time: 2020-03-11 17:41:38Z

Comment: Added get last item from db. kmeans initial new cluster function added

Time: 2020-03-13 13:52:50Z

Comment: Fixed silly list conversion bug

Time: 2020-03-13 13:55:30Z

Comment: Nearly working dynamic add functionality. Just provides wrong route

Time: 2020-03-13 14:56:17Z

Comment: Finds new route properly and shows on map. Still overlays on old map

Time: 2020-03-13 16:19:12Z

Comment: Broken for now, joins 2 routes together seemingly at random

Time: 2020-03-19 13:21:54Z

Comment: Problem was finding clusters after the item was added to db. Works woo!

Time: 2020-03-19 14:00:45Z

Comment: Tidied up some code. Routes sometimes didn't get depot added. Fixed.

Time: 2020-03-19 15:31:36Z

Comment: More code tidying

Time: 2020-03-19 16:15:17Z

Comment: Changed from max distance to max flight time. Gives more realistic outp.

Time: 2020-03-20 12:06:30Z

Comment: Added time heuristic to greedy best first

Time: 2020-03-20 12:48:19Z

Comment: Fixed program crash when adding a location before running

Time: 2020-03-24 12:31:24Z

Comment: Fixed GBF not working properly. Best leg wasn't properly held

Time: 2020-03-24 16:20:01Z

Comment: Input validation added for parameters

Time: 2020-03-24 17:46:38Z

Comment: some bug fixes

Time: 2020-04-07 14:21:15Z

Comment: Outputs time and dist to csv

Time: 2020-04-21 14:51:15Z

Comment: Fixed bug where changing DB didn't refresh the map

Time: 2020-04-24 10:31:11Z

Comment: Backing up documents + poster

Time: 2020-04-24 10:32:09Z

Comment: Added nice output to HTML file. Still needs some fixing

Time: 2020-04-24 14:59:49Z

Comment: Minor bugfix

Time: 2020-04-24 15:09:16Z

Comment: Fixed output so routes are grouped into however many drones there are.

Time: 2020-04-27 12:11:08Z

Comment: Outputs to table instead of text

Time: 2020-04-27 13:00:32Z

Comment: Styled table + fixed bug where it only ran once

Time: 2020-04-27 14:01:35Z

Comment: Minor bugfix

Time: 2020-04-27 14:53:25Z

Comment: Documents backup

Time: 2020-04-27 16:10:38Z

Comment: Fixed impossibleroutes not refreshing each run. bearing + w/e to out

Time: 2020-04-28 12:34:09Z

Comment: Presentation back up

Time: 2020-04-28 16:11:43Z

Comment: Final report structure

Time: 2020-04-30 11:13:20Z

Comment: Final writing

Time: 2020-04-30 12:07:42Z

Comment: Changed selection to tournament selection

Time: 2020-04-30 16:18:51Z

Comment: Report backup

Time: 2020-04-30 16:45:20Z

Comment: report backup

Time: 2020-05-01 10:51:18Z

Comment: First draught design complete

Time: 2020-05-0111:46:40Z

Comment: Added a bunch of diagrams. Implementation section completed

Time: 2020-05-05 15:28:20Z

Comment: Updated testing graphs python file. Generated a bunch of graphs

Time: 2020-05-06 14:45:16Z

Comment: Some testing complete. CSV output on GUI class added

Time: 2020-05-07 13:04:59Z

Comment: More testing, testing section written in report

Time: 2020-05-08 14:05:31Z

Comment: Tidied up directory

Time: 2020-05-11 14:48:10Z

Comment: Merge pull request #1 from DessyT/implementation

Implementation

Time: 2020-05-11 14:50:08Z

## Test results

### Finding optimal population size and no. generations

#### 3 Clusters

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Population Size | No. Generations | Route 1 | | Route 2 | | Route 3 | | Run time (s) |
|  |  | Distance(km) | Time  (s) | Distance(km) | Time  (s) | Distance(km) | Time  (s) |  |
| 50 | 50 | 6.88 | 474.72 | 10.03 | 671.52 | 9.94 | 631.5 | 1.16 |
| 150 | 50 | 6.88 | 474.72 | 10.03 | 671.52 | 9.49 | 585.18 | 3.72 |
| 300 | 50 | 6.88 | 474.72 | 10.03 | 671.52 | 9.49 | 585.18 | 7.9 |
| 50 | 150 | 6.88 | 499.48 | 10.4 | 699.65 | 9.49 | 585.18 | 3.52 |
| 150 | 150 | 6.88 | 499.48 | 10.03 | 671.52 | 11.2 | 688.61 | 11.11 |
| 300 | 150 | 6.88 | 499.48 | 11.05 | 717.1 | 10.25 | 700.34 | 23.82 |
| 50 | 300 | 6.88 | 499.48 | 10.03 | 671.52 | 9.69 | 594.61 | 6.99 |
| 150 | 300 | 6.88 | 474.72 | 10.03 | 671.52 | 10.25 | 700.34 | 22.31 |
| 300 | 300 | 6.88 | 474.72 | 11.05 | 717.1 | 11.5 | 711.21 | 47.76 |

Best Times

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Route | Distance | Time | Run Time | Pop Size | No. Gens |
| 1 | 6.88 | 474.72 | 1.16 | 50 | 50 |
| 2 | 10.03 | 671.52 | 1.16 | 50 | 50 |
| 3 | 9.49 | 585.15 | 3.52 | 50 | 150 |

#### 5 Clusters

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Population Size | No. Generations | Route 1 | | Route 2 | | Route 3 | | Route 4 | | Route 5 | | Run time |
|  |  | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Time  (s) |
| 50 | 50 | 6.7 | 404.03 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.72 | 544.23 | 1.34 |
| 150 | 50 | 5.54 | 348.73 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 9.11 | 559.83 | 4.32 |
| 300 | 50 | 5.54 | 348.73 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.68 | 541.22 | 9.49 |
| 50 | 150 | 5.54 | 348.73 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.68 | 541.22 | 4 |
| 150 | 150 | 5.54 | 348.73 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.95 | 556.4 | 13.06 |
| 300 | 150 | 5.54 | 348.73 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.72 | 544.23 | 28.63 |
| 50 | 300 | 5.84 | 383.1 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.72 | 544.23 | 8.08 |
| 150 | 300 | 5.54 | 348.73 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.72 | 544.23 | 26.07 |
| 300 | 300 | 5.54 | 348.73 | 6.26 | 408.22 | 4.31 | 287.36 | 6.37 | 408.3 | 8.68 | 541.22 | 57.73 |

Best Times:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Route | Distance | Time | Run Time | Pop Size | No. Gens |
| 1 | 5.54 | 348.73 | 4 | 50 | 150 |
| 2 | 6.26 | 408.2 | 1.34 | 50 | 50 |
| 3 | 4.31 | 287.3 | 1.34 | 50 | 50 |
| 4 | 6.37 | 408.3 | 1.34 | 50 | 50 |
| 5 | 8.68 | 541.2 | 4 | 50 | 150 |

#### 7 clusters

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pop Size | No. Gens | Route 1 | | Route 2 | | Route 3 | | Route 4 | | Route 5 | | Route 6 | | Route 7 | | Run time |
|  |  | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) |  |
| 50 | 50 | 6.7 | 406.54 | 5.71 | 373.8 | 5.34 | 307.6 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 1.51 |
| 150 | 50 | 6.7 | 406.54 | 5.71 | 373.8 | 5.34 | 307.6 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 5.06 |
| 300 | 50 | 6.27 | 374.02 | 5.71 | 373.8 | 4.52 | 306.5 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 11.31 |
| 50 | 150 | 6.7 | 406.54 | 6.0 | 373.3 | 4.52 | 306.5 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 4.6 |
| 150 | 150 | 6.27 | 374.02 | 6.0 | 373.3 | 4.52 | 306.5 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 15.17 |
| 300 | 150 | 6.7 | 406.54 | 6.0 | 373.3 | 4.52 | 306.5 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 34.21 |
| 50 | 300 | 6.7 | 406.54 | 5.71 | 373.8 | 4.52 | 306.5 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 9.3 |
| 150 | 300 | 6.27 | 374.02 | 5.71 | 373.8 | 5.34 | 307.6 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 30.8 |
| 300 | 300 | 6.27 | 374.02 | 6.0 | 373.3 | 4.52 | 306.5 | 4.31 | 282.3 | 6.14 | 381.6 | 5.0 | 343.3 | 4.82 | 306.2 | 68.14 |

Best Times:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Route | Distance | Time | Run Time | Pop Size | No. Gens |
| 1 | 6.27 | 374.02 | 11.31 | 300 | 50 |
| 2 | 6.0 | 373.3 | 4.6 | 50 | 150 |
| 3 | 4.52 | 306.5 | 4.6 | 50 | 150 |
| 4 | 4.31 | 282.3 | 1.51 | 50 | 50 |
| 5 | 6.14 | 381.6 | 1.51 | 50 | 50 |
| 6 | 5.0 | 343.3 | 1.51 | 50 | 50 |
| 7 | 4.82 | 306.2 | 1.51 | 50 | 50 |

### Multiple runs on same data

#### 50 x 50 Genetic Algorithm – 3 Clusters, 5 runs

NB – DONE ON DIFFERENT DAY

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Route 1 | | Route 2 | | Route 3 | | Run time (s) |
| Distance (km) | Time  (s) | Distance (km) | Time  (s) | Distance (km) | Time  (s) |  |
| 6.88 | 483.74 | 10.5 | 708.09 | 11.61 | 750.59 | 1.15 |
| 6.88 | 483.74 | 10.5 | 658.68 | 9.64 | 625.02 | 1.14 |
| 6.88 | 483.74 | 10.97 | 692.81 | 10.25 | 656.4 | 1.14 |
| 6.88 | 449.67 | 10.79 | 692.2 | 11.35 | 719.12 | 1.14 |
| 6.88 | 449.67 | 10.84 | 692.56 | 11.32 | 728.34 | 1.14 |

Best Times:

|  |  |  |
| --- | --- | --- |
| Route | Distance | Time |
| 1 | 6.88 | 449.67 |
| 2 | 10.5 | 658.68 |
| 3 | 9.64 | 625.02 |

#### 50x50 Genetic Algorithm - 5 clusters, 5 runs

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Route 1 | | Route 2 | | | Route 3 | | | Route 4 | | Route 5 | | Run time (s) |
| Dist (km) | Time  (s) | | Dist (km) | Time  (s) | | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) |  |
| 5.84 | 368.46 | | 6.26 | 411.21 | | 4.67 | 303.34 | 6.88 | 423.29 | 9.22 | 534.79 | 1.31 |
| 5.54 | 346.57 | | 6.26 | 411.21 | | 4.67 | 303.34 | 6.74 | 414.75 | 8.69 | 521.95 | 1.32 |
| 5.54 | 346.57 | | 6.26 | 411.21 | | 4.67 | 303.34 | 6.74 | 414.75 | 9.09 | 540.06 | 1.32 |
| 5.54 | 346.57 | | 6.26 | 411.21 | | 4.67 | 303.34 | 6.74 | 414.75 | 8.69 | 521.95 | 1.34 |
| 5.54 | 346.57 | | 6.26 | 411.21 | | 4.67 | 303.34 | 6.88 | 423.29 | 8.92 | 551.9 | 1.32 |

Best Times:

|  |  |  |
| --- | --- | --- |
| Route | Distance | Time |
| 1 | 5.54 | 346.57 |
| 2 | 6.26 | 411.21 |
| 3 | 4.67 | 303.34 |
| 4 | 6.74 | 414.75 |
| 5 | 8.69 | 521.95 |

#### 50x50 Genetic Algorithm - 7 clusters, 5 runs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Route 1 | | Route 2 | | Route 3 | | Route 4 | | Route 5 | | Route 6 | | Route 7 | | Run time (s) |
| Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) |  |
| 6.7 | 405.08 | 6 | 373.52 | 5.34 | 305.05 | 4.31 | 281.65 | 6.14 | 380.98 | 5 | 344.88 | 4.84 | 307.56 | 1.47 |
| 6.27 | 371.11 | 5.91 | 389.21 | 5.34 | 305.05 | 4.31 | 281.65 | 6.14 | 380.98 | 5 | 344.88 | 4.84 | 307.56 | 1.49 |
| 6.27 | 371.11 | 6 | 373.52 | 5.22 | 309.38 | 4.31 | 281.65 | 6.14 | 380.98 | 5 | 344.88 | 4.84 | 307.56 | 1.48 |
| 6.7 | 405.08 | 5.71 | 374.25 | 5.34 | 305.05 | 4.31 | 281.65 | 6.14 | 380.98 | 5 | 344.88 | 4.84 | 307.56 | 1.49 |
| 6.7 | 405.08 | 5.92 | 374.22 | 5.34 | 305.05 | 4.31 | 281.65 | 6.14 | 380.98 | 5 | 344.88 | 4.84 | 307.56 | 1.49 |

Best Times:

|  |  |  |
| --- | --- | --- |
| Route | Distance | Time |
| 1 | 6.27 | 371.11 |
| 2 | 6 | 373.52 |
| 3 | 5.34 | 305.05 |
| 4 | 4.31 | 281.6 |
| 5 | 6.14 | 380.98 |
| 6 | 5 | 344.88 |
| 7 | 4.84 | 307.56 |

#### Genetic Algorithm - 50 pop size x 150 generations - 3 clusters, 5 runs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Route 1 | | Route 2 | | Route 3 | | Run time (s) |
| Distance (km) | Time  (s) | Distance (km) | Time  (s) | Distance (km) | Time  (s) |  |
| 8.43 | 519.5 | 10.03 | 673.92 | 10.85 | 684.91 | 3.48 |
| 6.88 | 449.24 | 10.03 | 673.92 | 10.04 | 621.64 | 3.47 |
| 6.96 | 477.8 | 11.27 | 705.6 | 9.49 | 581.6 | 3.49 |
| 6.88 | 449.24 | 10.03 | 673.92 | 11.78 | 716.2 | 3.47 |
| 6.96 | 477.8 | 10.15 | 708.18 | 10.58 | 626.1 | 3.48 |

Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Route | Best | | Worst | |
|  | Distance | Time | Distance | Time |
| 1 | 6.88 | 449.24 | 8.43 | 519.5 |
| 2 | 10.03 | 673.92 | 10.15 | 708.18 |
| 3 | 9.49 | 581.6 | 11.78 | 716.2 |

#### Genetic Algorithm - 50 pop size x 150 generations - 5 clusters, 5 runs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Route 1 | | Route 2 | | | Route 3 | | | Route 4 | | | Route 5 | | | Run time (s) | |
| Dist (km) | Time  (s) | | Dist (km) | Time  (s) | | Dist (km) | Time  (s) | | Dist (km) | Time  (s) | | Dist (km) | Time  (s) | |  |
| 5.54 | 383.21 | | 6.65 | 409.3 | | 4.31 | 281.71 | | 6.93 | 430.79 | | 8.68 | 548.86 | | 4.04 |
| 5.54 | 383.21 | | 6.65 | 409.3 | | 4.31 | 281.71 | | 6.93 | 430.79 | | 9.15 | 563.1 | | 4.03 |
| 6.08 | 385.66 | | 6.65 | 409.3 | | 4.31 | 281.71 | | 6.68 | 457 | | 9.27 | 549.32 | | 4.06 |
| 5.54 | 383.21 | | 6.65 | 409.3 | | 4.31 | 281.71 | | 6.93 | 430.79 | | 8.85 | 539.57 | | 4.05 |
| 6.41 | 388.72 | | 6.65 | 409.3 | | 4.31 | 281.71 | | 6.93 | 430.79 | | 9.15 | 563.1 | | 4.03 |

Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Route | Best | | Worst | |
|  | Distance | Time | Distance | Time |
| 1 | 5.54 | 383.21 | 6.41 | 388.72 |
| 2 | 6.65 | 409.3 | 6.65 | 409.3 |
| 3 | 4.31 | 281.71 | 4.31 | 281.71 |
| 4 | 6.93 | 430.79 | 6.68 | 457 |
| 5 | 8.85 | 539.57 | 9.15 | 563.1 |

#### Genetic Algorithm - 50 pop size x 150 generations - 7 clusters, 5 runs

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Route 1 | | Route 2 | | Route 3 | | | Route 4 | | Route 5 | | Route 6 | | Route 7 | | Run time (s) |
| Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) | Dist (km) | Time  (s) |  |
| 6.7 | 404.83 | 5.92 | 374.16 | 5.34 | 304.69 | 4.31 | | 281.71 | 6.14 | 380.84 | 5 | 345.1 | 4.84 | 307.5 | 4.59 |
| 6.27 | 370.81 | 6 | 373.43 | 4.52 | 308.72 | 4.31 | | 281.71 | 6.14 | 380.84 | 5 | 345.1 | 4.84 | 307.5 | 4.58 |
| 6.59 | 383.52 | 6 | 373.43 | 5.34 | 304.69 | 4.31 | | 281.71 | 6.14 | 380.84 | 5 | 345.1 | 4.84 | 307.5 | 4.59 |
| 6.27 | 370.81 | 6 | 373.43 | 5.34 | 304.69 | 4.31 | | 281.71 | 6.14 | 380.84 | 5 | 345.1 | 4.84 | 307.5 | 4.6 |
| 6.7 | 404.83 | 6 | 373.43 | 5.34 | 304.69 | 4.31 | | 281.71 | 6.14 | 380.84 | 5 | 345.1 | 4.84 | 307.5 | 4.59 |

Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Route | Best | | Worst | |
|  | Distance | Time | Distance | Time |
| 1 | 6.27 | 370.81 | 6.7 | 404.83 |
| 2 | 6 | 373.43 | 5.92 | 374.16 |
| 3 | 5.34 | 304.69 | 4.52 | 308.72 |
| 4 | 4.31 | 281.71 | 4.31 | 281.71 |
| 5 | 6.14 | 380.84 | 6.14 | 380.84 |
| 6 | 5 | 345.1 | 5 | 345.1 |
| 7 | 4.84 | 307.5 | 4.84 | 307.5 |

## ‘Aberdeen’ dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Lat** | **Lon** | **Item** | **Time** |
| 1 | 57.1535 | -2.0831 | PC | 1578406178 |
| 2 | 57.1473 | -2.0836 | Laptop | 1578406178 |
| 3 | 57.1578 | -2.0901 | Headphones | 1578406178 |
| 4 | 57.1697 | -2.0939 | Phone | 1578406178 |
| 5 | 57.1603 | -2.0998 | HDD | 1578406178 |
| 6 | 57.1337 | -2.084 | Speakers | 1578406178 |
| 7 | 57.1372 | -2.0718 | Mic | 1578406178 |
| 8 | 57.1375 | -2.1257 | SSD | 1578406178 |
| 9 | 57.1491 | -2.111 | Aux Cable | 1578406178 |
| 10 | 57.1611 | -2.1267 | TV | 1578406178 |
| 11 | 57.1563 | -2.1454 | DVD | 1578406178 |
| 12 | 57.1687 | -2.1114 | Pens | 1578415503 |
| 13 | 57.1391 | -2.1049 | DVD | 1583419912 |
| 14 | 57.1328 | -2.1228 | Remote | 1583419912 |
| 15 | 57.1362 | -2.1049 | Phone | 1583419912 |
| 79 | 57.1485 | -2.1291 | test | 123123123 |
| 80 | 57.1381 | -2.1118 | test | 123123123 |
| 81 | 57.1459 | -2.1235 | test | 123123123 |
| 82 | 57.1609 | -2.1152 | test | 123123123 |
| 83 | 57.1611 | -2.0829 | test | 123123123 |
| 84 | 57.1551 | -2.0917 | test | 123123123 |
| 85 | 57.1532 | -2.1362 | test | 123123123 |
| 86 | 57.1574 | -2.1248 | test | 123123123 |
| 87 | 57.1465 | -2.1289 | test | 123123123 |
| 88 | 57.1514 | -2.1332 | test | 123123123 |
| 89 | 57.1407 | -2.0994 | test | 123123123 |
| 90 | 57.1668 | -2.1058 | test | 123123123 |
| 91 | 57.1508 | -2.0908 | test | 123123123 |
| 92 | 57.1625 | -2.0931 | test | 123123123 |
| 93 | 57.1467 | -2.1331 | test | 123123123 |
| 94 | 57.1416 | -2.132 | test | 123123123 |
| 95 | 57.1663 | -2.1154 | test | 123123 |
| 96 | 57.1561 | -2.1207 | test | 123123 |
| 97 | 57.1677 | -2.1241 | test | test |

## Poster



## Ethics Form

**STUDENT PROJECT ETHICAL REVIEW (SPER) FORM**

**The aim of the University’s *Research Ethics Policy* is to establish and promote good ethical practice in the conduct of academic research. The questionnaire is intended to enable researchers to undertake an initial self-assessment of ethical issues in their research. Ethical conduct is not primarily a matter of following fixed rules; it depends on researchers developing a considered, flexible and thoughtful practice.**

**The questionnaire aims to engage researchers discursively with the ethical dimensions of their work and potential ethical issues, and the main focus of any subsequent review is not to ‘approve’ or ‘disapprove’ of a project but to make sure that this process has taken place.**

The *Research Ethics Policy* is available at  [www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=706](http://www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060)0

|  |  |
| --- | --- |
| **Student Name** | Andrew Trail |
| **Supervisor** | Kit-ying Hui |
| **Project Title** | Drone Delivery Scheduler |
| **Course of Study** | CASD |
| **School/Department** | RGU |

|  |  |  |  |
| --- | --- | --- | --- |
| **Part 1 : Descriptive Questions** | | | |
| 1 | Does the research involve, or does information in the research relate to: | Yes | No |
|  | (a) individual human subjects |  | X |
| (b) groups (e.g. families, communities, crowds) |  | X |
| (c) organisations |  | X |
| (d) animals? |  | X |
| Please provide further details: | | |
|  |  |  |  |
| 2 | Will the research deal with information which is private or confidential? | Yes | No |
|  |  | X |
| Please provide further details: | | |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Part 2: The Impact of the Research | | | |
| 3 | In the process of doing the research, is there any potential for harm to be done to, or costs to be imposed on | Yes | No |
|  | (a) research participants? |  | X |
| (b) research subjects? |  | X |
| (c) you, as the researcher? |  | X |
| (d) third parties? |  | X |
| Please state what you believe are the implications of the research: | | |
|  | | |
| 4 | When the research is complete, could negative consequences follow: | Yes | No |
|  | (a) for research subjects |  | X |
| (b) or elsewhere? |  | X |
| Please state what you believe are the consequences of the research: | | |
|  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Part 3: Ethical Procedures | | | |
| 5 | Does the research require informed consent or approval from: | Yes | No |
|  | (a) research participants? |  | X |
| (b) research subjects |  | X |
| (c) external bodies |  | X |
| If you answered yes to any of the above, please explain your answer: | | |
| 6 | Are there reasons why research subjects may need safeguards or protection? | Yes | No |
|  |  | X |
| If you answered yes to the above, please state the reasons and indicate the measures to be |  |  |
| 7 | Has PVG membership status been considered? |  |  |
|  | (a) PVG membership is not required. | X |  |
| (b) PVG membership is required for working with children. |  | X |
| (c) PVG membership is required for working with protected adults. |  | X |
| (d) PVG membership is required for working with both children and protected |  | X |
| If you answered yes to (b), (c) or (d) above, please give details: | | |
| 8 | Are specified procedures or safeguards required for recording, management, or storage of data? | Yes | No |
|  |  | X |
| If you answered yes to the above, please outline the likely undertakings: | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Part 4: The Research Relationship | | | |
| 9 | Does the research require you to give or make undertakings to research participants or subjects about the use of data? | Yes | No |
|  |  | X |
| If you answered yes to the above, please outline the likely undertakings: | | |
| 10 | Is the research likely to be affected by the relationship with a sponsor, funder or employer? | Yes | No |
|  |  | X |
| If you answered yes to the above, please identify how the research may be affected: | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Part 5: Other Issues | | | |
| 11 | Are there any other ethical issues not covered by this form which you believe you should raise? | Yes | No |
|  |  | X |
|  |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Statement by Student  I believe that the information I have given in this form is correct, and that I have addressed the ethical issues as fully as possible at this stage. | | | |
| Signature | Andrew Trail | Date | 29/9/19 |

**If any ethical issues arise during the course of the research, students should complete a further Student Project Ethical Review (SPER) form.**

The *Research Ethics Policy* is available at  [www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=706](http://www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060)0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part 6: To be completed by the supervisor | | | | |
| 12 | Does the research have potentially negative implications for the University? | | Yes | No |
|  |  | ✓ |
| If you answered yes to the above, please explain your answer: | | | |
|  | | | |
| 13 | Are any potential conflicts of interest likely to arise in the course of the research? | | Yes | No  ✓ |
|  | If you answered yes to the above, please identify the potential conflicts: | | | |
|  | | | |
| 14 | Are you satisfied that the student has engaged adequately with the ethical implications of the work? [In signifying agreement, supervisors are accepting part of the ethical responsibility for the project] | | Yes  ✓ | No |
|  | If you answered no to the above, please identify the potential issues: | | | |
|  | | | |
| 15 | **Appraisal:** Please select one of the following | | | |
|  | The research project should proceed in its present form – no further action is required | | ✓ | |
| The research project requires ethical approval by the School Ethics Review Panel | |  | |
| The research project needs to be returned to the student for modification prior to further action | |  | |
| The research project requires ethical review by an external body. If this applies please give details | |  | |
| Title of External Body providing ethical review |  | | |
| Address of External Body |  | | |
| Anticipated date when External Body may consider project |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Affirmation by Supervisor | | | |
| **I have read the student’s responses and have discussed ethical issues arising with the student. I can confirm that, to the best of my understanding, the information presented by the student is correct and appropriate to allow an informed judgement on whether further ethical approval is required.** | | | |
| **Signature** | K. Hui | **Date** | 30/09/2019 |

## Project Proposal

|  |  |
| --- | --- |
| First Name: | Andrew |
| Last Name: | Trail |
| Student Number: | 1203154 |
| Supervisor: | Kit-ying Hui |

Defining your Project

**1.1 Project title**

**Help:** a brief statement about what you are actually going to do.

|  |
| --- |
| Drone Delivery Scheduler |

**1.2 Background**

**Help:** Provide the background to your project. This section should highlight the main topics in the area you are going to research. Essentially what is the project about, what has been done before and why is this project important? ~500 words

|  |
| --- |
| The aim of this project is to create an automated drone delivery scheduler. It will simulate delivery of objects via drones to customers in so called ‘last mile’ deliveries. The solution should find the optimal pattern of delivery based on a number of variables, such as wind speed/direction, ‘no fly zones’ and the weight of the object being carried.  There are several companies who are known to be working on a similar service, notable Amazons ‘Prime Air’, googles ‘Project Wing’ and FedEx. None of these companies have fully released a service to consumers yet, but Amazon is expected to roll out their service in the late months of 2019 for selected cities. It is currently doing a trial in Cambridge for select customers.  There is limited information surrounding these projects, however it is safe to assume they are using AI to schedule deliveries and calculate flight paths. I will be creating a similar solution.  The project is important as I believe it is the future of delivery services. Besides the companies mentioned above, there are many more who are working on delivery services using drones, with applications ranging from food delivery, to emergency supplies on the front line of a war.  The use of drones will only increase, and it is vital for companies using them for delivery to be able to have a reliable and optimal delivery scheduler. This saves money for the company which is an important factor for them. The customer is also happier when their parcel is delivered quicker than possible using a traditional van method.  I will be exploring the use of AI for this task, however it may not be feasible with current information available for me to research and adapt to this purpose. |

**1.3 Motivation**

**Help:** To whom is this project important? A project must address a question/problem that generates a small piece of new knowledge/solution. This new knowledge/solution must be important to a named group or to a specific client (such as a company, an academic audience, policy makers, people with disabilities) to make it worthwhile carrying out. This is the ***motivation*** for your project. In this section you should address who will benefit from your findings and how they will benefit. ~300 words

**Example** 1: If you intend to demonstrate that a mobile application that automates class registers at RGU will be more efficient than paper-based registers - the group who would be interested in knowing/applying these findings would be both academic and administrative staff at RGU and they would benefit by time saved and a reduction in their administrative workload.

**Example** 2: You are demonstrating that a particular 3D model design increases realism in 3D environments. The group that would be interested would be games designers or developers of 3D virtual environment applications. The would benefit from producing more realistic environments that could increase sales of their products.

**Example** 3: You have designed a new network topology for IrishOil plc’s new Aberdeen headquarters. The interested group would clearly be IrishOil. They would benefit from easier maintenance and improved security of their computer network.

|  |
| --- |
| There is no specific company that would benefit from my findings, as I am not working with a specific company. However, if I find that the project is viable, and produce a solution that works well, this could be used by any company to implement drone delivery for themselves. This has implications of saving the company money, as they wouldn’t have to employ delivery drivers to drive a van or pay a third party to do this for them. Additionally, it will help to prevent issues such as drones running out of battery mid-flight, and to improve delivery times, and give customers a more accurate estimated delivery time.  It also has implications for individuals and small companies. For example, a single person who sells homemade products like jewellery on ebay or etsy could benefit from purchasing a handful of drones and using the system for themselves to deliver to the area surrounding their home. |

**1.4 Aim & Objectives**

**Help:** Outline what are the main things your project is going to do and what steps or milestones will be used to achieve this aim. The Aim is unlikely to change throughout your project; however, the objectives are likely to adapt to your ongoing research and development. In particular it is highly likely that you may wish to split objectives into sub-objectives as work progresses. A good clear set of objectives give you something to evaluate your final project against.

**Example** : For the timetable app outlined above

Aim: To create a functioning attendance application that efficiently automates the taking of class registers.

Objective 1: study existing register system in place at RGU and identify weaknesses

Objective 2: research existing automation technology’s and identify and evaluate those that may be appropriate to taking in class registers

Objective 3: Implement chosen technologies to create prototype application

Objective 4: Conduct user trials to evaluate capabilities of prototype application

Objective 5: Create a refined application incorporating feedback from user trials

|  |
| --- |
| Aim: To create a drone delivery scheduler  Objective 1: Study existing implementations of delivery scheduling and see if they can be adapted and applied to drones.  Objective 2: Study limitations of drones that are currently available.  Objective 3: Research sources of data for the variables used to simulate the problem. Weather data, areas drones cannot fly etc.  Objective 4: Select the best sources of data and method of solving the problem, implement a solution.  Objective 5: Test the system extensively to find any faults.  Objective 6: Implement any findings from the testing and test the new optimised solution.  Objective 7: Evaluate the final solution and draw a conclusion about its fitness for purpose. |

**1.5 Key Techniques**

**Help:** Perform some initial research into the area and outline what techniques you my research in further detail here. The techniques you cover here should include references to the papers where you have sourced the information. The techniques mentioned here are very likely to become the section headers in your literature review.

|  |
| --- |
| A\* Travelling Salesperson. In this case, the drone can carry multiple packages and make multiple deliveries in one trip. This problem can be solved with an A\* search and a strong heuristic. The solution must find the shortest path for a drone to take. (Siregar et al ., 2017 https://iopscience.iop.org/article/10.1088/1742-6596/801/1/012038/pdf.)  Grouping delivery locations. I must research how to group delivery locations into groups that can then have a path determined and a drone assigned.  Simulated Annealing may be a useful technique. If there are a large quantity of packages to be delivered, the time taken to calculate the optimal route may take too long. If the system takes too long to find a perfect solution, it may end up taking more time to do this and perform deliveries then it could otherwise take with a sub optimal route. Simulated annealing can be used as a heuristic to help solve this issue. (Dorling et al., 2017 <https://ieeexplore.ieee.org/abstract/document/7513397>)  Vehicle Routing Problem – A problem that aims to find the optimal route for vehicles to use to deliver packages to customers. |

**1.6 Legal, Social, Ethical, Professional and Security issues**

**Help:** Here you should discuss any legal, social, profession and security issues that you believe may occur during the course of your project. It is not acceptable to write none in this box, all projects, regardless of focus will have to address issues in one, or more, of these categories. This is an extremely important part of your honours project to which there is no correct answer, this section must be fully discussed with your Honours Supervisor.

**Example 1** : In the class register example above – there would be a Legal and Security issue with the gathering and storage of student data. There may be a social constraint as you may be relying on a user to have access to a specific technology. There will need to be consideration of user accessibility.

**Example 2** : A 3D model design may have ethical considerations in its evaluation. What if your model made users feel nauseous. Social constrains may again be access to technology or accessibility issues.

**Example 3** : You network design need to adhere to specific company policies. You would need to consider the possibility that your design could be wrong, compromising the company’s security.

|  |
| --- |
| Legal:  Flying an unmanned drone, controlled by a piece of software is going to bring a host of legal issues. Who is responsible for the drone if it were to crash, the person who owns it, or the person who wrote the software? Additionally, how can it be certain that the drone is not going to fly through any restricted air space?  Social:  The proposed system may cause the loss of jobs. Last-mile delivery drivers who go from a depot in a city to peoples homes may become unemployed as a result of the service. Also noise of drones flying past and over peoples homes, and danger of the drone crashing into someone are all issues to take into account.  Security:  An initial security issue is that of hacking. If someone managed to gain control over the system, they could alter the way the drones function, causing them to steal packages by delivering them to a specified location, or cause injuries to people by crashing into them, onto roads for example.  Also the possibility of people managing to take drones out of the sky and steal what they are carrying. This also falls under a professional issue, as a company who’s products are routinely stolen will not be received well by the public. |

**1.7 Project Plan**

**Help:** This is the project plan as to how you will go about achieving the objectives of the project.

**Example**: In the class register example above the research plan may involve:

Collecting and analysing paper-based registers in a given class on five occasions.

Identifying the error rate average on these occasions

Researching existing automation techniques

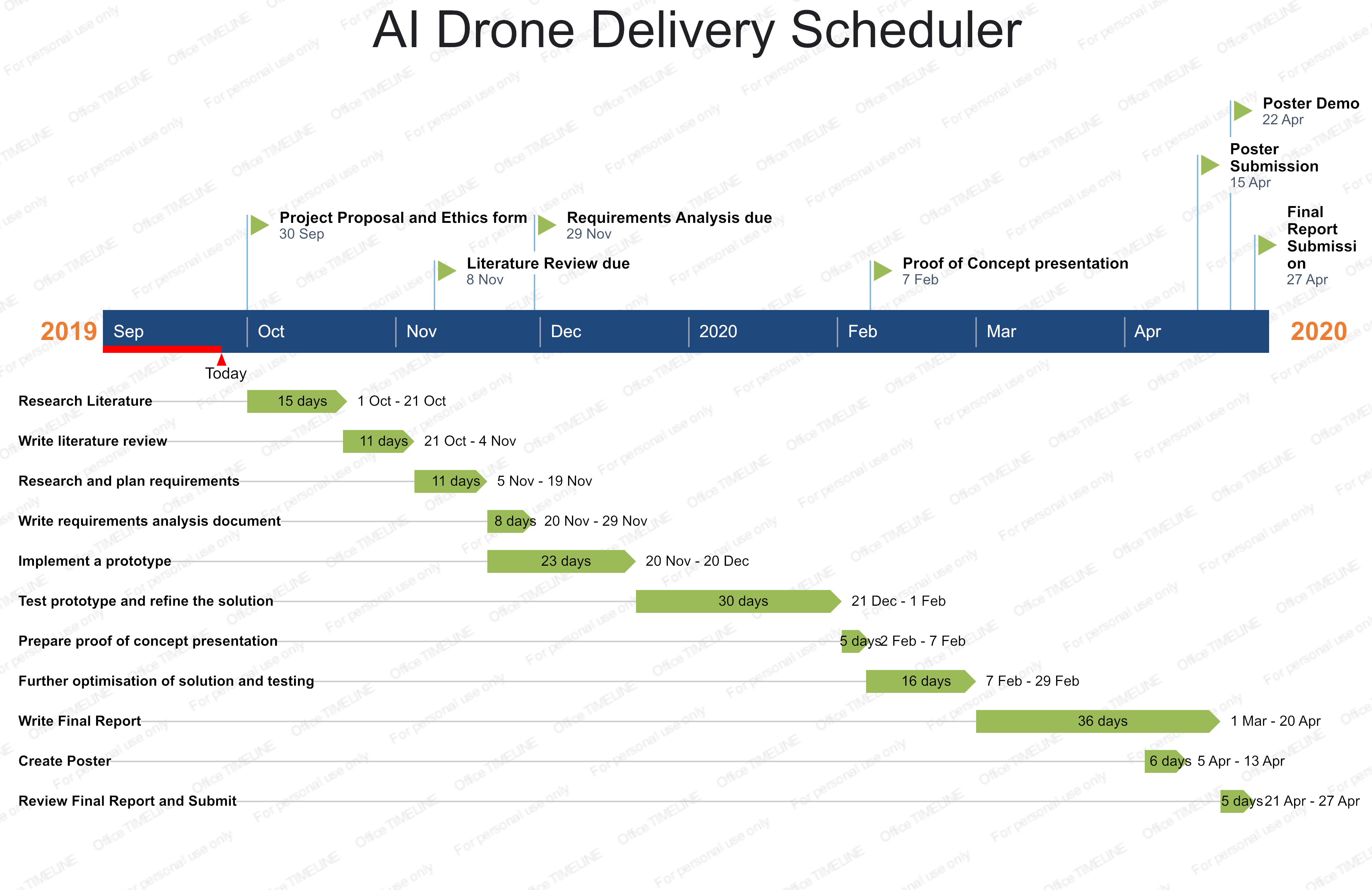
Designing and implementing a mobile application that automatically records attendance in class.

Deploying the application in the class on five occasions.

Identifying the error rate average of the mobile application on these occasions.

Comparison of data and summary of findings.

|  |
| --- |
| Research current implementations of drone delivery systems. Analyse how they have been implemented, if AI has been used.  Research drones that are currently on the market. How far can they fly, what weight can they carry, how much do they cost?  Research weather data APIs and laws surrounding drone flight specifically unmanned.  Select the best of the above findings and conduct further research to see if they are viable methods to solve the problem  Begin implementing a prototype  Optimise the solution based on findings from testing  Evaluate the solution vs the aim and objectives  Begin combining everything into a final report |

****

**1.8**

## Requirements Analysis

Introduction

The purpose of this project is to produce a system for scheduling delivery via drone. The system will have a GUI for the user where they can add or remove orders, adjust the number of drones being used and see the resulting routes in a visual way. The orders will first be placed into groups with similar locations. The system will then apply an algorithm to find good routes to allow drones to serve customers efficiently. This requirements analysis will lay out all the functional and non-functional requirements of the project.

Functional Requirements

Must:

Have a GUI for the user to interact with:

The main screen will have a real map of a city. A fictional depot will be marked on, as well as customer locations, groups that customers have been assigned and the route that each drone will take. This is needed to make the application usable and the results understandable to the average user.

The user will be able to input orders from the GUI. They will have the option of clicking a point on the map to select that as the location, and then entering details of the order. Alternatively, they will be able to input everything manually.

The user will be able to control some parameters in the system. These will include the number of drones and maximum flight distance per drone. This is essential as it allows a user to create a scenario and explore how many drones and of what power they would need for their business.

Divide all the customer locations into smaller groups. For this a clustering method should be used. The locations are to be grouped by proximity so that each drone only serves a tight area. This will result in more efficient delivery as flight time is reduced.

Routes must then be created for drones to take. There are several algorithms available to solve this problem, at least one will be available for the user. This is required as without it; the problem cannot be solved.

Should:

Give the user several algorithms to choose from for path finding. This will be useful for comparison of algorithms and to ensure that we find the best one for the job. Several algorithms were reviewed in the literature review so we will select from there.

Consider weather conditions. As the user interface will have a real location, we may take the weather into account when planning deliveries. The main factors taken into consideration will be wind speed and direction. If it is too windy a drone will take a long time, or even be unable to complete its route.

Allow for interactive user input. Ideally the application should be running constantly, waiting for orders to be placed. The user should be able to add, remove or edit an order while the system is running, and it will adapt its current groups and routes to accommodate the new order.

Could:

Display the current state of routes as they are created. It would be an interesting way of showing how the different path finding algorithms work and how quickly they converge on a good route.

When running, show the location of each drone as they move through their route.

Non-Functional Requirements

Must use python for development. The language has been chosen as there are several libraries available to aid development. Additionally, I am familiar with the language.

Could use scikit-learn library for clustering. The library provides a range of algorithms such as k-means and affinity propagation. Developing a clustering algorithm for ourselves would take a long time and is not the focus of this project thus the library.

Could use pyeasyga for the genetic algorithm. As with scikit-learn for clustering, the purpose of this project is not to explore implementing a genetic algorithm. While it would be interesting to do, it is not essential.

Could use pyqt to produce a GUI. Pyqt is a powerful tool for creating GUIs that will allow a clean user interface to be developed.

Should ship as a .exe file. This will allow portability over a range of hardware and make it easy to use.

Should use an appropriate API for weather data if this is implemented

Should have a fast runtime. As this is designed for delivering items to customers, a long wait time is not acceptable.

Summary

|  |  |
| --- | --- |
| **Concept** | **Reason for choice** |
| **Must have:** | |
| A GUI | This is essential to make the end product usable for anyone. It would be extremely difficult to show the user what is happening without doing so visually. |
| Real city map | A map of a real city makes the system more applicable to the real world and shows a realistic outlook on how drones may operate |
| User input from GUI | Without this it becomes a difficult tool for the average user to use and understand |
| User controlled parameters from GUI | A final item of the GUI that will make the system intuitive to use. The user can directly see the effect that different parameters have on the system |
| Clustering technique | For a delivery system, speed is essential. The use of clustering allows the system to more quickly find suitable routes for drones. It also ensures that drones are not flying large distances in between customers where they can avoid it |
| Path finding algorithm | Possibly the most crucial step of the whole process is finding the routes. With no pathfinding algorithm the end product would not solve the problem of scheduling delivery. |
| **Should have:** | |
| Solution produced in a reasonable timeframe | The data should be analysed quickly, and a solution provided. May not be possible if there is a large amount of orders placed at once. |
| Simple to use GUI | It is important that anybody should be able to use the software without guidance. |
| Explore alternative path finding algorithms | Differing algorithms would be an interesting feature so that users can see how they vary in quality and speed of solution |
| Weather conditions considered | Makes the solution more realistic and interesting |
| Interactive user input | It is possible to restart the process every time a new order is added. However, this is far from ideal, as the routes would all need to be recalculated every time a new order was placed. |
| Delivered as a .exe file | This allows portability across windows machines. Additional formats could be created for macOS and linux. |
| **Could have:** | |
| Display current state of routes | It would be interesting for the user to see how routes are generated with different algorithms. |
| Show where drones are on their route | Provides a more interactive experience for the user. |