Visualisation of Ant colony optimisation applied to routing problems
ECM3401 - Final Report & Demonstration

Jacob Maullin-Davies

April 2022

Abstract

The increased application of machine learning and artificial intelligence in the field of routing problems is one of the most widely researched areas, due to the potential applications in logistics, robotics and construction planning. Nature inspired algorithms are proposed for many path finding and route searching problems, as these types of algorithms are unique in their search methods. The Ant Colony Optimisation (ACO) is a well-known swarm intelligence technique, inspired by real world ants and their cooperative behaviour in the exploration of the world and finding food resources. A significant number of studies have focused on the organised behaviour of ants in the natural environment to develop effective systems for dynamic problems, such as route optimisation.

Search algorithms are highly dependent on the correct setting of parameters for the given problem. Different parameters can be better suited for certain optimisation problems. Visualisation offers a way of presenting the results of such search algorithms so that a user can understand how these types of nature inspired algorithms explore and search under constraints and demonstrate the algorithms characteristics. Visualisation further offers the demonstration of how the optimisation algorithm performs the process.

This report details the design, implementation and evaluation of a Visualisation system to demonstrate the ACO algorithm applied to routing problems. This project presents an application intended to offer insight to the process of route optimisation, by demonstrating the performance of the ACO algorithm optimising route optimisation of real world street network environments in an effective visualisation.

I certify that all material in this dissertation which is not my own work has been identified.

Contents

1 Introduction

To visualise an algorithm, results are not merely fitted data in a chart; there is no primary dataset that is chosen. Instead there are logical rules that describe behaviour for an algorithm, to show how it performs against a given problem. This is why algorithm visualisations are so unusual, where designers experiment with novel forms to better communicate the solution process [7].

In recent years, there has been an expansion of interest in route planning and path finding for its applications in logistics, mobile robotics, traffic transportation prediction, and other AI controlled motion technologies [2,5,22]. This rise in interest has explored different prevalent path planning techniques, most of which are nature inspired swarm algorithms and natural systems [3,4,6]. These types of algorithms have systematic social mechanisms that can enhance the design of typical algorithms and AI problem solving systems [3,6,21]. Many solutions to improve path routing use these ideas to create biologically inspired techniques such as particle swarm optimisation (PSO) and ant colony optimisation (ACO) [2,6].

The ACO algorithm is a heuristic algorithm with strong robustness and search capabilities and simulates the natural cooperative foraging behaviour of ants which derives the best path solution jointly planned by the ants [3,4,24]. This algorithm simulates this by artificially using ants, assisting and guiding other ants by using chemical signatures called pheromones for direct and indirect communication, and are therefore are a collective wisdom model [3,4,6].

An approach to making such techniques more readily understandable is to visualise the operation of the algorithms on a given problem, in a way such that a decision maker can identify operation characteristics that will assist them for a problem solution [7,22]. Specifically, the aim of this report and the final system application is to visualise the ant colony optimisation algorithm in different problem environments, where the algorithm parameters can be altered so that the algorithm user can identify which parametrisations offer best performance for their problem [7]. The parameter values of the ACO have a large impact on the ant decision making process [22].

The system that this project proposes is a novel and interesting visualisation method which can provide detailed insight into the ACO algorithm route optimisation solutions generated by behaviour of the ant population [22]. It effectively conveys the search algorithm's behaviour as it produces solutions, rather than typically showing the final output, and data on a chart [22]. Possible applications of such a system include: finding optimal paths during construction of road networks in a city, investigating traffic routes most likely taken and identifying high pollution areas, and delivery logistics in a city search area.

This project aims to successfully demonstrate the techniques of the ACO heuristic algorithm and show its performances in applied graphical routing problems. The ACO ant behaviour and process will be illustrated, and how it produces an optimal path. Visualisation offers a way of presenting the results of path planning so that a user can understand how the algorithm is performing [7]. By examining the path characteristics, such as optimal time/length of the algorithm's performance, convergence and diversity, the user can learn how effective the chosen algorithm parameters are [7].

Through the process of design and development, a system that visualises the ACO algorithm is applied to routing problems using OSMnx geo-spatial data models [1]. OSMnx is a Python package for downloading OpenStreetMap street network data and then constructing it into NetworkX graphs [1,20]. Different ACO algorithm solutions can be applied to the NetworkX graphs in path finding problems, and then the path solutions are visualised in real time [1]. The complete system demonstrates how the algorithm performs in the problem graphs compared to traditional route optimisation algorithms such as A* and Dijkstra's [11,13,25]. Testing of each algorithms performance will be visualised in real time.

The following report describes the design, project implementation and evaluation of the ACO visualisation software applied to routing problems. First, a literature review and project specification will explain the key concepts of the ACO model, how the algorithm works and how it is implemented in the literature. Then a specification describing the functional and non-functional requirements is listed. Design sections illustrate how the project was approached, detailing the abstract architecture of the systems functionality. Following this, the Development section demonstrates how each part of the project was implemented, problems encountered and solutions to solve these issues, and the

justification of modifications from the original specification. Testing sections outline the processes used to assess the quality of the system and the algorithms performance in path-finding. This is followed by an Evaluation of the results obtained from the testing and the subsequent design modifications that contributed to the project. The Critical analysis details any future work that may be undertaken further from this project. Finally, the Conclusion will assess the overall success and limitations of the project, with further analysis to determine if the final product has fulfilled the specification objectives for the project.

2 Summary of Literature Review and Specification

2.1 Literature Review

The ACO algorithm was initially proposed to solve combinatorial problems, as stated by Mirjalili [3]. In such problems, the objective is to find an optimal subset from a given finite set [3]. Most of combinatorial problems are NP-hard, therefore heuristic approaches have been always beneficial and popular [3,24].

Path planning and route optimisation aims to calculate the shortest and collision-free path from an origin point to a designated endpoint by avoiding static or dynamic obstacles, given an environment [2,3]. In the paper by Duan *et al.* [18], it is explained that an environment should be traversed under certain constraints when a search algorithm attempts to find the optimal path most commonly distance/cost, or time. The ACO algorithm tries to solve a given problem by generating numerous possible paths, and Seyyedabbasi and Kiani [23] explain that this mechanism of operation is unique in metaheuristic algorithms. This approach has a high proficiency in handling unknown environments as each ant agent presents one solution during the exploration phase of the algorithm, meaning there might be many possible solutions [21,24]. The best path, being the least cost of the path, is chosen from each iteration [3,18,24].

There is an extensive amount of literature in the field of route planning and path finding in general and especially for vehicle route planning which uses evolutionary algorithms [2,3, 24]. The most important feature concerns the solution representation, which can define the size of the search space and influence the efficiency of the algorithms [5].

Various representations, such as graph-based and grid-based have been suggested for visualisation of path finding problems [5,8,11]. Typically, there are two main approaches, explored by Weise and Mostaghim [5]. The first is a variable-length chromosome which is often used in combination with the graph-based problem representation in both topological and geometric based environments [5,14]. This approach represents a solution as a list of nodes, with edges consisting of different lengths connecting the nodes together which allows the formation of paths [5]. The geometric representation is the view of the actual environment, where the topological is the actual connectivity of the graph that the algorithm views [5,14]. This is highlighted effectively in Figure 1. The second approach is a fixed-length chromosome, representing the directions of travel together with a list of nodes in a graph or a list of grid cells [4,5]. Using a grid type of environment is a simple and effective way to demonstrate the capabilities of path finding demonstrations, but lacks the detail of real world environments. Path representations using a grid environment explored by Chen *et al.* [4] and Ajeil *et al.* [8] are a simple rendition of an environment problem, representing real-world problems by discretizing the problem [9,25].

Topological and geometric graph based representations have more advantages than the simple grid method [5]. Weise and Mostaghim [5] state that generally, geometric graph-based representations allow higher flexibility in representing real-world problems, which can be considered heterogeneous, compared to grids which are usually homogeneous as defined by the square structure of the environment [14]. Graph based are more complex and show true dynamic environments compared to static based grid environments [4,8].

Real world environments were successfully graphically visualised and effectively implemented by Aisham *et al.*'s study [2]. Route optimisation using ACO was explored where the implementation of ACO algorithm was used as a solution to minimise time delay for a campus map traversal for university students [2]. To create the environment for the ACO algorithm, possible routes were obtained by using

Google Maps API to collect students' residence coordinates [2,10]. A polygon border was established within a limit area for the data search space [2]. The defining area from the latitude and longitude provided set searching point boundaries in the algorithm to perform without searching outside the defined area. Each point in the search space is then used as a possible node point that can be traversed.

This project aims to create a system that can model a real world environment, and using ACO optimise paths. From the review and assessment of the literature, this project's specification has been altered over the project. Time and complexity have also affected the project's requirements. Visualisation of ACO has remained the main focus, but the design of the system requirements has changed to achieve this.

2.2 Functional Requirements

• Environment graph generation: Data point geographical data will be used to generate an environment, in addition to the creation of a graph that can be traversed using route optimisation algorithms. This allow algorithms to search and process paths to produce optimised routes which can then be shown on the geographical environment. This environment will then be transferred into a graphical representation for the algorithms path finding demonstrations. The visualisation of the area environment must be complex and correctly rendered. Origin and destination nodes will need to be correctly defined.

• ACO path finding: Route optimisation is a fundamental requirement and has many applications in logistics or building infrastructure (roads). This project will implement the Ant Colony Optimisation algorithm, which should then be able to traverse the environment according to its naturalistic functionality. The ACO model will use the heuristic techniques and nature inspired behaviour to locate paths to the objective(s) where this is displayed in the environment. Numerical and visual evaluations of the paths will be generated to determine the success of the output paths. Additionally, baseline search algorithms will be implemented to evaluate the performance.

• Real-time performance visualisation: Each path finding techniques will be visualised through each iteration as it explores the environment. Systems must be created in order to successfully visualise this exploration of the environment using the paths generated by the ACO algorithms. The system must show each path trail in real time (in an acceptable time frame rate) with adjustable scaling of the speed of the paths visualised. At the end of the iteration process, the end best path must also be shown distinctively.

2.3 Non-Functional Requirements

Additionally, an implemented user interface (UI) would allow seamless interaction with the area environment selection and the path finding visualisation. UI features would include real world map interaction and selection, allowing the definition of the area size. The origin and target selection allows custom defined targets within the search area. Path finding ACO variables and environment parameters could be defined before execution of program, to explore the effects of the variables on the output. ACO techniques could include different alterations to the basic models, to demonstrate effectively how the ACO algorithms perform under the rules of its own functionality. Interactive charts for data analysis purposes could also be added to the UI, allowing the view of the path progress overtime. Since these non-functional objectives involve a UI, and additional algorithms, these will all affect the final outcome of the system. Therefore, as a non-fundamental objective, these will objectives will be evaluated.

2.4 Evaluation Criteria

This project's implementation will be evaluated against previously mentioned functional and non-functional requirements and the extent to which they are successfully met. If the project implemented is able to take a map area and process the environment point data into a graph, then the environment creation will be a success. If this graph can visually display the ACO behavioural routing techniques to show the best found path in an efficient and and effective manner, then this area will be a success.

This visualisation will be successful if the displayed paths are rendered in real time (appropriate amount of time for the environment size) and reflect the real world street networks. For the route optimisation paths, these must conform to the ACO techniques with the removal of less travelled paths using the ACO algorithm techniques. This behaviour must be reflected in the visualisation.

Gathered testing of different areas and the performance of the ACO will demonstrate the success of the implemented ACO models and architecture, and its adaptability over new areas. Different sizes and node complexity will be tested for run time environment generation performance and path finding search performance, and optimal path convergence. These will be tested against an integrated Dijkstra's algorithm [4,5,8] for a baseline performance. These combined tests are performed to provide data analysis which will determine the overall measure of success of the ant paths generated, and the extent to how well the ACO performs.

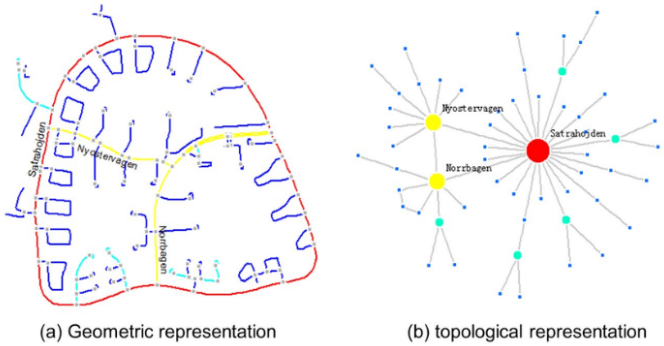
3 Design

The design section outlines the implemented methods and system architecture, and the functionality is split into 3 stages: environment gathering and generation, path finding algorithm design and path proposal visualisation.

1. Map graph acquisition and are-processing - an area will undergo pre-processing to generate isochrone area maps and gather street network data before it is passed onto the ACO search algorithm [10]. This ensures that the area selected has appropriate road areas to traverse, and the target/origin is able to be reached within the area.
2. ACO graph traversal - the program must be able to traverse the graph. To allow this, an adjacency grid must be generated from the nodes. This grid determines which nodes of a graph are connected, and the edges determining the direction of travel
3. Path visualisation - the generated ACO node paths must be shown on the map environment.

Discussed above in the literature review, there are numerous ways to visualise and implement path finding techniques each with their own benefits and drawbacks in different applications. This report's design section will outline the justification for choices of technique and implementation of systems and algorithms.

3.1 Graph Acquisition



(a) Geometric representation (b) topological representation

Figure 1: Environment visualisation and algorithm search representation [14]

Reflecting on the literature and shortcomings of grid based environment representations, graph based environments are effective and the topologically representation is accurate for path finding in real world applications. In order to reflect the complexity of real world paths and environments, a geometric graph must be constructed for the route optimisation algorithms path traversal outputs to be shown. The ACO techniques will be able to explore the topological environment, and use the heuristic pheromone behaviour to converge to an optimal path, where these behaviours are reflected in the geometric graph, seen as which paths are taken.

To generate a graph environment for traversal and visualisation of the system, an area must be defined as the search space.

In this section the basic notation is introduced, with further explanation of how the road network is represented by a graph and then reduced to a complete graph [15]. Provided is a mathematical

formulation of the environment formulation. A graph is an abstract representation of a set of elements and the connections between them [1,15]. The elements are interchangeably called vertices or nodes, and the connections between them are called links or edges [1,5,15].

An undirected graph's edges point mutually in both directions, but a directed graph, or digraph, has directed edges (i.e., edge i,j points from node u to node v, but there is not necessarily a reciprocal edge j,i) [1 12]. A self-loop is an edge that connects a single node to itself [1,14]. Graphs can also have multiple edges between the same two nodes (can travel the same road but both ways) [1].

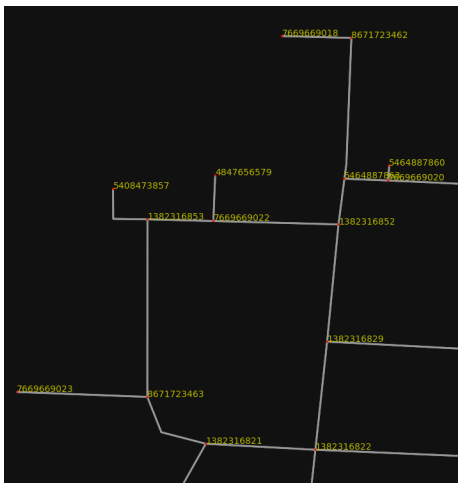


Figure 2: Red markers are a node location, identified by a unique number. The OSMnx data has the respected directions of adjacency, if the edges to other nodes are passable or not [1].

Using world road-map data, nodes can be generated from intersections and any road network can be expressed as a directed connected graph $G(N, E)$ where N is a set of nodes and E is a set of edges connecting them. A link between nodes i and j is denoted as $E_{i,j}$, effectively shown in Figure 2 [3,11]. The path-finding cost $C_{i,j}$ function is the distance or time to travel the link [15,18]. This cost function will determine the ranking of a solution path among the others proposed [18].

For the ACO implementation, an intensity of pheromone from one node to another is stored on the edge connecting them. This intensity of pheromone is the main functionality of the ACO algorithm. This link however is dependant on the direction of travel. The link $E_{i,j}$ cannot have a single pheromone stored for path traversal, and must reflect the road map geometry. For example, all possible paths from a node i to nodes (h, j, k) will have a pheromone id to the links $E_{i:h,j,k}$. Nodes h,j,k will have a pheromone value each link to node i. Therefore, each edge will have an independent pheromone for each edge. This must be efficiently stored and access for larger maps to ensure path visualisation is performed in the respected time frame.

The python package OSMnx can be used for data, creation and analysis of simple street networks with consistent and automatable from the perspectives of graph theory, transportation, and urban design [1].

OSMnx is a free, open-source Python package that downloads political/administrative boundary geometries, building footprints, and street networks from OpenStreetMap [1,20]. It has the ability to easily construct, project, visualise, and analyse non-planar complex street networks consistently from an area box [1].

3.2 ACO Design

For the ACO visualisation system to autonomously navigate the environment graph it must be able to traverse every possible node edge with simplicity and efficiency. The model must identify shorter/more traversed paths, while also not falling into a local optimum. The combined ant traversal should converge to a best path solution in a reasonable time frame (for the size of the area map selected).

To solve NP-Hard combinatorial problems, the implementation of the ACO has been designed with 3 main phases: path-search/construction, pheromone-update, pheromone-evaporation.

Algorithm 1 The pseudo code of the ACO algorithm [3]

```
Environment Initialisation
while not End Condition do
    Search Environment
    Return Path Finesses -> Update Pheromone
    Evaporate Environment
end while
Return Best Solution
```

ACO as a bionic intelligence algorithm is a heuristic based algorithm used to find an optimised path within a graph [4]. In path planning, ACO finds the path of the starting point to the target point by simulating the foraging behaviour of an ant colony [3,4,6]. When searching for a path, the pheromone of the ant will be left on the path it travels. The more a path is travelled, the more pheromones are left and the higher the pheromone concentration will be [4,9]. Thus, the higher the probability of the ant choosing that path. Below are the equations defined for the ACO traversal of node paths.

$$\triangle\tau_{ij} = \begin{cases} 1/d_{ij}, \text{ point i to point j is walkable} \\ 0, \text{ otherwise} \end{cases} \quad (1\ ,\ [3,\ 4])$$

$1/d_{i,j}$ is the heuristic factor from node i to node j representing the degree of influence of the pheromone value on the path selection, where $d_{i,j}$ is the length of the edge i, j [3]. Pheromone choices of available possible edges from a node are selected and a node is chosen. Edges with a higher value are more like to be chosen; but not an absolute choice. $\triangle\tau_{i,j}$ is the total amount of pheromone deposited on the i, j edge [4].

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \triangle\tau_{ij} \qquad (\text{ where } \triangle\tau_{ij} = \alpha + \beta) \quad (2,\ [3])$$

$\tau_{i,j}$ is the pheromone update value from node i to node j. M is the number of ants that passed the edge, and $\triangle\tau_{i,j}$ is the change of pheromone that will be added to the edge. α is a parameter to control the influence of pheromone and β is a parameter to control the influence of the edge weight [2,12]. α is represented and simplified to 1/edge cost [18]. β is a degree to which the path is updated as a constant value (e.g. 0.1). When all ants have finished searching the environment once, the length of the path taken by each ant is calculated and the shortest path is saved. At the same time, the pheromones on each edge are updated according to the heuristic factor weight. The shorter the iteration path, the more pheromone is distributed.

Outlined in the literature, there are many process and different ACO implementations. As visualisation aims to observe different parametrisations of the ACO algorithm, 3 unique ACO methods will be implemented. The overall structure of the models will use the Max-min system [3,4,9].

$$\tau_{ij} = [\rho * \tau_{ij} + \triangle\tau_{ij}^{best}]_{\tau min}^{\tau max} \quad (3,\ [3,4])$$

ρ is the pheromone volatility coefficient, and after all ants have updated the path, the evaporation phase reduces the values across the whole environment. The pheromone evaporates with each iteration, therefore the concentration of pheromone (heuristic factor) for the shortest path would be higher than longer paths [2,5,3,6]. This method of depositing pheromones and evaporation over time will lead to better paths, eventually converging on an optimal path. Eq. 3 controls the maximum and minimum pheromone amounts on each edge trail. To avoid stagnation of the search algorithm, the range of possible pheromone amounts on each trail is limited to an interval [τmax, τmin]. All edges are initialised with a high pheromone amount to force a higher exploration of initial solutions. When the search phase has ended, edge pheromone values can only be reduced to τmin when evaporated, to ensure that paths remain viable. If the system had no τmin value, then paths not visited would be reduce to near 0, and never be chosen. This methods ensures all paths are viable in the heuristic path choices.

The 3 algorithms this project will implement are:

- Daemon Phase [3]

The ants in the ACO algorithm communicate locally using the phenomenon component. Only the iteration best ant path is allowed to add pheromone to its trail. The best path in each iteration is maintained for each ant. This step is very similar to elitism in evolutionary algorithms [3]. The iteration best ant solution deposits pheromone on its trail after every iteration (even if this trail has not been revisited). The elitist strategy has as its objective directing the search of all ants to construct

a solution to contain links of the current best route [9]. This is chosen because each iteration should propose better solutions every time, and this is the original version of ACO developed.

• Rank-based ant system [3,11]
All solutions are ranked according to their length. Only a fixed number of the best ants in this iteration are allowed to update their trials (e.g the top 5%). The amount of pheromone deposited is weighted for each solution, such that solutions with shorter paths deposit more pheromone than the solutions with longer paths. This combined pheromone distribution allows more coverage of the environment with the next iteration, thus more likely better paths. This was chosen because of its reflection to how real world ants create paths.

• Local A* heuristic [4]
The A* algorithm has a faster merit-seeking speed because of its unique valuation function, where the A* path-finding will preferentially move towards the endpoint each time [4,11,13].

$$\eta_{ij}(t) = \frac{Q}{g(n) + h(n)} \qquad (6, [4,11,13])$$

g(n) is the distance of the current location from the origin point and h(n) is the distance of the current location from the destination point [4,11,13]. The valuation function is used as heuristic information in order to search the global path and speed up the convergence. The distance is calculated by euclidean distance. The edge choices are ranked by using equation 6 and the pheromone value is biased for the higher ranked edge, where the random choice function will choose using the updated pheromone values.

3.3 Visualisation Design

This section outlines the experimental setup for the visualisation system [7]. The aim of this project is to visualise the paths made throughout each iteration of the ACO, showing how the ant explores the environment using the above equations to perform each path iteration. The visualisation will reflect the ACO behaviour of pheromone deposit, where a path that is traversed moreover will have an increased pheromone amount. Subsequently, the evaporation of pheromones will be reduced (limited by the minimum pheromone value), where less travelled paths will be eventually removed.

The purpose of this is to allow a decision maker to bring their own intuition about how the problem works so that they can better inform the route optimisation process through interaction [7]. The methods to allow such an implementation will include the Leaflet.js library [17]. Leaflet.js is the leading open-source JavaScript library for interactive maps and is a power-full tool [17]. Real world map paths can be explored by the route optimisation algorithms, and the output of the iteration paths will be seen throughout the process, ultimately over each iteration a clear best path is produced. The process of visualisation can be applied to many research industries, such as traffic simulation, construction or mobile robotics. Illustrating the paths gives insight as to why certain paths were taken. For example, analysis of logistic route data provides understanding by revealing different aspects of the data to help inform decisions.

The designed system is to be integrated in Django, a full-stack web development framework for Python [16].This choice was motivated primarily due to Django's HTTP url request handling and has front-end design tools [16]. This compatibility with the Leaflet.js library [17] allows the creation of HTML web pages with interactive visualisation, and using integrated functions will allow fast render requests for processes and outputs from the ACO algorithms that can be rendered at the same time.

3.4 Measuring performance

In order to demonstrate the potential of the visualisation tool, it will be demonstrated with data describing the optimisation of test problems of differing complexity [5, 18]. Each test problem will have its own characteristics that are designed to make the problem challenging in a specific way for

the ACO route optimiser to solve. Real world maps and road network have no limit on applicable difficulties of route optimisation [5].

Problems will be selected to show the ACO working well in some cases, and poorly in others, so that the reader can observe the difference and understand the process behind the success or fail using the selected parameters to create such outcomes. These problems will be defined as areas of a city or town differing in access ability (walking or driving). These areas will differ in node size and edge amount, as towns will have less compared to large cities. The discussion around the example visualisations presented in the Testing section is structured in such a way that the visual properties and the ACO search behaviour demonstrated are explained in the context results. Testing will also convey how the ACO paths are found during the optimisation process [5,7].

In the paper by Walker and Craven [7], other types of meta-heuristic optimisation algorithms have been explored with discussions to approaches with parameter tuning [12]. The ACO algorithm behaviour is all automatic, therefore the reflected algorithms explored should have a determined parameter tuning ability that the algorithm follows.

Aisham *et al.* [2] and Sangeetha *et al.* describe multiple combinations of different parameters through testing, and demonstration the effect on the output. However, the exploration of the environment and the process of how the paths were made are never seen throughout the literature. Only the final output length or iteration time taken is presented. There is a gap in the literature for path-finding visualisation, the processes in-between the final solution are not detailed and the parameter acquisition are explained only as optimal for the given area problem [12]. In order to measure the performance and meet the evaluation criteria, multiple dataset areas will be selected, each with unique node sizes, edges and network type. Different parameters will be used to find the most optimal selection for each type of ACO route optimisation algorithm.

4 Development Process and Implementation

This section describes the implementation and the process construction of the final project result. The following sections focus upon the creation of the software components designed and the implementation of the operation procedures developed in order to meet the specification. It describes how each element was produced along with any challenges that were encountered. The choices of methods used that were made during the process, including the architecture of this project, will be justified. For a visualisation application, this section also discusses and justifies the user interface design choices.

4.1 Django system

This section covers the system application architecture implementation. As previously stated in the design, this project's two major frameworks are Leaflet.js and Django [16,17]. The installation and setup of these was critical in the success of this project. Leaflet.js used for interactive maps can be implemented to a front-end framework by using a script import. This can then be used in a JavaScript file that utilises the Folium map packages in the library for the map display and interaction.

The challenge of visualisation must allow seamless and quick response requests from the logic components to the display. To solve this, the base HTML template implements JavaScript to handle the request responses using AJAX, meaning the page does not have to be refreshed for the changes to be reflected on the web page. Each ACO path visualisation would be lost if the page refreshed, through initial prototyping using HTML maps would be extremely slow using a get request to refresh and update the next chosen for all previous paths. Using AJAX allows the update of the same page for each iteration, the storage of page variables without loosing them, and path processing speed.

Offloading computation to the front-end, such as area selection verification, functions can utilise the built in Leaflet functions to verify an area of OpenStreetmap data [1,20]. This overcomes the problem of many response requests to the back-end and these methods allows quick front end response and verification while on the page before sending the necessary functionality data to environment generation for the ACO logic.

Creation of a functional application in Django requires consideration of techniques before implementation, including the pipeline of the responses from the ACO path finding logic [16]. Particular URL paths must be thought out in order to call function-based views. These views are what render the HTML files and are a process pipeline providing the context data from the back-end to the front-end.

Python framework for Django is used to build the back-end of the application [16]. Django Rest is a robust and adaptable toolkit for creating and building web applications with very few lines of code [16]. Django was chosen since the back-end needs to run the ACO heuristic learning model to produce solutions for the visualisation. Python was chosen for the simplicity of integration with the OSMnx package [1] and numerous useful in-built libraries for the implementation of the route optimisation algorithms.

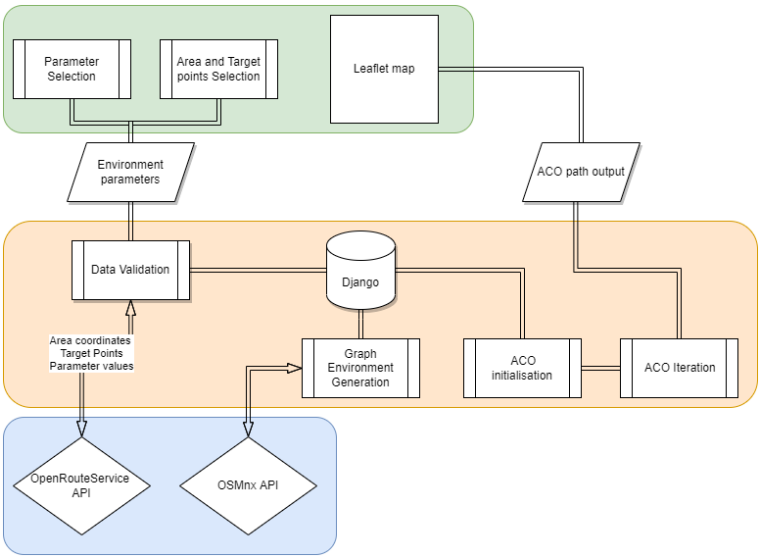


Figure 3: Simplified architecture of system. Green encapsulates the front end functions. Orange encapsulates the Django and ACO processes [16]. Blue encapsulates the API request interaction. Parallelogram represents data

The OSMnx API allows the generation of area point data in the form of a JSON object response. This was selected because raw data point modelling from OpenStreetMap [20] and OpenRouteService [19] would have required filtering and preprocessing. Through initial prototyping, this would have taken too much time to successfully implement. The OSMnx API provides data for world road and street coverage in a variety of methods. Additionally, OpenRouteService [19] was chosen as this service offered a free API Key for isochrone area environments [10]. This searches a given location in a radius for every possible street that can be accessed. This was extremely useful for the area validation before the creation of the environment, especially if the area is large. This also allowed the integration into the front-end validation. Using a back-end validation response would be slower.

Figure 3 illustrates how the different technologies interact with one another to produce a fully functional application. The front-end allows an interacting user to select the environment and parameters. After validation, this data is pipe-lined using AJAX to the correct Django Url path. The area environment data can then be requested and using the OSMnx response the data is used to generate the graph with the initialised parameters. The ACO logic will then return route coordinate data for the paths taken, which can then be visualised in the Leaflet map [17].

4.2 Environment modelling

Before creating any path finding architecture, a suitable environment must be created first that allows such algorithms to be developed. This environment must also be able to process and visualise the output to the same environment. The main problem encountered was how to successfully achieve an environment that can be traversed by an ACO model while being visually represented simultaneously to accurate degrees of real world environments. Therefore prior to making the environment an analysis of available geometric datasets and geographical packages were explored.

From the literature review, Aisham *et al.* [2] used data-point modelling to create every possible path in an area, and these points acted as nodes that the ACO algorithm could utilise to traverse this environment. To create an undirected graph that would reflect the real world roads, a mesh grid matrix was generated from a given area, and each target location was uniformly generated. However this was clearly to much of a simplification and had no real accurate descriptions of data. The produced graph was a simple grid based environment, which justified above is not suitable for this application.

This challenge was overcome by using the OSMnx python package. An area can be taken from a defined bounding box area, and the API response JSON data of the map is used to construct street networks from OpenStreetMap [1,20].

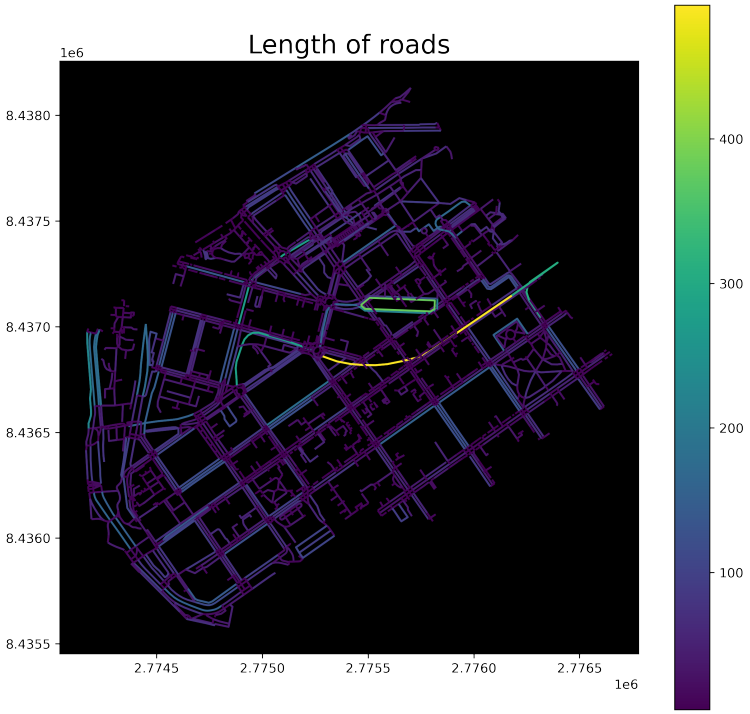


Figure 4: Lengths of edge data [1]

An environment must be generated reliably, and for the specification it must have real world data. Initially, the OpenRouteService API was used to generate the data-points and the boundary area for the defined environment [19]. To ensure that the area was defined well enough, isochrone area wraps were generated from the bounding area box (Bbox) in a 100m radius. This was a future proof choice in the event that if a user selected a target too close to the boundary, the available road network might not reach the target point. Isochrone areas are bounds enclosing every possible street reachable within a range [10]. Therefore adding this extra area layer would allow further exploration and possible better paths, at the cost of a larger search area previously defined [18]. This area is also beneficial, as the OpenRouteService isochrone API request will give a JSON response for the map area, thus having a built in verification of the chosen area [10, 19]. Each point, including the origin/destination points, are used to calculate an array of isochrone data points, where the extrema points can be defined.

OSMnx geocodes this search area using OpenStreetMap's Nominatim API and constructs a polygon from its boundary geometry [1,20]. It then buffers this polygon and downloads the street network data within this buffer from OpenStreetMap's Overpass API [20]. Next it constructs the street network node data. For one-way streets, directed edges are added from the origin node to the destination node. For bidirectional streets, reciprocal directed edges are added in both directions between nodes, as a normal road [1].

OSMnx then corrects the topology, calculates accurate degrees and node types, then truncates the network to the original, desired polygon road lines [1]. This ensures that intersections are not considered dead-ends simply because an incident edge connects to a node outside the desired Bbox geometry. The geographic area of Kamppi, Helsinki, Finland is accurately plotted, weighted by road length shown in Figure 4.

For the graphical traversal, Figure 5 outlines how the network is constructed and connected. OSMnx performs topological correction and simplification automatically. Simplification is essential for correct topology as OpenStreetMap nodes are inconsistent: they comprise of intersections, dead-ends, and all the points along a single street segment where the street curves [20]. For visualisation, this is important as nodes connected must match the real world data (one node edge cannot 'teleport' to another across the map).

Using this data, a directed graphical environment can be created. For the directed graph G(N,E), each node (N) is created as an object, with a node number, and it geographical [x,y] location. Denoted by the topology of the graph, each node has an adjacency list, consisting of the other nodes it is connected to. This adjacency are the edges (E) connecting the graph together, which also stores the geometry lines for each edge connection. This edge data stores the length of the path, and for the

ACO model a pheromone value. The alpha update will be calculated using this data.When the graph is initialised, nodes are stored as one object.

To perform visualisation of path finding techniques, the origin and destination (target) points must also be initialised. In order to achieve this, the target points coordinates can be processed against the graphs node locations. The nearest node calculator uses a growing radius circle distances to find the first node that contacts the area of the circle. This nearest node will be refactored into the origin/destination respectively.

When the complete graph is created, the pheromone values are added in preparation for the graph traversal. These values are randomised at the beginning of the graph creation. The chosen maximum and minimum values for the pheromone are also initialised, including the evaporation constant and the heuristic (alpha, beta) values.

In the initial testing development of the path finding systems, the program would be slower to converge in the iteration run time search, often missing the destination node while trying to reach it. To solve this problem, inspired by the natural ant behaviour, the area surrounding the destination node is more enticing (area surrounding food would have high pheromone deposits). Each adjacent node edge connected to the destination the pheromone value is constant, set to 1 after the pheromone evaporation to ensure the ants reach the objective efficiently.

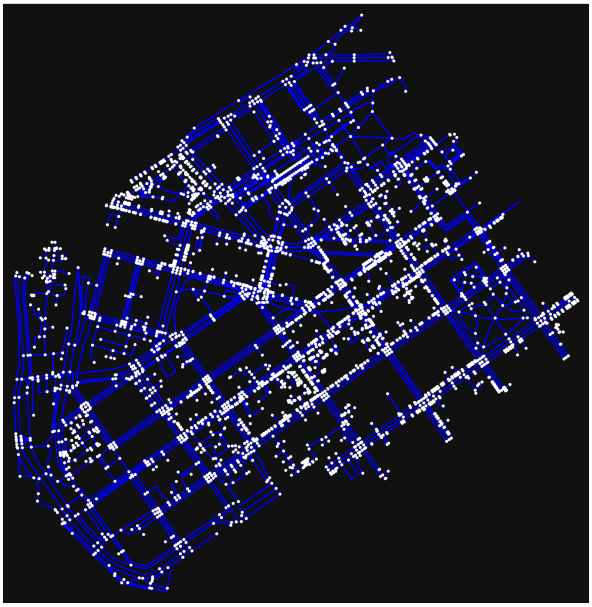


Figure 5: Environment visualisation representation [1]. The white indicators represent passable nodes, where the blue links follow the real world geometry to other reachable nodes

Furthermore, while testing the ACO behaviour another problem was encountered. In some instances of the OSMnx data, one way streets did not have nodes adjacent, as these existed outside the area and never generated. The ant cannot simply 'turn around' and is therefore stuck in this 'dead' node. One solution would go through each adjacency list in the graph initialisation to remove this node. However this takes more time and for larger areas this is not ideal. The solution created evaluated if an ant encounters a 'dead-end' one way street, the running ant search is stopped and all path data is reset for the next iteration. The edge will be removed from the previous node adjacency list.

4.3 Ant colony optimisation search

As detailed in the design section, the scope for this project is to create a system that can traverse a graph using the heuristic framework of the ACO algorithm to produce an optimal path. Therefore the creation of ACO logic that can access and traverse the graph efficiently must be created. The simulated ant objects are initialised at the origin node, where the ACO behaviour is performed to produce passable routes.

Every ant is allowed to traverse the environment, choosing its own path defined by the pheromone heuristic behaviour. Therefore a function must be implemented to allow this operation. Input of the the current node the ant is located on will have possible edges to choose from: for a node N, with possible edges i,j,k. From the possible edges the pheromones of each each are selected denoted by i_p, j_p, k_p. Each value is input into a random choice function, where the chosen edge is influenced by the pheromone value. A higher value is more likely to be chosen. The chose output edge is saved in the ant's memory, to determine which edges where chosen. This edge choice runs continuously, stopping if the ant reaches the destination node where the path taken is recorded.

To determine the best path, each edge distance is totalled throughout the individual ant search

and the best path is the shortest distance out of all the other proposed solutions. However, this project attempts to visualise multiple ACO techniques. Therefore, different evaluation functions must be created to determine these paths for the respective ACO algorithm technique. The elitist method and A* local search identifies one best path [13]. The rank method will determine the top few best paths. The fitness of a path represents how good a solution is, where paths which are shorter in total length have a better fitness than paths which are longer.

For the determined best fitness path, a function is constructed to update the graph pheromones. Each edge visited on the path has the pheromone value updated influenced by the alpha and beta rate. After this the entire graph is evaporated in preparation for the next iteration.

A challenge encountered reflected the real world behaviour of the ant colony. The program searches for the target node until it is found, like in nature where an ant continuously searches. This loop, where an ant will keep searching until it has found the destination target is inefficient. The ant may take too long to find a solution in complex environments, and if the target is found the returned proposed path is usually too overlapped making the pheromone update ineffective for the next iteration. This overlap slowed the convergence by a significant margin, where the final solution is either to scattered or clearly not optimised (going down a street and back again). One solution would be to integrate a post process to the path, where nodes traversed more than once would cut the path between. However this method deviated too far from an ACO model, as the path is being checked twice with takes more computation and time. Another solution would be to increase the evaporation value, but this was also ineffective as good solutions would be lost too fast for the other iterations.

The chosen solution simply limits the ant's traversal ability. As an ant would run out of energy in nature, the simulated ant can traverse up to a limit of the total nodes in the graph. However there is an additional problem. Ants that have reached the targets must be sorted against ants which have not. This is because ants that do not reach the target might have a smaller path than those which have, and therefore being pooled into the same best fitness function, it would return those ants as the best path. Therefore, if the limited energy value is exceeded and the target has not been found, the ant is 'killed' and no path is proposed. A method such as removing the ant from the colony list when 'dead' would not be efficient for the reason the ant object would have to be re-initialised for the next iteration. In the chosen solution, 'dead' ants are instead marked using a Boolean value to establish it has not found the target, and therefore cannot update the graph. These solutions for the ant search allowed better search phases. When a solution is proposed, the path fitness is concise (not too random or sparse) so that for the next iteration, the pheromone deposit is effective. Ants that have reached the target will be pooled together, and the best fitness path is found. If no ant has reached the target, then no graph pheromone update is performed but the best ant for the iteration is still visualised to show the algorithms search progression.

4.4 Djikstra and Depth First Search

An issue faced with the development of the path finding systems was there was no way to show the absolute performance of the algorithm or justify a successful path. It is beneficial to provide specific indicators when comparing algorithms such as running time, computational complexity, accuracy and quality. As this project aims to successfully visualised route optimisation, additional path-finding algorithms were implemented to show the performance. Dijkstra [25] and Depth First Search (DFS) were implemented, to ensure the performance of the ACO path.

Dijkstra's algorithm applies a greedy strategy that employs a breadth-first search to settle the single-source shortest path [25]. A function was created to capture each node span search in the order of the search. As Dijkstra has no iteration procedure the process of visualising this algorithm is shown in this search order [25]. Each node searched is highlighted at its location on the map. The result demonstrates the branching out strategy of the algorithm. A similar process was made for DFS, where each node is searched as deep as possible in the available adjacency list. These two algorithms better show the capability of the ACO algorithm and show how the nodes are visualised on the map. This also allows a better understanding of the fitness of the paths proposed and the computational complexity of each for the environments.

4.5 Front-end visualisation

Once the best fitness path(s) are decided, the path must be processed to visualise the how the ant traversed the environment. From a returned ACO path, the memory of the paths taken are shown as each node number. Every node object has a node number and an adjacency list of the other node ID's it is connected to. Therefore, the returned path acts like a linked list, in which each node will have the next node in its adjacency list. To provide a visualisation of the paths, this node data must be shown on the Leaflet map [17]. To achieve this, OSMnx provides line polygon objects for each edge in a nodes adjacency list. These line objects can be converted into latitude and longitude partitions, being split up into an array of plot points that will match up the geometry of the real world street networks.

There were many challenges in the creation of the front-end visualisation. In order to create a search graph, an area of a map must be selected. Therefore a page was created that displayed an interactive map. Clicking the map generated an area, from which points can be placed inside. To ensure validation, the area must first be converted into a polygon with bounding edges. These edges area sent in an XMLHttpRequest() to the OpenRouteAervice API [19] to check that the area has reachable road data. This ensures that no problems are encountered with the area.

Through initial designs and tests to show the ACO paths, OSMnx was used to provide a method for route path visualisation. The OSMnx API provides matplotlib graphs for the environment (similar to Fig 5). ACO iteration routes are an array of the node ID's visited in the search. Using the node identification and plotting each node visited in order overlapped on the map, the ant path can be visualised. However, this method was extremely slow to update the plotted graph combined with relying on the OSMnx API to return chart plot, and then update the figure. Furthermore, this was vastly incomparable to the specification interaction requirement for the visualisation. The choice to use the line-object data points for the path geometry was significantly advantageous with the ability to display the data in the Leaflet map [17].

To construct the environment graph, another HTML page was created to show the output of the route optimisation data on the Leaflet map [17]. On this page the parameters and the ACO model selections were later implemented, which are used in the generation and construction of the environment and the ACO algorithm. UI interaction functions were created to allow the active user to select system parameter option. The parameters are parsed in the AJAX get request and pipe-lined through the Django Urls to the back-end. When an iteration of the ACO is completed, the geometric data is sent back through the Url and received as a JSON response. This minimises the packet size and JSON is simple to integrate into the JavaScript logic.

The next challenge was how to successfully show the ant path data. Initially, the returned paths were pre-rendered as a HTML file, and then loaded. This was still not ideal as refreshing the page to show the map made functionality very slow and difficult. Instead, by using an Async window timer interval function, the line-object data can be shown without stopping the back-end functions from running. Get requests receive the new path data without needing to refresh the HTML map. However, this could not be rendered all in one execution. Therefore a speed setting was integrated into the visualisation, which allowed the individual nodes visited to be shown at the desired speed.

One essential function of the ACO algorithm is the evaporation of the graph. Hence, an evaporation function was implemented in the front-end to represent the evaporation of the pheromones in the environment. Over each iteration this function demonstrates the convergence to an optimal path by removing the the old less traverse iteration paths from map that have a lower fitness. Furthermore, the paths were then adjusted to different colours and opacity. The more an ant traversed an edge, the layers of opacity would increase. Inversely, edges not traverse much would be seen as opaque and not a desired location, eventually being evaporated. Combined with the evaporation, the edge importance and paths are demonstrated through each iteration.

To confirm the convergence of the program and show the progression of the path fitness overtime, chart.js was implemented to demonstrated this. Iteration fitness is plotted on the graph to show the progression overtime. This plot was implemented for better understanding of how the visualisation is correlated to the paths fitness. This also provided specific numerical indicators when comparing algorithms.

5 Testing of Parameters and Visualisation

This section outlines the processes of evaluating the functionality of the code to ensure that the software application meets the requirements. Three different map locations were used with differing node and edge sizes. The locations were tested for the alternate driving, walking and bike data available. The high environmental diversity of cities, towns and parks facilitates testing of how well the system performs in the generalised environments. The varying complexity (difficulty) of the locations provides a good scale to incrementally test the performance of the system. Larger environments are expected to be initially difficult for the ACO as is is a larger search space, but should converge on an optimised path at the end of the iterations (given optimised parameters).

Parameters	Values	Constant
Number of Ant objects	10, 50, 100	50
Number of Iterations	25, 50, 100	75
Beta rate	0.05, 0.1, 0.15	0.1
Evaporation rate	0.2, 0.4, 0.7, 0.9	0.8
Minimum Pheromone	-	0.05
Maximum Pheromone	-	0.91

Table 1: Table of the parameter testing

5.1 Datasets

Areas of varying size and node complexity were acquired using the visualisation system for the purposes of demonstration the ACO path-finding and route optimisation visualisation and to inspect the ACO parameter functionality. The areas selected are converted into a search graph by the projects Django system [16], using the OSMnx [1] API to generate the road map data. This area is denoted by an area boarder which is then validated to determine if the area has road map data. There were 23,946 unique nodes and 54,918 different edges collected in total. As shown in Appendix B, the following figures are the three datasets that are used in this section. To demonstrate different node environments the street network type are set to walk, bike and drive for the consecutive dataset maps. The system location and graphical representations of the environment are also generated for each dataset.

5.2 Experiments

Due to the time constraints encountered during the development of this project, the tests made were unable to carry out large data tables and user testing. Instead this section focuses on the efforts into exploring the performance of the ACO route-optimisation, and how well this is visualised. After this test, a focused set of mathematical tests will be explored into the effects of the parameters. Table 1 lists the parameter settings for the computational experiments. Datasets were acquired from map data discussed in subsection 5.1 [2]. The Constant value used for the ACO parameters testing are outlined in the last column.

The computational results are based on various numbers of artificial ants, number of iterations, beta rate and evaporation rate to evaluate performances of the ACO algorithms. The objective function aims for a minimum travelling distance in meters [2].

Firstly, the visualisation process is tested to demonstrate the capabilities of the ACO program and show the paths through the iterations. For this demonstration, each iteration best ant path is visualised on the map. The line objects are drawn with a set opacity. The more a path is traversed, the deeper the colour gradient. Through evaporation the less travelled paths become more transparent, then removed completely. The ACO algorithm eventually converges on an optimal path, highlighted by a red path as the final iteration.

Figure 6 presents the visualisation of the performance of the ACO algorithm instances of the route optimisation process. These figures show 5 snapshots taken during the execution. This visualisation uses the elitist method of ACO. where the single best ant path is shown throughout the iteration.

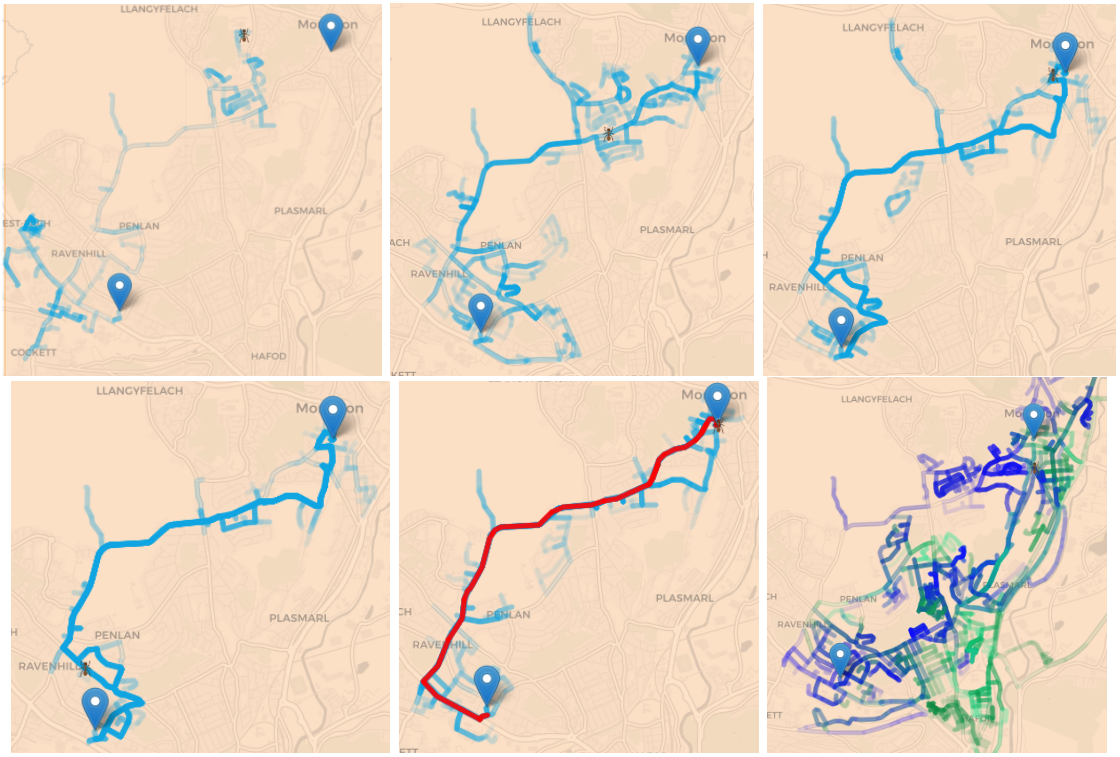


Figure 6: The process of the visualisation and search methods of the ACO algorithm. (Parameters = ants: 50, iter: 75, beta: 0.1, evap: 0.75, min-max: 0.05-0.9 for the Plasmarl, Swansea area

The first figure at iteration 3 is the initial search phase where the ant objects search at random for the destination target. The second figure at iteration 10 shows the ant location being found. The third figure at iteration 25 shows the collection of ant producing a visible path. The fourth figure at iteration 50 shows the evaporation of the less travelled paths more significantly reduced/removed and the fifth figure at iteration 75 with the final optimised path [7,2]. The sixth figure demonstrates the exploration of the other ants in the colony, each with a differing colour path. For the elitist ACO algorithm, only the best path for the ACO algorithm is allowed to update the graph. This shows how the environment is collectively explored initially to present multiple different paths. From these, the best path is chosen resulting in the eventual convergence to an optimal path, as better solutions are found each iteration.

5.3 Experiments Validation

		Number of Ants			Number of Iterations			Evapouration rate				Beta rate		
Dataset Area	Algorithm	10	50	100	25	50	100	0.01	0.35	0.75	0.99	0.05	0.1	0.2
A	ACO-Basic	3716	2618	2302	3095	2535	2468	30040	13398	2096	25991	2406	2356	2514
	ACO-Leaders	2188	9608	16360	6394	4905	4574	14497	3853	4855	20335	2534	10898	11415
	ACO-Astar	5812	2635	3012	2701	2266	2042	34772	34128	2707	16532	7477	2262	2401
	Djikstra	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400	1400
	DFS	6467	6467	6467	6467	6467	6467	6467	6467	6467	6467	6467	6467	6467
B	ACO-Basic	331290	13022	8309	14540	11206	9571	198828	161696	8474	91418	11532	8669	9669
	ACO-Leaders	16058	12632	94136	15323	12534	9530	300430	19109	14908	176195	11285	12070	13377
	ACO-Astar	253713	185307	9335	17692	11187	10653	113488	56040	10364	107475	143642	12864	9463
	Djikstra	6554	6554	6554	6554	6554	6554	6554	6554	6554	6554	6554	6554	6554
	DFS	26454	26454	26454	26454	26454	26454	26454	26454	26454	26454	26454	26454	26454
C	ACO-Basic	147358	19478	13408	20605	21061	17413	492218	186836	16576	385658	48110	13709	18567
	ACO-Leaders	180381	61102	24460	123938	70881	35160	249074	42681	37447	233267	36926	83423	281500
	ACO-Astar	402244	21423	15836	41418	28002	12563	438043	451763	30536	287145	92039	18462	49234
	Djikstra	11091	11091	11091	11091	11091	11091	11091	11091	11091	11091	11091	11091	11091
	DFS	124851	124851	124851	124851	124851	124851	124851	124851	124851	124851	124851	124851	124851

Table 2: The computational experiment results form the datasets

Through the iterations of the ACO algorithm, this system demonstrates how the simulated ant search and collectively combine the path data to form a smooth optimised route (no repeated node edges, or a path loop). In Figure 6, the system effectively conveys this method of path-finding through the visualisation iterations. The desired paths can be seen to be more heavily outlined and the evaporation process removing the less traversed paths are steadily removed (depending on the pheromone evaporation value). At the end of the iterations, a clear path is outlined.

The computational experiment results are shown in Table 2. The result demonstrates that each individual ACO parameter can affect the result of the travelling distance and its affect on the optimal paths that needed to be taken [2]. Dijkstra path search was implemented as well as Depth First Search with the performance of the ACO [25]. The larger dataset area allowed more possible path selections and possible routes that could be one of many best paths. Through testing, certain parameters for each ACO algorithm allowed the algorithm to perform better and was faster to converge on an optimal path.

5.4 Final Results

Following the initial testing, the ACO can be tested on the each dataset of increasing complexity and area using the most optimised combination of parameters for each ACO algorithm. This will give an insight into the overall ability of the best performance of each ACO model, and how effectively this is conveyed through the visualisation process. The fitness overtime is plotted in the visualisation system's graph, shown in Appendix C. All tests were conducted over 75 iterations an an range of 0.05-0.9 for the minimum and maximum pheromone deposited.

Dataset	Algorithm	Number of Ant objects	Beta rate	Evaporation rate	Best fitness
A	ACO-Basic	55	0.1	0.75	1854
	ACO-Leaders	25	0.08	0.8	2031
	ACO-AStar	55	0.1	0.75	1943
B	ACO-Basic	65	0.15	0.75	7945
	ACO-Leaders	45	0.04	0.75	8122
	ACO-AStar	65	0.12	0.7	7819
C	ACO-Basic	75	0.12	0.85	13328
	ACO-Leaders	45	0.11	0.75	14563
	ACO-AStar	65	0.15	0.8	13145

Table 3: Table of the parameter testing

The number of ants affects how much of the graph can be explored and thus the generation of path to the target. Higher ACO ant numbers performed better overall, as this allowed a better search of the area for both ACO-Basic and ACO-Astar. However ACO-Leaders suffers from overpopulation due to the top 1% being able to add pheromones to the graph. This results in too many paths that are too diverged to accumulate into an overall path. The A* heuristic function determines the best direction from the distance to the target, where a chosen path might be close but lead to a dead-end [13]. This ultimately slowed the convergence speed due to this.

High evaporation rates results in more exploration behaviours of the ants, which results in ants getting lost. Conversely, low evaporation rates result in less exploitation behaviours, which would lead to in an inability to acquire the optimal path as pheromone trails which represent the previous bad solutions are not eliminated quickly enough by the pheromone evaporation that they are not discovered again [2]. Thus, the adaptation capabilities of ACO rely heavily on the pheromone evaporation mechanism. Seen in the visualisation, the paths are not removed enough to visibly see the best paths. The parameters typically used in the literature are normally fixed, which means they are more generalised to solve problems. By allowing alterations to different values, the performance is seen and allows a better perception of how the ACO mechanism explores and generates better paths.

Real-world data is usually more heterogeneous[5]. Therefore the algorithm's performance might depend on the underlying structure of the street networks of the area. Through testing, as the

complexity of the environment increased, the performance of the visualisation system slowed due to the increase of the node size, as the ants explored the environment. As the ant objects started to use exploitation of the paths, this resulted in less running time for exploration which quickly increased the performance.

6 Final product Description

The final product of the system is as follows. Shown in Appendix A, these figures demonstrate the final look of the systems interface for the Django front end, where the interactive Leaflet map is initialised [16,17]. On this page, an area can be selected by clicking the map and the area size is defined using the area range selector. When a valid area is selected, the toggle switch allows the selection of the target points within the area. The type of node generation is selected from the drop down (drive, walk and bike). If the target points are not desired, the 'Clear' button will remove them. If area, type and points are desired, then the 'Get points' button will generated the environment area.

On the Visualisation page, selected from the navigation bar, the generated area is shown along with the two target points. On this page, algorithms and parameters can be selected before running the path-finding and visualisation systems. 'ACO-Basic' will set the daemon phase ACO program [3], 'ACO-Leaders' will set the Rank based ACO program [3] and 'ACO-Astar' will set to run the A* heuristic ACO [4,13]. When correct parameters and an algorithm is selected, the 'Run Visualisation' button will initialise the node environment and start the path-finding system, where returned paths are shown on the Leaflet map [17]. The paths generated can be altered to the appropriate speed setting to increase or decrease how fast the paths are shown in real time. The 'Reset' button will clear all paths currently on the map (not ending the run time). All iteration path fitness values are recorded on a chart below the Leaflet map on the visualisation page [17].

7 Final Product Evaluation

This section will evaluate the final system using the results from the testing section, and compares the system to existing literature. Firstly, the system was able to visualise the process of the ACO path finding algorithm, and show each iteration for the pheromone distribution and evaporation. The paths traversed were correctly shown, using the transparency line-objects to correctly visualise the ACO path heuristic behaviour. In figure 6, each iteration defines the process of the path-finding effectively and clearly. The heavier line-objects are defined from the rest of the paths generated using the pheromone deposit function, and the evaporation adds to this process by removing these less travelled paths. Furthermore, the colony behaviour is correctly shown using the system toggle method, which shows the ants with the best fitness paths traversing the environment, outlined using differing colour gradients. From further testing, by showing the collective ant behaviour the best defined bath is still visibly shown, uniquely by the combination of all the colours of the ant paths. Overall, the results are promising and useful to demonstrate the iterations for the visualisation of route optimisation.

Compared to existing literature, the visualisation is far more detailed than the presented environments by Chen *et al.* [4]. The proposed grid based graph does not offer the same detail for real world environments and the size of the nodes search-space is significantly less. Weise and Mostaghim [5] successfully explored using simplified OSMnx data used by path-finding benchmarks where final route outputs were shown. The area had a significantly larger node area, but the process of the optimisation was not shown. Similarly Aisham *et al.* [2] used point data maps and applied ACO for route planning in university campuses. While parameter testing was demonstrated, the final path outputs were not. This project system solves these issues, with visualisation, the affects of the parameters can be seen through the optimisation process.

When exposed to more complex environment datasets, the ACO still performed well even with a higher node density. The resulting increase of nodes meant there was more possible edge paths to go, which therefore had an expected eventuating slower convergence. All optimised parameter best paths produced were significantly better compared to Depth first search. However, when the ACO is

compared to the results from the Dijkstra algorithm, the path fitness were always above the end value [25].

It was hypothesised that a combination of the following factors can be attributed to the performance outcome of the ACO. Due to the ACO nature of path randomness, the path with a more direct route (smaller node amount yet same distance covered) is preferred to route with a larger node size. With more nodes the ACO is more likely to go astray, as more nodes means more edges where the randomness of the ACO edge choice means there is a still a possibility to go the non-ideal way with so many of these choices. Paths which have these early additional disordered edges will have a higher path length and never are allowed to rest pheromones. There are solutions that could improve these issues, discussed in section 8.1. Nevertheless, the ACO still performs well and does return a smooth final path. From the test, an interesting advantage to the ACO is that multiple paths can be generated with similar path finesses, where Dijkstra only returns one path [25]. Therefore ACO has a better adaptability and generalisation of an environment, in the event a path is not possible in the future.

Both visualisation and path-finding would not be possible without the OSMnx data environment generation [1]. The performance of the systems correctly gathers real world data and create a mirror geographical environment for the ACO programs to traverse. The environment contains the same road regulations such as bridges, one-way or two-way streets, and can perform under different distributions of certain road types, reflecting the hierarchical properties of real-world examples [5]. The system effectively captures the true heterogeneous characteristics of real world street maps, and effectively removes any possible edge errors.

8 Project Critical Assessment

The following critical assessment of the route optimisation visualisation project will be comprised of three sections. Initially, the project's successes and drawbacks will be assessed against the project's requirements set out in section 2.2 to 2.3 set previous to the implementation of the project. The assessment will be continued against the explicit evaluation criteria in section 2.4. Any departures from the original requirements will be justified.

The projects's limitations will then be stated, alongside potential strategies to solve these shortcomings. Further improvements for potential future work will be outlined, joining a reflection outlining any potential changes that would need to be considered if the project were to be continued or attempted differently by a another party.

8.1 Assessment

Initially, the functional requirements outlined that an implementation of the ACO algorithm would be used to visualise route optimisation. This has been successfully achieved. The ACO algorithms implemented: ACO-Basic, ACO-Leaders, and ACO-Astar all use the naturalistic and heuristic behaviour of the ACO algorithm in their unique functionality constraints, and successfully have been implemented for path finding. Furthermore, the functional requirement also describe baseline search algorithms to test and compare performances to ensure successful functionality. The Dijkstra's algorithm and Depth First Search were also implemented, thus completing this requirement further [25]. These search algorithms (with optimised parameters) were all able to generate optimised paths to between the target nodes. All the algorithms were implemented using Django through validation of initial designs and testing, as this pointed to the best method of realising the requirements [16]. This also an important consideration when designing the nonfunctional requirements, which stated a User interface. This requirement justified the system that would be created to allow the implementation of each functional requirement. From an personal code experience standpoint, Django was the best way to implement this project.

Additionally, the OSMnx [1] API could be implemented in the Python programming language, which compelled further with the choice of Django [16]. The design of each requirement during the development was heavily influenced by the Django system structure. OSMnx provided the data for the real world graph generation, which allowed the creation of the search graph [1]. The project system creates an accurate reflection of the data in a form that allows path finding to be possible. This is

therefore a successful implementation of the graph generation environment, meeting the respective functional requirements. From each of the datasets, the graph is far more complex than that of a grid based environment explored in the literature review and evaluation, and the produced graph was correctly rendered for the visualisation system.

Leaflet.js was chosen for its tools and its interactive capabilities [17]. Additionally, its integration into the Django front-end graphical user interface was simple and effective. The creation of a base HTML file allowed the addition of the leaflet API and map, and thus allowing additional functions for the Leaflet path visualisations and parameter selections [17]. Linking the necessary CSS and JavaScript plugins, the layout and structure of the pages is uniform across both pages, shown in Appendix A. In terms of layout elements, the interactive maps, menus, buttons and parameter inputs have been implemented successfully, fulfilling the user interface requirements respectively.

The visualisation system effectively demonstrates the ACO paths on the Leaflet map [17]. The paths are distinctive for which paths are desired and traversed more using the transparency overlays of the edges traversed. The evaporation of less fit paths is effectively shown, with the convergence to a best path demonstrated. The final route is is clearly shown, and thus satisfies the visualisation requirement.

It can be concluded that the implementation has succeeded in all respects when analysed against the project requirements. The system performs each of the three main functional requirements, with unified cohesion and efficiency. The UI system that was specified in the non-fictional requirement were also successfully implemented, where parameter options for the visualisation of the route optimisation system can be effectively demonstrated.

8.2 Limitations and Proposed Solutions

While successful in the fulfilment of the specified requirements, there are significant improvements could be implemented if there was additional time to work on the project.

The system graph generation could be improved. Collection of the data from OSMnx has to be converted into a usable connected graph. For larger datasets, this takes longer to convert the data (not a large time frame but an undesired length). Currently the limit of choice area is 5000m x 5000m, as any larger area starts to affect the performance. A possible solution would utilise the isochrone areas, by drawing a connecting line between the target points and finding isochrone areas along intermittent points upon the line. This would create a direct path area section, and thus a more direct area to origin and destination where this area would look more rectangular in shape. For the current system, unless the target points are in a corner, most of the area is never fully traversed (for larger datasets), and the energy function ensures a limits of search distance. This solution would create a better graph map, which could allow a greater distance of traversal with the same node size as an area which is square.

Another limitation is the ACO traversal path choice. As the ACO behaviour is random, it is seen from the testing that a path with a higher node count results in a higher possibility of non-desired edges being chosen, which reduces the fitness of the path. This results in less direct paths yet shorter are selected. Improvements to the ACO models would assist in convergence speed, and solve this limitation. Implementing a direction function would allow the ACO models to determine a better edge choice. If a possible path had many branches, then utilising a direction function would allow the ant objects to better choose a path in addition to the pheromone choice. The ACO-AStar heuristic function determines the best node from the distance to the target, but suffers from being unable to determine dead ends. Therefore, a direction function would greatly benefit the ACO algorithm, but would have to be tested heavily to determine the benefits.

Furthermore, the ACO is iteration based and each ant object searches and returns a path. When all ant objects have been run, the best path is determined. Then the path is visualised. Understandably, this is not efficient in the run-time performance, as the next ACO iteration waits until the visualisation is finished before running the next iteration, in a waiting switch function. This switch function ensures data is not overwritten or erased. To overcome this limitation and increase the performance of the system, each iteration path(s) returned would be added to a queue. The visualisation function would take the first element to process, while simultaneously the ACO path finding is still running and

adding the new paths to the queue.

8.3 Potential Future Improvements and Additional Features

The creation of the solutions in the above section would adversely improve the overall project, and solve many of the challenges that the current system faces. This section will briefly focus on possible future improvements and additional features that do not exist in the system yet.

While the UI performs to a good standard and satisfies the requirements, improvements could be made. The option to save path solutions would allow comparisons of differing solutions, and see which solution best suited the type of situation. The solution could additionally be pre-loaded to see if there are further options available. The history of each iteration could also be saved, to see the specific progress of the path. Adding further UI systems would improve the interaction of the system. A database structure would need to be created in order to achieve this. This feature would not be too difficult to implement in the existing Django system, as databases are supported [16]. UI designs would further be changed to a more user friendly design, such as a more defined colour scheme and further text descriptors for each of the programs. Furthermore, to enhance the ACO algorithm an addition of more feature control for each system would be beneficial. For example, controlling the ant energy function would force better solutions to be discovered sooner or allow more searching behaviour in the environment. Through doing so, more paths could be generated under these constraints. Finally, more target points could be added to the graph to allow more searching abilities or force a more specific route optimisation. The ACO model would have to be significantly altered to implement this, and further testing to ensure correct performance.

9 Conclusion

In conclusion, the main objective of this project has been to create a fully functional system that visualises the ACO algorithm searching behaviour applied to route optimisation. The literature review summary has introduced the fundamentals of the ACO techniques and methods used in path finding applications and existing visualisation systems. In addition, the functional and non-functional requirements were stated and were taken into consideration when developing the preliminary project and when evaluating final product. Functional requirements consisted of the mandatory features of the application that were critical to the performance and success of the project, while the non-functional requirements consist of criteria for the system's attributes.

The final project system has been achieved through the development of the Design, Implementation and Testing phases over the academic year. Initially various techniques were tested and prototypes were created to decide on the best methods to implement this project successfully. Ultimately, the decision was made to implement the Django and Leaflet application with the outlined ACO algorithm integration to achieve this [16,17].

These development choices were selected because of the simplicity and functionality of the separate systems and APIs that could be joined through the Django system efficiently [16]. The design divided this project into three components: the ACO path-finding algorithms, the graph data collection and generation and the visualisation render application. The application visualisation front-end took advantage of the JavaScript Leaflet [17] library for the map interactions and OpenRouteService API [19]. In the development phase, the combined functionality of the overall system was then and tested successively, where cases of errors and vulnerabilities were identified and prevented from carrying forward.

Overall, this project has been successful in the creation of a visualisation system for the ACO algorithm. This project has contributed to existing knowledge for the ACO algorithm and behavioural exploratory analysis of the parameter tuning in complex visualised environments. This project also offers a detailed view for more heterogeneous environments than those in existing systems explored in the literature review. This project also has potential for future work, which may provide more applications and insights of path finding and route optimisation using the ant colony optimisation algorithm.

References

[1] Boeing, G. 2017. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks", *Computers, Environment and Urban Systems*, vol 65, pp. 126-139. DOI:10.1016/j.compenvurbsys.2017.05.004

[2] Aisham, R., Yusoff, M. and Kadir, N. 2019. "Traffic Route Optimization for University Students with Enhanced Ant Colony Optimization Algorithm", *International Journal of Advanced Trends in Computer Science and Engineering*, vol 8, pp. 2668-2672. DOI: 10.30534/ijatcse/2019/121852019

[3] Mirjalili, S. 2019. "Ant Colony Optimisation", *Evolutionary Algorithms and Neural Networks. Studies in Computational Intelligence*, vol 780, pp. 33–42. DOI: 10.1007/978-3-319-93025-1_3

[4] Chen, L., Su, Y., Zhang, D., Leng, Z., Qi, Y. and Jiang, K. 2021. "Research on path planning for mobile robots based on improved ACO" in *Conf. 2021 36th Youth Academic Annual Conference of Chinese Association of Automation (YAC)* pp. 379-38. DOI: 10.1109/YAC53711.2021.9486664

[5] Weise, J. and Mostaghim, S. 2022. "A Scalable Many-Objective Pathfinding Benchmark Suite", *Transactions on Evolutionary Computation*, vol 26(1), pp. 188-194. DOI: 10.1109/TEVC.2021.3089050.

[6] Roy, S., Biswas, S. and Chaudhuri, S.S. 2014. "Nature-Inspired Swarm Intelligence and Its Applications", *International Journal of Modern Education And Computer Science*, vol 6(12), pp. 55-65. DOI: 10.5815/ijmecs.2014.12.08

[7] Walker, D.J. and Craven, M.J. 2020. "Identifying good algorithm parameters in evolutionary multi-and many-objective optimisation: a visualisation approach", *Applied Soft Computing*, vol 88, Article 105902. DOI: 10.1016/j.asoc.2019.105902

[8] Ajeil, F.H., Ibraheem, I.K., Azar, A.T. and Humaidi, A.J. 2020. "Grid-based mobile robot path planning using aging-based ant colony optimization algorithm in static and dynamic environments", *Sensors*, vol 20(7), Article 1880. DOI: 10.3390/s20071880

[9] López-Matencio, P., Vales-Alonso, J. and Costa-Montenegro, E. 2017. "ANT: Agent Stigmergy-Based IoT-Network for Enhanced Tourist Mobility", *Mobile Information Systems*, Article 1328127. DOI: 10.1155/2017/1328127

[10] Khodnenko, I., Kudinov, S. and Smirnov, E. 2018. "Walking distance estimation using multi-agent simulation of pedestrian flows", *Procedia Computer Science*, vol 136, pp. 489-498. DOI: 10.1016/j.procs.2018.08.256

[11] Akka, K. and Khaber, F. 2018. "Mobile robot path planning using an improved ant colony optimization", *International Journal of Advanced Robotic Systems*, vol 15(3). DOI: 10.1177/1729881418774673

[12] Sangeetha, V., Krishankumar, R., Ravichandran, K.S. and Kar, S. 2021. "Energy-efficient green ant colony optimization for path planning in dynamic 3D environments", *Soft Computing*, vol 25(6), pp. 4749-4769. DOI: 10.1007/s00500-020-05483-6

[13] Yu, X., Chen, W.-N., Gu, T., Yuan, H., Zhang, H. and Zhang, J. 2019. "ACO-A*: Ant Colony Optimization Plus A* for 3-D Traveling in Environments With Dense Obstacles", *IEEE Transactions on Evolutionary Computation*, vol. 23(4), pp. 617-631. DOI: 10.1109/TEVC.2018.2878221.

[14] Jiang, B. 2015. "Geospatial analysis requires a different way of thinking: The problem of spatial heterogeneity", *GeoJournal*, vol 80(1), pp. 1-13. DOI: 10.1007/s10708-014-9537-y

[15] Vodák, R., Bíl, M. and Křivánková, Z. 2018. "A modified ant colony optimization algorithm to increase the speed of the road network recovery process after disasters", *International journal of disaster risk reduction*, vol 31, pp. 1092-1106. DOI: 10.1016/j.ijdrr.2018.04.004

[16] Django, 2022. *Django Software Foundation - Django Framework v3.2 Documentation.* [Online] Available: `https://www.djangoproject.com/` [Accessed 11/04/22]

[17] Leafletjs.com. 2022. *Leaflet — an open-source JavaScript library for interactive maps.* [Online] Available: `https://leafletjs.com/` [Accessed 11/04/22]

[18] Duan, H., Yu, Y., Zhang, X. and Shao, S. 2010. "Three-dimension path planning for UCAV using hybrid meta-heuristic ACO-DE algorithm", *Simulation Modelling Practice and Theory*, vol 18(8), pp. 1104-1115. DOI: 10.1016/j.simpat.2009.10.006

[19] Openrouteservice.org. 2022. *Openrouteservice.* [Online] Available at:`https://openrouteservice.org/` [Accessed 13/04/2022].

[20] OpenStreetMap. 2022. *OpenStreetMap.* [Online] Available: `https://www.openstreetmap.org/` [Accessed 13/04/2022].

[21] Sarker, I. H., 2021. "Machine learning: Algorithms, real-world applications and research directions", *SN Computer Science*, vol 2(3), pp. 1-21. DOI: 10.1007/s42979-021-00592-x

[22] Pryke, A., Mostaghim, S., Nazemi, A. 2007. "Heatmap Visualization of Population Based Multi Objective Algorithms". Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds) "Evolutionary Multi-Criterion Optimization", EMO 2007, *Lecture Notes in Computer Science*, vol 4403, pp. 361–375. DOI: 10.1007/978-3-540-70928-2_29

[23] Seyyedabbasi, A. and Kiani, F. 2020. "MAP-ACO: An efficient protocol for multi-agent pathfinding in real-time WSN and decentralized IoT systems", *Microprocessors and Microsystems*, vol 79, Article 103325. DOI: 10.1016/j.micpro.2020.103325

[24] Miao, C., Chen, G., Yan, C. and Wu, Y. 2021. "Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm", *Computers & Industrial Engineering*, vol 156, Article 107230. DOI: 10.1016/j.cie.2021.107230

[25] Ab Wahab, M.N., Nefti-Meziani, S. and Atyabi, A. 2020. "A comparative review on mobile robot path planning: Classical or meta-heuristic methods?", *Annual Reviews in Control*, DOI: 10.1016/j.arcontrol.2020.10.001

A Figures of application

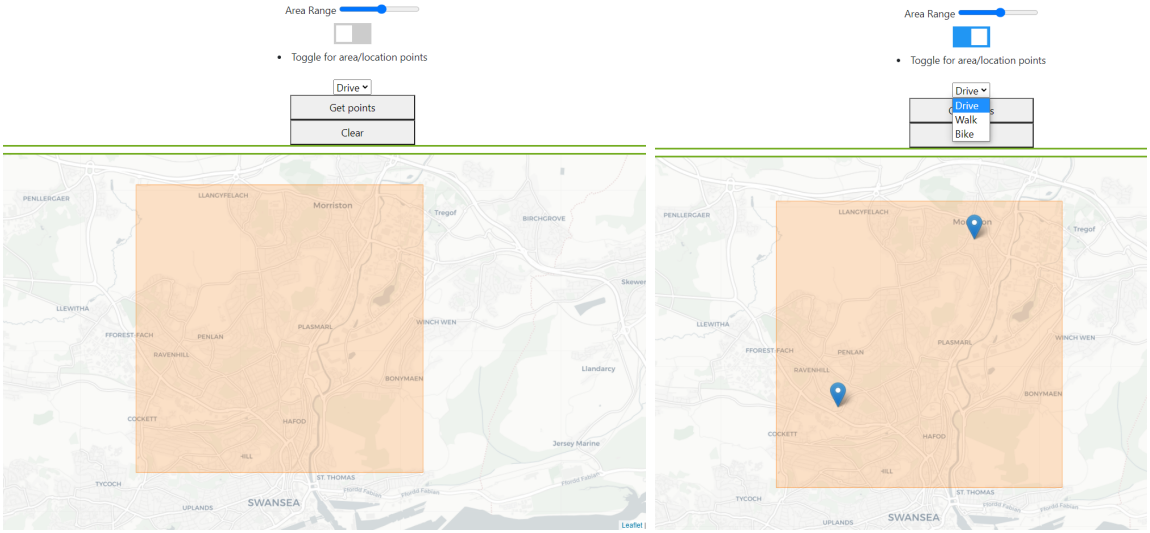


Figure 7: Area is selected, outlined within the bounding area. Then points inside are selected. The available map type is selected in the drop down

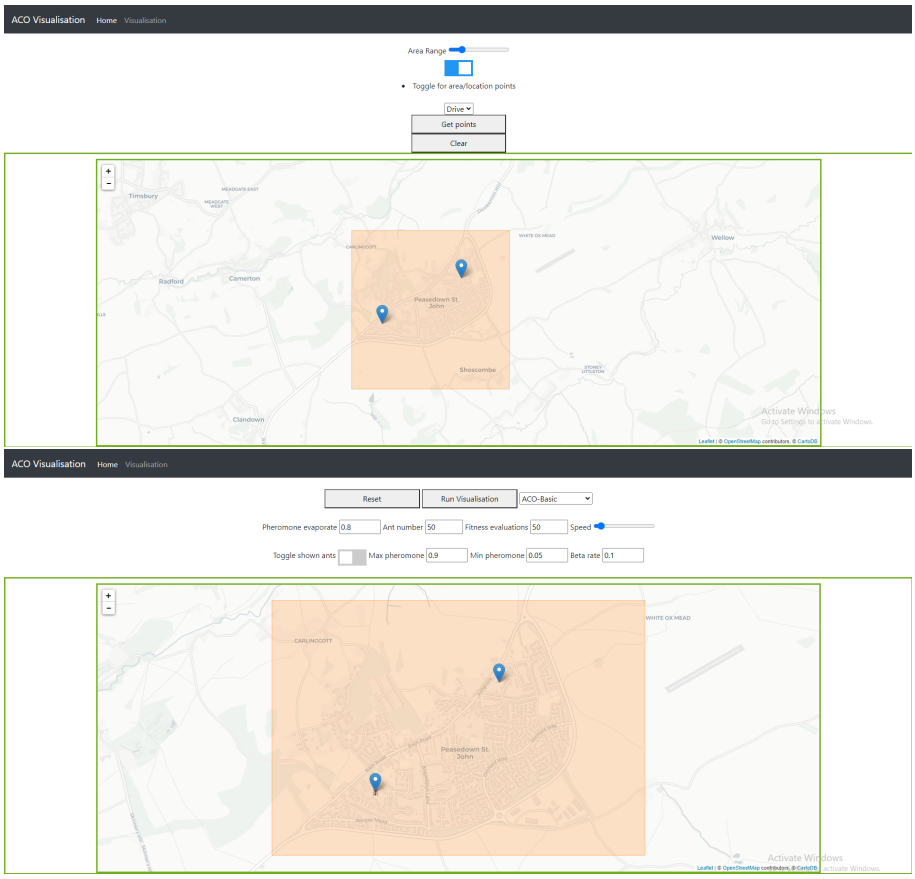


Figure 8: 1: Index page. Leaflet map used to interact and display area for visualisation
2: Visualisation Page. Page allows parameter selection for the environment and ACO models

B Figures of area tests

B.0.1 Dataset A

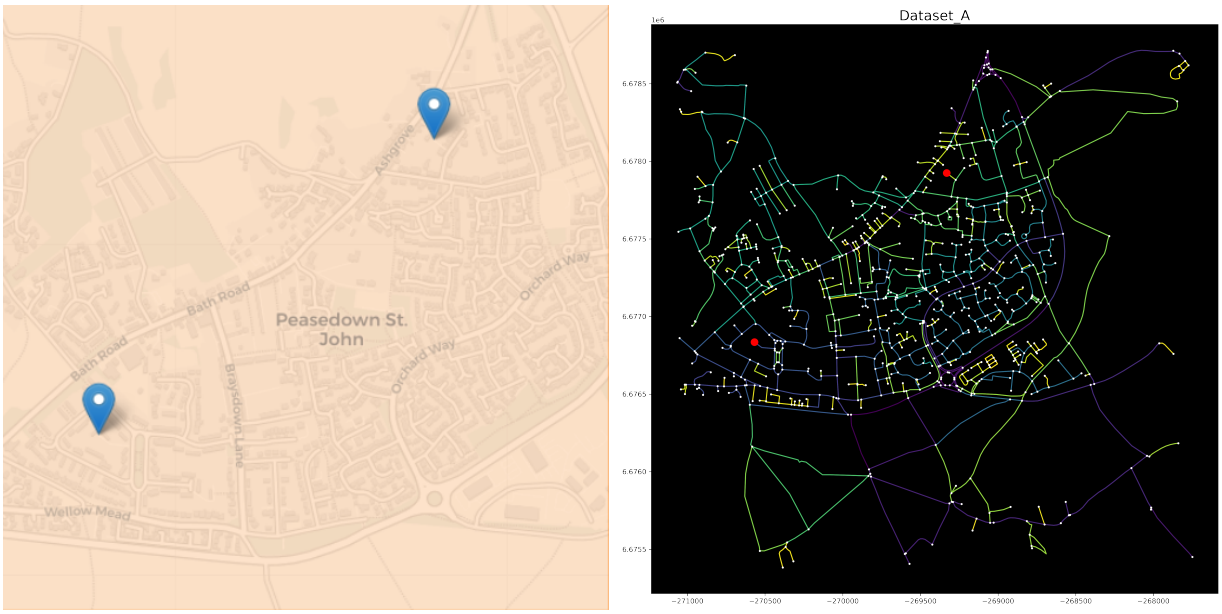


Figure 9: Dataset A: Coordinates Origin = [51.311864397068035, -2.42746789008379] and Destination = [51.319207308630425, -2.4184342101216316]. 879 nodes in the graph, 108927.1370000003 total edge length

B.0.2 Dataset B

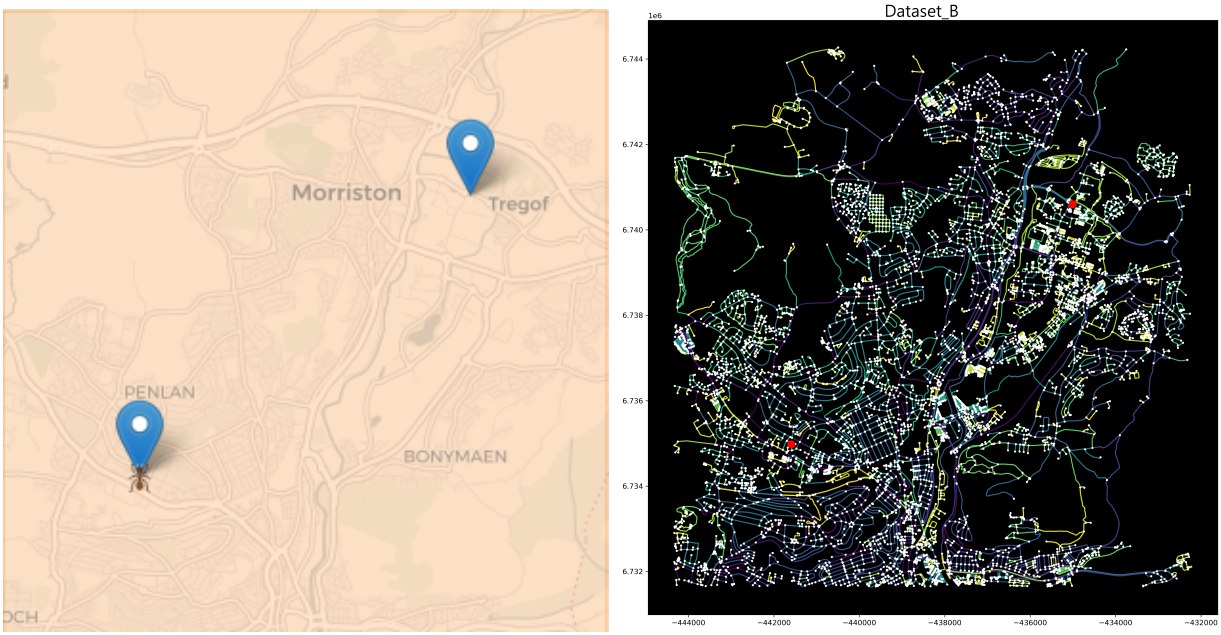


Figure 10: Dataset B: Coordinates Origin = [51.63816503729968, -3.966952690934704] and Destination = [51.66958513378711, -3.9077252792594663]. 7455 nodes in the graph, 1255130.158 total edge length

B.0.3 Dataset C

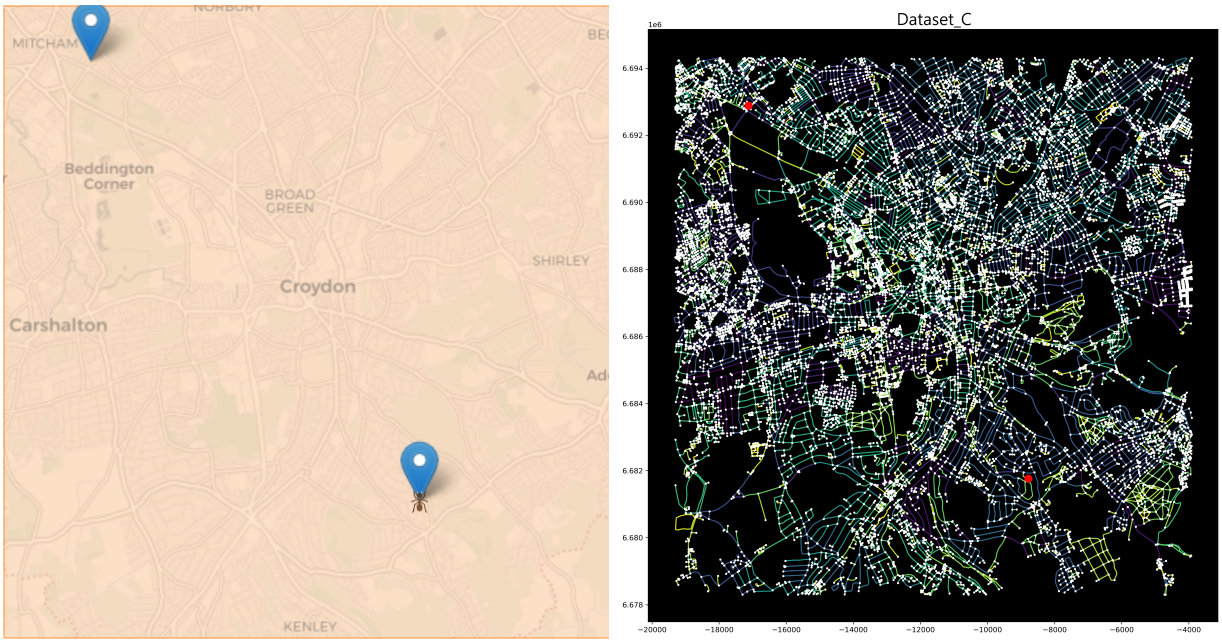


Figure 11: Dataset B: Coordinates Origin = [51.340558777309404, -0.07895149290561676] and Destination = [51.40297065880249, -0.15380004420876503]. 15612 nodes in the graph, 2464640.536 total edge length

C Fitness graphs

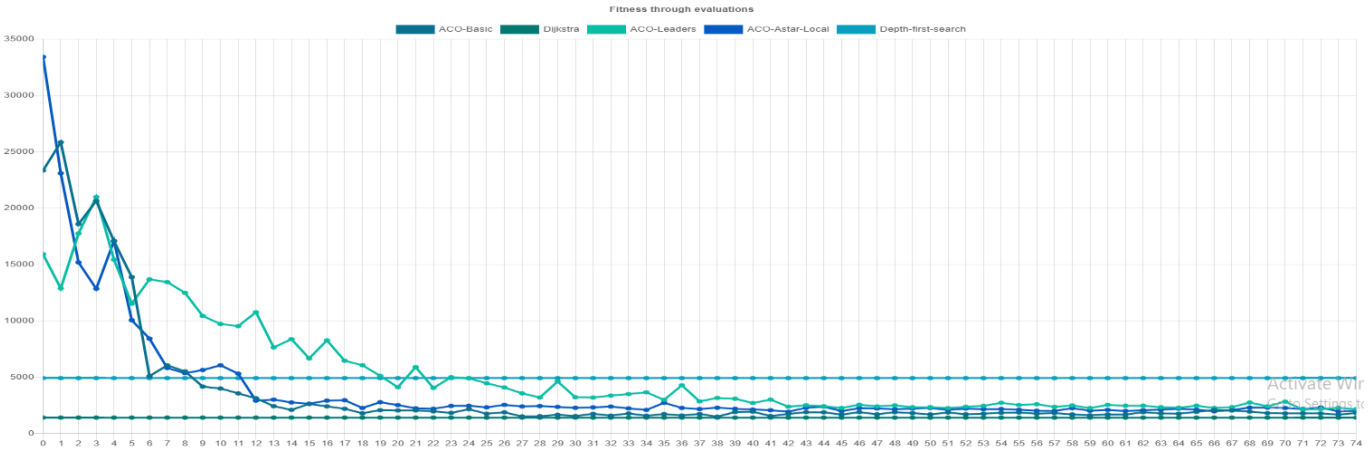


Figure 12: Dataset A: Fitness overtime for 75 iterations

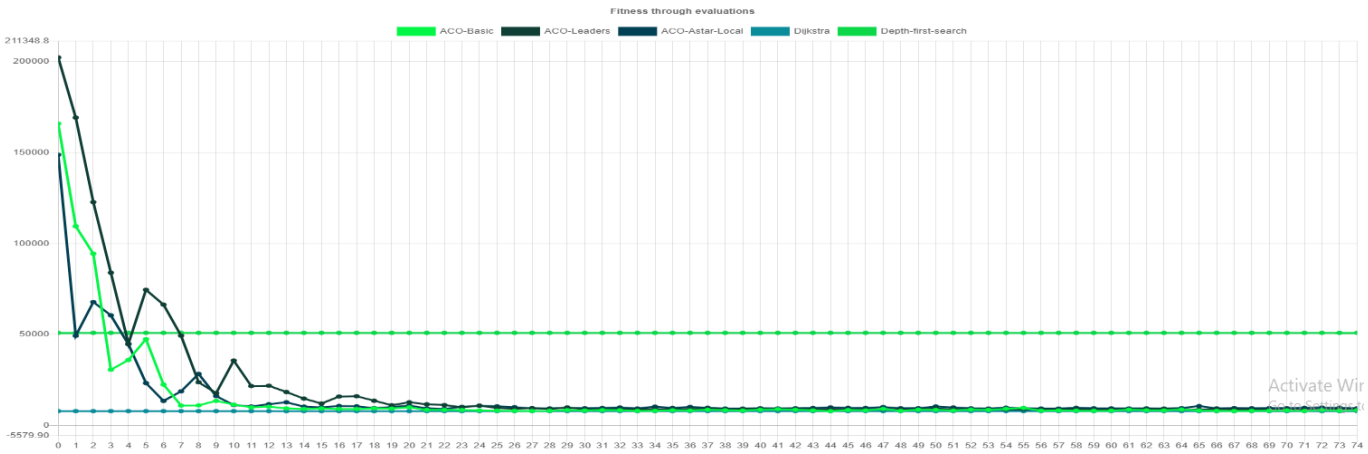


Figure 13: Dataset B: Fitness overtime for 75 iterations

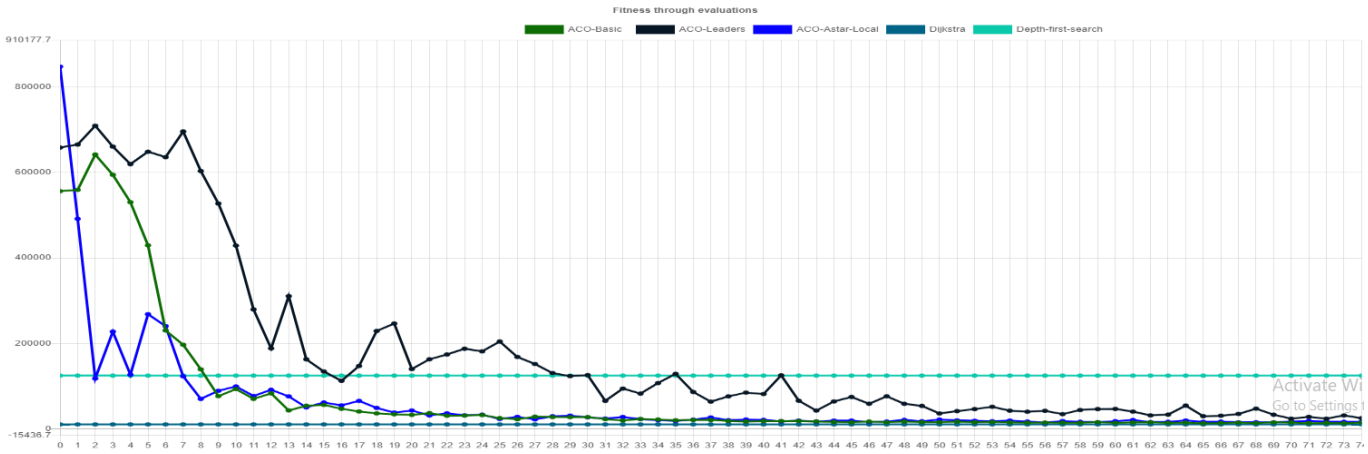


Figure 14: Dataset C: Fitness overtime for 75 iterations

D Dijkstra and Depth First Search

Figure 15: Dijkstra Iteration Search visualisation

Figure 16: Depth First Search Iteration Search visualisation