

University Of Hertfordshire

School of Computer Science

BSc Honours In Computer Science (Artificial
Intelligence)

COMPUTER SCIENCE PROJECT

**Autonomous pathfinding
algorithm using Geographic
Information Systems**

Author: B R Raindle

Supervisor: Manal Helal

SN: 19039840

April 2024

0.1 - ABSTRACT

Do you ever wonder what the future of autonomous vehicle capabilities is? This paper explores how vehicles can use Geographic information systems to produce routes and directions based on your desired metric. Would you like your journey to use the least amount of petrol possible, while you don't mind how long it might take? Or would you prefer to get there as fast as possible, at the expense of a higher fuel consumption?

We start with the basics: finding the right data and figuring out what matters most about each dataset. Discussing the benefits and disadvantages of each option, and then performing basic pre-processing procedures on the chosen data. whilst it might be the least exciting part of the process, understanding the data you are working with is the first step to a successful project.

Next, Using the object-oriented, high-level programming language, 'Python', I developed a fully comprehensive system that harnesses the data produced in the earlier stages of the report, to demonstrate NetworkX's spatial analysis capabilities. And produce some nicely visualized routes that suitably reflect the desired features. To do this, I had to perform several steps of calculations and computations, developing my formula for fuel consumption, and using pre-existing methods of retrieving the rest.

The resulting artefact produced in this project is a GUI with tools and buttons that provide the functionality to request the shortest path between 2 locations, with multiple options for optimised value.

0.2 - ACKNOWLEDGEMENTS

I would like to thank Manal Helal For the extensive support, feedback and reinforcement she provided during some stressful periods throughout my final year.

I'd also like to thank my family, and my friend Pamela Buda for proofreading the report – each providing extremely appreciated feedback.

CONTENTS

0.1 - Abstract	1
0.2 - Acknowledgements	1
Provided Resources	4
Glossary	4
Chapter 1 - Introduction	4
1.1 - Overview	4
1.2 - Project Aims, Objectives and Targets.	5
1.3 – Motivation	5
1.4 – Report Structure	5
Chapter 2 – Background Research	6
Chapter 3 – Data Selection	8
3.1 – Contour Data	8
3.2 – Road Network Data	9
3.3 – Water Link Data	10
3.4 – Protected Land Reserves	11
3.5 – Location Chosen	11
Chapter 4 – Data Visualisation, PreProcessing and Analysis	12
4.1 – Contour Data	12
4.2 – Road Network Data	16
4.3 – Water Link Data	18
Chapter 5 – Artefact Development	19
5.1 - Choice of Approach	19
5.2 - Importing the data	21
5.3 - Setting up The UI	22

5.4 - Calculate Shortest Path	25
Chapter 6 – Testing.....	34
6.1 – Covering a short distance	34
6.2 – Covering a Long Distance	35
6.3 – Comparison with Trusted Software	37
Chapter 7 – Evaluation	39
7.1 – Project Management Review	39
7.2 – Future Additions	40
Chapter 8 References & Extra Information	41
8.1- References	41
8.2 – Appendix	43

PROVIDED RESOURCES

GLOSSARY

Phrase	Definition
A-Star	An algorithm That Searches using a heuristic-based system, using predictive values to determine whether the nodes have a lower estimated distance from the target
CRS	Coordinate Reference System (Used by Geo-Spatial Analysis)
Dijkstra's	An algorithm That Searches each edge from a node, and continues to search that path until it proves more expensive than previous alternatives.
DEM	Digital Elevation Models
GUI	Graphical User Interface
GeoPackage	a file format of encoded geometries, multiline strings and Polygons.
Graph	a Vector based "Edge Node" data structure, used in several Spatial analysis methods
QGIS	Quantum Geographic Information Systems

CHAPTER 1 - INTRODUCTION

1.1 - OVERVIEW

The future of transportation is autonomy, with an increasing reliance on the widely accessible Geographical information systems(GIS), and spatial analysis tools. The goal of this project is to integrate QGIS and NetworkX into a fully comprehensive system which can address complex spatial challenges, such as finding the shortest path connecting 2 geographical locations while taking into consideration various parameters and limitations. Such parameters include road networks, incline limitations, and river crossing, with the possibility of modular base additions, with each dataset about a layer of the database.

1.2 - PROJECT AIMS, OBJECTIVES AND TARGETS.

- Develop a comprehensive understanding of QGIS and NetworkX, including its functionality surrounding spatial data analysis and manipulation.
- Collate The data in a layered fashion, using Geo-Packages, importable by Python libraries.
- Integrate the routing algorithm with QGIS and NetworkX to enable data processing and visualisation.
- Using the Same datasets, implement a node graph pathfinding algorithm with Python, using A* or Dijkstra Algorithms for pathfinding, using Weighted connections, calculated using the parameters absorbed from the data layers that precede the Node placement.

1.3 – MOTIVATION

From Planetary surface exploration to mining utility, any industry that shares the common issue of off-road navigation will benefit from a versatile and efficient implementation of route planning. The death rate in mountaineering is 0.2-40 per 1000 mountaineers per year (Windsor et al., 2009). Many of these deaths are a result of the time taken to receive emergency care in life-and-death situations. With the technology for automated patrol routing and mounted sensors for detecting people in distress, a tool for dangerous terrain rescue scenarios becomes possible, with the capabilities of utilising road networks to reach hospitals or whatever service is required.

1.4 – REPORT STRUCTURE

Chapter 1 – Introduction, Provides a summary of the Project, with aims and objectives, with the motivation behind the project.

Chapter 2 – Background Research, A Literature review that looks into several projects that were resourceful and enlightening.

Chapter 3 – Data selection, A comprehensive explanation of choices regarding Data and locations

Chapter 4 – Data Preprocessing and Visualization, Displaying each dataset into fully comprehensible visualizations, with attribute table analysis.

Chapter 5 – Data Handling and System Programming within Python – Importing datasets into a NetworkX graph, then performing search algorithms, and further preprocessing steps to reach a functional model

Chapter 6 – Testing – Testing each function, producing analytical tools for evaluation.

CHAPTER 2 – BACKGROUND RESEARCH

This literature review examines the fundamental issues within the shortest path on geographical data, considering factors like incline limitations and road networks. This topic is present within applications in various fields, particularly in route navigation systems, I have found several studies on computing the shortest path in various types of networks.

The A-Star Algorithm is a particularly popular search algorithm due to its heuristic properties, however, it can struggle with uneven terrains because it only considers a Hypothesised Euclidean distance between points, not the actual incline. Hong et al. (2020) propose an improved A-Star algorithm for off-road path planning. The conventional algorithm is known for its efficiency in finding the shortest path on a graph. However, it has limitations when applied to uneven terrains. Hong et al. (2020) address this by incorporating a terrain data map and feasible slope thresholds into the A-Star algorithm. The algorithm uses this information to calculate the slope between two points and ensure that the path stays within the vehicle's incline limitations, making it suitable for off-road scenarios with gradient restrictions. Implementing an incline to the heuristic value of the network would allow my system to perform with a similar functionality

Dijkstra's algorithm, developed in 1956 by Edsger W. Dijkstra, is a systematic and guaranteed method for finding the shortest path in a weighted graph. It works by iteratively exploring neighbouring nodes, keeping track of the shortest distance encountered so far. This ensures that the final path is indeed the most efficient. However, Dijkstra's algorithm doesn't take into account the destination's location. It explores all paths outwards from the starting point with equal priority, which can be computationally expensive for large networks.

The A* algorithm, introduced by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968, builds upon Dijkstra's foundation. It incorporates a heuristic function that estimates the remaining distance to the goal node. This "informed search" prioritizes exploring paths that seem closer to the destination based on the heuristic. While A* isn't guaranteed to find the absolute shortest path in all cases (the heuristic might be inaccurate), it often finds a very close approximation much faster, especially in situations where the goal location is known beforehand. The problem with this approach is that I would require significantly more data to justify the Heuristic value.

<i>Feature</i>	<i>Dijkstra</i>	<i>A Star</i>
<i><u>Approach</u></i>	Systematic Exploration	Informed Search Using Heuristic Attribute
<i><u>Guaranteed Shortest Path</u></i>	Yes	Not always
<i><u>Efficiency</u></i>	Slower for large Data networks	Faster in most scenarios
<i><u>Knowledge of Target</u></i>	Not Required	Required To calculate Heuristic Attribute

Table 1 – Comparison between Dijkstra and A Star Algorithms

Based on this analysis seen in Table 1, I think a Dijkstra-based approach will be more suitable, as the data is still a relatively small portion of the UK Map, and the ultimate goal is the absolute shortest path possible, which A Star does not always provide.

My project could potentially benefit from a different approach, Like Node2vec (Nayak, 2019) Which uses a Deep learning method. In the anticipation of working with Large road networks and extensive datasets, that can cover vast regions of the world, the A* algorithm becomes extremely computationally expensive. 'Deep learning approaches like Node2vec can potentially offer faster processing times by learning efficient node representations' (Nayak, 2019) It is, however, important to consider the limitations of Deep neural network implementations, Nayak (2019) highlights that while Node2vec works extremely well at approximating shorter paths, the accuracy wavers when it comes to long-distance calculations.

The trade-off between speed and accuracy is a large factor when the applications of this technology could potentially rely on extremely precise information. If a mountain rescue vehicle is provided with a route that is 20 metres inaccurate, it could be the difference between life and death. In this scenario, a more precise technique would be more suitable. It could potentially be possible to integrate the A Star algorithm into the Deep neural network, as a preprocessing stage, which would help reduce the searched area and computation time. Alternatively, a Digital elevation model (DEM) could be utilised as a way of representing the offroad environment, which introduces the concepts required for off-road pathfinding. An article (Li et al., 2017) offers insight into how my project could be improved by incorporating some other impactful features. By considering fuel efficiency as a factor, it would offer an alternate use for the system, for vehicles that perform several, recurring journeys, like lorries and heavy goods vehicles. It could help 'optimise routes for the vehicles range limitations' (Li et al., 2017)

CHAPTER 3 – DATA SELECTION

3.1 – CONTOUR DATA

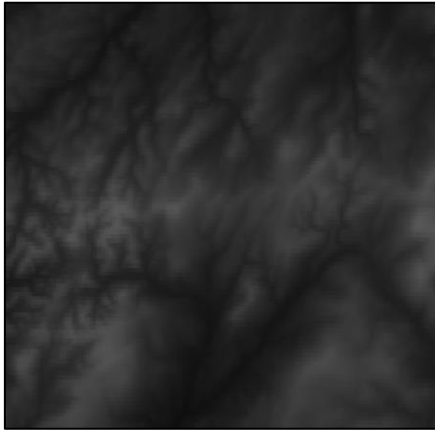


Figure 3.1.1- screenshot of Raster Data Section

When selecting the contour dataset for Topographic analysis, it was decided that the GMTED-2010 dataset would prove most appropriate. Firstly, its raster format contains encoded information in each pixel, enabling detailed terrain analysis and visualization. It uses the Coordinate reference system(CRS): EPSG:4326, which facilitates an easy conversion to the project default CRS: EPSG:27700. Additionally, the data is easily accessible from the USGS website, with downloadable sections of data, depending on the user's specific requirements.

It is essential, however, to consider the limitations of GMTED-2010. The main disadvantage is that its last update was in 2011. It could contain inaccuracies or discrepancies, particularly in rapidly changing landscapes. For instance, urban expansion, land use change, or natural land changes and disasters could alter the features of the terrain, affecting potential analysis outcomes. Furthermore, preprocessing raster files often involves more steps than handling raw Vector data, increasing effort and computational resources.

An alternate approach worth considering would be the UK Open-Data 'Terr50', a GeoPackage directly downloadable from the Open Data website. Terr50 comes as a GeoPackage format (.gpkg) containing Contour lines, and peak points, in vector form. This leads to directly usable information that requires less pre-processing. This unique approach to terrain analysis provides several advantages, for example, easily retrievable contour polygons, which can then be used for attractive visualisations, and topographic analysis.

Ultimately, it was conclusively decided that GMTED2010 Would be used as the raster format allowing for point extraction for encoded information, allowing the development of versatile polygons, through a process called "point-clustering", this wouldn't have been possible with the Contour line GeoPackage provided by the Terr50 dataset.

3.2 – ROAD NETWORK DATA

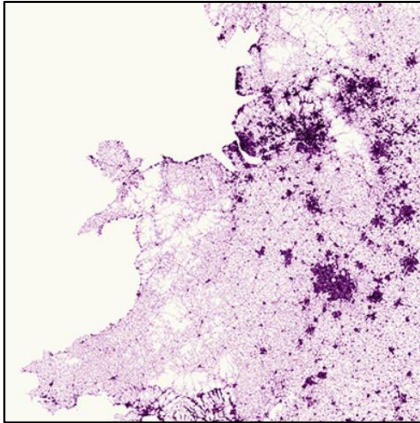


Figure 3.2.1 – Open Roads Cover image

The inclusion of road network data is crucial for enhancing the system's capabilities in route planning and optimizing traversal across various terrains. Among the available options, Open Roads (OS Open Roads - Vector Map Data for GIS - Free OS Data downloads) stands out as the most suitable candidate for this project. Developed using Ordnance Survey mapping data, this dataset offers a comprehensive vector representation of roads in the UK, packaged in a GeoPackage file format.

One of the key strengths of Open Roads lies in its alignment with the same data format, EPSG:27700, as the other datasets being analysed. This functional symphony ensures seamless integration and high cohesion between different layers of spatial data, facilitating effortless interoperability within the system. Furthermore, the dataset covers the entirety of the UK and provides detailed features, including major highways, secondary roads, and local streets, thereby offering accurate depictions of the transportation infrastructure.

Moreover, Open Roads benefits from regular updates and widespread accessibility, in accordance with the Open Government License. However, it is important to note that the dataset may exhibit variations in quality, particularly in rural areas where road data may be less detailed. Additionally, periodic revisions to Open Roads may introduce inconsistencies or discrepancies over time. To address these challenges and enhance the system's longevity, it would be vital to consider implementing mechanisms for version handling. This would involve managing and tracking revisions to the dataset, ensuring that the system remains up-to-date and reliable even as new versions of Open Roads are released.

3.3 – WATER LINK DATA

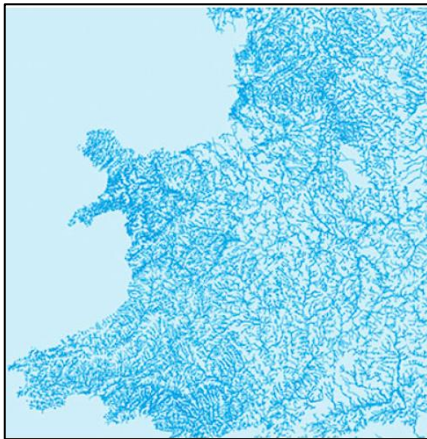


Figure 3.3.1 – Open Rivers Cover image

Water network information is integral to ensuring the safety and integrity of a vehicle's operation, particularly concerning ground clearance and potential water depth. In this project, water crossings are evaluated as a crucial parameter within the algorithm, represented as a boolean value to determine if a vehicle can safely traverse a body of water based on its ground clearance or equipped support infrastructure such as snorkels or raised air intakes.

To achieve this, the selected dataset to represent the river network is OS OpenRivers, accessible through the OS Open Data Hub. This dataset provides detailed formations of water links across the UK, facilitating an accurate depiction of water bodies and their connectivity. While OS OpenRivers lacks precise depth readings for each water connection, it categorizes water as either crossable or non-crossable, aligning with the project's requirement to determine the feasibility of water crossings.

However, it's important to acknowledge a limitation of the OS OpenRivers dataset: the absence of accurate depth readings may reduce the application's robustness, particularly in scenarios where precise depth measurements are necessary for assessing water crossings. Despite this limitation, there remains the potential to enhance the software's capabilities in future iterations by incorporating more detailed datasets or additional parameters to refine water-crossing assessments.

Alternative options, such as the UKCEH Digital River Network of Great Britain, were considered. However, this dataset, operating at a scale of 1:50000 and last updated in 2011, presents limitations compared to the more recently updated (2023) OS OpenRivers dataset. The UKCEH dataset's potential for inaccuracies due to its scale and outdated information renders it less suitable for the project's requirements, highlighting the superior suitability of the OS OpenRivers dataset for the current application.

3.4 – PROTECTED LAND RESERVES



Figure 3.4.1 –Protected Areas Cover image

[Special Protection Areas \(England\) - Special Protection Areas \(England\) - Natural England Open Data Geoportal.url](#)

The decision to utilize the Special Protection Areas (SPA) dataset from the Natural England Open Data Geoportal for representing protected areas of the UK is informed by several critical considerations. Despite the relative sparsity of these areas, compared to other geographic features, their significance cannot be undermined, it is essential to take into account lands designated as protected to ensure their preservation and prevent desecration.

SPAs primarily encompass the coastal regions of the UK but also include scattered wildlife reserves across the country. These areas serve as crucial habitats for various protected species and play a vital role in biodiversity conservation. While these reserves may be small in size, their ecological impact necessitates careful consideration in any spatial analysis or routing algorithms.

One of the key factors guiding the selection of the SPAs dataset is the representation using polygon geometries. This format aligns seamlessly with other spatial datasets utilized in the project, ensuring consistency and compatibility in data integration and analysis. Additionally, the use of the EPSG coordinate reference system further enhances interoperability, facilitating accurate spatial analysis and routing calculations across all datasets.

Furthermore, the inclusion of the SPA allows for the consideration of access permission based on specific criteria. For example: while certain vehicles like rescue vehicles may have authority to traverse these preserved lands, activities such as automated patrols, mining or agricultural uses may be restricted. Incorporating this information into the routing algorithm ensures adherence to legal and environmental regulations governing protected areas, promoting responsible and ethical land use and conservation practices.

3.5 – LOCATION CHOSEN

It's important that this project can demonstrate all capabilities, whilst being possible to produce within a reasonable time frame, and with datasets with the size of 9GB, some procedures can take up to or more than 5+ Hours, which also significantly increases the resource requirements, mainly computer memory for accessing/recalling data.



Figure 3.5.1 – Google map covered section of map extent

For this project, Use the 'clip by extent' function in QGIS. By reducing the data size to a confined extent, the project gains full control over the amount of data being handled, thereby reducing computation time and power, this will prevent excessive delays in project development. The selected sector, as depicted in Figure 3.5.1, contains all essential

features necessary to reflect robust functionality, focusing on a delimited map extent, the project can concentrate testing efforts on specified terrain types, road networks, and water bodies relevant to the objectives. This ensured a thorough evaluation, without the use of unnecessary duplication and redundancies.

The confined map extent, as represented in Figure 3.5.1, offers the potential to demonstrate both off-road and on-road capabilities effectively. In the southern sector, where large areas of off-road terrain are present, the project can showcase the algorithm's proficiency in navigating and recognising limitations within diverse landscapes. Simultaneously the inclusion of a large road network within a confined extent enables the demonstration of on-road capabilities, highlighting the algorithm's ability to optimize routes if they can be improved by utilizing established roadways.

CHAPTER 4 – DATA VISUALISATION, PREPROCESSING AND ANALYSIS

4.1 – CONTOUR DATA

To preface, The entire QGIS Project is available through Appendix B, including each post-processed Dataset, and raster file.

The contour data preprocessing stage turned out to be the most challenging aspect of this entire project. After importing the Raster file into QGIS, I was returned with a black-and-white image, as seen in Figure 4.1.1. Each pixel/sector contains encoded information, representing the elevation values stored at that location. Extracting the information presented in Figure 4.1,2, using the "Identify Feature" tool, provided by QGIS, the 'Band 1' value is the raster elevation value. The derived information includes the X and Y coordinates. Highlighting the pixel in question with a red box.

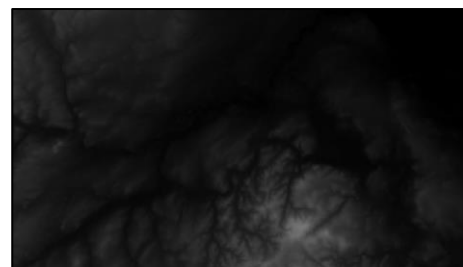
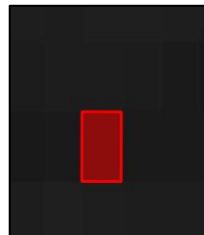


Figure 4.1.1 – Snippet of Raster Layer Visualised

The goal for preprocessing the contour data is to return a GeoPackage/shapefile, containing a collection of polygon geometries, with the allocated elevation, encoded.

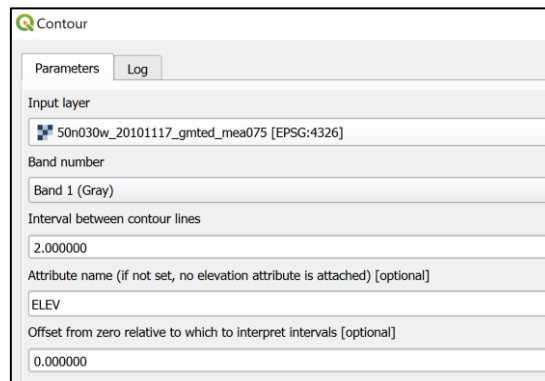
There were several options on how to reach this goal. Firstly, the 'Raster> Extract> Contour' Tool, built-in with QGIS, was used. As parameters, the raster layer is used as the input layer, If the data has multiple colour bands, select the one that represents the elevation value, for this Dataset, it only included Band 1. I chose to use intervals of 2.00 Meters are the default unit of measurement for this Tool. 2 meters is the maximum accuracy that can be detected with the CRS EPSG:27700, The ELEV Value is what is going to be used to determine what signifies the contour line value. After generating 12052 Line-string geometries, QGIS returns the plotted lines, as seen in Figure 4.1.4. From here, we want to turn those lines, into polygons, which contain the elevation.

One method of getting this is the 'lines To polygon' Tool from QGIS Using the "raster_Contour_2m" layer as an input, it returns Polygons seen in Figure 4.1.5. This is not the desired outcome. L.V Lucchese (2021) Encounters this issue and describes that "you should start with the regular 'Fix Geometries' Algorithm". This, however, just outputs the same result, With similarly broken geometries, which means the issue could lie in the contour lines generated in previous steps, or possibly the data itself.



Feature	Value
50n030w_20101117_gmted_mea075	0
50n030w_20101117_gmted_mea075	
Band 1	184
(Derived)	
(clicked coordinate X)	386008
(clicked coordinate Y)	631727
Column (0-based)	13332
Row (0-based)	6922

Figure 4.1.2 – Raster Pixel Example Data



Q Contour

Parameters Log

Input layer
50n030w_20101117_gmted_mea075 [EPSG:4326]

Band number
Band 1 (Gray)

Interval between contour lines
2.000000

Attribute name (if not set, no elevation attribute is attached) [optional]
ELEV

Offset from zero relative to which to interpret intervals [optional]
0.000000

Figure 4.1.3 – Contour Tool Provided With QGIS

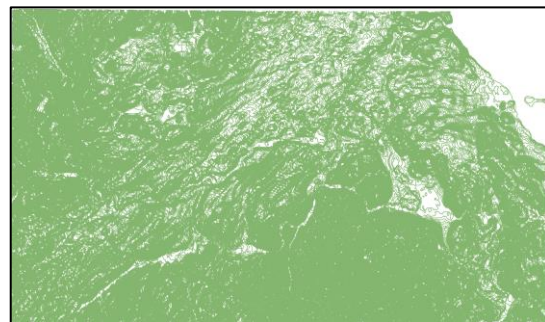


Figure 4.1.4 – Contour Lines Visualised

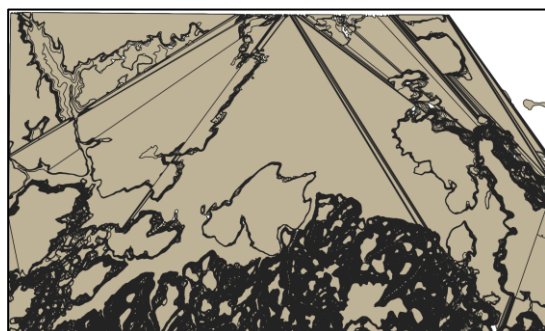


Figure 4.1.5 – First attempt at Polygonised Contours

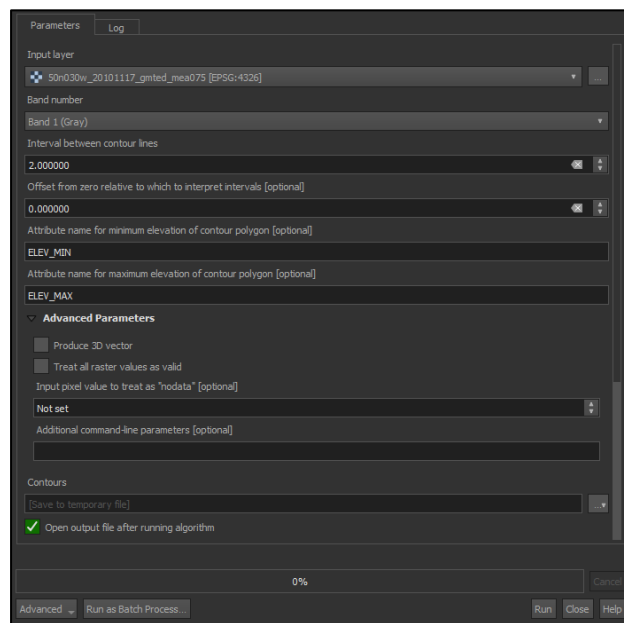


Figure 4.1.6 – Contour Polygons (combined) Tool Provided with QGIS

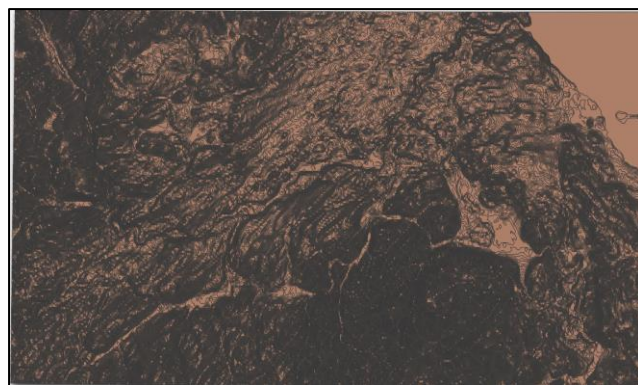


Figure 4.1.7 – Output from the Contour Polygons Tool in Figure 4.1.6

with 0 offset. The ELEV_MIN and ELEV_MAX select 2 Elevation values from the range, and turn them into 1 'range' E.g. '0-2', '2-4', '4-6'. These will become the boundaries that are used to create the polygons. The elevation in the target area ranges from 0-816, so there will be a total of 408 output features, resulting in the visualisation seen in Figure 4.1.7.

Now, to make it easier to comprehend, I edited the Symbology of the GeoPackage, By changing it from a Single symbology, to a graduated symbology, taking the Value ranges, and assigning their own, customisable visualisation, I used the Elev_Min Value as the classifying variable, and gave it a colour ramp that goes from light yellow to dark purple, and each value entry ('0-2', '2-4', '4-6') is given its colour along that scale. I then instructed it to control the feature render order in ascending order of Elevation.

One of the biggest issues with this approach to polygonising is that each of the elevation levels overlaps each other, like a pyramid, the goal is for neighbouring polygons to be formed, in a way they can recognise their neighbouring polygons.

Since this solution does not give the desired output, another approach was followed. This time, the 'GDAL>Raster Extraction>Contour Polygons' Tool was used, as seen in Figure 4.1.6. This takes a raster as an input and turns it into polygons that encompass boundaries of data values.

Firstly, I needed to crop the Raster to the extent of the map I was testing on so that I could use it as the input layer. Using the "Clip raster by extent" QGIS built-in tool, I put that new clipped raster as an input, then select Band 1, this information contains the elevation data encoded into the pixels. The system needs as high of an accuracy as possible, so the interval between contour lines is set to 2.000 with meters as the units,

The results form an extremely understandable elevation map (Figure 4.1.8), containing separate polygons for each elevation value, with the higher elevated areas displayed using the darker colours, and as they tend towards white, the elevation has decreased.

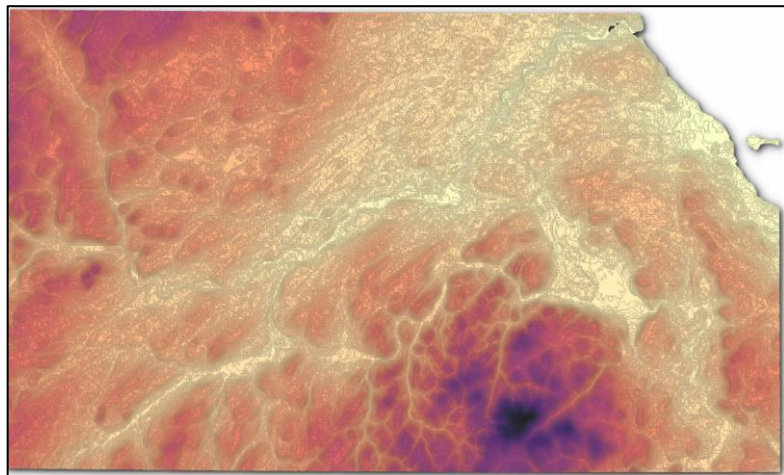


Figure 4.1.8 – Fully visualised Contour Map, Differentiated Elevation colours

When the River Roadmap is overlaid on this data, as seen in Figure 4.1.9, it can be seen that all of the river links, fall in areas where the elevation is significantly lower than the average elevation of the surrounding areas, denoted by the lighter colours amongst very dark patches. This will be useful when implemented into the pathing algorithm, as it helps classify impassable locations.

To understand exactly what I have created Geometrically, Figure 4.1.10 Shows a Zoomed in the area near the peak of the extremely dark patch, I have visually disabled several polygons to show that each boundary ('0-2', '2-4', '4-6') is represented as a Ring polygon, that encompasses all surface area that falls within that elevation range. Each polygon is a 2-meter change, the thinner the polygon, the steeper the hill. Theoretically, this could work if a system was in place to measure the thickness of a polygon at the point of



Figure 4.1.10 – Separated Polygons

interception when attempting to traverse that section of terrain. This project however, will rely on neighbouring polygons to tell what terrain is traversable, the current structure suggests that all neighbouring polygons fall within 2m of each other, so there is no way to tell whether the polygon is too steep for traversing with this information alone. I needed to generate clustered data polygons. I can extract the data from each pixel of the Raster file, and inject that data into a point (or node). That point will have an allocated geographic

location, using x and y coordinates based on the CRS of the project, and the newly allocated Elevation value from the encoded pixel.

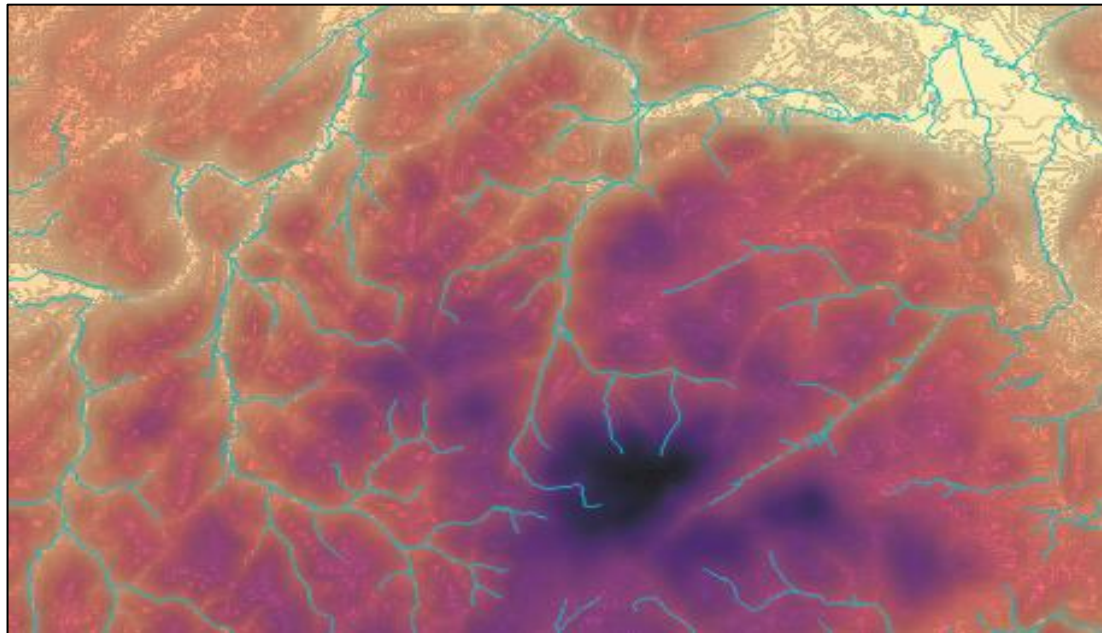


Figure 4.1.9 – River data overlaid ontop of the contour polygons

Disappointingly, This project does not reach the stage in which the Contour data is used for the search algorithm, Methods I wish I had the chance to attempt will be addressed in chapter 8 (Future aspirations).

4.2 – ROAD NETWORK DATA

The road network dataset from Open Data: ‘Open Roads’ was first put through the Clipping tool, Found under Vector>GeoProcessing_Tools> Clip... It is important to reduce the size of the data being worked with, as it reduces the run time of certain tools. The extent polygon generated when selecting the Location will be used as the Overlay Layer, and the new Clipped data will be saved as a GeoPackage, named “Clipped_RoadLinks.gpkg”.

The attribute table for this contains many useful pieces of information, including ‘start_node’

start_node	5EAB582B-0CD2-4EAD-B0D5-0423C7062CAC
end_node	B6D571B8-A4F7-40ED-A94C-E85EEF76E05A

Figure 4.2.1 – Example of Start/End node identifier.

and ‘end_node’, as seen in Figure 4.2.1 This information will be useful when the information is fed into a NetworkX graph, as it provides an identifiable location where each section of road meets.

Another useful element of the attribute table is the ‘road_function’. It is provided in the form of a string which contains any of the following: ‘A Road’, ‘B Road’, ‘Local Road’, ‘Minor Road’, ‘Restricted Local Access’ and ‘Secondary Access Road’. This can be used to inform the

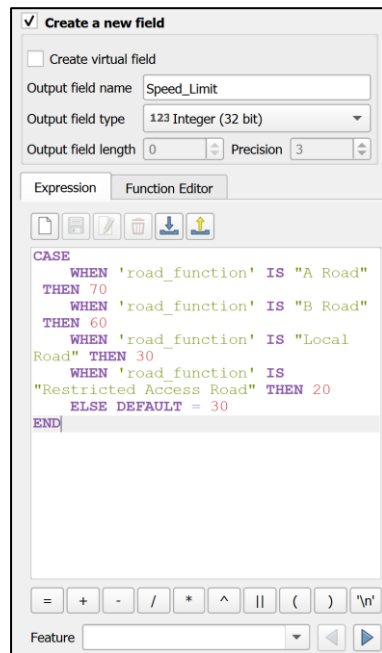


Figure 4.2.2 – Field Calculator for Road Data for road_function. In this scenario, 30 was used. By creating this new field, the speed that can be reached on a road can be used as if it were a heuristic value, in conjunction with road section length. Now, Using the time and speed values for each road link, a calculation can be performed to calculate the Time taken to navigate each road link, which represents the weight input for the chosen search algorithm.

QGIS reads the encoded geographic location of each start and end node and places it in the 'world' based on the CRS (Coordinate reference system)

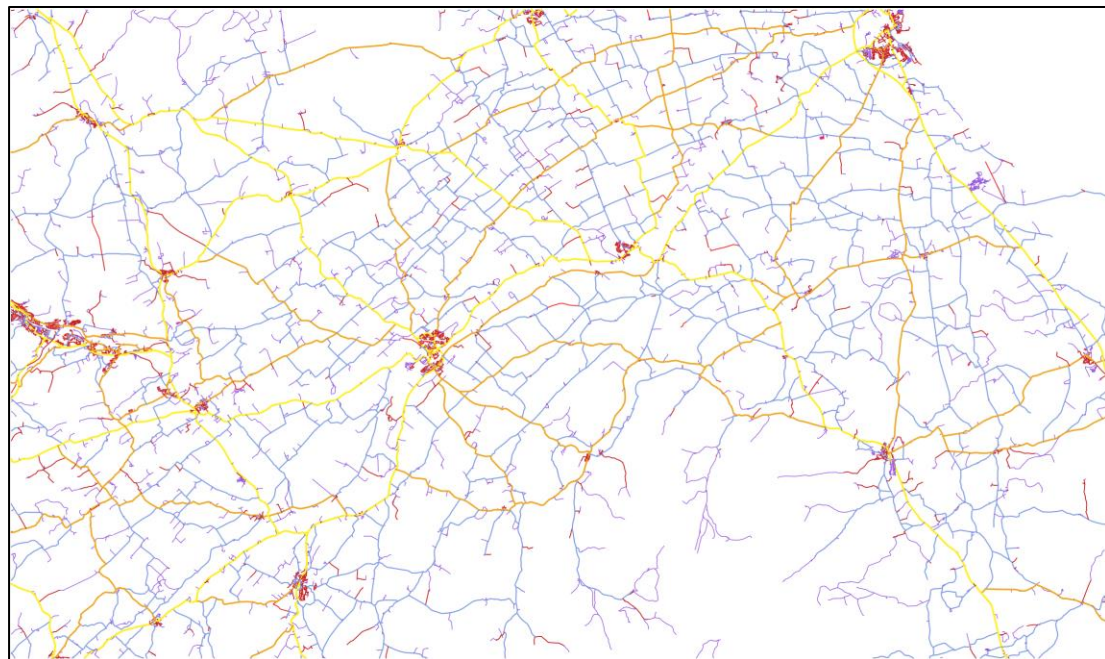


Figure 4.2.3 – Fully visualised Road data, road type differentiated by colour

Using a categorised symbology, assigning all the different road types a unique visual representation. Starting with A roads, I set their colour to 'yellow', and assigned a Width of 0.400, so the line will stand out against the rest. Next, I assign a darker orange colour to the B road and a width of 0.350, as it has a similar speed limit and functionality, but it still needs to be differentiated from the others. Local access roads and Local roads are both found commonly in Towns and settlements, so I allocated the colour 'red' to them, with a width of 0.200, since the roads are more densely clustered, and if the width is too large, the visualization will overlap each other. Restricted access roads are private roads that shouldn't be accessed by the general public without authorization, they are represented by the purple lines with a width of 0.150. Finally, Minor roads, including country roads and anything similar, are represented by the blue lines, with a thickness of 0.200.

4.3 – WATER LINK DATA

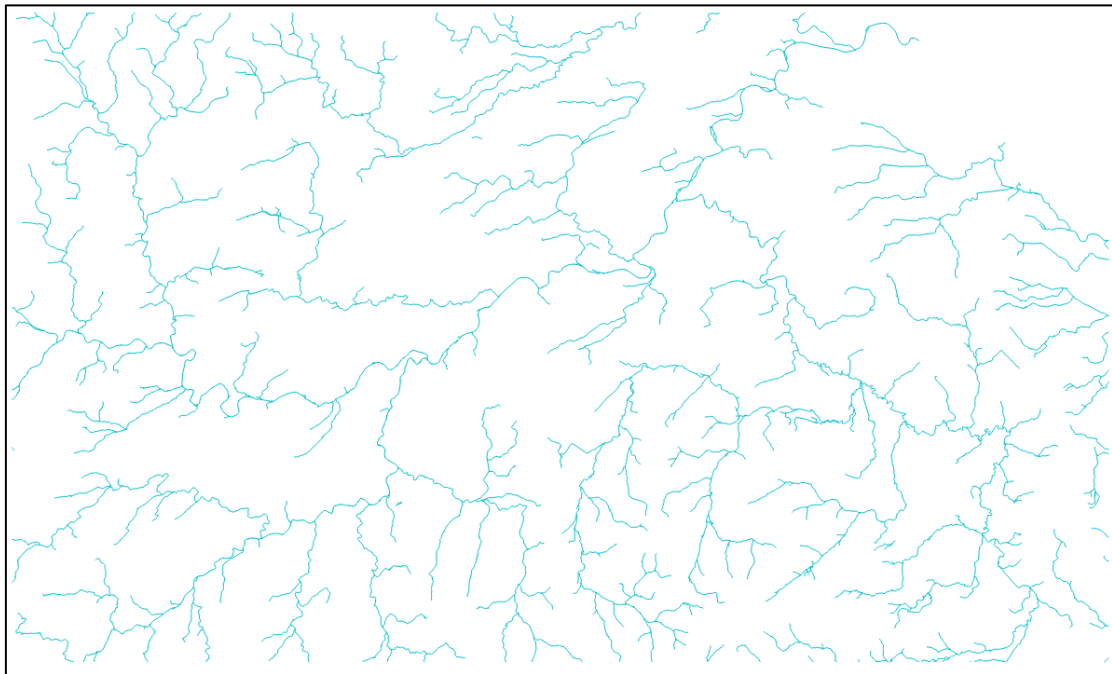


Figure 4.3.1 – River data visualised using QGIS.

Despite the fact The system was not progressed enough to factor the Water Link data, into the prototype, it was still visualised and conceptualised the impact it would have as an additional attribute for the algorithm to use.

Figure 4.3.1 is a representation of the Water networks as a Multi-line String. In theory, it would be desirable that when analysing the contour data, and performing search algorithms,

if a potential path intercepts a river network, it would not attempt to cross them, and instead, re-route around, or find a crossing.

CHAPTER 5 – ARTEFACT DEVELOPMENT

5.1 - CHOICE OF APPROACH

In the domain of Spatial analysis and routing algorithms, using the right Python library can significantly improve the chances of success and, the efficiency of the project. In this project case, we aim to leverage graph structures to manipulate and process GIS data to produce the most effective path using designated features as important factors, for example, Fuel efficiency, or Distance.

Firstly, let's explore why NetworkX is the best library for a route-finding application. NetworkX is extremely versatile and user-friendly, NetworkX's accessibility for network and graph creation, analysis, and visualization offers an easy to work with interface with straightforward methods dedicated to manipulating graphs of data. The benefits of this are reflected in the learning stages of spatial analysis. Step 1 is understanding the data. Even with a low proficiency in this area, NetworkX proved easy to use and extremely accessible. It also integrates seamlessly with QGIS, with the ability to directly import spatial data from QGIS into a NetworkX graph. Using GeoPackage and Shapefiles. There are also several plugins for QGIS, such as "OSMnx," which facilitate the integration of NetworkX functionalities for transportation network analysis using OpenStreetMap data. This extended compatibility between software has the potential to benefit my project directly. NetworkX is an open-source library, which means anyone can customize it to fit their needs and contribute to its ongoing development. This open nature fosters a large community and makes a wealth of information about NetworkX readily available.

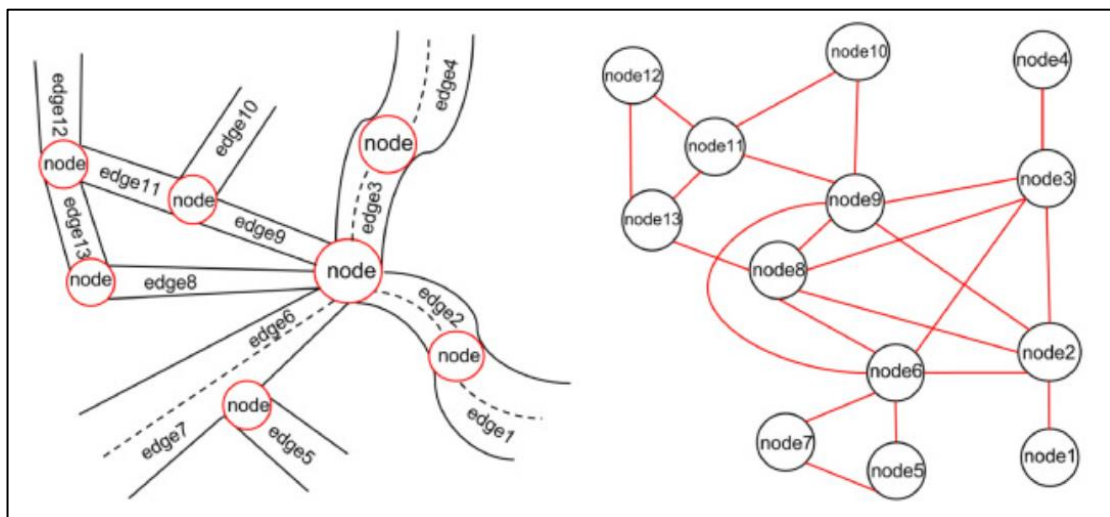


Figure 5.1.1- Roads represented by Nodes and Edges' – (Z. Gharaee et. al. (2021))

NetworkX Uses intuitive graph structures to represent the datasets, by exploiting nodes and edges with embedded attribute data for multiline strings, and the capabilities of implementing polygon geometries for elevation data.

The node and edge feature will be exceptionally helpful when creating the road link connections, As seen in Figure 5.1.1. There are limitations in my dataset that make the system infeasible for real-world implementation, for example: The road link Nodes, do not contain an attribute defining what type of connection it is, and NetworkX will not be able to consider it without an implicit attribute that represents the function of a road connection. This could include junctions, Roundabouts, or cross-sections (as sparse as they are in the UK). Each of these features behaves differently and would directly impact the calculations when it comes to the weight structure of the search algorithm.

NetworkX provides a collection of built-in graphing algorithms, tailored for spatial analysis, and optimization. This includes the Shortest path algorithms like Dijkstra's algorithm and A* algorithm. With access to these pre-built algorithms, I can feed the node/edge graphs through the search algorithms, with defined start and finish positions, with descriptions of which factors and constraints to include. I would like to produce software that can generate 3 alternate paths, with different attributes contributing to the search heuristics. Fuel economy, distance, and time. Each route will also measure the alternative values for the full journey, so they can be compared and evaluated.

I chose NetworkX over the alternatives, mainly because of the uniquely efficient and accessible graph structure. Other options like 'Igraph', 'graph-tool' or even my implementation of the structure and search algorithm, would most likely use an adjacency matrix, or some form of linked array. They would also have a much steeper learning curve, Furthermore, none of them come with built-in algorithmic processes like shortest path.

Functional Requirements

- Display The map available through the testing data
- Take 2 clicks as inputs for start and end locations
- Plot the Shortest path line
- Give enough control to erase lines, and replot them
- Choice between multiple attributes to optimise
- Return Information and statistics on a route.

Non-Functional Requirements

- Enough accuracy to provide a concise and obvious journey
- System will be able to compute the shortest path promptly
- Modular structure, should be able to add new elements and datasets to the system to expand

5.2 - IMPORTING THE DATA

Taking data from QGIS directly into a Python file can be done in several ways. This project uses GeoPackage to store the encoded Geometries. Geopandas handles the reading and manipulation of geospatial datasets, while Matplotlib provides powerful plotting capabilities for informative visualizations. By organizing the data loading process into a method, the code promotes modularity and reusability, allowing for easy integration into larger projects and facilitating maintenance and future enhancements, like those mentioned in Chapter 7.

Step 1: Loading the files and organizing.

The method starts by defining a list of GeoPackage files that will be imported ('gpkg_files'), Containing ['Clipped_Contour_2m.gpkg', 'clipped_roadlinks.gpkg', 'clipped_waterlinks.gpkg'], Each file represents a specific type of geographic data, such as road links, contour lines, or water links. The order matters here, as they will be rendered in Matplotlib in order of their relative positions. These are then allocated colours, for visual representation. I used a dictionary type to assign a colour, as a string, to each file name, so when it is called, it can also allocate the corresponding colour.

Step 2 Loading In With Geopandas.

Within the loop iterating over each GeoPackage file, the code utilizes Geopandas (gpd) to read the contents of the file (gpd.read_file()). Geopandas is a powerful asset for working with geospatial data, providing functionalities to read, write, manipulate, and visualize geographic datasets, which came in handy.

```
Index(['id', 'fictitious', 'road_classification', 'road_function',
      'form_of_way', 'road_classification_number', 'name_1', 'name_1_lang',
      'name_2', 'name_2_lang', 'road_structure', 'length', 'length_uom',
      'loop', 'primary_route', 'trunk_road', 'start_node', 'end_node',
      'road_number_toid', 'road_name_toid', 'speed_limit', 'geometry'],
```

Figure 5.2.1 – Columns Of the Road Link Dataset

Figure 5.2.1 Represents all attributes of the Road Link Dataset. The most important Attributes in this project will be Road classification, length, start_node, end_node, speed_limit, and geometry. The rest of the attributes are not necessary for the project and therefore are redundant. They could be helpful however, in future amendments, which could implement a way to recognise the path based on the name of the road, so instead of inputting a Coordinate using click interaction, you could provide a String name of a road or location, and the system would still recognise the targets.

After loading the data, the code sets the geometry column to ensure proper handling of geometric information (data.set_geometry('geometry', inplace=True)). "inplace=True:" This argument specifies whether the operation should modify the data-frame in place or return a

new data-frame with the changes applied. When 'inplace' =True, the operation modifies the original data frame (data) itself rather than returning a new data frame. This means that the data frame is altered directly without creating a copy, which can be more memory-efficient for large datasets. For line features, including multi-line strings, the "geometry" column would contain the coordinates of the vertices that define the lines. Each line segment would be represented by a sequence of coordinate pairs. For example, a line might be represented as [(X1, Y1), (X2, Y2), ...] where (X1, Y1), (X2, Y2), etc., are the coordinates of the vertices defining the line segments, as shown in Figure 5.2.2.

```
0      MULTILINESTRING ((393696.000 641418.000, 39370...
1      MULTILINESTRING ((397089.120 648647.740, 39708...
2      MULTILINESTRING ((398548.000 651067.000, 39852...
3      MULTILINESTRING ((398042.000 651036.000, 39799...
4      MULTILINESTRING ((357884.970 631529.130, 35788...
...
11766  MULTILINESTRING ((352704.770 627427.040, 35272...
11767  MULTILINESTRING ((407327.960 617186.480, 40732...
11768  MULTILINESTRING ((357721.670 627254.190, 35771...
11769  MULTILINESTRING ((372636.000 634949.000, 37263...
11770  MULTILINESTRING ((365427.640 621054.560, 36544...
Name: geometry, Length: 11771, dtype: geometry
```

Figure 5.2.2 – Multi-LineString Geometry Format

Step 3 Data visualization.

I chose to use matplotlib since it is a widely-used plotting library in Python, and by utilizing it alongside Geopandas, the system achieves seamless integration of geospatial data visualization with access to customizable plot settings, like zoom tools, buttons and inputs, giving me access to real-time interaction with the map for closer analysis and testing. The loaded data is plotted on a subplot (ax) using Matplotlib (data.plot(ax=ax, color=color)). Each GeoPackage file is plotted with a specific colour based on the mapping defined in the colours dictionary. This allows for easy differentiation of different types of geographic features in the visualization. (Note: In code, Colour, is written as 'color' as Matplotlib is an American-made library, And ax is used to refer to the plotted graph, standing for Axes.).

5.3 - SETTING UP THE UI

QtGUI is ideal for a simple prototype project like this that incorporates Matplotlib for several reasons. Firstly, QtGUI provides a comprehensive set of tools and widgets for building intuitive and interactive graphical user interfaces. Secondly, its seamless integration with Matplotlib allows for embedding plots and visualizations directly within the application, facilitating dynamic data visualization. Additionally, QtGUI offers cross-platform compatibility, enabling the prototype to be deployed and tested across different operating systems without significant modifications. Overall, QtGUI's ease of use, flexibility, and compatibility with Matplotlib makes it a suitable choice for rapidly prototyping applications

with graphical interfaces. Qt also offers a module called Qt for Mobile (formerly known as Qt for Android and Qt for iOS), which enables me to create mobile applications using Qt's cross-platform framework, this would be extremely useful in the future for a more accessible version of the software when it becomes fully refined.

Layout Configuration

The UI layout is managed using a QVBoxLayout, facilitating a vertical arrangement of widgets. This layout provides a clean and organized structure, ensuring optimal space utilization while accommodating various components.

Start and End Location Inputs:

Start Location	(367814.7780287815, 644381.5143687386)
End Location	(386114.7379102339, 643198.8639002094)

Figure 5.3.1 - Start/End Locations input

Figure 5.3.2 – Example inputs after clicking

QLineEdit widgets are employed for users to input the start and end locations for pathfinding. Placeholder text offers guidance and enhances usability/accessibility by prompting users on the expected input format, as seen in Figure 5.3.1. By providing distinct fields for start and end locations, the UI enables users to specify their desired route accurately. Considering this System currently just uses Geographic information, using Coordinate reference systems, the input is in the form of x/y coordinates on a graph, as seen from QGIS and any CRS, and as an example, in Figure 5.3.2. However, in future adaptations I would like the system to recognise postcodes, and street names – but the dataset would need to be drastically more detailed.

Reset Button: The QPushButton labelled "Reset" is mostly organisational. Upon activation, the reset button initiates a sequence of actions to reset user inputs. This includes clearing the text fields associated with start and end locations. By invoking the clear() method on these input fields, any previously entered coordinates are promptly removed, ensuring a clean slate for subsequent input. This improves the testing functionality and accessibility, by allowing the user to correct inputs or start afresh without cluttering the interface. This is important because the system takes around 60 minutes to warm up and launch, then another 60 seconds to complete a process. In addition to managing user inputs, the reset button also handles the removal of visual feedback elements. Notably, when a path calculation is performed and a yellow line representing the shortest path is displayed on the map, activating the reset button triggers its removal. This is achieved by removing the plotted line from the graphical representation using the remove() method.

On original creation of this feature, The screen would freeze whenever clicked, forcing the user to 're-size' the window (minimise, maximise)to refresh the display. To mitigate freezing

issues during path calculation, the system uses a Python threading mechanism. Despite the computational load involved in calculating the shortest path, users experience uninterrupted interaction with the GUI. The reset button seamlessly operates within this threaded environment, ensuring consistent behaviour across all system states, including a robust error-handling system to handle unforeseen scenarios. Whether it's dealing with unexpected input data or encountering issues during visual feedback removal.

An essential aspect of the reset button's functionality is its interaction with the canvas used for map visualization. When clearing user inputs and removing visual feedback, the canvas activates a redraw operation. By calling the draw() method on the canvas, any updates or modifications made to the graphical representation are immediately displayed, providing users with real-time feedback. If this is missed, then the previous yellow line will remain until the window is resized, refreshing the frame.

Weighted Shortest Path Options:

Distance
Time
Fuel Efficiency

Figure 5.3.3 – Weight selection Menu

The inclusion of weighted shortest path options (distance, time, and fuel efficiency) reflects the diverse needs and preferences of users when planning routes. Each option offers a different optimization criterion, allowing users to prioritize factors such as minimizing travel distance, travel time, or fuel consumption based on their objectives or constraints. As seen in Figure 5.3.3, The functionality of this is as follows: When clicked, the button will display the line generated using that relevant weight as the focal point of the algorithm. E.g. When the time button is clicked, it will display the path with the most efficient time value, whereas when the Fuel Efficiency button is clicked, it will display the best Fuel Economy route.

GeoPackage Data Loading: The method `load_geopackage_data()` is invoked within the UI setup, loading spatial data from GeoPackage files using Geopandas for preprocessing and visualization, as seen in Chapter 5.2 of this report. This integration enables seamless incorporation of geographical data into the UI, providing users with relevant context for route planning and visualization.

Map Visualization: The map visualization section utilizes Matplotlib's Figure and Figure Canvas to embed a plot within the UI (Figure 5.3.4). This integration visualizes geographical data and routing results directly within the application, which helps keep all the information visible at once. The features are rendered in order of relevance, as of current the System does not utilise the Land or Water networks, as they require more development to be

functional, so they are rendered underneath the road network, so it is best displayed for understandability.

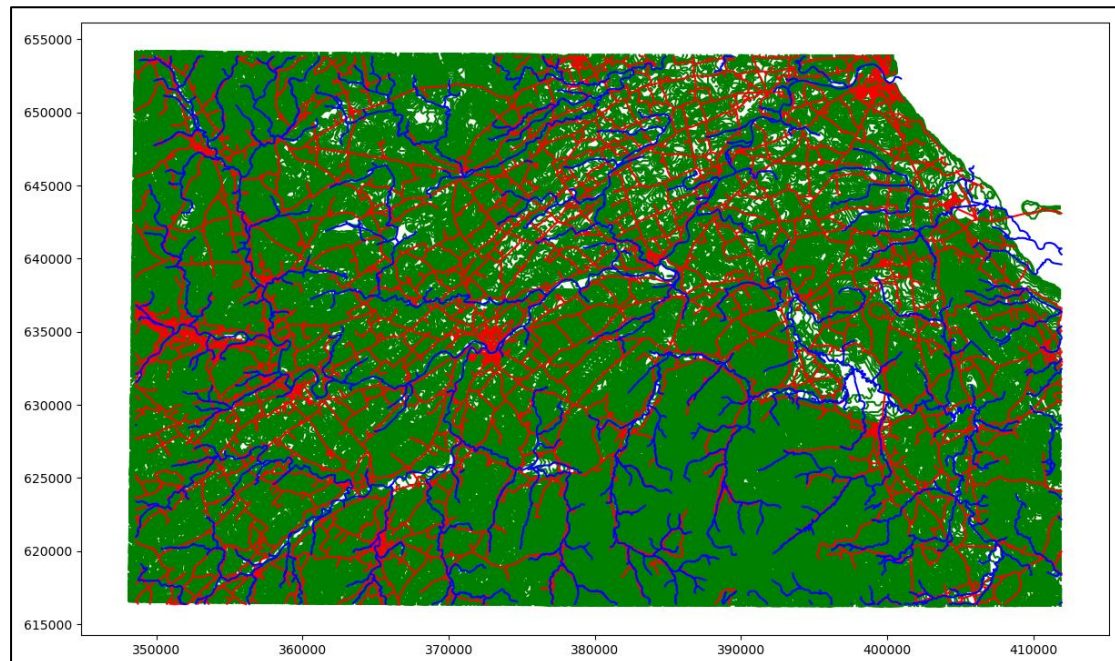


Figure 5.3.4 – Displayed map, with no alterations.

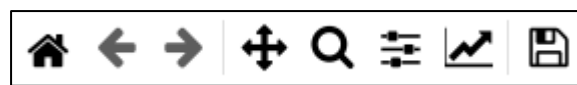


Figure 5.3.5 – Matplotlib UI

Navigation Toolbar: A Navigation Toolbar is added to the UI (Figure 5.3.5), providing me with interactive navigation controls for exploring the plot. This feature

improves usability by offering functionalities such as zooming, panning, and saving the plotted map, empowering users to interact with the visualization effectively, now we can review generated paths in closer detail to spot inaccuracies and flaws, helping to improve the system's overall in the long run.

The entire Code is available through 'Appendix A', a cloud-shared file containing all runnable files, the datasets required for the software, and the complete code used to produce the artefact.

5.4 - CALCULATE SHORTEST PATH

Firstly, we need to build a NetworkX graph. Starting by reloading the road link data into a variable named 'road_links'. A light amount of exception catching and error handling is

```
if road_links.crs != 'EPSG:27700':
    road_links: Any = road_links.to_crs('EPSG:27700')
```

Figure 5.4.1 – Normalising CRS with project

required to ensure that the data is consistent with the project default, so the Coordinate reference

system is assigned to EPSG:27700, which is possible to do inside Python itself. A for loop is constructed to iterate through every element of the dataset, adding the edges to the

```
start_node = 'start'
end_node = 'end'
G.add_node(start_node, geometry=Point(self.start_location))
G.add_node(end_node, geometry=Point(self.end_location))
node_coordinates['start'] = (self.start_location[0], self.start_location[1])
node_coordinates['end'] = (self.end_location[0], self.end_location[1])
```

Figure 5.4.2 – Start and End node creation

GeoDataFrame. For each road link in the GeoDataFrame, the centroid coordinates are extracted using the 'centroid.X' and 'centroid.Y' attributes. These coordinates are used to construct line geometries representing the road segments.

Since this method is only called when the start and end locations have been assigned, we can also build nodes for each of them as seen in Figure 5.4.2. these will be used when finding the nearest node.

Finding the nearest nodes

The 'find_nearest_road_link' function plays a crucial role in the system by identifying the nearest road link nodes to a given point. This function is utilized in the process of connecting the start and end nodes created above, to the road network graph.

By iterating through the road links in the dataset, the function calculates the distance between each road link and the provided point using geometric operations. It then selects the road link with the minimum distance as the nearest node, considering both the start and end nodes of the road link. After identifying the nearest road link node, the function stores this information in a dictionary and returns it to the calling function. This information is subsequently used to establish connections between the chosen locations and the road network graph.

```
def find_nearest_road_link(self, road_links, point) -> dict[str, Any] | None:
    nearest_node = None
    min_distance = float('inf')
    for idx, road_link in road_links.iterrows():
        distance: Any = road_link['geometry'].distance(point)
        if distance < min_distance:
            min_distance: Any = distance
            nearest_node: dict[str, Any] = {'start_node': road_link['start_node'],
                                           'end_node': road_link['end_node'], 'distance': distance}
    return nearest_node
```

Figure 5.4.3 – Find the nearest road link method

If this step isn't taken, unless the user clicks directly on top of the Road link node(the edge start and end coordinates), the system will not be able to provide a route as it will not recognise the desired start and end coordinates as part of the graph.

Creating our attributes

For the intended functionality, the dataset requires 2 new attributes, fuel consumption, and time cost. Let us start with the time cost.

Time :

```
for u, v, d in G.edges(data=True):
    length: Any = d['length']
    speed_limit: Any = d['speed_limit']
    if speed_limit == 0:
        d['speed_limit'] = 30
    else:
        d['time'] = length / (speed_limit * 0.44704)
```

To first create our time attribute, we need to understand what information we have access to and how it can

Figure 5.4.4 – Creating time attribute

be helpful. Each edge has a spectrum of data attributes embedded inside, including the length, in meters of the edge. The speed limit assigned to this edge is based on what type of road we are investigating. The for loop seen in Figure 5.4.4, iterates through every edge stored inside 'G' Our GeoDataFrame (NetworkX graph). And sets the data to True, allowing the for loop to access all attributes associated with an edge, storing them in a temporary dictionary "d".

Next, we extract the information that will be useful to us. In this situation, we desire the time it will cost to travel across 1 edge. I have decided to keep this model simple and assumed that the vehicle will accelerate to the speed limit instantaneously, whilst inaccurate and albeit lazy, it would require far more time and research to produce a model that can accurately represent the realistic parameters governed by the laws of physics. We would require the acceleration value for every vehicle the system would be used by and a way to identify this. We would also require things like the road incline and various other natural factors.

$$Time = \frac{Distance}{Speed}$$

We have access to length, and we have access to the speed limit, so we have all the information we need. `"length = d['length']"` Retrieves the length attribute of the current edge. This length represents the physical distance of the road segment.

`"speed_limit = d['speed_limit']"` Retrieves the speed limit attribute of the current edge. The speed limit indicates the maximum legal speed allowed on the road segment. If the speed limit is recorded as 0, it may indicate a missing or undefined value. To ensure consistency and avoid division by zero errors, a default speed limit of 30 mph (standard speed limit on the majority of roads) is assumed for such cases. This value can be adjusted based on domain knowledge or system requirements.

`"d['time'] = length / (speed_limit * 0.44704)"` Calculates the time attribute for the edge. This calculation takes the road segment length and divides it by the speed limit in meters per second (1 mph = 0.44704 m/s). The resulting value represents the estimated travel time along the road segment in seconds.

The time attribute allows the system to consider the varying travel speeds on different road segments when computing the shortest path. By factoring in the speed limit, the algorithm can identify routes that minimize travel time, providing users with efficient navigation directions.

Fuel :

Producing a fuel consumption attribute is somewhat more complicated than the time attribute. Fuel consumption varies between every car, and varies at different speeds. Whilst this project does not wish to address applications for every single vehicle, we can use the average road vehicle's fuel consumption as a control variable, and use researched information to build the idea on top.

"Optimal efficiency can be expected while cruising at a steady speed and with the transmission in the highest gear (see Choice of gear, below). The optimal speed varies

with the type of vehicle, although it is usually reported to be between 35 and 50 mph (56 and 80 km/h)” (Wikipedia, 2017)

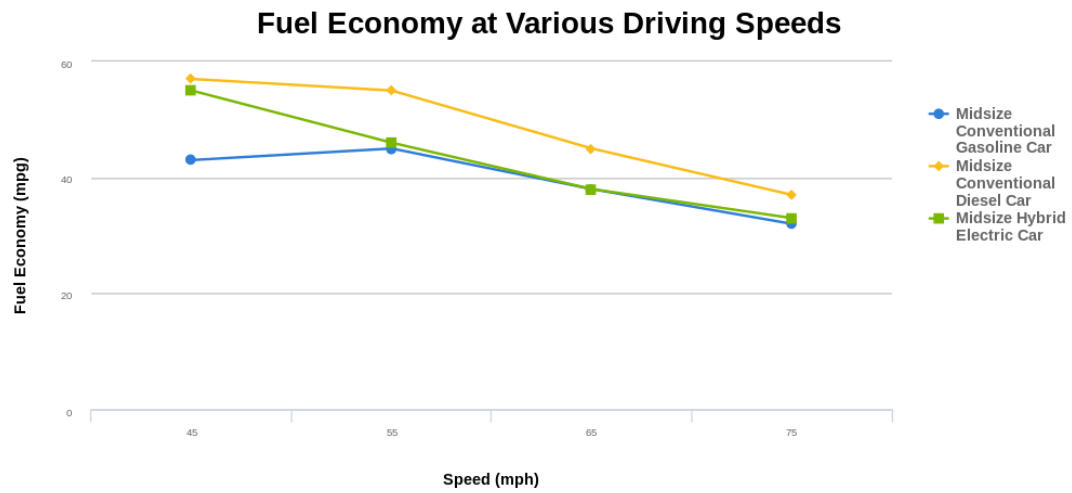


Figure 5.4.5 – (“Alternative Fuels Data Center”, 2021)

This study from Oak Ridge National Laboratory shows that the optimal fuel consumption speeds are roughly around 45 miles per hour, give or take. Now all we need is the average car fuel consumption, and we can build our formula. "Petrol cars usually average around 36mpg, while diesel cars average around 43mpg." (Cinch, 2023) With this information, we are ready to design our formula. for

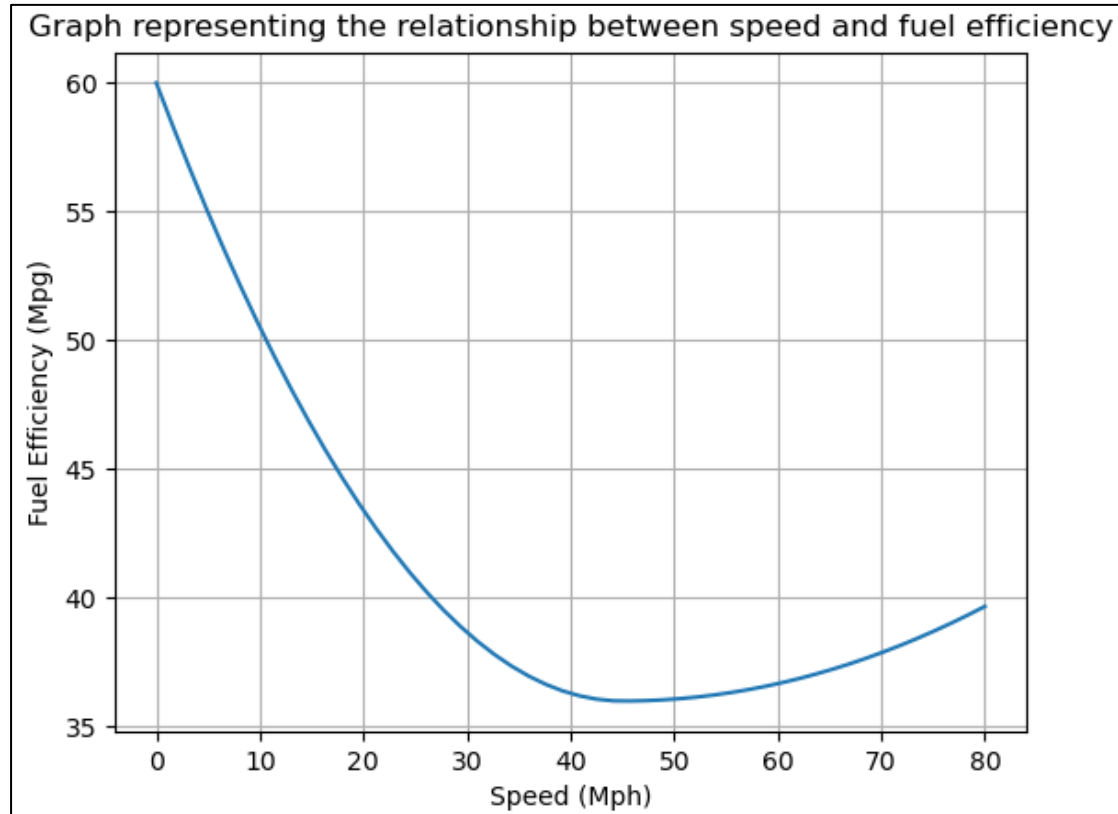


Figure 5.4.6 – The relationship between Speed and Fuel Efficiency.

calculating the consumption. Knowing the optimal speed is 45mph and the average fuel consumption is 36mpg, I will consider that to be when the consumption is optimal, and at its lowest. Also with basic knowledge about cars and physics, I know that cars will have a faster fuel consumption rate when they have a higher Revs per minute (Rpm). For speeds ranging from 0-30, the consumption should be considerably higher since the vehicle will be in a lower gear. Then as it approaches 45mph, the consumption will progressively decrease, when 45 mph is reached, the consumption will then gradually climb back up, but at a much slower rate as the vehicle is in a higher gear.

Figure 5.4.6 shows a visual representation of the formula I hand-designed to represent fuel consumption at various speeds, explicitly following the conditions explained previously.

$$((\text{Speed in MPH}) - 45)^2 \times a + 36$$

'a' is a value that represents the rate of change. When the Speed of a vehicle is less than 45, the rate of change is assigned a value of 0.01, for any other value, 'a' is considered 0.003 (1/3 of speeds below 45mph). this helps to create a slower incline after a speed of 45mph is achieved. Using this formula, we can calculate every single road edge's fuel consumption.

The next step is to discover the amount of fuel that is consumed, in 1 meter worth of distance travelled, at the exact consumption rate of the vehicle at a given speed. We do this by translating Miles per gallon, into gallons per meter. Knowing that there are 1609.4 Meters in a mile, we can first translate it into meters per gallon, For example, if a vehicle's fuel consumption is 36 miles per gallon, that would equate to 57,938.4 Meters per gallon. Then we simply have to multiply this value by the power of '-1'. This translates the value from Meters per gallon to gallons per meter. The 'Calculate_fuel_consumption' Method returns this value when called.

Finally, with the required information obtained, we can assign a fuel consumption attribute to the edges. To do so, we follow a similar procedure as the Time attribute, except we perform a different calculation. Using the newly found Gallons per meter value, I multiply it by the length of a road segment, which will return the value for the fuel consumed across that section of road. This will result in an extremely small value, but when combined with every road segment in a route, it adds up.

Choice Of Search Algorithm

Several factors need to be considered before settling on an algorithm of choice for this project. What is the main goal, To find the shortest possible route to the destination? So I need to make sure I consider all of the options. I also want to maintain continuity with NetworkX, which provides a handful of search algorithms built in.

Dijkstra's algorithm is a method for finding the shortest path between nodes in a graph. It operates by iteratively selecting the node with the lowest distance from a set of unvisited nodes and updating the distances of neighbouring nodes. The algorithm maintains a set of distances for each node, initially set to infinity, and gradually explores the graph, updating distances as it progresses. It ensures that the shortest path to each node is determined before exploring further. Usage in Route Planning: In the context of route planning, Dijkstra efficiently computes the shortest path between the start and end nodes, considering various factors, in application to my system, Fuel Consumption, Distance, and Time cost are considered to be the weights used in Dijkstra's search algorithm.

I did have a few other options when choosing an algorithm, for example, NetworkX provides A-Star search algorithm functionality in their shortest path methods.

While A-Star search is another popular algorithm for pathfinding, it is typically preferred in scenarios where heuristic information is available and can be utilized to guide the search process more efficiently. A-Star incorporates heuristic estimates of the remaining distance to the goal node, allowing it to prioritize exploration in promising directions. This can lead to faster convergence and more efficient pathfinding, especially in large graphs or complex environments. However, it does not search all of the options, unlike Dijkstra's. For this main reason, This project will use Dijkstra's to perform all route-finding operations.

Shortest Path Computation

```
shortest_path_distance: ArrayLike = (nx.dijkstra_path(G, source=start_node, target=end_node, weight="length"))
shortest_path_distance: list = [node for node in shortest_path_distance if node not in ['start', 'end']]
print(len(shortest_path_distance))

shortest_path_time: ArrayLike = (nx.dijkstra_path(G, source=start_node, target=end_node, weight="time"))
shortest_path_time: list = [node for node in shortest_path_time if node not in ['start', 'end']]
print(len(shortest_path_time))

shortest_path_fuel: ArrayLike = (nx.dijkstra_path(G, source=start_node, target=end_node, weight="fuel"))
shortest_path_fuel: list = [node for node in shortest_path_fuel if node not in ['start', 'end']]
print(len(shortest_path_fuel))
```

Figure 5.4.7 – Shortest Path computational Code

Figure 5.4.7 shows the process of producing the desired shortest paths, using their designated weight metrics: distance, time, and fuel. The 'nx.dijkstra_path()' function from the NetworkX library is utilized to compute the shortest paths. Performing this method 3 separate times, with each unique weight, allows for live comparison between each proposed choice. This provides me with the ability to prove each method is correctly producing the desired outcomes, optimising their respective weights.

The 'weight' parameter in the Dijkstra path method indicates which metric it plans to compute using, with "G" representing the Graph we created previously, containing all of the edges and nodes in the GeoDataFrame. We also previously created start and end nodes, which contain the Graphical coordination for the desired start and end positions. The output from this function is an Array List, that contains every edge the shortest path would contain. After computing each shortest path, the code filters out the start and end nodes from the resulting path lists. This step ensures that only the nodes contributing to the actual route are retained for further analysis. For test reasons, I also print the length of each path, to verify they are unique from each other. It also provides an insight into how many road segments are required when situational circumstances vary.

If I wanted to adapt this system to use the A-Star approach, I would substitute the 'nx.dijkstra_path()', with NetworkX's Alternate method "nx.astar_path()" Which instead of requesting a weight parameter, would require a Heuristic. This value would have to accurately estimate the distance from the start node, and the End node, this could be done using the Euclidean distance between each other, or a more advanced formula, taking into consideration time and speed.

Informative Display

After computing the shortest path, and storing each in a named variable, we need to select the appropriate one based on the request from the user. This is done by checking the Mode of the system. Starting with a default value of 'distance' meaning if no input is provided about what type of value needs to be optimised, then it assumes you desire the shortest distance. If a line already exists, and you desire an alternate option but wish to retain the

same start and end positions, then clicking each button will change the mode of the system and display the other paths that were generated.

```
-----
104
120
107
Mode: distance
Time taken: 0:, 42:, 34
Distance: 22.678177387 Miles
Fuel Consumed: 0.5647667720414925 Gallons
-----
104
120
107
Mode: time
Time taken: 0:, 20:, 13
Distance: 23.06529152 Miles
Fuel Consumed: 0.608585711871461 Gallons
-----
104
120
107
Mode: fuel
Time taken: 0:, 43:, 44
Distance: 22.701168114 Miles
Fuel Consumed: 0.5635818198529754 Gallons
-----
```

Figure 5.4.8 – Example Terminal Statistical output.

Next, some basic analytical processes are performed. The code initializes variables to store statistical information about the chosen route. These variables include 'total_time', 'total_distance', and 'total_fuel', which represent the cumulative time taken, distance covered, and fuel consumed along the path, respectively. The time taken is converted into hours, minutes, and seconds for better readability. Additionally, the total distance covered is converted from meters to miles for user-friendly presentation. These metrics, along with the total fuel consumed, are then displayed in the console As seen in Figure 5.4.8, providing users with information about the selected route. This is also useful for testing and the information can be used to plot graphs for comparing efficiency.

To visually represent the selected shortest path, the code plots a line on the open Window display like the one seen in Figure 5.4.9 (this line represents the data generated in Figure 5.4.8[Mode: Fuel]). This line, drawn in yellow colour, corresponds to the chosen route and helps users visualize the path directly on the map or graphical interface. The coordinates of the path nodes are retrieved from the node coordinates, ensuring accurate positioning of the line. Because the system relies on node coordinates quite frequently, we must ensure all data uses the same coordinate reference system in the pre-processing steps.

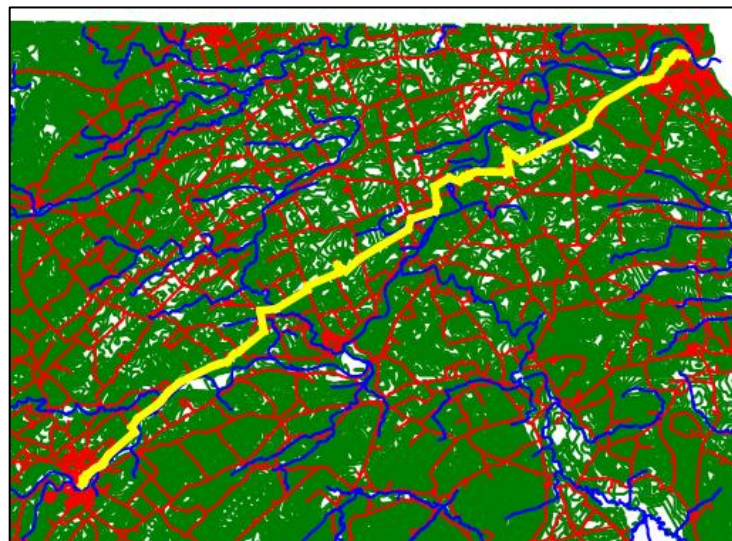


Figure 5.4.9 – Generated 'Shortest Path' line from 2 user inputs.

CHAPTER 6 – TESTING

6.1 – COVERING A SHORT DISTANCE

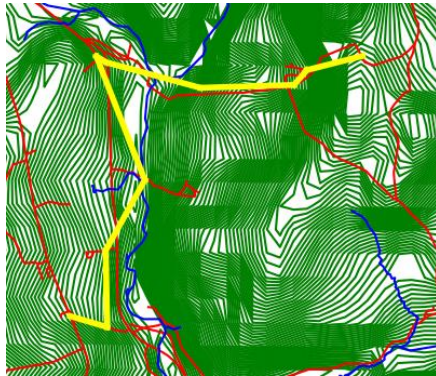


Figure 6.1.1 – small example of generated line

Firstly, I made sure that I would be able to judge the testing area using my intuition and common sense, whether a route is optimal or unreasonable, so I zoomed into my data and found somewhere with minimal turns, but a clear ‘best’ choice.

Immediately, looking at the shortest path output (Yellow) in Figure 6.1.1, it does not follow the lines of the original Road link data ... it is misaligned and seems to wander from the supposedly correct location. Yet it still tries to conform to the same

generic structure. This could be caused by the dataset missing certain sections, or specific segments of the road being made redundant in the pre-processing stages. It could also just be that the graph plotting technique used for the Shortest path output is less precise than

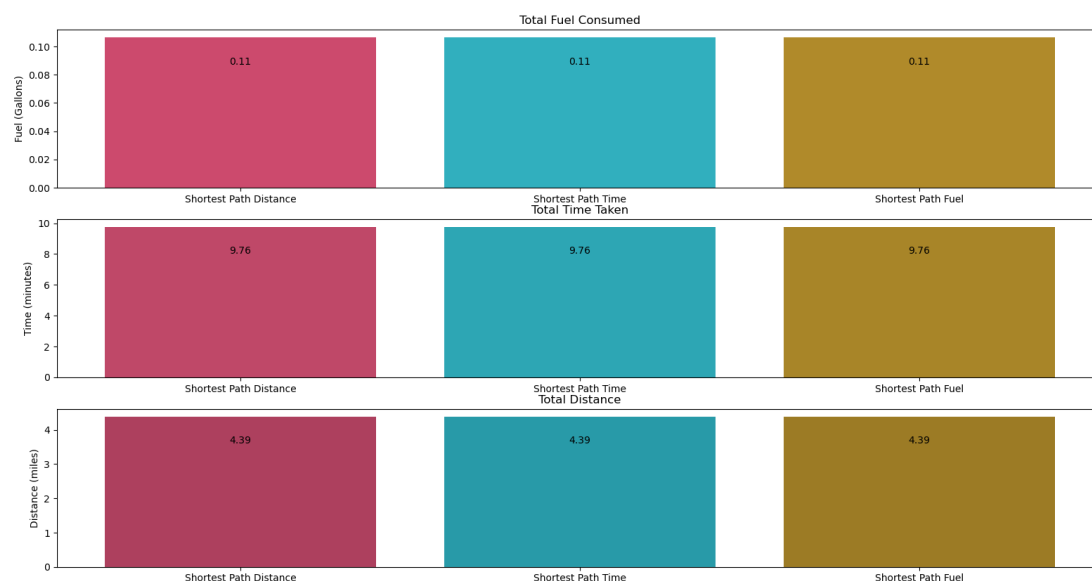


Figure 6.1.2 – Short distance path Statistics (Time, Distance, Fuel consumption)

the method used for displaying the original dataset. In future editions of this software, I will ensure continuity, so this can be avoided. From initial inspection, it seems to have generated a relatively suitable path considering the options provided. I would also agree that the system has chosen an optimal route. Based on the Values portrayed in Figure 6.1.2, There was little to no variation between paths generated, optimizing the various attributes. However it is still obvious that the system has generated the shortest possible path, which is

a good sign, testing over longer distances will be a better way to analyse variation in path optimisation, as there are more opportunities for unique choices.

6.2 – COVERING A LONG DISTANCE

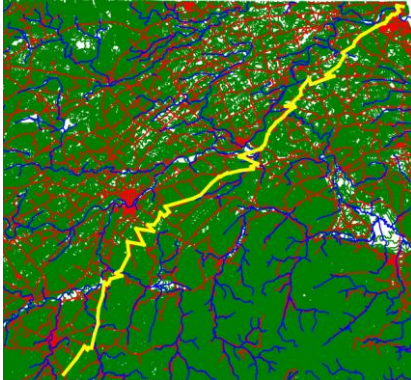


Figure 6.2.1 – Shortest Path Distance

My starting location was the furthest most top right section of the road that I had access to, and my target destination was a random road segment in the southeastern region of the map. Logical assumptions predict that the path should follow as direct a route as possible. However, there may be alternative routes that pose unexpected benefits.

Figure 6.2.1 shows the most optimal path regarding distance travelled. There are several points which could be questioned, for example around 2/3 of the way down there is a large zig-zag that isn't directly represented in the road link data, this could be faulty data or misinterpreted visualisation. It seemingly optimises the distance to the naked eye.

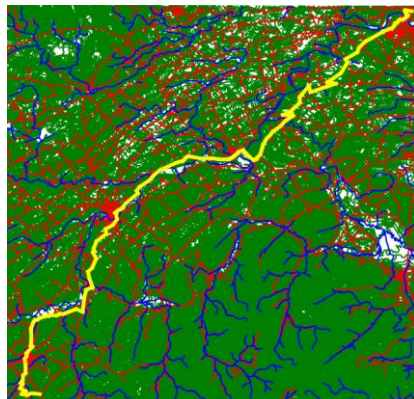


Figure 6.2.2 – Shortest Path Time

Figure 6.2.2 shows the most optimal path regarding the time taken to complete the route. It differs from the distance route significantly, with a huge deviation from the Euclidian path that the distance path followed. This could suggest that the time path favours roads that offer a faster speed limit, allowing the car to save time by significant amounts. It does however sacrifice in the distance department, and due to the increase in speed, it is predictable that the fuel consumption would have risen significantly.

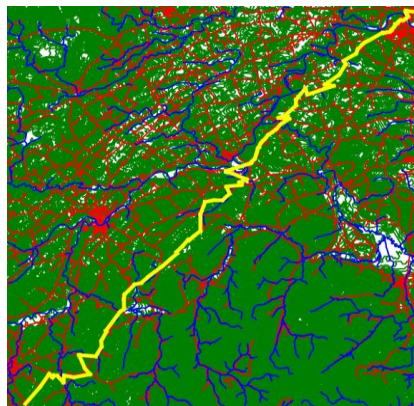


Figure 6.2.3 – Shortest Path Fuel

Figure 6.2.3 Represents the line for optimising Fuel consumption it seems extremely similar to the distance optimising path. It would be fair to assume the fuel-efficient route will mostly follow slower road segments, as they can conserve fuel, running at lower engine RPMs.

It would be interesting to see if the project could implement a weight-based decision-making path, integrating all 3 features into 1 overall value, with time potentially being the most impactful attribute considering the human requirement for punctuality.

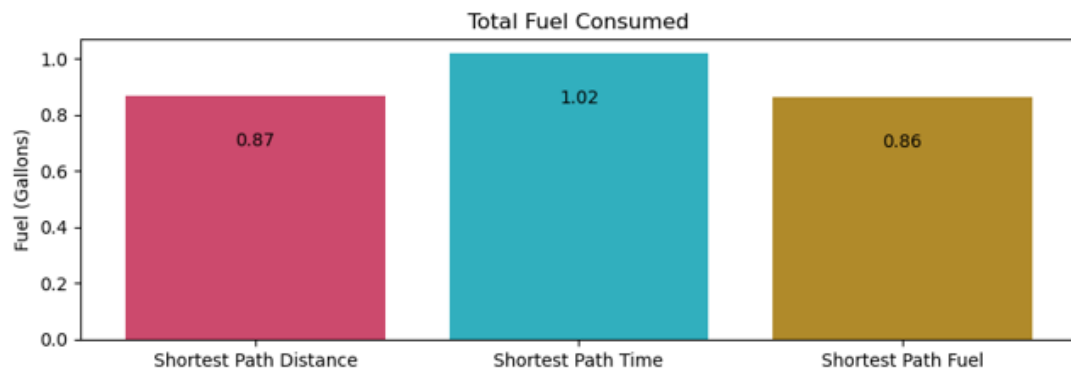


Figure 6.2.4 – Total fuel consumption for all 3 paths Bar Graph

From Figure 6.2.4, it is clear to see that the Time path consumed significantly more than both other paths, as predicted. But it is reassuring that the Path that most efficiently optimised for fuel consumption, Was the 'Shortest Path Fuel', which reinforces the structure of the algorithm.

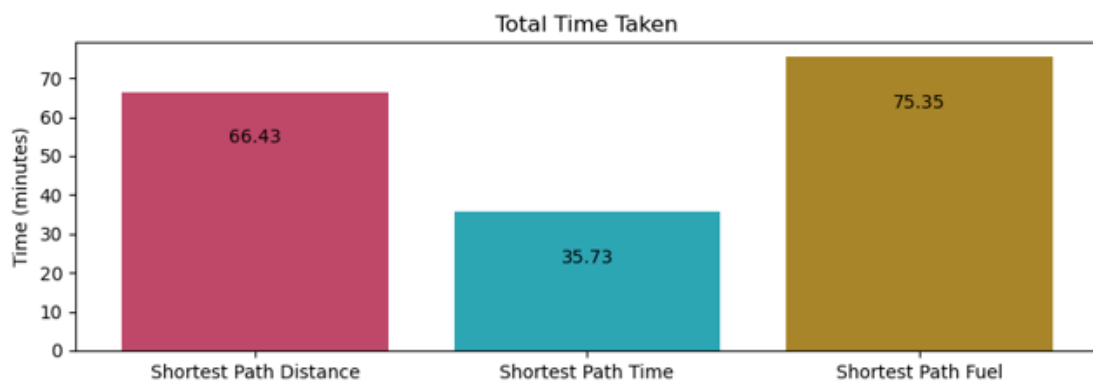


Figure 6.2.5 – Total time taken for all 3 paths Bar Graph.

Once again, the system was capable of optimizing the time best using the 'shortest path time', as seen in Figure 6.2.5. If you compare the shape of the road in Figure 6.2.2, and the Yellow Road Representing an A-Road in Figure 4.2.3 allowing for faster travel, despite it being somewhat of a detour. The 'Shortest path distance' and 'Shortest path fuel' both have much longer journeys, probably because the most direct route in terms of distance and land coverage travels through villages, and uses country roads to reach the target.

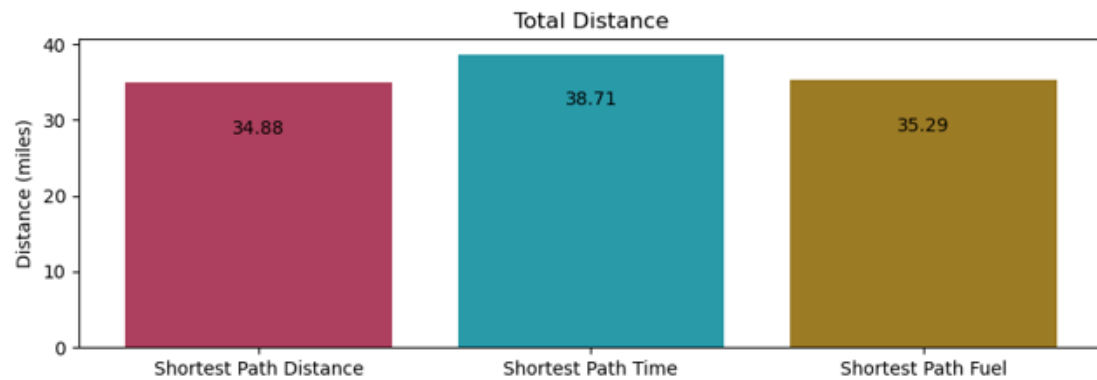


Figure 6.2.6 – Total Distance coverage for all 3 paths Bar Graph

The final attribute, Distance, also shows that the algorithm generated the lowest total distance, which is the 'Shortest Path Distance'. All 3 paths have now produced an output that positively reflects their intended optimised attributes. The distance for the Time path is marginally longer, by roughly 4 miles, yet by driving the extra 4 miles and consuming a small amount more fuel, it manages to save 30 minutes compared to the other paths, which is almost Half the time of the other paths.

6.3 – COMPARISON WITH TRUSTED SOFTWARE

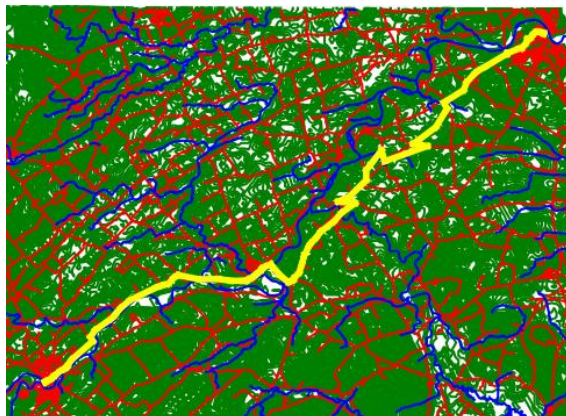


Figure 6.3.1 – Time optimised route for

```
Mode: time
Time taken: 0:, 19:, 55
Distance: 23.038572567 Mile
Fuel Consumed: 0.6079901975360344 Gallons
```

Figure 6.3.3 – Stats relating to figure 6.3.1

Figure 6.3.1 Is an image of the path generated using my system From a location called Tweedmouth, with a target destination of a place called "Tweed Tranquillity", in the Time optimisation mode. Figure 6.3.2 Is a route planned using Google Maps. They share a resounding resemblance, however, it could be argued that my generated path aligns better with the 36-minute path offered by Google. Instead of the Decidedly better

option from Google, presenting a more attractive 34 minutes. Figure 6.3.3 shows the statistics related to my generated path. While the distance is almost identical, at a near round 23 miles, compared to Google's 23.1 miles, There is a drastic difference in the Time taken. The reason for this is that Google

takes into account Traffic and road blockages. Whereas my system simply runs off a graph of edges and nodes with no real-time update. Perhaps this is where the PostGIS approach would be more suitable, as it allows for the implementation of several plugins, including Live traffic information. With this I could add a traffic weight to each edge, making them less

desirable for time algorithms. However, the fact that my system generated a nearly identical path, means that the traffic is probably negligible when it comes to affecting the decision-making of the search algorithm and would impact any route it generates proportionately to any difference. The only way it could have a significant impact is in the occurrence of a traffic jam or road accident



Figure 6.3.2 – Google Maps (2024) Generated Path

CHAPTER 7 – EVALUATION

7.1 – PROJECT MANAGEMENT REVIEW

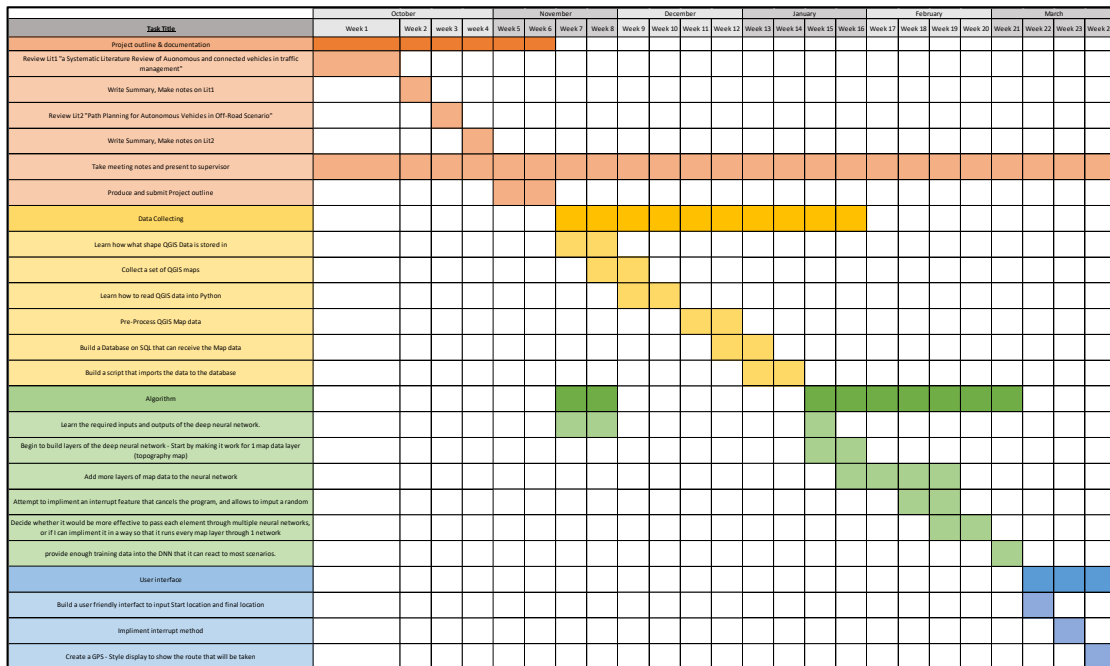


Figure 7.1.1 – Original Gantt Chart for project (last edited 13/11/2023)

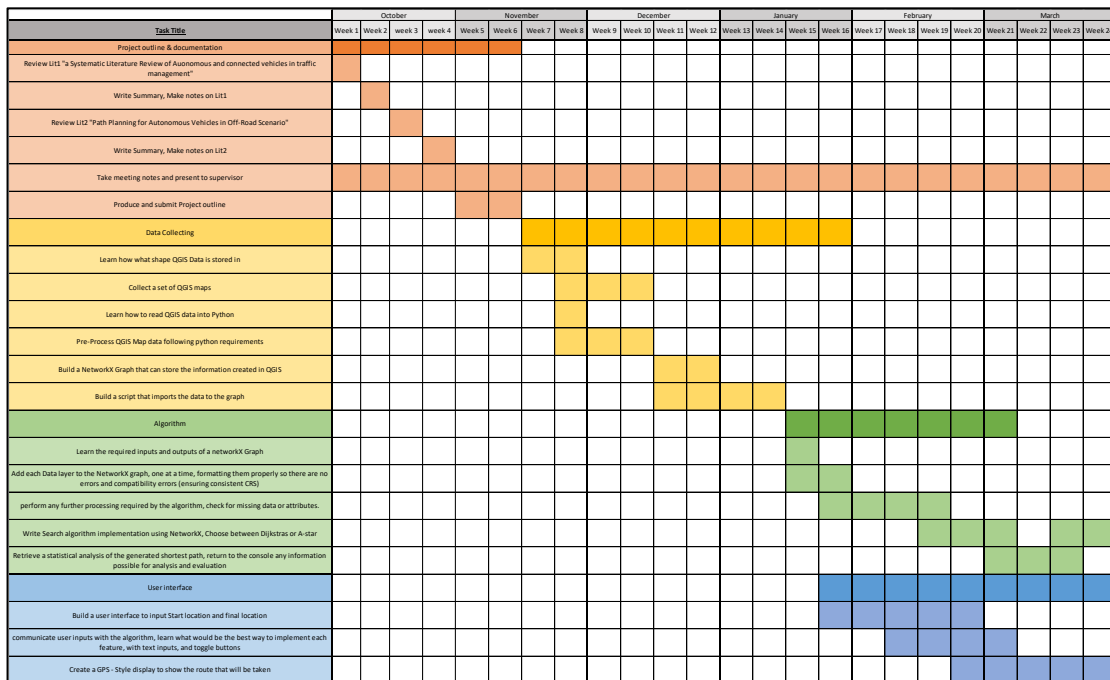


Figure 7.1.2 – Most recent Gantt Chart for project (last edited 03/04/2024)

Both Gantt charts have been made available through Appendix C as a OneDrive ShareFile for closer inspection.

In the first few weeks/months of the project, the scope wasn't fully realised, so I was only able to speculate on decisions and aspirations, which in hindsight, I was extremely optimistic about, as can be seen when comparing the old Gantt chart (Figure 7.1.1) to the new Gantt chart (Figure 7.1.2). Originally I wanted to produce a Deep neural network approach to the GIS routing algorithm, this was however quickly realised around week 8/9 that it wouldn't be feasible, as I also had to consider other modules undertaken in the Third year of university.

A decision was made to downscale the project, and instead Focus on the implementation of a search algorithm-based approach. This decision allowed me to focus on a more specific detail, and research and learn about it in greater detail. For example, I gained a much deeper understanding of Dijkstra's algorithm, and how It can be used with GIS data and Libraries like NetworkX. I was never able to achieve a neural network, which I think would still improve this system, it would require a significant amount of practice and learning, however, which wasn't possible during the time scale of this project.

Figure 7.1.2 describes the plan to add every single layer to the NetworkX graph, I sadly was unable to achieve what was truly intended by this statement. Whilst I was able to add them visually to the system, they functionally, are redundant, However, the system was still developed in a way that can be added to in future iterations, so it wasn't completely neglected.

I didn't expect the GUI to overlap so much with the algorithm, but I found that as I made changes to the algorithm, I would have to adapt a GUI in tandem to ensure it remained functional throughout the whole process, instead of waiting till the end to implement an interactive visualization. The Gantt chart did impact my ability to perform this as having the written plan kept me disciplined and reminded me of the requirements and goals I set for myself.

Much of the original Gantt Charts Data collection stage was similar to my Newer Gantt charts data collection, this is because both desired approaches used the same shape data to begin the project, so I wasn't forced to decide on project direction till quite late into the pre-processing stages, but it was still early enough that I was able to adapt the project, to go in an alternative direction, still producing a interesting analysis of GIS data use in route finding. The data I prepared affected the decision unexpectedly, as I realised I wouldn't have enough information and usable data to provide a neural network with the information it required.

7.2 – FUTURE ADDITIONS

Improving Current Road Network Data

Enhancing the accuracy of the dataset is crucial for improving the reliability and effectiveness of the routing algorithm. This can be achieved by adding more detailed information about the road network, including the precise locations of junctions, intersections, and other special road features As mentioned in 6.1. By incorporating this additional data, the routing algorithm can make more informed decisions, such as choosing optimal turning points and navigating complex road layouts more efficiently. To increase dataset accuracy, I would consider using crowdsourced data sources, satellite imagery, or alternative governmental databases to gather detailed information about the road network. Additionally, advanced data processing techniques such as machine learning algorithms

could be utilized to automatically identify and map road features from raw data sources. However, this exceeds the scope of this project.

Real-Time Traffic Information

Another important enhancement would be the addition of a real-time traffic plugin to the system. Real-time traffic data provides up-to-date information on road conditions, including congestion, accidents, and road closures. Integrating this data into the routing algorithm enables the system to dynamically adjust routes based on current traffic conditions, thereby optimizing travel time and improving overall efficiency. Without the Real-time Traffic information, The system's output is unreliable, as discussed In 6.3.

To implement this, one could explore integrating APIs provided by traffic data providers such as Google Maps or TomTom, allowing the system to access real-time traffic information and incorporate it into route planning. Additionally, machine learning algorithms could be employed to predict traffic patterns and optimize routes proactively.

Adding the Next Dataset: Contour (Elevation)

One significant improvement could involve integrating contour data into the search algorithm. Contour data provides valuable information about the elevation changes across the landscape, which can significantly impact routing decisions, especially for vehicles susceptible to changes in gradient. To implement this, one could start by creating data cluster polygons representing different elevation levels using Geographic Information System (GIS) software such as QGIS. These data clusters can then be loaded into the NetworkX graph, allowing the routing algorithm to consider elevation changes along with road networks. By incorporating contour data, the system can generate more efficient routes that minimize fuel consumption and travel time, especially in areas with varied terrain.

In conclusion, the future aspirations for the project encompass a range of enhancements aimed at further optimizing the routing algorithm and improving the overall user experience. By integrating contour data, real-time traffic updates, and increasing dataset accuracy, the system can deliver more efficient and reliable route navigation tailored to specific user preferences and environmental conditions.

CHAPTER 8 REFERENCES & EXTRA INFORMATION

8.1- REFERENCES

Alternative Fuels Data Center: Maps and Data - Fuel Economy at Various Driving Speeds [WWW Document], 2021. . U.S. Department of Energy - The Alternative Fuels Data Center (AFDC). URL <https://afdc.energy.gov/data/10312> (accessed 4.2.24).

Cinch. (n.d.). "What is a good MPG? Average car MPG explained." Retrieved from <https://www.cinch.co.uk/guides/ask-the-experts/what-is-a-good-mpg>

Diken, Chris; Erica Francis. "Ten fuel-saving tips from a hypermiler". NBC News – available at https://en.wikipedia.org/wiki/Energy-efficient_driving#cite_note-10

EPSG. (2021.). WGS 84 (G1762). Retrieved from <https://epsg.io/6326-datum>

GeoPandas. (2024.). Documentation. Retrieved from <https://geopandas.org/en/stable/docs.html>

Gharaee, Z., et al.(2021) Graph representation learning for road type classification. Pattern Recognition, Volume 120, 108174, ISSN 0031-3203. Available at <https://www.sciencedirect.com/science/article/pii/S0031320321003617>

Google Maps (2024) United Kingdom, Scale: 2km Available at - <https://www.google.com/maps/@55.6864138,-2.2406424,11.33z?entry=ttu> Retrieved on (Accessed 19.3.24)

Hong, Z., Sun, P., Tong, X., Pan, H., Zhou, R., Zhang, Y., ... & Xu, L. (2020). Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map. ISPRS International Journal of Geo-Information, 10(11), 785. – <https://www.mdpi.com/2220-9964/10/11/785>

Li, J., Sun, C., & Wang, H. (2017). An Algorithm for Path Planning of Autonomous Vehicles in Unknown Environments. International Journal of Vehicle Autonomous Systems, 19(4), 319-330. - <https://www.inderscienceonline.com/doi/abs/10.1504/IJVAS.2017.087148>

L.V. Lucchese (2021) " Trick to fix broken polygon geometries in QGIS 3.X" Available at: <https://www.luisalucchese.com/post/fix-broken-polygon-geometries-qgis/>

Matplotlib Development Team. (2024.). Matplotlib Documentation. Retrieved from <https://matplotlib.org/stable/index.html>

Nayak, A. (2019, August 27). Shortest Path Distance Approximation Using Deep Learning: Node2Vec. Towards Data Science. <https://arxiv.org/pdf/2002.05257>

National Archives. Open Government License (Version 3). Retrieved from <https://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/>

Nature England (n.d.) Special Protection Areas (England). Contains public sector information licensed under the Open Government Licence v3.0. - [Special Protection Areas \(England\) - Special Protection Areas \(England\) - Natural England Open Data Geoportal.url](https://www.nature.com/open-data/geoportal/special-protection-areas-england)

NetworkX Developers. (2023.). NetworkX 3.2.1 Reference. Retrieved from <https://networkx.org/documentation/stable/reference/index.html>

Ordnance Survey. (n.d.). Road network data - OS Open Roads - Vector Map Data for GIS - Free OS Data downloads. Contains public sector information licensed under the Open Government Licence v3.0. Retrieved from [OS Open Roads - Vector Map Data for GIS - Free OS Data downloads.url](#)

Ordnance Survey. (n.d.). Terr50 Contour map data - OS Terrain 50 - Visualise Landscapes in 3D - Free OS Data downloads. Contains public sector information licensed under the Open Government Licence v3.0. Retrieved from [OS Terrain 50 - Visualise Landscapes in 3D - Free OS Data downloads.url](#)

Ordnance Survey. (n.d.). Water network data - OS Open Rivers - OS Data downloads - OS Data Hub. Contains public sector information licensed under the Open Government Licence v3.0. Retrieved from [OS Open Rivers - OS Data downloads - OS Data Hub.url](#)

Riverbank Computing Limited. (2023.). PyQt5 Documentation. Retrieved from <https://www.riverbankcomputing.com/static/Docs/PyQt5/>

Shapely Contributors. (2024.). Shapely User Manual. Retrieved from <https://shapely.readthedocs.io/en/stable/manual.html>

United States Geological Survey. (2010). GMTED2010 – Contour data. Retrieved from https://topotools.cr.usgs.gov/gmted_viewer/viewer.html

Windsor, J.S., Firth, P.G., Grocott, M.P., Rodway, G.W., Montgomery, H.E., 2009. Mountain mortality: a review of deaths that occur during recreational activities in the mountains. Postgraduate Medical Journal 85, 316–321. <https://doi.org/10.1136/pgmj.2009.078824>

8.2 – APPENDIX

Appendix A =

As the Code section is 400+ lines of code, and uses Imported data files, it has been made available as a Zipped Folder using one-drive using the following link:

- https://herts365-my.sharepoint.com/:u:/g/personal/br20aab_herts_ac_uk/Ed2Y5A7-DuhNgIBBm6RG8BcBzI4pt4OKCQDzThxr6ocOCQ?e=0meJiE

Appendix B =

The QGIS Project has been made available including every GeoPackage in a Zipped Folder on one-drive using the following link:

- https://herts365-my.sharepoint.com/:f:/g/personal/br20aab_herts_ac_uk/EvC0atHtORVHuNIR9-RLqH4BsWHfcN5xT-wnv8TB1WY8PQ?e=Q3dIO1

Appendix C =

Both Gantt Charts presented as png images – Shared Via Zipped folder on one-drive using the following link

- https://herts365-my.sharepoint.com/:f/g/personal/br20aab_herts_ac_uk/EghuRrT5-CxFjRnUjpuvf4EBY_1k0dQ5z_003Y6H7JM0jg?e=EZmjUR