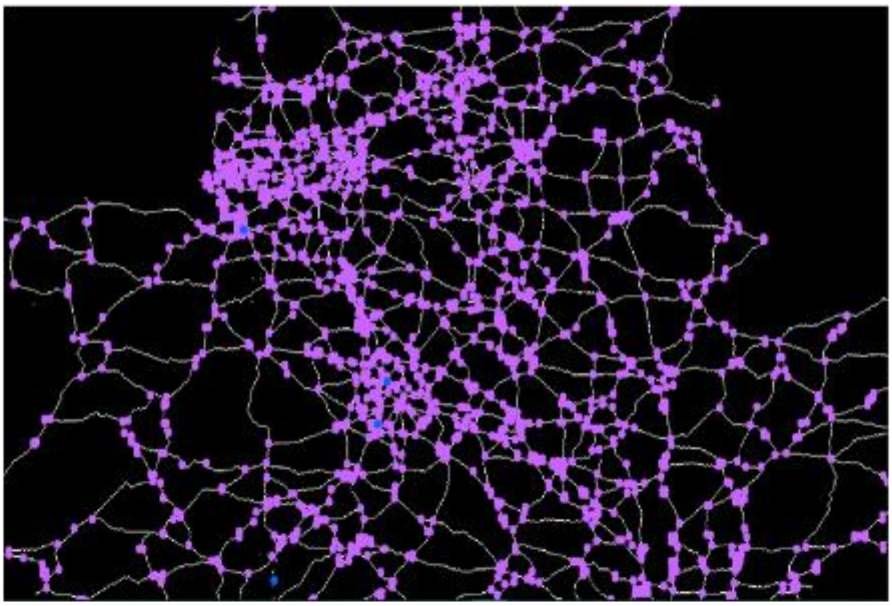
University of Essex

BSc Computer Science

CE301 Capstone Project



The Pathfinding Visualizer

Author	Supervisor	Second Assessor
Emmanuel Oladipo Olaoye 1905600	Dr Themistoklis Melissourgos	Dr Riccardo Poli

23 August 2023

Acknowledgments

First and foremost, I want to thank the Lord God Almighty for his help and guidance throughout this project. Without him, none of this would be possible. Secondly, I would like to express my sincere gratitude towards my project supervisor Dr Themistoklis Melissourgos for his invaluable advice, comments, and suggestions over the course of this project especially while I was changing the topic. I would also like to extend my thanks to my second Assessor Dr Riccardo Poli for crucially pointing me in the right direction regarding thresholding and skeletonization. The completion of the Computer Vision Section of this project could not have been possible without Dr Adrian Clark's teaching of the Computer Vision Module.

Abstract

Pathfinding is the process of finding the most optimal path between two points. This project takes the guesswork out of manually planning a journey and plotting the route towards a desired destination.

My project aims to find the shortest route between two destinations on a map for a traveller. This will be achieved by taking in an image of a road and then masking out the lines of the road, therefore removing unnecessary items such as rivers, forests, and lakes. Subsequently, the masked image is converted into a binary image where the process of skeletonization will then reduce the binary image into a 1-pixel-wide skeleton. From the skeleton, the program will recognise the road intersections and place them into a neighbourhood representation which Dijkstra's algorithm can process to return the shortest path to finally display in a user-friendly UI to the user.

Existing Solutions

Project Name	*Comments*
The Pathfinding Visualiser By Clement Mihailescu [1]	This is the most popular pathfinding visualiser on the internet. It is a web- is a web-based application that allows a user to place a start and destination node anywhere on a 2d grid. While visually displaying the search process, the program calculates the shortest distance. The application used 6 different pathfinding algorithms which allow users to place walls to try and block the algorithm forcing it to calculate an alternate path. Available at Link
Pathfindout By Dean DeHart [2]	Similar format to the Pathfinding Visualiser however [2] added terrain generation which attributes a low cost to green nodes which represent grass, medium cost to yellow nodes which appears to look like sand and a high cost to grey and blue nodes which represent rocks and water. Available at Link
Pathfinding Visualizer By Kelvin Zhang [3]	The most unique methods of pathing visualisation. The application displays the shortest path between two points in a city. Unsure how it reads the map, but it returns the shortest walking route Link
Google Maps By Google [4]	Undoubtedly one of the most extensively utilized applications, it boasts an exceptional user interface, enabling users to pinpoint any location on the planet and obtain the shortest route. This route can be measured in terms of travel time or distance, with real-time updates available during the journey. For further details, refer to this live update link.
Skeleton Tracing By Lingdong Huang [5]	This I s an application that allows you to interact with binary image skeletons in real life. You can navigate through the page look select what type of image you want skeletonised and using you mouse, you can draw on it turning you newly created into its skeletonised representation. Available Link.

Comments on existing solutions

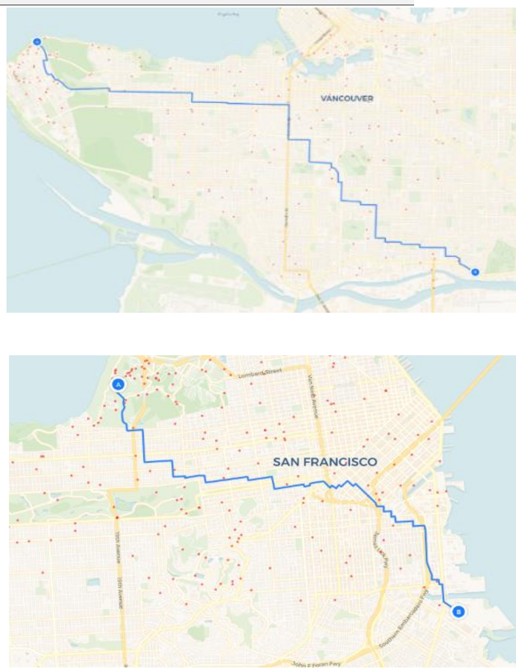


Figure 1 Screenshots of "Pathfinding Visualiser" [3]

As demonstrated in the references [1][2][3][4], a majority of the existing solutions appear to adhere to a similar format

1. Giving users the ability to select which area/ region the wish to explore.
2. Select the Point(s) for their desired route.
3. Displaying the most optimal route by means distance/ time back.

I was most impressed with Kelvin Zhang's "Pathfinding visualiser" solution [3]. They gave the user the ability to place a start and end node anywhere within "real world" cities. The user would then choose between a wide array of pathfinding algorithms. After clicking the "visualise" button the application would start calculating the shortest path while displaying the visited nodes in real-time. While impressive, Im unsure of how the application was able to calculate the shortest path or what the developers used to represent their nodes/ graph. Did they get the graph by analysing the image or use pre-existing database such as google maps API to create the graph?

Introduction

Pathfinding involves determining the most efficient route between two points. Leveraging computer vision capabilities, this project eliminates the need for manual journey planning by automatically plotting routes to desired destinations. As mandated by project requirements [6], this final report will offer an overview of graph theory and computer vision, emphasizing their relevance to the project. Utilizing figures, diagrams, code snippets, walk-throughs, and pseudocode, I will highlight key software features, followed by comprehensive technical documentation that justifies each aspect of the developed product.

Origins of the Project

This project initially began on a smaller scale, drawing inspiration from Clement Mihailescu's "Pathfinding Visualiser" project [1]. The original objective [6] was to create a simple web application that visually demonstrated how various path-finding algorithms discover the shortest route. However, upon examining existing solutions, I realized that my proposed approach might not be unique enough to warrant development, considering the prevalence of similar solutions. This prompted a shift in strategy. I opted to highlight the potential of computer vision in identifying the shortest path, drawing inspiration from Google Maps [4] ' utilization of satellite imagery. Initially, my intention was to employ "raw" satellite imagery as input and train a machine learning algorithm for road and street extraction, akin to the efforts of researchers at [7]. Yet, practical constraints became apparent, including the need for substantial training data to process images, the necessity of longitude and latitude data for image location, and the demand for accurate detection. Factors such as satellite image altitude, scaling, and aperture focal points further complicated the endeavour.

The goal of my project is to determine the shortest path between two destinations on a given map:

1. The program takes one or more maps as input.
2. It devises a methodology to ascertain distances between roads and intersections.
3. Utilizing these distances, the program computes the optimal path between user-defined points.

Technical Documentation

Image processing and computer vison

Figure 2 Panoramic image of Seiser Alm plateau in South Tyrol, Italy [9]

Digital image processing or **Image processing** involves the manipulation of digital images using computer algorithms to improve their quality, extract valuable information, and automate tasks. It employs mathematical models and algorithms to enhance and analyse digital images, aiming to make images more useful and informative. The ultimate objectives of digital image processing are to enhance image quality, extract meaningful data, and automate various tasks through computational methods [8]. Most people will and have utilised image processing throughout their daily lives by, taking and editing photographs, panoramas, photocopying, driving autonomous vehicles and even paying games that use depth sensing cameras such as the Xbox Kinect [10]. Computer Vision (**CV**) employs the various Image processing techniques. Computer vison relies on image processing techniques to process and analyse and return useful information the information from these images like how your brain tries to interpret the visual information given to you by your eyes. Computer vision can by typically categorised into 3 levels:

4. **Low Level vision:** This level involves the initial processing and analysis of visual data at the pixel level particularly, colour processing, segmentation, thresholding, colour masing skeletonization, and region labelling and interpretation [low level vison slides][11].
5. **Intermediate level vison:** this level goes beyond the basic pixel level by extracting more complex information to distinguish and describe objects in scenes. It involves identifying finding and matching features such as edges and contours. [12]
6. **High level vision:** This is the most advanced level. Here, we rely on machine learning algorithms to recognise object in images and use high level decision making to understand scenes.[13]

N.B. It should be of note that the techniques and algorithms are not isolated to the levels. There can and is a lot of overlap between them; their categorising depends on the context of the problems/ solutions being found.

This project utilised image processing/computer vision low-level vison techniques by way of the OpenCV library to extract the intersections of the roads in a map so that they can be represented as nodes in an adjacency matrix to then find the find the shortest path between them. The computer vison methodology can be broken down as follows:

1. Inputting and preparing the image
2. Colour masking
3. Thresholding the image
4. Converting the colour image into a binary image
5. Skeletonising the binary image
6. Locating the intersections from the image skeleton

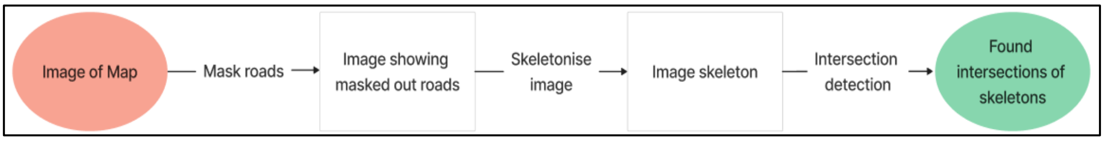


Figure 3 Flowchart of Computer Vison Methodology

Binary images and thresholding

Figure 4 An image of Eiffel Tower (left) as a binary image (right) [15]

Binary images or bitmaps are images that haves with exactly 2 possible values typically black (whose pixels are displayed as 0) or white (whose pixels are displayed as 1 or 255) with no shades of grey in between [14]. Binary images are used for various tasks dues to their simplicity and efficiency as there is no need for the functions/ algorithms to account for the rest of the colour values in the spectrum. Binary images are typically created by thresholding a greyscale or colour images.

Thresholding is a technique used in image processing and computer vision to separate objects from the background of an image. The basic idea is to convert a grayscale or colour image into a binary image, where each pixel is either black or white [14]. Thresholding works by comparing the intensity values of each pixel in the image to a threshold value. If the pixel's intensity is greater than the threshold value, it is assigned a white colour. If the intensity value of a pixel is less than or equal to the threshold value, it is assigned a black colour. In Figure 4, the threshold level would be the peaks between the blue and green levels turning the blue-sky pixels in the background black and the rest of the pixels white. Thresholding is a simple and effective technique used by for a wide range of applications, including object detection, segmentation, and tracking. There are several different types of thresholding algorithms, the most common type of algorithm is the Otsu's method. Otsu's method (see Figure 5) is a way of automatically thresholding images. However, it only works for images where the foreground and background have similar number of pixels.

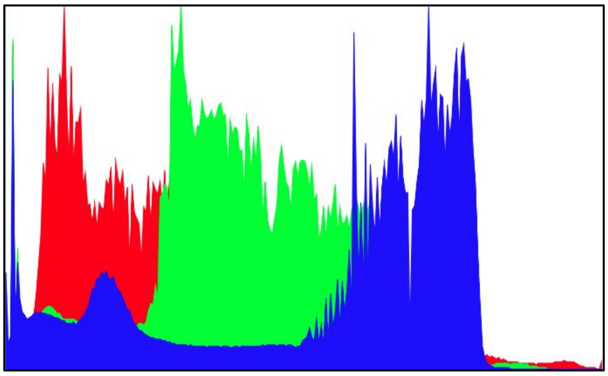


Figure 5 Histogram of the Eiffel Tower [15]

```
function otsu(im):
    // Determine the threshold by Otsu's method.

    // Initialization
    nr, nc, nb = dimensions of im
    npixels = nr * nc * nb
    ngreys = round(im.max() + 0.5)
    hist = array of size ngreys filled with zeros

    // Compute histogram
    for each pixel (r, c, b) in im:
        v = round(im[r, c, b] + 0.5)
        hist[v] += 1

    sum = sum of all pixel values in im
    sumB = totB = threshold = max_var = 0

    // Iterate over histogram bins
    for t in range(0, ngreys):
        sumB += hist[t]
        if sumB == 0: continue
        sumF = npixels - sumB
        if sumF == 0: break
        totB += t * hist[t]
        mB = totB / sumB
        mF = (sum - sumB) / sumF
        var = (sumB / sum) * (sumF / sum) * (mB - mF)^2
        if var > max_var:
            max_var = var
            threshold = t

    return threshold
```

Figure 6 Pseudocode of Otsu's method algorithm interpreted from [16]

Skeletonization

Figure 7 Skeletonised binary image on Los Angeles

Skeletonization or thinning in image processing, is a morphological algorithm that transforms an object in binary image into its “medial representation.” [17] it does so by reducing these object’s boundaries until they are 1 pixel wide. However Skeletonised objects (**skeletons**) retain their original size as the end points of the skeleton extends all the way to the edges of the original object in the image.[18] Properties of a skeleton include:

- Skeletons must have preserved the topology and connectivity of the original object.
- Skeletons must be thin at most 1 pixel wide.
- Skeletons must represent the medial axis of the original object.
- They must preserve the primary shape and features of the object.

The properties ensure accuracy for shape analysis, pattern recognition, and other image processing tasks.

Skeletonization Techniques and Methods

1. **Medial Axis Transform**: this method finds the central point inside an object by measuring distances, showing where the object's skeleton would be. This helps simplify the object's shape while keeping its structure [19]. This method can produce accurate skeletons that closely match the object's true shape and typically used to extract the skeletons of complex or irregularly shaped objects [20]. However, it can be computationally expensive/inefficient and may produce noisy results. Due to the use of a Euclidian Distance map, the method does not guarantee “error free” connected skeletons. [21]
1. **Thinning**: Using algorithms like Zhang-Suen, Guo-Hall, or Rosenfeld-Pfaltz, thinning Iteratively erases the foreground pixels of an object to return a 1-pixel wide skeleton. This method is used to extract the skeleton of a wide range of objects [22]; however, it can produce skeletons that may not accurately represent the object's true shape.
2. **Morphological Skeletonization**: this method applies morphological operations like erosion and dilation to derive the object's skeleton. This technique can be used to extract the skeleton of a wide range of objects particularly “fuzzy” objects. However, it can produce thin skeletons that may not accurately represent the object's true shape. [23]
3. **Voronoi Diagram**: Utilizes the Voronoi diagram to find points equidistant from the object's boundary, from which the skeleton is extracted. This method can produce accurate skeletons that closely match the object's true shape and is particularly well-suited to extracting the skeletons of geometrically rigid and flexible objects and road maps. However, more work needs to be done [24]

Skeletonization Applications

Skeletonization has many applications in computer vision, image processing, and computer graphics. Examples of the most common applications include:

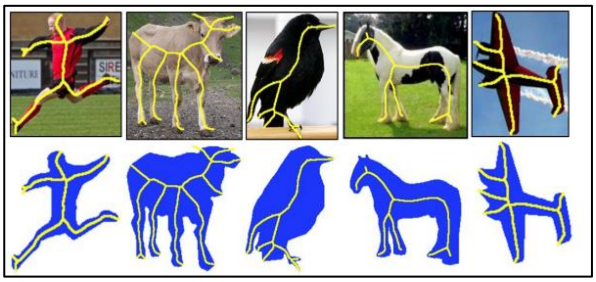


Figure 8 Skeletons of natural images used and skeleton reconstruction of images (coloured blue) [26]

1. Object recognition: Skeletonization can be used to extract the key features of an object, which can then be used for object recognition and classification see fig. [24][25]
2. Shape analysis: Skeletonization can be used to analyse the shape of an object, and to extract geometric features such as curvature, length, and width. [25]
3. Medical imaging: Skeletonization can be used to extract the structure of blood vessels, nerves, and other anatomical features from medical images such as [20], X-rays, CT scans, and MRI scans.
4. Robotics: Skeletonization can be used to extract the structure of objects in a scene, which can then be used for robot navigation and manipulation. [27]
5. Agriculture: Skeletonization is used in agriculture to analyse the structure of plants (Figure 8) and to extract features such as stem thickness, leaf shape, and fruit size fields, and is a fundamental tool in computer vision and image processing [28]

Figure 9 Skeletons of plants [28]

Skeletonization - Medial axis transform

The Medial Axis Transform (MAT) is a mathematical and image processing technique that is used to find the skeleton of a shape or object in an image. The skeleton represents the medial axis of the object, which is a set of points that are equidistant to the object's boundaries in all directions [29]. This technique is often used in image analysis and computer vision for various applications such as shape analysis, pattern recognition, and object representation.

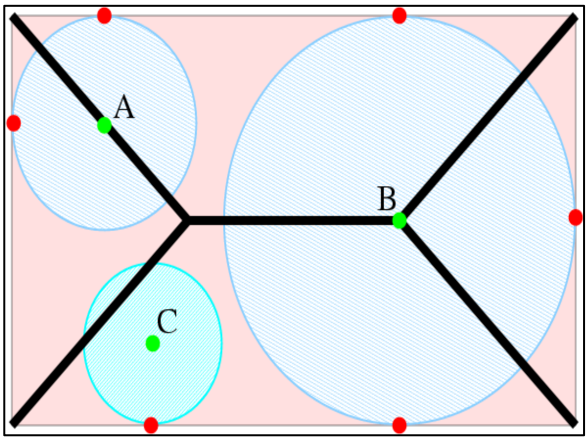


Figure 10 image showing a skeleton (black) of a rectangle (shaded red) using medial axis transform. Points A and B are skeletal, but C is not.
Courtesy of [30]

The Medial Axis Transform skeletonization process:

Image Pre-processing:
The process begins with a binary image where the object of interest is represented as white pixels on a black background. The object's boundaries should be well-defined in the binary image.

Distance Transform:
The first step is to compute the Distance Transform of the binary image. The Distance Transform assigns each pixel a value corresponding to its distance to the nearest background (black) pixel. This effectively "inflates" the object, creating a gradient-like image.

Finding the Medial Axis Points:
The points that constitute the Medial Axis are those where the distance to the object's boundaries is maximized. These points lie at the centre of the object, and they are equidistant to the object's boundaries in all directions.

Skeletonization:
The skeletonization process involves thinning the computed Medial Axis points to create a one-pixel-wide representation of the object's centreline. This is done by iteratively removing pixels from the edges while preserving the connectivity of the skeleton.

Result Visualization:
The result of the Medial Axis Transform is a skeletonized representation of the original object (see figure above). This skeleton captures the essential structural information of the object while reducing its representation to a simplified form.

The Medial Axis Transform skeletonizations are particularly useful for objects with complex and irregular shapes[fuzz], as it provides a concise representation that retains important geometric features. In applications like image recognition and shape analysis, the skeleton can be utilized for object matching, deformation analysis, and other tasks that require robust shape descriptions. While Medial Axis Transform is a powerful technique, the quality of the skeletonization can vary based on factors such as object shape, noise in the image, and the algorithm used. Different algorithms and variations exist to compute the Medial Axis, each with its own strengths and limitations.

Skeletonization implementation

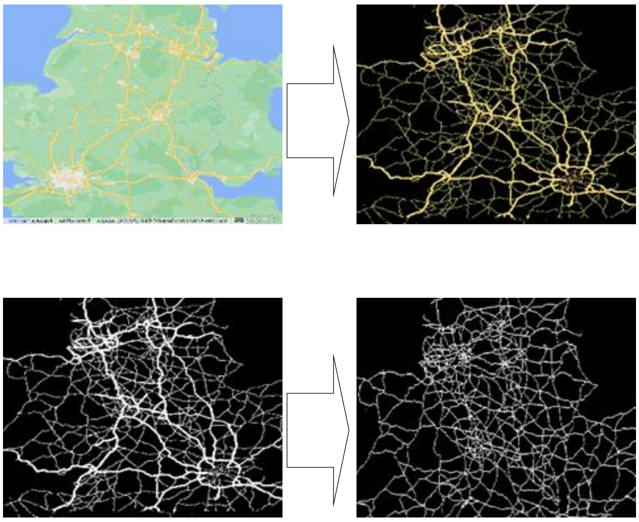


Figure 11 Stages of the threshold function (starting from the top-left) (1.) original image (2.) filtered out yellow areas (3.) binary image (4.) skeletonised image.

The code starts by importing necessary libraries and modules, including **NumPy**, **cv2** (OpenCV), **skimage** (scikit-image) fun. **low_yellow** and **high_yellow** define a range of HSV (Hue-Saturation-Value) values that correspond to yellow colour in the image.

In the main body of the file, the **Threshold ()** function takes an input image (**im**) as its parameter. It starts by converting the input image to a binary image using a threshold level grey level of 127. Then, it converts the input image from BGR to HSV colour space using **cv2.cvtColor()**. A mask is created using **cv2.inRange()** to filter out colours within the defined yellow range. The **cv2.bitwise_and ()** operation is performed to retain only the yellow areas from the original image. **filters.threshold_otsu()** calculates an Otsu threshold to create a binary mask. The binary mask is then processed using **morphology.medial_axis()** to find the skeleton of the mask. **img_as_ubyte()** converts the processed mask to an 8-bit image. Finally, processed image is returned as output (see Figure for an example output).

Edge detection implementation

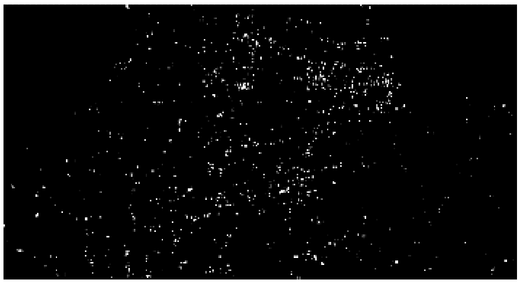


Figure 12 Image of the intersections from the joints image.

The function takes one parameter **im** the input image. Firstly, the input image has a Gaussian blur applied using **cv2.GaussianBlur()** with a kernel size of **(3, 3)** to reduce noise. The blurred image is thresholded using **cv2.threshold()** with the **cv2.THRESH_BINARY + cv2.THRESH_OTSU** flags. This converts the image into a binary image where pixels are either 0 or 255. The structuring elements (**verticalStructure** and **horizontalStructure**) are defined using **cv2.getStructuringElement()** for morphological operations. These elements are used for detecting vertical and horizontal lines. Morphological opening is then applied using **cv2.morphologyEx()** to detect vertical and horizontal lines separately. **vertical** and **horizontal** represent the images after applying morphological operations.

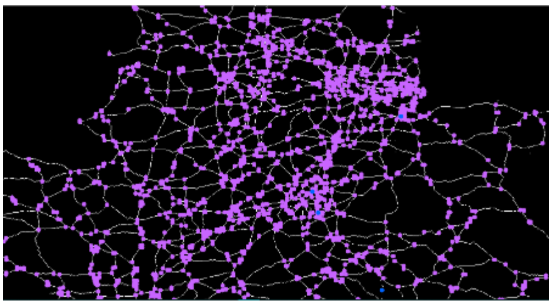


Figure 13 Image of the detected intersections (highlighted in purple).

To detect the intersections of lines a bitwise AND operation (**cv2.bitwise_and()**) is applied between the horizontal and vertical images (**horizontal** and **vertical**). The result is stored in **joints**. The function **np.argwhere(joints == 255)** returns the indices (pixel coordinates) where intersections are detected in the binary image. Subsequently, the code iterates through the pixel coordinates where intersections are detected. If the corresponding pixel in the original image (**img**) is white (255), it adds the **(y, x)** coordinates of the intersection to the **listOfIntersections**.

Finally, the function returns a list of **(y, x)** coordinates where intersections are detected in the original input image.

Graph theory

In computer science and mathematics**, Graph** is a network that helps define and visualise relationships and networks.

In Figure 14 the circles represent the components in the network while the lines connecting the components represent the relationships between the components. The components are called graphs/**nodes** and the connecting line represent **edges.** Graph theory involves studying the properties of these networks and the practical applications that can be solved.

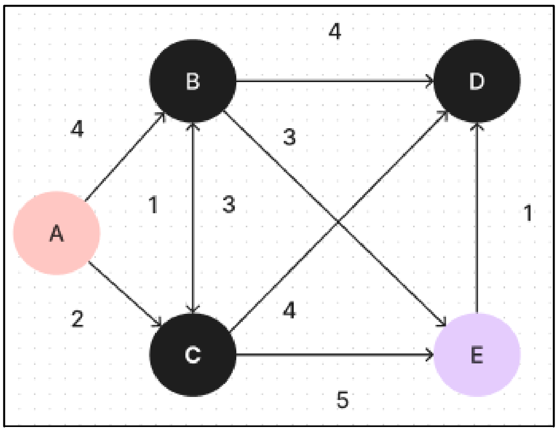


Figure 14 Example of a graph

What is a Graph?

A graph G = (V,E) is a pair of sets where V is a set of elements called vertices, and elements called edges with each edge a having a connection between 2 vertices [31]

What is a node?

Nodes (**V**) are the components represented as points in the graph. Nodes can represent various things, such as people in a social network, cities in a map, junctions between roads or any other objects of interest. [31]

What is an edge?

Edges (E = v, w): are the connections or relationships between nodes {v, w}, represented as lines connecting the nodes. Edges can have a direction (in directed graphs) [31] or no direction (in undirected graphs). Throughout the project I referred to graphs as undirected graphs. The associated weights or costs of an edge are quite arbitrary representing the strength or distance of the relationship between the connected nodes.

Graphs are incredibly versatile and find applications in areas like social networks [32], transportation and routing systems, recommendation systems, computer networks, project management [33], and more. They provide a powerful and intuitive way to represent and analyse interconnected data, making them an essential tool in many domains.

Graph representations

The 2 main methods of representing graphs as stated by [34] are adjacency matrix and **adjacency lists.**

Adjacency lists

According to [34], the properties of an adjacency list include a graph which is represented as a collection of lists with each list containing a node and a set of neighbours and their weights. It is used to describe what are a particular node's neighbours and their associated weights. To represent an adjacency list in Python, I used nested dictionaries.

```
adjacencyList = {
    'A': {'B': 4, 'C': 2},
    'B': {'D': 4, 'E': 3},
    'C': {'D': 4, 'E': 5},
    'D': {},
    'E': {'D': 1},

}
```

Figure 15 Using nested dictionaries to represent Adjacency lists.

Graph traversal?

Graph traversal refers to the process of systematically visiting all the nodes and edges of a graph, covering all its elements. It is a fundamental operation in graph theory and plays a crucial role in various applications across computer science and real-world scenarios. Traversal algorithms help explore the relationships between nodes and gather information about the structure of the graph.

Graph traversal methods

The 2 main methods of graph traversal are Breath first Search and Depth First Search (DFS):

Breadth First Search (BFS) is a graph traversal algorithm that systematically explores a graph or tree by visiting all the nodes at the current level before moving to the next level [35]. It uses a queue data structure to manage nodes, ensuring that nodes closer to the starting point are explored before those farther away. BFS was invented in the late 1950s by E. F. Moore, who used it to find the shortest path out of a maze [36]. BFS is commonly employed for tasks like shortest pathfinding and analysing the connectivity of graphs

Depth First Search (DFS) is a Traversal algorithm used for traversing a tree or graph data structures. The algorithm starts at the start node and keeps exploring the neighbouring nodes until it cannot any longer before backtracking and exploring again [35]. It was first investigated by Charles Pierre Trémaux in the 19th Century as a form of solving mazes [36]. Using DFS, I was able to find and traverse the white pixels of a binary Image to find all the neighbouring skeletal intersections of any given intersection.

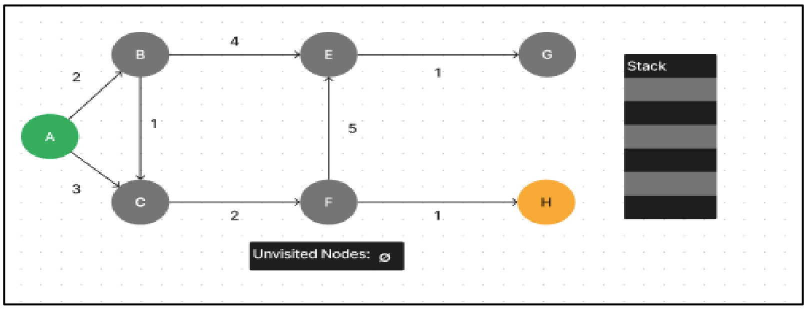


Figure 17a Graph using Depth-First Search

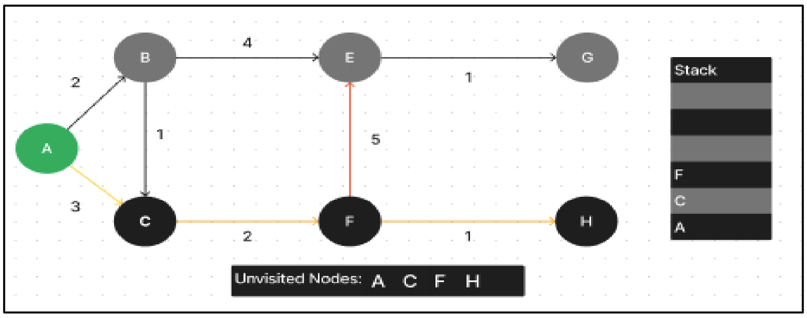


Figure 17b Graph using depth-First Search examining node F.

Let Figure 17a Be a graph with A as the start node and H as the destination node. There is also an empty stack for the node that will be explored and an empty list of unvisited nodes. Once DFS is ran, it will start by examining the node A. Since A is not visited, it will be placed in the visited list and pushed on top of the unvisited stack. Next, the algorithm will examine its neighbouring nodes B and C.

Now the algorithm will examine node C as the current node, it will also add the node to the visited list and push on top of the stack. Next, the algorithm will then look at current node C's neighbours which will be node F. The algorithm will also add F to the visited nodes list and push the node to the top of the stack. At this point the neighbouring nodes are E and H. Choosing node H would mean that the algorithm would have found the destination node, However, not all nodes would have been visited as the stack would still have nodes A, C, F and H in the stack (in figure 17b) meaning the whole graph has not been explored and there still may be alternate paths to the destination node. To find these paths, the algorithm will backtrack by popping the node H on top of the stack therefore removing it from the stack and revealing node F as the new node on top of the stack. Subsequently, the algorithm will now examine F's neighbours again. Since Node has already been visited, the algorithm will visit node E and push it. E only has node H as its neighbour so it will examine and push it to the top of the stack. Like Node H, G has no neighbours so the algorithm will back track again back to E and since E has no unvisited neighbours it will keep back tracking until it finds a node will unvisited neighbours. In this case, the algorithm will back track to node A and visit the only unvisited neighbour node B and add and add it to the list and end the algorithm.

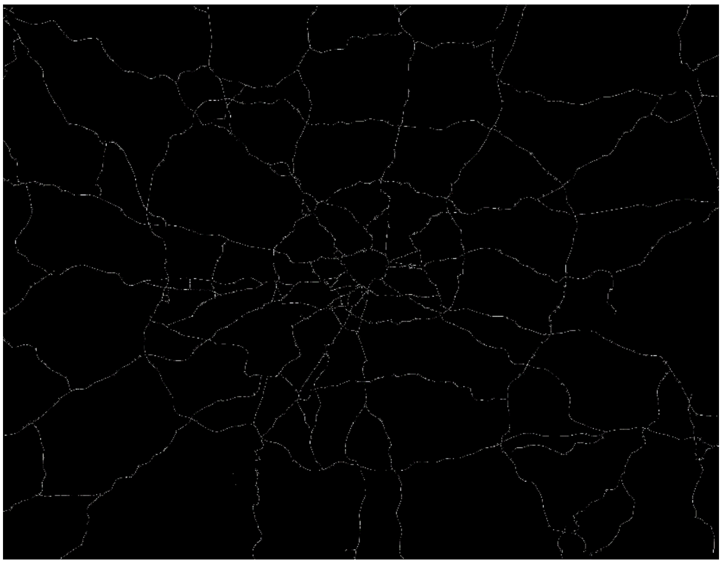


Figure 18 Skeletonised image of London

Implementation Justification

By the midpoint of my development of the project, I had created a Dijkstra's algorithm function that used a neighbourhood representation in the form of an adjacency list and used image processing techniques to mask out, threshold, and skeletonize the major roads of a map. I then created a function that found the pixel coordinates of the skeletal intersections. To figure out the shortest path between 2 intersections, I now faced the problem of figuring out how to find the neighbouring intersections for all the skeletal intersections in an image. Having already understood graph traversal through my research, I needed to find an appropriate solution. I then began to break the problem down into smaller, simpler problems. The first problem was figuring out how to travel along a skeletonized image. My solution was to use coordinate geometry to move in all 8 directions. Since I knew that my image's skeleton would only be 1 pixel wide, I knew that I only needed to move along one plane, i.e., from left to right, up, and down, and in one diagonal plane (north-west to south-east, and south-west to north-east). By also creating a function that returned the list of intersections as their pixel coordinates, I knew exactly where each intersection would be, guaranteeing my start point and endpoint for exploration.

My next idea was to build a function that started at a particular intersection and kept exploring along the adjacent white pixels until it found another intersection or ran into a dead end. At first, I planned to create a brute force function that would check if it was connected to every intersection, but I quickly realised that it would be extremely inefficient as it would take O(N), where N was the number of intersection nodes in the image. However, from this solution, I realised that what I was logically trying to create was a DFS implementation that started at a particular intersection, kept exploring until it found another intersection, and would then explore using another possible direction. This is where I began to start creating my DFS implementation.

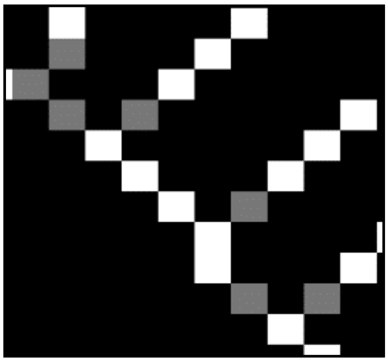


Figure 19 Snippet of 1-pixel wide binary image skeleton with the intersections in grey

At first, my plan was to implement DFs iteratively; however, I realised that I needed to keep track of all the possible paths, and keeping track of such paths was difficult. This was where I then decided to recursively implement DFs, as it was easier to keep track of and branch out to a new path if necessary. This way, for each path to an intersection, I would be able to return a list containing the order of the nodes visited while not ending the programme.

```
Def DFS (node, path):

    if node is an intersection:
        save path
    else:
        for all neighbours in nodeNeighbours(node):
            nextNode = neighbour
            if node is visited or if node != white or node not in image:
                mark node as visited

            else:
                Dfs(nextNode, path)

```

Figure 20 Pseudo code of recursive Depth first search

At this point, I was able to formulate all the possible paths to the neighbouring intersections, and all I needed to do was wrap the function in a for loop. For each intersection in a list of intersections, it returned all the paths to all its neighbours. Having every connecting path meant that all I needed to do was add the Euclidian distance for each node along the path to get the total distance between each intersection, finding its connecting edge weight. I solved by taking the first node in the connecting path list, finding the distance to the next node in that list, and adding it to the total along the way. However, not all distances are created equal. Moving left, right, up, and down meant that the Euclidian distance would be equal to 1, but diagonally would be the square root of 2. I considered this when adding the distances. The last step would simply be placing the nodes, neighbours, and distances into their respective neighbourhood representations.

DFS implementation

The **DFS()** function is a recursive Depth-First Search algorithm implementation that explores the paths between intersections in a binary image skeleton. It builds a graph where intersections are nodes, and paths between intersections are edges. The graph can be used to analyse connectivity and distances between intersections in the skeletal image.

returnAllPaths Function:

This function acts as a wrapper for the DFS process: It initializes **visitedNodes** and **path** lists. It then calls the **DFS()** function with these initialized lists and other required parameters.

DFS Function

The function takes 5 parameters: **node**, **image**, **path**¸ **visitedNodes**, and **listOfIntersections.**

- **node:** Current node.
- **image:** Binary skeleton image.
- **path:** Current path.
- **visitedNodes: a** list of visited nodes.

listOfIntersections: List of intersection nodes.

Secondly, the function starts by adding the current node to **visitedLists** and **paths.** the algorithm immediately checks If the current node is an intersection (excluding the starting node), a path has been found between intersections. If so, it creates a copy of the current path and adds it to the list of all paths (**allPaths**). This signifies that a path has been found.

If the current node is not an intersection or if it is the starting node, the function will then continue to examine the nodes neighbours. It iterates through 8 possible directions (up, down, left, right, diagonals) For each direction, the **nextNode** is calculated using the **next_node ()** function. The algorithm then validates the **nextnode** by checking if the node is within the image's boundaries (the **inbounds()** check), Is part of the binary skeleton **(is_white()** check) or if it Has not been visited before (**nextNode()** not in **visitedNodes**). If the **nextNode** passes all validity checks, the **DFS** function is called recursively with the **nextNode** as the new current node. This is the core of DFS: exploring as deep as possible before backtracking. Once all paths from the current node are explored (all possible directions have been checked), the algorithm backtracks. It removes the current node from both **path** and **visitedNodes.** This allows the algorithm to explore other paths from the previous node. The algorithm continues exploring all possible paths and backtracking until all paths from the starting node have been explored.

Helper Functions:
A set of helper functions assists in navigation, validation, and operations within the algorithm:

1. **next_node**: Defines how to move in different directions based on a direction index.
2. **move_up**, **move_down**, **move_left**, **move_right**, etc.: Functions that calculate new coordinates when moving in specific directions.
3. **inbounds**: Checks if a given node's coordinates are within the image boundaries.
4. **is_white**: Determines if a node in the image is part of the binary skeleton.
5. **is_intersection**: Checks if a node is one of the intersections.
6. **euclidianDistance**: Calculates the Euclidean distance between two nodes.

Subgraph and Graph Generation:
Within the **recursiveNeighbourhoodGraph ()** function, an empty dictionary called **graph** is created. The function then iterates through each intersection node present in the **listOfIntersections**. For every intersection, an empty list named **allPaths** is initialized to store paths between intersections. The **returnAllPaths** function is employed to generate these paths, and subsequently, an attempt is made to construct a subgraph associated with the current intersection. This involves calculating the lengths of the generated paths using the **pathLength** function. The first node in each path is as a key, while the path length functions as its corresponding edge-weight in the subgraph. Through the **subgraphAdder** function, these subgraphs are incorporated into the main **graph**. The intersection nodes serve as keys, and their respective subgraphs constitute the values in the **graph** dictionary. Once all intersections have been processed, the fully assembled **graph** is returned as the result of the function, effectively mapping intersection nodes to their connected paths and linked subgraphs.Top of Form

Pathfinding

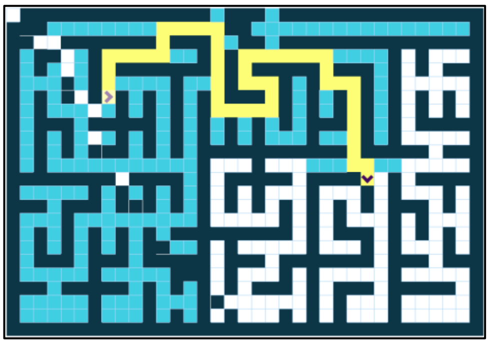


Figure 21 pathfinding to solve a maze.

Path finding is the plotting, by a computer application, is the process of finding to a path between two points. it starts from a goal node and reaches a goal node by repeatedly searching for the goal node [37]. However, there may be a case where there are multiple paths to the goal node, in this case finding the optimal path where the distance or sum of the path's edges are minimised should be selected. This is also referred to as the ***"the shortest path"*** [37] states that the 2 primary problems of pathfinding are finding a path between 2 nodes and the shortest path.

Algorithms used in path finding.

- **Dijkstra's algorithm** is a graph search algorithm that finds the shortest path from a starting node to all other nodes in a weighted graph. It maintains a set of nodes with known minimum distances and iteratively selects the node with the smallest distance, updating its neighbours' distances if a shorter path is found. The process continues until distances to all nodes are determined. (Explained in detail below.) [38]
- **A* search** is a prominent and versatile pathfinding algorithm that efficiently finds optimal paths by evaluating nodes based on a weighted combination of their current path cost and a heuristic estimate. The algorithm intelligently combines aspects of Dijkstra's algorithm and selects nodes with the lowest total cost, and its variations include dynamic weighting, bidirectional versions, and the use of pattern databases for heuristics. [39]
- **Greedy Best-First Search** is an AI algorithm that seeks the most promising path between a start and a goal. It assesses potential paths based on their costs and expands the one with the lowest cost, continuing this process until the goal is achieved.[40]
- **Breadth First Search (BFS)** is a graph traversal algorithm that systematically explores a graph or tree by visiting all the nodes at the current level before moving to the next level [35]. It uses a queue data structure to manage nodes, ensuring that nodes closer to the starting point are explored before those farther away. BFS is commonly employed for tasks like shortest pathfinding and analysing the connectivity of graphs. [36].
- **Depth First Search (DFS)** is a graph traversal algorithm used to explore and navigate through a graph or tree structure. Starting from a chosen node, it explores as far as possible along each branch before backtracking [35] It is often implemented recursively and is used for tasks like pathfinding, topological sorting, and solving puzzles. N.B. it **does not** guarantee the shortest path. (Explained in detail above) [35].

Applications of pathfinding

- **Logistics and Supply Chain Management:** Did you know (its commonly said) that ups drivers rarely ever turn left. This is due to their navigation software called Orion. [41] Pathfinding is crucial in navigational software by helping its users to find the shortest route not only in distance but also in time.
- **Maze Solving**: Pathfinding algorithms solve mazes, aiding in puzzle-solving and automated exploration. [35]
- **Air Traffic Control**: Pathfinding aids in managing flight routes, ensuring safe and efficient air travel.
- **Navigation Systems**: GPS devices and mapping applications employ pathfinding to provide users with accurate directions and estimated travel times [42]
- **Video Games**: Pathfinding is integral to character movement, enemy AI, and generating optimal routes in video games. It ensures lifelike and strategic behaviour. [45] [42]
- **Robotics and Autonomous Vehicles**: Pathfinding guides robots and self-driving vehicles in navigating environments, avoiding obstacles, and reaching destinations efficiently [43].
- **Emergency Response Planning**: Pathfinding helps responders navigate disaster-stricken areas, aiding in search and rescue operations [47].
- **Virtual Reality and Augmented Reality**: Pathfinding enhances user experiences by enabling virtual and augmented objects to navigate real-world spaces realistically [48].

Dijkstra’s algorithm

Dijkstra’s algorithm was developed by Edwin Dijkstra in 1956 [49]. It is a graph search algorithm that guarantees the shortest path from a start node to a goal. However, it can only calculate the shortest path based on the paths edge weights and is commonly referred to as the **shortest distance**. The weights can only be non-negative [46].

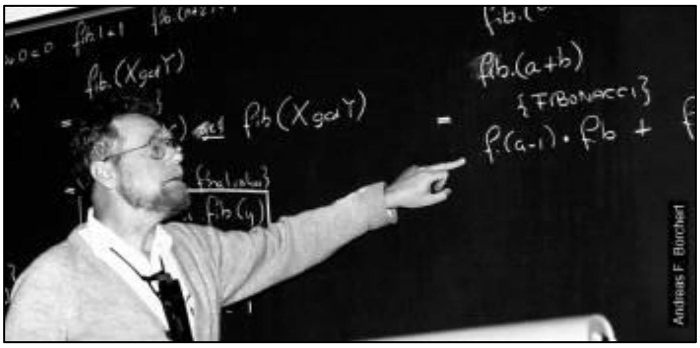
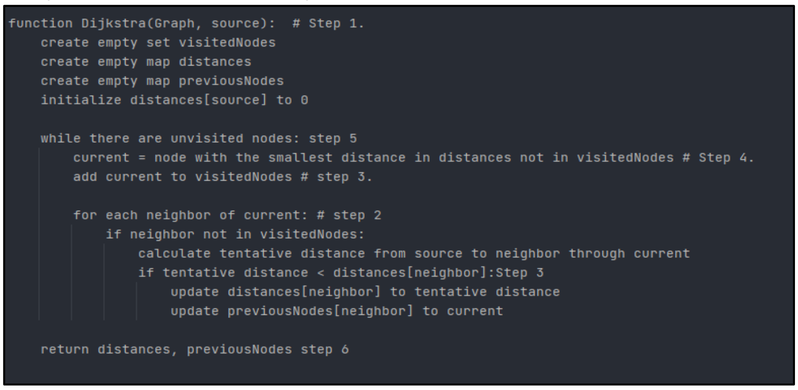


Figure 22 Edwin Dijkstra [50]

Algorithm

1. **Initialization:** Starting with a graph, start node and destination node, the algorithm creates an empty set called “unvisited nodes” for all the unvisited nodes and the algorithm add all nodes to this set. The algorithm then assigns a provisional distance of 0 to the start node and assigns infinity to all other nodes.
2. **Visiting Neighbouring Nodes:** For the current node, the algorithm will examine all the unvisited neighbouring (or adjacent) nodes. For each neighbour, the algorithm will calculate the distance between the neighbour and start node through the current node. With this newly calculated distance, the algorithm will compare the distance to one already assigned to its neighbour. If the newly calculated distance is shorter than the already assigned distance, the algorithm will replace it with the newly calculated shorter distance.
3. **Marking Visited nodes:** after examining all the neighbouring node of the current node the algorithm will mark the current node as visited and removes (or pops) it from the visited set. The visited nodes will not be examined again.



```
function Dijkstra(Graph, source):  # Step 1.
    create empty set visitedNodes
    create empty map distances
    create empty map previousNodes
    initialize distances[source] to 0

    while there are unvisited nodes: step 5
        current = node with the smallest distance in distances not in visitedNodes # Step 4.
        add current to visitedNodes # step 3.

        for each neighbor of current: # step 2
            if neighbor not in visitedNodes:
                calculate tentative distance from source to neighbor through current
                if tentative distance < distances[neighbor]:Step 3
                    update distances[neighbor] to tentative distance
                    update previousNodes[neighbor] to current

    return distances, previousNodes step 6
```

Figure 23 Dijkstra’s algorithm pseudocode

4. **Finding the next current node:** the algorithm then selects the node with the shortest known distance as the new current node.
5. **Repeat steps 2-4**
6. **End:** if the destination node has been marked as visited the algorithm ends. If there is a case where the shortest provisional distance if infinity, then the algorithm also ends; however, this means that there is not a known path between the start node and the unvisited paths.

Dijkstra’s algorithm Walkthrough

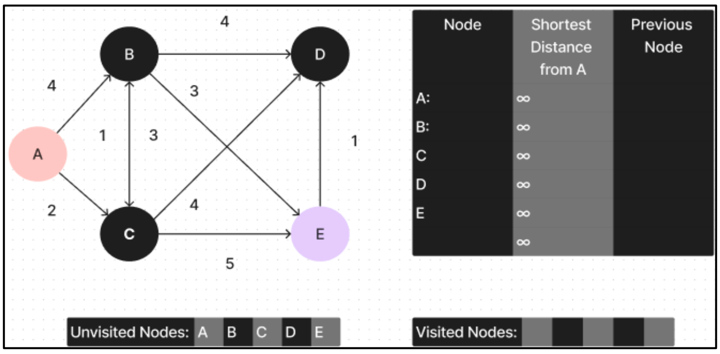


Figure 24 Visual representation of a graph

let Figure 3a be a visual representation of a graph where A is the start node and E is the destination node. There is also a table of unvisited nodes containing the nodes in the graph, a visited table which is empty to start. Once Dijkstra’s Algorithm has run it will generate the table containing the shortest distances from a starting node to all current nodes. so, for example the shortest distance from A to D is 6 making the previous node would be C. Since C will also have its own shortest distance and the previous node means that once you have reached E, you would be able to back track via the previous vertex to find the shortest path. Because all the nodes from A are unexplored, the algorithm will set all the distances in the table to infinity (a high incalculable number).

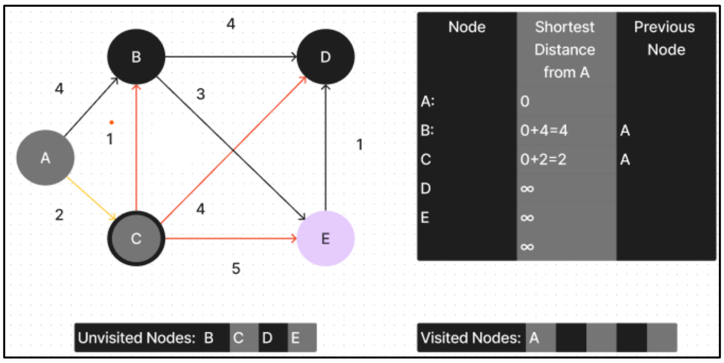


Figure 26 Examining node C.

Once ran, Dijkstra’s Algorithm would start by setting the current node to A and then examine the unvisited node with shortest distance from A. Since it is itself, it will mark it down as simply 0 on the table. Then it will examine the unvisited neighbouring nodes which are B and C. from there it will calculate the distance from each neighbour from the start vertex which will be 0+4=4 for be and 0+2=2 for C. and the since the calculated distance for B and C are less than the known distance on the table. The algorithm will add the distances and the previous node A to the table accordingly. Since all the neighbouring nodes distances from the current A have been calculated and added, the algorithm will then add A to visited and move on. Subsequently, the algorithm will then set the current node as the node with the shortest known distance from A which is C with 2. From there, the algorithm will then examine the unvisited neighbouring nodes B, D and E. The algorithm will also calculate the distance from each node to the starting node which will be 2+1=3, 2+4=6, 2+5=7 for B, D, E, respectively. Since node B’s calculated distance from A is less than its known distance 3, The algorithm will replace it with the new calculated distance and change the previous node to C. The algorithm will also update all the D and E and mark C as visited. From here, the algorithm will continue to loop by setting the current node as the neighbour with the shortest known distance and find the distances if it is neighbours until it reaches the goal node. From here it will backtrack to the previous node adding the shortest distance as it goes on until it reaches the start node.

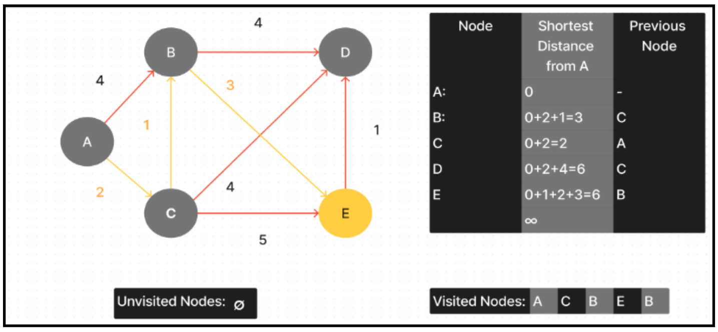


Figure 25 Dijkstra's algorithm result

By the end of the process the shortest path distance between A and E will be calculated to be 6 with shortest path being A->C->B->E as shown in Figure 4

Dijkstra's Algorithm of Implementation

The **DijkstraAlgorithm()** functions starts by taking 3 parameters: **graph, start**, **goal.** The graph parameter represents the graph on which the function will iterate through. **Start** and **goal** represent the starting and ending nodes for finding the shortest path. Firstly, the function initializes several variables:

- **Infinity** is set to the maximum integer value in the system, representing an infinite distance.
- **Shortest_distance**: A dictionary that will store the shortest distance from the start node to each node in the graph.
- **Track_predecessor**: A dictionary that will keep track of the predecessor node for each node in the shortest path.
- **Track_path**: An empty list that will be used to track the final shortest path from the start to the goal node.
- **Unseen_nodes**: A dictionary that initially contains all nodes in the graph. It is used to keep track of the nodes that are yet to be visited during the algorithm's execution.
- **Tempgraph**: since the algorithm will use the **pop()** function to delete the contents of graph a copy of graph and **unseen_nodes:** graph that holds all the unseen variables takes that copy to be used freely:

Secondly, the function initializes the **shortest_distance** dictionary so that each node is set to infinity, except for the start node, which is set to 0. a priority queue called **priority_queue** with a single element: a **tuple (0, start)**. A priority queue efficiently extracts the node with the shortest distance.

The main part of the algorithm starts with a while loop that iterates as long as the **priority_queue** is not empty. There, the algorithm takes the node with the minimum distance from **priority_queue**, which is stored in. The algorithm checks if the "minimum distance" of **min_distance_node** is greater than the current shortest distance for the node **shortest_distance**[**min_distance_node**].

If so, it means that this node has already been visited and updated with a shorter distance, and the algorithm skips and does not continue examining this node by using **continue**. If the node is not skipped, the algorithm then iterates though all the neighbour nodes of **min_distance_node** (**path_options**) and updates their distances if it finds a shorter distance.

This is achieved by exploring all of **min_distance_node** the neighbouring nodes **neighbouring_node,** here, the algorithm compares the current distance of **neighbouring_node** (which is stored in the **shortest_distance**[**neighbouring_node**]) to the possible shortest distance of the **neighbouring_node.** This distance is obtained by taking the edge weight '**weight'** between **min_distance** and **neighbouring_node** and adds it to the shortest distance from **start** to **min_distance** (which is stored in **shortest_distance[min_distance_node]**). The result of this gives the total distance from **start** to **neighbouring_node** forming the path **start** -> **min_distance_node -> neighbouring_node.**

At this point, the algorithm will compare if this distance is smaller than the current shortest distance of **neighbouring_node** (i.e., **weight + shortest_distance[min_distance_node] < shortest_distance[neighbouring_node]**) then, the algorithm has found a shorter path. There, the algorithm updates **neighbouring_node** shortest distance to the new smaller distance (at **weight + shortest_distance[min_distance_node]**). The algorithm also updates the previous node for **neighbouring_node** to the **minimum_distance_node**, marking that the shortest path from **start** goes **minimum_distance_node** and then stores this in the **track_predecessor** dictionary. Finally, since the shortest path distance to **neighbouring_node** has been changed and may need to be changed in the future the program adds **neighbouring_node** as well as the shortest distance to the to the **priority_queue** as a tuple (**shortest_distance[neighbouring_node], neighbouring_node**). This process continues until the priority queue is empty, and all nodes have been visited.

After finding the shortest path, the algorithm reconstructs the path from **goal** to **start** by backtracking through the **track_predecessor** dictionary. The path is stored in the **track_path** list. The function finally returns the **track_path** list, which represents the shortest path from **start** to **goal**, along with the shortest distance from start to goal.

Time complexity

In a typical directed graph of with each vertex having a non-negative edge weight, the time complexity of Dijkstra's algorithm if implemented with a binary heap is $O((|E+V|) \log|V|)$ where V is the number of vertices and E is the number of edges in the graph [51]. This is because the binary heap allows for an efficient extraction of the node with the shortest distance in each iteration in turn contributing to the algorithm's overall time complexity. In my iteration of Dijkstra's algorithm my function uses a binary heap as a priority queue.

To calculate the time complexity of my iteration of Dijkstra's Algorithm with a priority queue, the algorithm must be broken down in to three parts:

1. Building the initial priority queue
2. The main loop operations
3. Backtracking the path

Building the initial priority queue: initializing the variables and setting the distances to infinity takes $O(|V|)$ where $|V|$ is the number of vertices in a graph. Since the initialised **priority_queue** starts with just node, the time complexity of initialising the **priority_queue** is $O(1)$.

The main loop operations: The for loop continues as long as **priority_queue** is not empty. In each iteration of the loop, the algorithm performs:

1. Taking the minimum distance from the priority queue takes $O(\log|V|)$ since the heap must be readjusted after a node is removed.
2. Iterating through all the neighbouring node edges. In the worst case the algorithm will iterate though each edge once taking $O(E)$ time where E is the number edges.
3. Updating the distance and the previous nodes of **neighbouring_node** requires **priority_queue** to be updated which will take $O(\log V)$ for each node.

Therefore, makes the total time complexity for the main loop operations $O(V*(\log V+E*\log V))$.

Backtracking the path: In the worst case the algorithm backtracks from **goal** to **start**, the time complexity is **O(V)**.

Adding up all time gets:

1. $O(V)$ (*Building the initial priority queue*)
2. $O(V * (\log V + E * \log V))$ (*The main loop operations*)
3. $O(V)$ (*Backtracking the path*)

Simplifying further achieves:

$O(V * \log V + V * E * \log V + V) = O((V + V * E) * \log V) = O((E + V) * \log V)$

In the case where the **graph** is dense (V^2) the complexity of the graph is $O((V^2 + V) * \log V) = O(V^2 * \log V)$ where the graph is at its most inefficient. [51]

In the case where the graph is sparse the is $\mathbf{O((E + V) * \log V)}$ where the graph is at its most efficient.

User interface

Key user Gui features include:

- The ability to select any node on the map.
- Easy / instantaneous switching of map
- Having nodes change colour when hovered.
- Displaying the country flag when then region button is hovered over, or page is active.
- Being able to select nodes by clicking them.
- A preview of the map shown and the flag.
- Having the shortest path displayed instantly after selecting the last node.
- Being able to clear the map to start selection points again.
- Selecting the maps
- Exit page confirmation.

Figure 27 The London map page

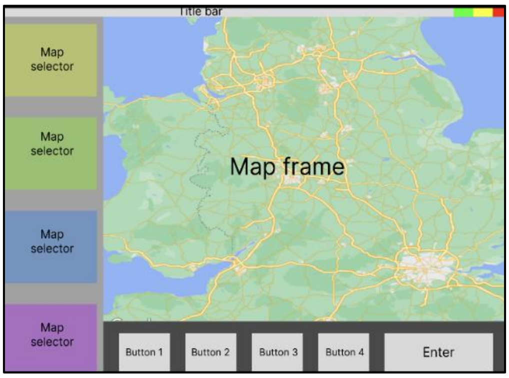


Figure 28 Gui wire frame

The user Interaction

Convenience and ease of use way an important philosophy that I tried to implement throughout the development phase of the Gui. Initially my plan was to implement a utility button row at the bottom of the window to handle selecting the start button, however, through user feedback from asking my friends how they would prefer to use the application, their unanimous consensus was that they would rather just the map display the shortest path after selecting their desired destination on the map. This from there I decided to remove the menu bar.

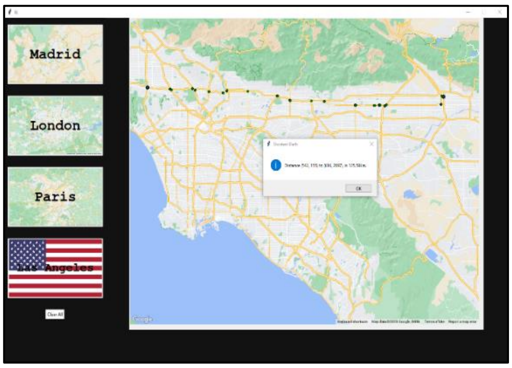


Figure 31 Map of Los Angeles with the shortest

This philosophy also extended to the design of user controls, where complete interface manipulation is achieved through the computer's mouse. The selection of buttons and nodes involves hovering the mouse cursor over the respective buttons/images and clicking to access a page. Additionally, the mouse cursor is utilized for choosing both the starting and destination points of the desired route.

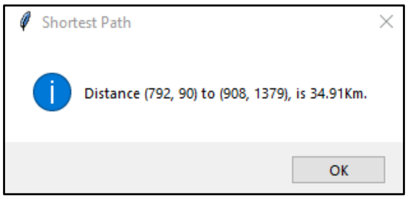


Figure 30 Shortest path message box.

After selecting the desired start and destination points on the map, the application displays the shortest route along with a pop-up message box showing the coordinates of the selected points and the shortest path distance in kilometres (refer to images).

Figure 29 Clear all Button.

If the application is unable to calculate a path between the points, an error message box will pop up to alert the user. After closing the message box, they can still select other points by clicking the “clear all” button or navigating to another page entirely, free from errors.

Upon closing the message box, the shortest path stays displayed. The user can either click on the “clear all” button (see figure) to remove the path and display the selectable points again or choose a different map page.

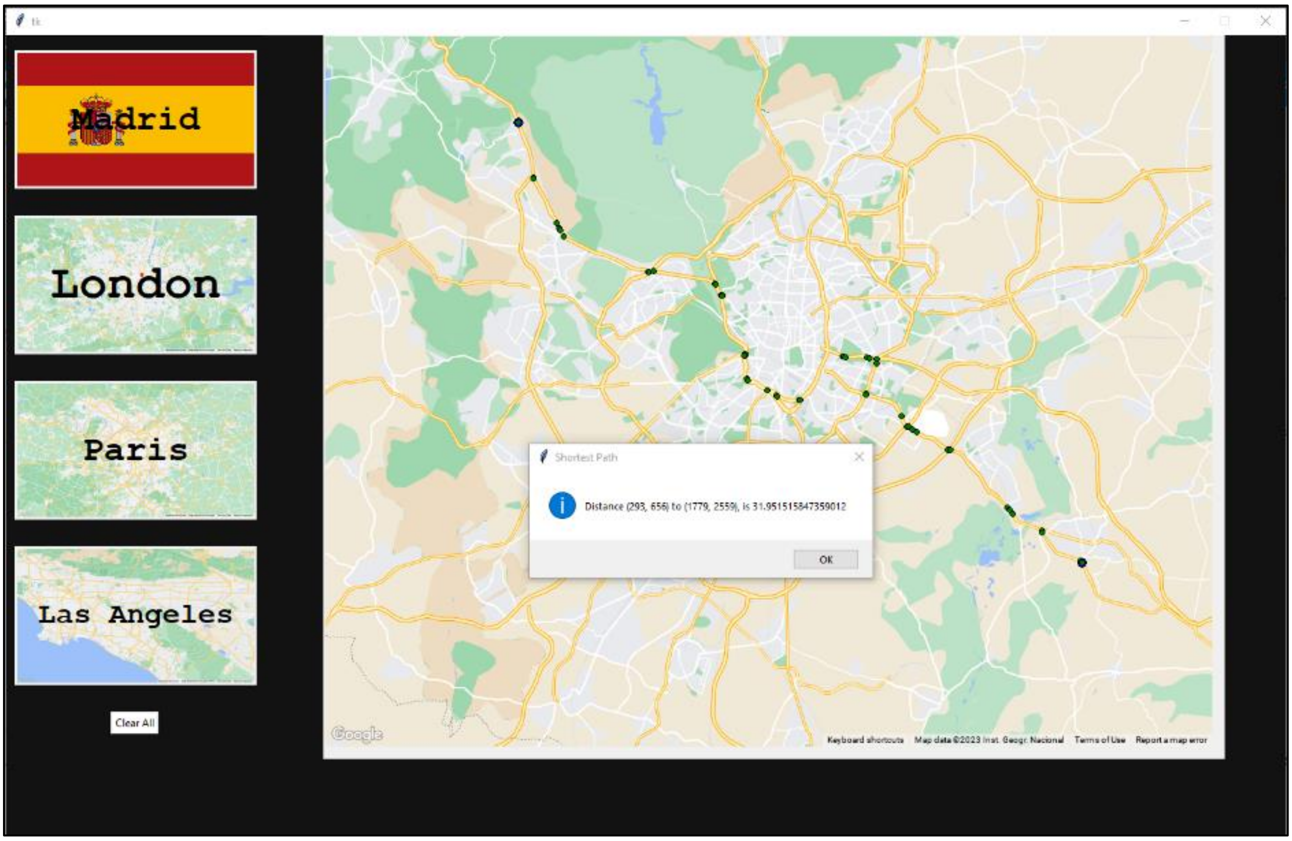


Figure 33 Map of Madrid with the shortest path displayed.

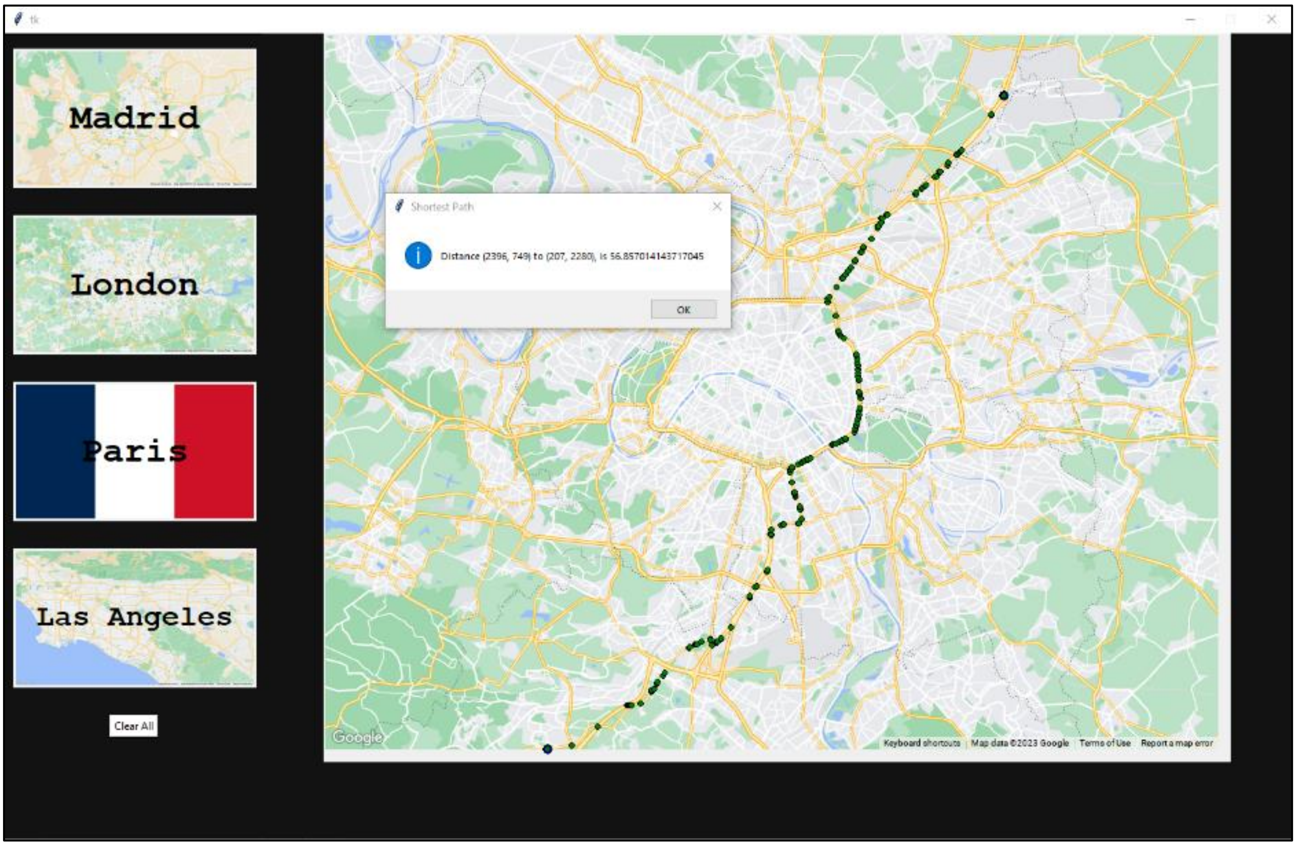


Figure 32 Map of Paris with the shortest path displayed.

Multiprocessing

Multiprocessing is the ability of a computer to execute multiple tasks of processes simultaneously, fully taking advantage of the multicore processors in modern computers and optimizing resource utilization. Multiprocessing enhances the efficiency and runtime of specific programs, especially those involving tasks divisible into smaller, independent units of work. In Python, multiprocessing primarily occurs at the CPU level during runtime execution rather than at the compilation stage. Because Python's Global Lock limits "true" multithreading, multiprocessing achieves its goals by utilizing processes. These processes can be thought of as distinct Python interpreters operating in their own isolated memory spaces and CPU cores. [52]

Multithreading is the ability of a computer to execute multiple threads within a process simultaneously. A thread is a lightweight unit of an execution within a process. Multithreading allows for multiple tasks to perform simultaneously. Threads within the same process share memory space, making communication and data sharing between threads relatively straightforward. Nevertheless, due to Python's Global Interpreter Lock, which permits only one thread to execute at a time, it might prove inefficient for CPU-bound tasks demanding substantial computational resources. [53]

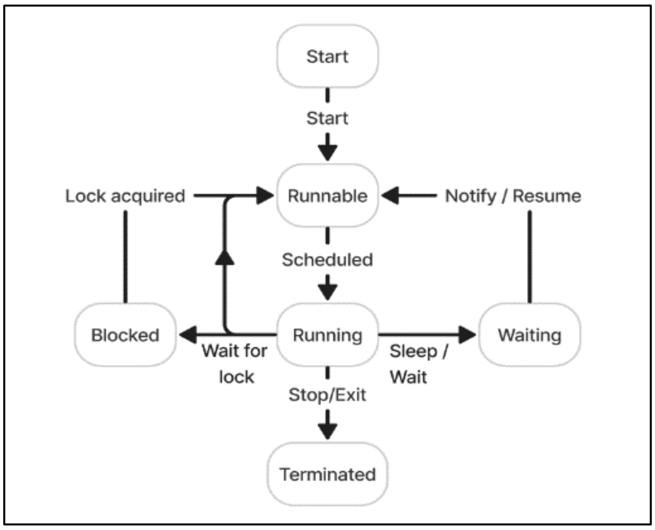


Figure 34 Flowchart of a thread [53]

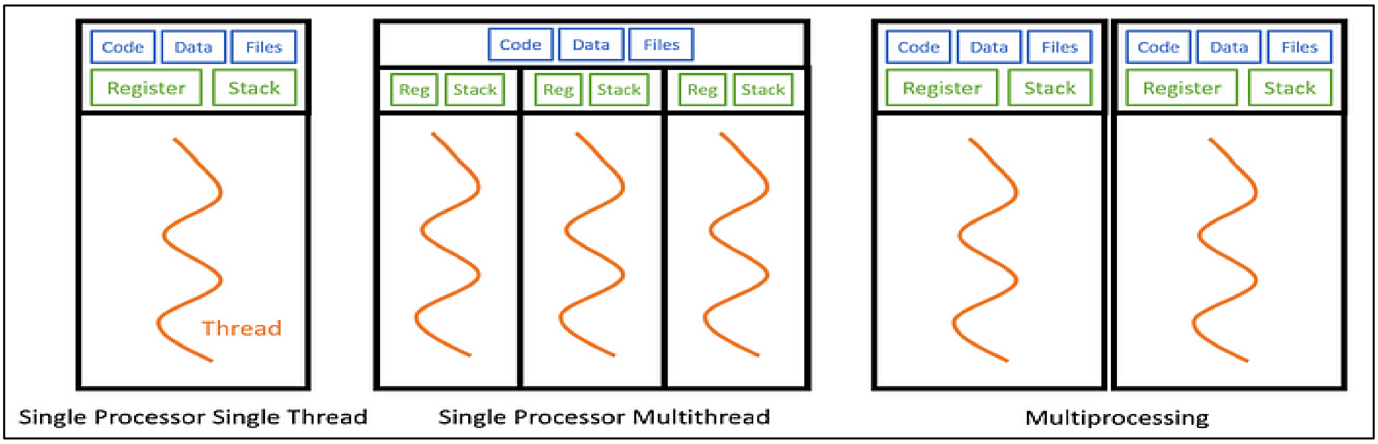


Figure 35 Multithreading vs multiprocessing [54]

Multithreading Implementation

The Windows operating system lacks the capacity to spawn a new child process that is an exact duplicate of the parent process. Consequently, the child process needs to rerun the code within the "**main**" block. The "if **name** == "**main**":" block acts as the program's entry point, facilitating the execution of multiprocessing when the script is run directly. This setup enables the operating system to differentiate the starting point of the Python script and permits new processes, working independently, to execute the entire script directly. This design prevents potential problems like infinite recursion.

The Main function and Multiprocessing implementation

Firstly, the necessary modules are imported to facilitate key functionalities. The modules **Tkinter, Graph** and **gui** from the **Algorithms.Gui** package are integrated, responsible for graph management and visualization and the **tkinter** module is employed for GUI development.

Main Function
The **main()** function serves as the primary program entry point. It initializes a list of countries with the variables **"fr", "es", "gb", and "us",** to enable subsequent processing. This list plays a crucial role in the concurrent generation of graph data for multiple maps.

Multiprocessing
Within the **Multiprocessing with Manager** module, concurrent processes are facilitated using the **Manager()** function. A shared list named **nestedGraphs** is initialised to hold the graphs as they are created.

Iterative Process Launching
The program iteratively processes the countries listed, initiating dedicated processes. Each process is directed to the **UpdateGraph** function, with the current **country** and the **nestedGraphs** list as arguments. This parallel approach significantly expedites the generation of graph data as it takes over 30 seconds to return a single graph. Effective process synchronization is achieved by employing the **process.join()** method. This ensures that the main process awaits the successful completion of all spawned processes before proceeding.

Compilation of Graph Data:
The process of compiling graph data entails appending the **nestedGraphs** list with data from individual processes. This aggregated data is then sorted into distinct arrays corresponding to **countries**, **graphs**, and **indices**.

Graph Data Generation
The **UpdateGraph()** function, targeted by the specified **country**, dynamically generates the graph. Finally, the resulting data is added into the shared **nestedGraphs** list.

Graphical User Interface (GUI) Initialization:
Using **tkinter**, a graphical user interface (GUI) window is initialised. An instance of the **gui.GUI** class is created, facilitating the creation of the graph. **win.mainloop()** activates the main event loop of the GUI.

N.B. Unlike Unix, the Windows operating system lacks the capacity to spawn a new child process that is an exact duplicate of the parent process. Consequently, the child process needs to rerun the code within the "main" block. The **"if name == "main":** block acts as the program's entry point, facilitating the execution of multiprocessing when the script is run directly. This setup enables the operating system to differentiate the starting point of the Python script and permits new processes, working independently, to execute the entire script directly. This design prevents potential problems that may arise like infinite recursion.

File structure

This project comprised of over 4,000 lines of code. Properly indexing files into their appropriate directories was crucial to avoid clutter and reduce the likelihood of spaghetti code. This approach also enabled me to assert functions without encountering import and packaging errors. As per the handbook requirements, which stipulated that all work should managed on the GitLab server, an organized file structure simplified pulling, committing, and pushing files, minimizing the risk of merge conflicts. In the Figure 36, colored borders delineate different directories that categorized the project. Each color denoted the following:

- (Black) Files located outside the program.
- (Purple) Graph traversal algorithms.
- (Brown) Main Function
- (Grey) Computer vision/image processing files.
- (Light blue) GUI handler files.
- (Yellow) GUI buttons.
- (Dark blue) Scratch files developed but not utilized by the program. *Some files were authored by others and hence excluded.*
- (Green) Text files containing place names for the map.
- (Red) Maps downloaded from Snazzy Maps.
- (Orange) Test images from the program used to monitor its outputs, *although not utilized by the program itself.*

Figure 36 The projects filing structure.[96] The coloured borders represent the different project directories.

Technical Achievements

I developed the entire project entirely in Python using the OpenCV, and Tkinter libraries. OpenCV (**cv2**) is an open-source computer vision and machine learning software library. It provides a range of tools to enable the development of image processing and computer vision programs. Tkinter (**tk**) is a GUI toolkit that provided the development of GUI applications for python.

Keeping in line with the project handbooks requirements [6], below I have explicitly stated what technical achievements implemented by me and what I copied, modified, or implemented.

Key achievements made with OpenCV and skimage:

Function/ Class	**Description**
threshold()	Colour masking[58], thresholding, binary image generation [59] and skeletonization suing skimage [30].
returnIntersections()	Modified a morphology operation from a stack overflow answer [55] to generate skeletal intersection points [61].
image_reader()	Creation of 2 image objects.

Key achievements made with Tkinter:

Function/ Class	**Description**
MapWIndow()	Displayed map widgets. Handling of all maps related functions and events, such as displaying buttons on the map. Handling user requests and displaying shortest path and distance. Handled possible user error. [61]
GUI()	Displaying the main GUI. Handles button press events. Created MapWindow() objects . [61]

Key achievements made/ implemented by me:

Function/ Class	**Description**
DijkstraAlgorithim()	Implemented Dijkstra's algorithm [56] which returned the shortest path between 2 points using an adjacency list. I adapted the to use a Priority Queue to efficiently extract the known shortest distances.
generate_graph()	Using a skeleton's intersections, found all its neighboring intersections by traversing the white pixels using an iterative implementation of the Depth-First Search algorithm [65].
recursiveNeighbourhoodGraph()	Using a skeleton's intersections, found all its neighboring intersections by traversing the white pixels using a recursive implementation of the Depth-First Search algorithm [64].
Graph.getGraph()	Converted a binary images skeleton' intersections into an adjacency list and returned the adjacency list, its list of intersections and country of origin into an array. [63].
Image.get_image()	Found the absolute path of an image's file name.
main()	Modified an answer from stack overflow to implement multiprocessing [57] to execute the generation of multiple adjacency list (generate_graph()) concurrently [66]

Programming Paradigms

Due to the versatility of the Python language, I was able to leverage multiple programming paradigms, primarily functional and object-oriented programming (OOP). Initially, during the development of the graph traversal/pathfinding algorithms and computer vision functions, I chose to employ functional programming. The static nature of these programs rendered their conversion unnecessary and cumbersome, as I only needed to invoke each of them once throughout the project. I also contemplated incorporating them as methods in other class files, but this approach would have complicated the testing of each algorithm. This is because I would have had to retest them after embedding them in larger class files. Moreover, such a practice would violate the principle of clean code, which states that objects and classes "should only have a single responsibility."

Conversely, object-oriented programming inherently aligned with Tkinter's design philosophy and offered substantial advantages for the development of the GUI in comparison to functional programming. Adopting a functional programming approach would have led to a violation of the "Don't repeat yourself" principle. OOP's encapsulation and abstraction helped maintain the program's organization and reusability. Furthermore, Tkinter's class-based structure seamlessly matched with my knowledge of OOP, allowing me to create multiple widgets with shared attributes and methods. This significantly reduced code's redundancy, making updates simpler and more efficient. This approach fit with Tkinter's event-driven paradigm, empowering widgets with event handlers and methods for improved autonomy and responsiveness. In essence, I found that OOP's compatibility with Tkinter's design principles, along with its support for encapsulation, inheritance, and event-driven design, made it my preferred approach for developing the GUI.

Planning

Challenge Week – 15th of October

Initially my plan was to create a Portfolio optimization tool. I had done the required background reading and had and gave an informal presentation to my supervisor. Based off his feedback I realized that continuing working on that project would not be I wise idea as (despite my interest) I didn't possess the perquisite knowledge on the subject and trying to catch would eat up precious time that would be better spent working on a project that I already had a lot of background knowledge of. Additionally, he advised that student who he had previously supervised and had a firm understanding of the subject, struggled to complete the project.

Consequently, I wasted my challenge week. As a result of this advice, I realized that switch to a project I had a good understanding of already. This is when I decided to look at the other project that I had shortlisted. I chose the pathfinding visualizer proposal by Dr David Richerby as I had always wanted to create an application which would find the shortest path between two points. The next week, I then presented this project topic to my supervisor and began immediately researching viable solutions.

During this time, I achieved the following:

- Began background reading on my new topic [67].
- Researched existing solutions. [68]
- Made a rough plan of the structure of my program. [69]
- Presented my new plan and risk assessment to my supervisor.[60][70]
- Creating user stories and tasks for the backlog in Jira [71]
- Made planned out the MVP .

Work towards the MVP between 15th of October and Interim oral review

During this time, I allocated Wednesday solely for on the project as it was the only day, I had no classes. I also scheduled my weekly capstone meeting on this day also. Around this time a realised that my project was novel enough due to the sheer number of existing solutions and talking to with classmates who were also making their similar solutions. The format of selecting nodes on a 2D grid [73] and having the shortest path and visually animating the program searching for the destination was an already thoroughly tried a tested method. The mean that to truly separate myself the other existing projects and convey the knowledge gained over my time at university, I need to change how I represented the graph. My supervisor also agreed with my sentiment and implored me to figure out a method finding out the shortest path between who regions on via the roads on map. This is where I began to research computer vison [74].

During this time, I developed a working MVP which achieved the following [77]:

- Successfully created my own implementation of Dijkstra's algorithm using an adjacency list [75].
- Created a method of masking out the roads on map.[58]
- Thresholding and generating a binary image from the masked-out roads.
- Researched into potential solutions of finding the intersections from the roads in from binary image. [76]
- Presented and demonstrated the MVP to my supervisor and second assessor as required in the Interim oral interview[78].

During the feedback section of the Interim oral interview[79], my second assessor advised me to research using skeletonization as a solution of extracting road intersections. This advice proved to be crucial in the development of the project as I had been struggling to find a viable solution to this problem.

Work done between Christmas Period – deadline day and absence from university.

I decided to use the time to complete up on all other outstanding coursework and take advantage of my free time to work on this project. However, during period due some personal issues left absent from

University between the beginning of January to the middle of march. Due to me not being absent at university I was unable to make any noteworthy progress on the project.

Despite my extended period of absence, I was still able to complete the following:

- Created a function that skeletonised objects in a binary image. [88]
- Utilized a set of morphological operations to return a list of intersections. [89]
- Created a poster featuring the work already completed. [86]
- Wrote my abstract with a proposal of my completed project. [87]
-

Work done between my return to university and the Reassessment Deadline

An unfortunate consequence of my absence from university meant that when I returned me with only a month to complete the project and submit the final report and the start of my final year examination period. My supervisor advised me to focus on preparing for my exams and to wait till I completed them before resuming work. Once I completed my examinations [90], I immediately focused completion of project where I achieved:

- Planned and created GUI [91]
- Completing development of the project (making minor changes by fixing bugs, changing naming filing schemes) [92]
- Demonstrated my completed project to my supervisor [93].
- Created a 6000-word draft of the final report [94].
- Worked on finishing off the final report [95].

Testing

Due to my modular design philosophy and testing was done at every stage of the development of my project. I only marked a task as done in Jira only if it the function works as intended. This was achieved by using simple scaled down example inputs to test functions. I also let friends who had no prior knowledge of the software to see if there were any unexpected errors.

Risk Management

T	Key	P	Summary	Description
	C301289-64	=	DFS function dosnt work as intended	at this point in the development of my program, I have reached a point where I cannot continue working on the project without figuring out how to fix the DFS function. the function seems to just return "null" every time I try to execute it. I've tried going through it with the debugger and it just seems to skip right through the node validators. I need to find a solution as there doesn't seem to be any time left as the deadline is less than 3 weeks away and i still need to work on the GUI
	C301289-34	=	Workload pill-up	ever since I've been reinstated my main focus has been to keep catch up on all my work. how ever, the work load has pilled up. the more I look into the things I've missed, the more things I have to catch up on. in the meantime I'm trying the 10am - 10 Pm work schedule and working at stem on at the CSEE labs to make sure I'm not procrastinating at home. i cannot Let my self get distracted or the work load will pill up.
	C301289-48	=	broken laptop	i broke my laptop and I've got to get a new one

Figure 37 Screenshot is risks logged in Jira.

Figure 37 above ([80]) represents major risks regarding the development of my code and the details surround them.

The biggest risk related to my project was the immense pressure [81] I was in was after I returned to university. At that point I had less than 3 weeks to complete the project and deliver the final report. That was not including the other outstanding coursework and studying for my exams. I decided to adopt a 12-hour workday starting at 10am and ending at 10pm to attempt catch up on all work missed. Thankfully, I was granted the ability to finish the course work in the summer and I was just able to focus on revision.

Around mid-December 2022, I accidentally broke my computer [82]. This meant that I was unable to do work my project at home while I was waiting the arrival of a new laptop. However, the possibility of this risk was accounted for in my risk assessment delivered in October [60. I regularly backed up all work, so I was able to work continue working on the project using the CSEE labs on campus. Accounting for this risk only slightly slowed down my progress as I was not flexible with when I could work on campus.

Struggling to successfully implement a DFS solution to create an adjacent list was the only risk that I could have led to an incomplete delivery of the software [80]. Despite it being planned for in my risk assessment [60], The only viable solution was to debug or try and find another method. However, this was to towards the end of the development of the project and the deadline was fast approaching. Luckily, I was able to figure out the problem and complete the project with ample time to spare for writing the final report.

Project planning with Jira

Jira is a widely used web-based project management and issue tracking tool developed by Atlassian. It helps teams plan, track, manage, and report on work across various projects and tasks using the **agile software development methodology**. With the use of **kanban boards,** Jira provides features for creating and organizing tasks, assigning responsibilities, setting priorities, tracking progress, and collaborating within teams.

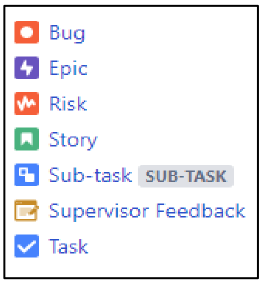


Figure 38 Image of types of Issues in Jira

Issues and The Kanban Board

Jira was vital throughout the development of my project. Using a Kanban board, I was able to visually plan, organize and keep track of goals by setting several types of issues depending on the task at hand.

Types of issues I used in Jira include:

- **Task:** A piece of work that requires completion
- **Bug**: Used to report and track software defects or unexpected behavior.
- **Story:** A user-centered requirement that represents a small piece of work that delivers value to users.
- **Epic:** A Major Task that can span multiple tasks or stories.
- **Sub-task:** A smaller piece of work that contributes to completing a larger task or story.
- **Risk:** Used to identify, assess, and manage potential risks that could impact the project's success or timeline.
- **Supervisor Feedback:** My projects supervisor input/feedback.

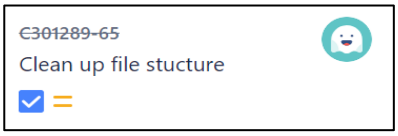


Figure 39 Done Jira task.

We organize the Kanban board using the following swim lanes:

- **Backlog:** For pending issues and ideas.
- **Selected for Development:** Issues chosen for immediate work.
- **In Progress:** Issues currently requiring.
- **Done:** For completed and finalized Issues .

	Issues	%
Task	32	50%
Story	9	14%
Sub-task	7	10%
Bug	6	9%
Epic	5	7%
Risk	3	4%
Supervisor Feedback	2	3%

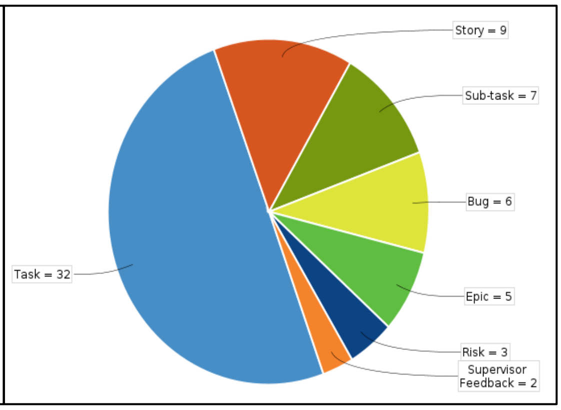


Figure 40 Pie chart and key of all issues submitted to Jira.

I used Jira extensively throughout the development of my project. I helped keep track of all my commits as I only marked an issue as complete it successfully passed testing. I used task names and ID numbers from Jira to label my commits. Here are a few examples of the issues I created in Jira:

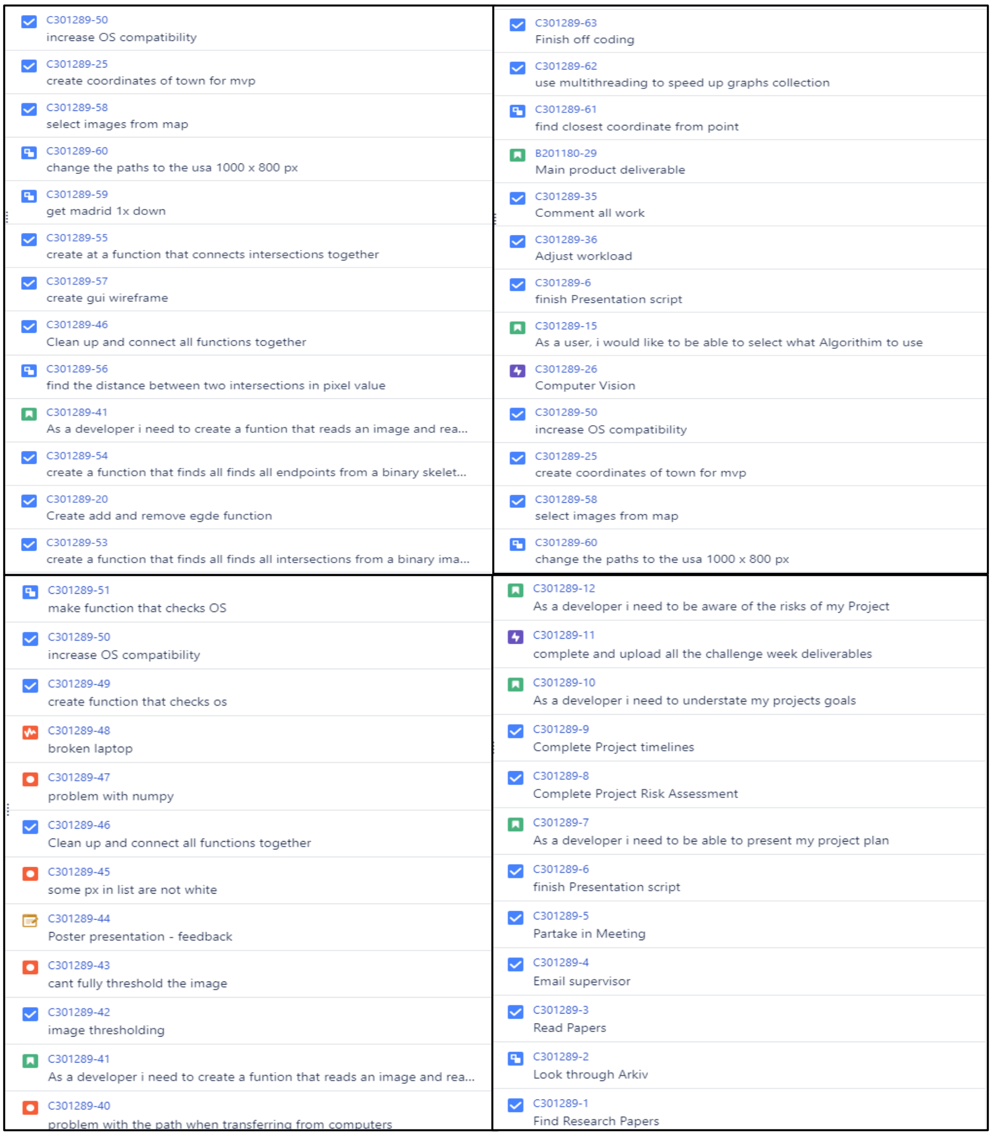


Figure 41 Screenshots of Jira Issues

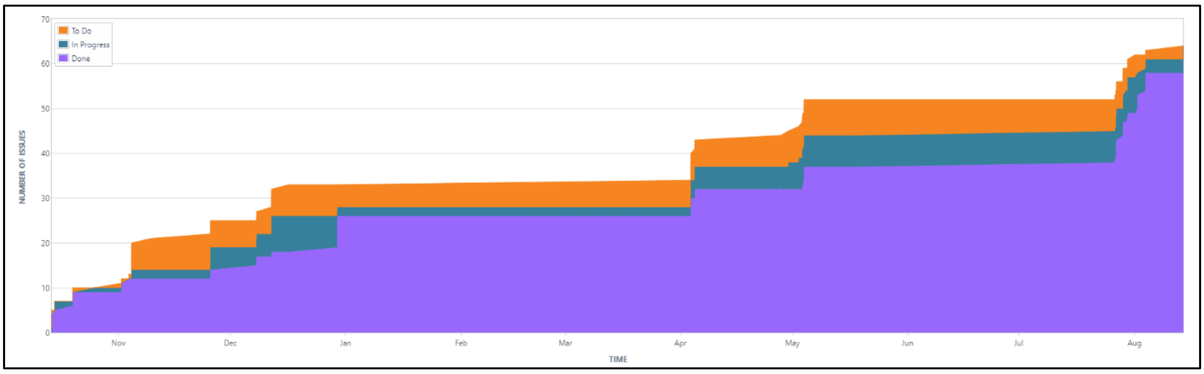


Figure 42 Project' s Cumulative Flow Diagram

Project source code management with Gitlab

GitLab is a web-based source code management platform for software development, allowing teams to collaboratively manage and track changes to source code, collaborate on projects. Git lab allows teams to push, pull and commits software files hosted remotely.

How I accessed GitLab

I chose to use PyCharm an Ide primarily used to develop Python applications. Here I cloned my repository (allocated for me by the university) using Pythons Git Remote feature. This seamlessly allowed me to interact with the repository without having to interact with GitLab natively.

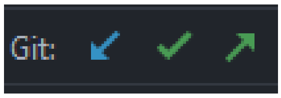
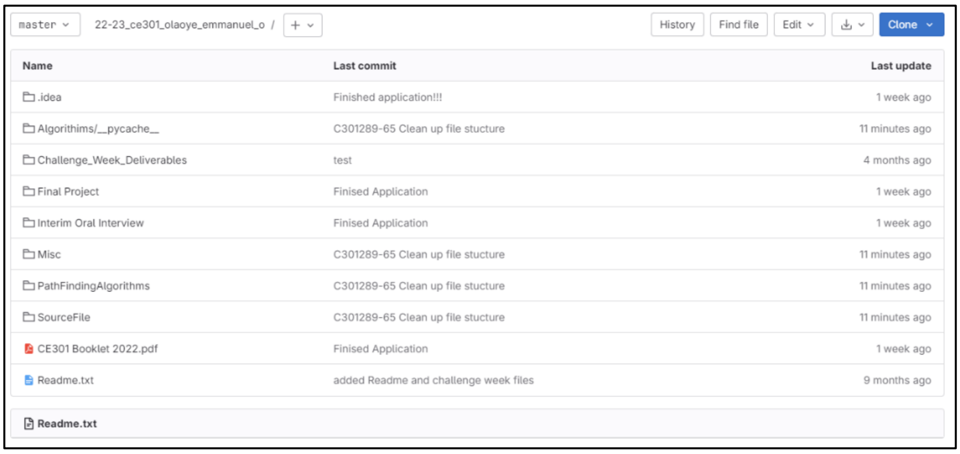


Figure 43 GitLab VSC commands (from left to right) pull commit and push, respectively.

I used the following version control commands (in figure shown):

• Commit: Creating a record of changes made to files in a local Git repository. (Commit message was Jira title)
• Pull: Fetching and merging changes from a remote repository to a local one.
• Push: Sending local commits to a remote repository to update its content.

The following is my projects GitLab file structure (might not be final):



Name	Last commit	Last update
.idea	Finished application!!!	1 week ago
Algorithms/__pycache__	C301289-65 Clean up file stucture	11 minutes ago
Challenge_Week_Deliverables	test	4 months ago
Final Project	Finised Application	1 week ago
Interim Oral Interview	Finised Application	1 week ago
Misc	C301289-65 Clean up file stucture	11 minutes ago
PathFindingAlgorithms	C301289-65 Clean up file stucture	11 minutes ago
SourceFile	C301289-65 Clean up file stucture	11 minutes ago
CE301 Booklet 2022.pdf	Finised Application	1 week ago
Readme.txt	added Readme and challenge week files	9 months ago

Figure 46 GitLab file structure

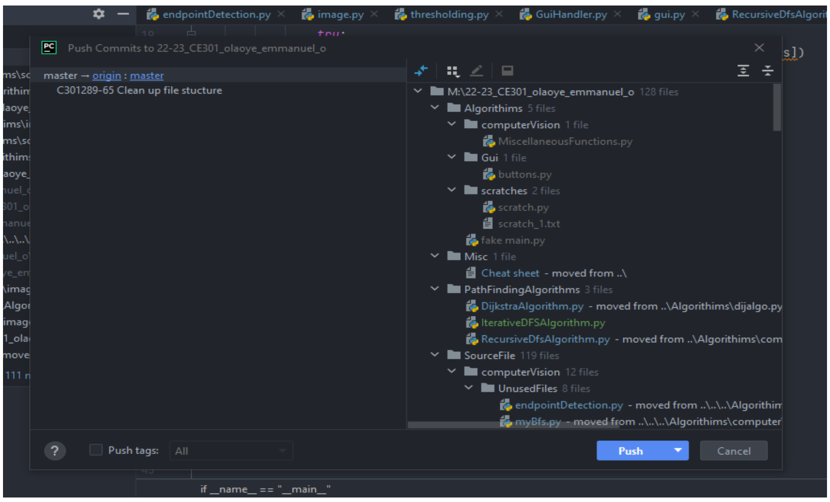


Figure 45a PyCharm Push to branch menu. [96]

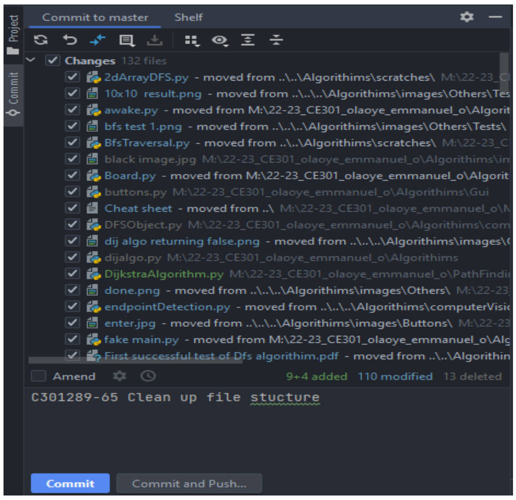


Figure 45b PyCharm Commit menu. (Note the commit message is a Jira title) [96]

Conclusion

The program effectively fulfils the initially set goals. It adeptly analyses images containing motorways and accurately presents the shortest path upon request. Furthermore, the program's capability to navigate an image skeleton to ascertain distances between nodes is a unique feature, not commonly observed. Users can seamlessly select any points, a functionality akin to that of Google Maps [4]. Throughout the project duration, I delved into the fascinating realms of computer vision and graph theory, gaining profound insights into their potential. Despite encountering challenges, exploring this topic proved to be both enjoyable and immensely rewarding.

Below is a table of requirements set out to reflect on what features and goals that were and were not achieved based on the goals proposed in October:

To create a pathfinding algorithm visualiser.	✅
Developed with python, the app must be interactive preferably a webpage.	✅
User will be able to select the start and end node.	✅
Be able to read any road and intersections	✅
Implement multiple maps	✅
Create a simple	✅
Calculate the distances between intersections	✅
Implement Dijkstra and Depth first Search.	✅
Animate the nodes as they were being visited.	❎
Weights must also be implemented and be considered by all algorithms.	❎
User must also be able to place wall nodes on the grid by mouse clicking also.	❎
Maze generated on the same grid and the app must be able to find a path to the end node.	❎
Implement A*, Bidirectional, Breath-first search, Swarm.	❎

References

[1] C. Mihailescu, “Pathfinding Visualiser,” *Pathfinding visualizer*, 2018. [Online]. Available: https://clementmihailescu.github.io/Pathfinding-Visualizer/. [Accessed: 15-Aug-2023]

[2] D. DeHart, *Pathfinding Visualizer*. [Online]. Available: https://pathfindout.com [Accessed: 15-Aug-2023]

[3] K Zhang, *Maps Pathfinding Visualizer*. [Online]. Available: https://pathfinding.kelvinzhang.ca/. [Accessed: 15-Aug-2023]

[4] *Google maps*. [Online]. Available: https://www.google.com/maps/. Google [Accessed: 15-Aug-2023]

[5] [1]L. Huang, “Skeleton- tracing tool” *GitHub.com*. [Online]. Available: https://github.com/LingDong-/skeleton-tracing. [Accessed: 15-Aug-2023]

[6] moodle.essex.ac.uk, “Project Handbook - Chapter 12 - Final Report,” 2022. [Online]. Available: https://moodle.essex.ac.uk/mod/book/view.php?id=622901&chapterid=11747. [Accessed 08 04 2022].

[7] Y. Zhang, J. Zhang, T. Li, and K. Sun, "Road extraction and intersection detection based on tensor voting," 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Beijing, China, 2016, pp. 1587-1590, Doi: 10.1109/IGARSS.2016.7729405.

[8] *GeeksforGeeks.com*, “Greedy best first search algorithm,” *GeeksforGeeks*, 02-feb-2023. [Online]. Available: https://www.geeksforgeeks.org/digital-image-processing-basics/https://www.geeksforgeeks.org/greedy-best-first-search-algorithm [Accessed: 15-Aug-2023]

[9] A. Krivec, “Amazing landscape of Seiser Alm plateau in South Tyrol, the northern province of Italy. Old-school living on the pastures in idyllic cottages.” Seiser Alm plateau, South Tyrol, Italy, 2023 [Online]. Available: https://unsplash.com/photos/yIkD6xtaUgk. [Accessed: 09-Mar-2023]

[10 /16] A Clark, “Computer Vision”, Moodle.Essex.ac.uk, Jan 12, 2023 [Online]. Available: https://moodle.essex.ac.uk/pluginfile.php/1454487/mod_resource/content/5/notes.pdf [Accessed: 09-Mar-2023]

[11] A. Clark, “CE316 - Computer Vision - Lecture slides (Unit 4: Low-level Vision),” 18 01 2023. [Online]. Available: https://moodle.essex.ac.uk/mod/resource/view.php?id=803332 [Accessed: 04-Aug-2023]

[12] A. Clark, “CE316 - Computer Vision - Lecture slides (Unit 6: Intermediate-level Vision),” 18 02 2023 [Online]. Available: https://moodle.essex.ac.uk/mod/resource/view.php?id=815813 [Accessed: 04-Aug-2023]

[13] A. Clark, “CE316 - Computer Vision - Lecture slides (Unit 9: High-level Vision),” 22 03 2021. [Online]. https://moodle.essex.ac.uk/mod/resource/view.php?id=815813 [Accessed: 04-Aug-2023]

[14] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, “Binary images,” *Binary Images*, 2003. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/binimage.htm#:~:text=Binary%20images%20are%20images%20whose,1%20or%20255%20for%20white. [Accessed: 15-Aug-2023]

[15] The Eiffel Tower. Paris, France: Britannica, The Editors of Encyclopaedia, Encyclopaedia Britannica 2023 [Online]. Available: https://www.britannica.com/topic/Eiffel-Tower-Paris-France. [Accessed: 12-Aug-2023]

[17] P. K. Saha, G. Borgeforsand G. S. di Baja, “Skeletonization: Theory, methods and applications”, Aug. 10, 2017. Available: https://books.google.com/books?hl=en&lr=&id=x7oIDgAAQBAJ&oi=fnd&pg=PP1&dq=skeletonization+image+processing&ots=OBCEV-lSj2&sig=xzRp0VPgWufszYPDQQ9RP-YqSTU [Accessed: 09-Mar-2023]

[18] H. Chatbri, K. Kameyamaand P. Kwan, “A comparative study using contours and skeletons as shape representations for binary image matching”, Pattern Recognition Lett ers, Aug. 10, 2016. Available: https://www.sciencedirect.com/science/article/pii/S0167865515001245

[19] N. Amenta and S. Choi and R. Kolluri. The Power Crust, Unions of Balls and The Medial Axis Transform. Int. J.

[20] D. Jin and P. K. Saha, “A new fuzzy skeletonization algorithm and its applications to medical imaging”, Image Analysis and Processing--ICIAP 2013: 17th International Conference, Naples, Italy, September 9-13, 2013. Proceedings, Part I 17, Sep. 09, 2013.

[21] M. Ilg and R. Ogniewicz, "The application of Voronoi skeletons to perceptual grouping in line images," Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol. III. Conference C: Image, Speech and Signal Analysis, The Hague, Netherlands, 1992, pp. 382-385, Doi: 10.1109/ICPR.1992.202004

[22] W. Abu-Ain, S. N. H. S. Abdullah, B. Bataineh, T. Abu-Ainand K. Omar, “Skeletonization Algorithm for Binary Images”, Procedia Technology, Aug. 10, 2013. Available: https://www.sciencedirect.com/science/article/pii/S2212017313004027

[23] P. K. Saha, D. Jin, Y. Liu, G. E. Christensen, and C. Chen, "Fuzzy Object Skeletonization: Theory, Algorithms, and Applications," in IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 8, pp. 2298-2314, 1 Aug. 2018, Doi: 10.1109/TVCG.2017.2738023.

[24] Jiangui Wu, H. Duan and Qi Zhong, "3D image skeleton algorithms," 2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification, Xiamen, China, 2011, pp. 97-100, doi: 10.1109/ASID.2011.5967425.

[25] R. Szeliski, Computer vision: Algorithms and applications. Cham: Springer, 2023 [Online] .Available: https://essex.primo.exlibrisgroup.com/ [Accessed: 12-Aug-2023]

[26] Shen, Wei & Zhao, Kai & Jiang, Yuan & Wang, Yan & Bai, Xiang & Yuille, Alan. (2016). “Deep Skeleton: Learning Multi-Task Scale-Associated Deep Side Outputs for Object Skeleton Extraction in Natural Images. IEEE Transactions on Image Processing.” PP. 10.1109/TIP.2017.2735182.

[27] Tsung-Ying Sun, Chih-Li Huo, Shang-Jeng Tsai and Chan-Cheng Liu, "Optimal UAV flight path planning using skeletonization and Particle Swarm Optimizer," 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress

on Computational Intelligence), Hong Kong, China, 2008, pp. 1183-1188, doi: 10.1109/CEC.2008.4630946.

[28] W. Abu-Ain, S. N. H. S. Abdullah, B. Bataineh, T. Abu-Ain, and K. Omar, 'Skeletonization Algorithm for Binary Images', Procedia Technology, vol. 11, pp. 704–709, 2013 [online] Available: https://www.frontiersin.org/articles/10.3389/fpls.2020.00773/full. [Accessed: 12-Aug-2023]

[29] [skimage documentation] https://scikit-image.org/docs/stable/auto_examples/edges/plot_skeleton.html

[30] K. Palágyi, "Skeletonization Techniques," Skeletonization. [Online]. Available: http://www.inf.u-szeged.hu/~palagyi/skel/skel.html#:~:text=The%20notion%20skeleton%20was%20introduced,least%20two%20closest%20boundary%20points. [Accessed: 12-Aug-2023]

[31] L. W. Beineke, M. C. Golumbic, and R. J. Wilson, "Preliminaries," in Topics in Algorithmic Graph Theory, L. W. Beineke, M. C. Golumbic, and R. J. Wilson, Eds. Cambridge: Cambridge University Press, 2021, pp. 1–16. [Online] Available: https://www-cambridge-org.uniessexlib.idm.oclc.org/core/services/aop-cambridge-core/content/view/8CD9C9A611EDE3CC5E588826770C888F/9781108492607int_1-16.pdf/preliminaries.pdf [Accessed: 04-Aug-2023]

[32] S. Catanese, P. D. Meo, E. Ferrara, and G. Fiumara, Analysing the Facebook Friendship Graph. 2011. [Online]. Available: https://arxiv.org/abs/1011.5168

[33] R. Fang, H. Liao, Z. Xu, and E. Herrera-Viedma, "Risk assessment in project management by a graph-theory-based group decision making method with comprehensive linguistic preference information," Economic Research-Ekonomska Istraživanja, vol. 36, no. 1, pp. 86–115, 2023, doi: 10.1080/1331677X.2022.2070772.

[34] D. Richerby, "CE204 Data Structures and Algorithms - Lecture slides (Unit 5: Graphs and Graph Algorithms)," 18 11 2021. [Online]. Available: https://moodlearchive.essex.ac.uk/2021/pluginfile.php/1730070/mod_resource/content/2/unit5-slides.pdf [Accessed: 04-Aug-2023]

[35] M. C. Golumbic, "Graph algorithms," in Topics in Algorithmic Graph Theory, L. W. Beineke, M. C. Golumbic, and R. J. E. Wilson, Eds. Cambridge University Press, 2021, pp. 17–32. doi: 10.1017/9781108592376.004.

[36] M. J. Mirchev, "Applications of spectral graph theory in the field of telecommunications". [Online] Available: https://epluse.ceec.bg/wp-content/uploads/2018/09/20160102-01.pdf [Accessed: 04-Aug-2023]

[37] G. E. Mathew, "Direction based heuristic for Pathfinding in Video Games," Procedia Computer Science, vol. 47, pp. 262–271, 2015. doi: 10.1016/j.procs.2015.03.206

[38] S. J. Russell, P. Norvig, and M.-W. Chang, Artificial Intelligence: A modern approach. Harlow, United Kingdom: Pearson, 2022 [Online]. Available: https://essex.primo.exlibrisgroup.com/discovery/fulldisplay?docid=alma991006902109707346&context=L&vid=44UOES_INST:UOES&lang=en&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=any,contains,Artificial%20intelligence%20a%20modern%20approach&pfilter=rtype,exact,books&offset=0

[39] M. Needham and A. E. Hodle, "4. Pathfinding And Graph Search Algorithms," in *Graph algorithms: Practical examples in Apache Spark and neo4j*, O'Reilly [Online]. Available: https://essex.primo.exlibrisgroup.com/discovery/fulldisplay?docid=alma991008469986307346&context=L&vid=44UOES_INST:UOES&lang=en&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=any,contains,pathfinding%20algorithms&pfilter=rtype,exact,books&offset=0

[40] *GeeksforGeeks.com*, "Greedy best first search algorithm," *GeeksforGeeks*, 04-Apr-2023. [Online]. Available: https://www.geeksforgeeks.org/greedy-best-first-search-algorithm [Accessed: 15-Aug-2023]

[41] A. Beltyukova, "Why UPS trucks (almost) never turn left," *CNN*, 23-Feb-2017 [Online]. Available: https://edition.cnn.com/2017/02/16/world/ups-trucks-no-left-turns/index.html. [Accessed: 05-Aug-2023]

[42] Rafiq A., Kadir T.A.A., Ihsan S.N. Pathfinding Algorithms in Game Development. IOP Conf. Ser. Mater. Sci. Eng. 2020;769:012021. doi: 10.1088/1757-899X/769/1/012021.

[43] S. J. A. Raza, N. A. Gupta, N. Chitaliya, and G. R. Sukthankar, Real-World Modelling of a Pathfinding Robot Using Robot Operating System (ROS). 2018. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1802/1802.10138.pdf

[45] G. E. Mathew and G. Malathy, "Direction based heuristic for Pathfinding in Video Games," 2015 2nd International Conference on Electronics and Communication Systems (ICECS), 2015.

[46] S. J. Russell, P. Norvig, and M.-W. Chang, Artificial Intelligence: A modern approach. Harlow, United Kingdom: Pearson, 2022 [Online]. Available: https://essex.primo.exlibrisgroup.com/discovery/fulldisplay?docid=alma991006902109707346&context=L&vid=44UOES_INST:UOES&lang=en&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=any,contains,Artificial%20intelligence%20a%20modern%20approach&pfilter=rtype,exact,books&offset=0

[47] P, Sundaravadivel, Lee I., Mohanty S., Kougianos E., Rachakonda L. RM-IoT: An IoT based rapid medical response plan for smart cities; Proceedings of the 2019 IEEE International Symposium on Smart Electronic Systems, ISES; Rourkela, India. 16–18 December 2019; Manhattan, NY, USA: IEEE; 2019. pp. 241–246

[48] M. Ranieri, "A new sense of direction with live view," Google, 01-Oct-2020. [Online]. Available: https://blog.google/products/maps/new-sense-direction-live-view/. [Accessed: 12-Aug-2023]

[49] Dijkstra E.W. A note on two problems in connexion with graphs. Numer. Math. 1959;1:269–271. doi: 10.1007/BF01386390.

[50] [1]A. F. Borchert, *Edsger Dijkstra at ETH Zurich in 1994*. ETH Zurich, Switzerland, 2011 [Online]. Available: https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

[51] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," in *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 263-, Feb. 1998, doi: 10.1109/12.663776.

[52] Z. A. Aziz, D. Naseradeen - Abdulqader, A. B. Sallow, and H. Khalid Omer, “Python Parallel Processing and Multiprocessing: A Rivew”, ACAD J NAWROZ UNIV, vol. 10, no. 3, pp. 345–354, Aug. 2021.

[53] T. Papastylianou, “Threads and Synchronisation,” CE303 Advanced Programming, University of Essex, Colchester [Online]. Available: https://moodle.essex.ac.uk/course/view.php?id=3664&section=5

[54] K. J. Wong, “Multithreading and multiprocessing in 10 minutes,” Medium, 11-May-2023. [Online]. Available: https://towardsdatascience.com/multithreading-and-multiprocessing-in-10-minutes-20d9b3c6a867. [Accessed: 12-Aug-2023]

[55] stackoverflow.com, “How to find the junction points or segments in a skeletonized image Python OpenCV?” 09 05 2022. [Online]. Available: https://stackoverflow.com/a/72165866 [Accessed 10 04 2021].

[56] youtube.com “Dijkstra's Algorithm in Python Explained.”, 26 04 2019 . [Online]. Available: https://www.youtube.com/watch?v=Ub4-nG09PFw&t=759s [Accessed 10 04 2021].

[57] stackoverflow.com, “Multiprocessing of shared list?” 09 05 2022. [Online]. Available: https://stackoverflow.com/a/23623891 [Accessed 10 04 2021].

[58] Jira, “Facility to create and modify a graph,” 04 04 2023. [Online]. Available: https://cseejira.essex.ac.uk/browse/A301079-10. [Accessed 10 01 2023].

[59] Jira, “create a function that finds all finds all intersections from a binary image” 27 07 2023. [Online]. https://cseejira.essex.ac.uk/browse/C301289-53 [Accessed 14 08 2023].

[60] Jira, “create a function that finds all finds all intersections from a binary image” 27 07 2023. [Online]. https://cseejira.essex.ac.uk/browse/C301289-53 [Accessed 14 08 2023].

[61] Jira, “As a developer I need to be aware of the risks of my Project”14 10 2022. [Online]. https://cseejira.essex.ac.uk/browse/C301289-12 [Accessed 14 08 2023].

[62] Jira, “Development of the Website or Application” 04 08 2023. [Online]. https://cseejira.essex.ac.uk/browse/C301289-53 [Accessed 14 08 2023].

[63] Jira, “Create the get Graph function” 14 08 2023. [Online]. https://cseejira.essex.ac.uk/browse/C301289-27 [Accessed 14 08 2023].

[64] Jira, “create at a function that connects intersections together” 28 07 2023. [Online]. https://cseejira.essex.ac.uk/browse/C301289-55 [Accessed 14 08 2023].

[65] Jira, “DFS function does not work as intended” 28 07 2023. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[66] Jira, “use multiprocessing to speed up graphs collection” 01 07 2023. [Online]. https://cseejira.essex.ac.uk/browse/C301289-62 [Accessed 14 08 2023]

[67] Jira, “Look through Arkiv” 01 10 2022. [Online] https://cseejira.essex.ac.uk/browse/C301289-2 [Accessed 14 08 2023]

[68] Jira, “Find Research Papers” 01 10 2022. [Online] https://cseejira.essex.ac.uk/browse/C301289-1 [Accessed 14 08 2023]

[69] Jira, “As a developer I need to understate my projects goals” 01 11 2022. [Online]. https://cseejira.essex.ac.uk/browse/C301289-10 [Accessed 14 08 2023]

[70] Jira, “Partake in Meeting” 01 11 2022. [Online]. https://cseejira.essex.ac.uk/browse/C301289-5 [Accessed 14 08 2023]

[71] Jira, “finish Presentation script” 01 11 2022. [Online]. https://cseejira.essex.ac.uk/browse/C301289-62 [Accessed 14 08 2023]

[72] Jira, “As a developer I need to be able to present my project plan” 01 11 2022. [Online]. https://cseejira.essex.ac.uk/browse/C301289-30 [Accessed 14 08 2023]

[73] Jira, “Create Grid Class” 04 11 22. [Online]. https://cseejira.essex.ac.uk/browse/C301289-18 [Accessed 14 08 2023]

[74] Jira, “learn about CV2“, 12 12 22. [Online]. https://cseejira.essex.ac.uk/browse/C301289-37 [Accessed 14 08 2023]

[75] Jira, “create dij algo “, 12 12 22. [Online]. https://cseejira.essex.ac.uk/browse/C301289-22 [Accessed 14 08 2023]

[76] Jira, “as a developer I need to create a function that reads an image and reads the intersections “, 12 04 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-41 [Accessed 14 08 2023]

[77] Jira, “Interim Oral Interview “, 12 12 22. [Online]. https://cseejira.essex.ac.uk/browse/C301289-28 [Accessed 14 08 2023]

[78] Jira, “create presentation”, 12 12 22. [Online]. https://cseejira.essex.ac.uk/browse/C301289-32 [Accessed 14 08 2023]

[79] Jira, “l Week 11 Interim Oral Feedback “, 12 12 22. [Online]. https://cseejira.essex.ac.uk/browse/C301289-33 [Accessed 14 08 2023]

[80] Jira, “Jira risks “02 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64?filter=-9&jql=issuetype%20%3D%20Risk [Accessed 14 08 2023]

[81] Jira, “Workload pill-up “03 04 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-34 [Accessed 14 08 2023]

[82] Jira, “broken laptop “02 08 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-48 [Accessed 14 08 2023]

[84] Jira, “DFS function doesn’t work as intended “31 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[85] Jira, “Development of the Website or Application “04 08 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-14 [Accessed 14 08 2023]

[86] Jira, “create poster “27 04 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-66 [Accessed 14 08 2023]

[87] Jira, “Write up abstract “27 04 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-67 [Accessed 14 08 2023]

[88] Jira, “skeletonise a binary image “31 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[89] Jira, “create a function that finds all finds all intersections from a binary image “27 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-53 [Accessed 14 08 2023]

[90] Jira, “Complete all outstanding coursework “31 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[91] Jira, “create gui wireframe “28 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[92] Jira, “finish off project “09 08 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[93] Jira, “demonstrate project to supervisor “05 08 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-73 [Accessed 14 08 2023]

[94] Jira, “complete at least 5000 words “05 08 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[95] Jira, “finish off final report “14 08 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-74 [Accessed 14 08 2023]

[96] Jira, “Clean up file structure “31 07 23. [Online]. mhttps://cseejira.essex.ac.uk/browse/C301289-65 [Accessed 14 08 2023]

[97] Jira, “Interim Oral Interview “31 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[98] Jira, “skeletonise a binary image “31 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]

[99] Jira, “skeletonise a binary image “31 07 23. [Online]. https://cseejira.essex.ac.uk/browse/C301289-64 [Accessed 14 08 2023]