Problem Solving

COMPUTATIONAL INTELLIGENCE

Authored by: Prerna Singh Shahi

Table of Contents

Problem Formation

A navigation program tailored to wheelchair users that assists them in planning trips, based on factors such as kerb ramps, sidewalk width, slope, and indoor lifts, doesn't currently exist. Google Maps is a popular mainstream navigation platform, and a key functionality is that it finds the shortest path, however, it doesn't consider the factors listed above. These factors are vital for wheelchair users. The aim of this project, and thus the problem to be solved, is to design a navigation map for wheelchair users that considers accessibility factors and finds the shortest path between two points.

The developed map will enable wheelchair users to navigate safely and efficiently, from their initial location to their desired location. The map will mainly be focused on the suburb called Cranbourne, located in Victoria, Australia, with a postcode of 3977. But it will also cover some facilities of relevance in surrounding suburbs such as Cranbourne East, or Cranbourne North. These suburbs also fall in the category of southeast suburbs in Melbourne. The map may not cover all the facilities in this area, but it will focus on the facilities most essential for wheelchair users' navigation needs. So, facilities related to healthcare, work, bathrooms, and some entertainment.

Wheelchair users cannot drive, so they will use the map to help them navigate in their wheelchair. On average, manual wheelchair users have a speed of 0.77 meters per second (Wheelchair Usage, n.d.), so this information will be used when finding the 'distance' between locations. The map will aim to help wheelchair users find the shortest route for them and indicate the landmarks they will pass through in their journey.

My Approach

The route planning methodology in this project is aligned with different computational intelligence concepts and problem-solving techniques.

The goal-based agent (Lee, 2024) is the program/algorithm which has the aim of searching for the shortest path, whilst also considering accessibility factors.

The environment is the map itself, which contains the nodes (locations).

The search strategy used will be the A* algorithm, which combines the benefits of the greedy search algorithm and the uniform-cost algorithm. Thus, when finding the shortest route, paths that are already 'expensive' will not be explored by the program.

For comparison purposes, an agent that uses Dijkstra's algorithm instead of the A* search algorithm, will also be developed at the end.

The evaluation function for the A* algorithm is f(n) = g(n) + h(n).

- g(n) is the cost to reach n so far.
- h(n) is a heuristic to calculate the estimated cost to the goal node from n.
- f(n) is the estimated total cost of the path through n to the goal.

Both g(n) and h(n) will continue to be evaluated in the A* algorithm through each node that is taken to reach the goal. (Lee, 2024)

Also in this project, n means the node/location at which the algorithm has reached so far.

This search algorithm is implementable, as observed in practicals during weeks 3 and 4. In the week 3 practical an A* search was used to find the route between suburbs (SIT215_Prac_W3 (AFTER CLASS), 2024). In the week 4 practical different heuristics were used within the A* search algorithm (SIT215_Prac_W4 (AFTER CLASS), 2024). In particular, the week 3 practical demonstrated that the A* search is relevant for graph-based problems, which further justified using this search to solve this problem in this project.

In class discussions examples such as the *Robotic Assembly Problem* were discussed, in terms of computational intelligence terms such as states, actions, goal test and path cost (Lee, 2024). These terms can also be used to define and guide the problem-solving in this project.

The **Goal Test** will be that a given state is a goal state, because if it is then that means that a route has been found. The **States** will be the node of the graph, so a location. The **Initial State** will be the starting node of the user. The **Actions** will be the edges between nodes, since they represent possible paths. The **Path Cost** will be the cumulative cost of travelling through a path using the A* algorithm. The **Transition Model** is when an adjacent node is chosen by the program the node will be arrived at.

Google's My Maps feature to find and annotate facilities and routes of interest (see Appendix A). This data was then exported and moved into an Excel Workbook where the necessary calculations could be performed. To find the associated costs between locations, it was necessary to first find the distance between each location in kilometres, this was researched through Google Maps, and placed in an Excel Sheet (see Table B1). Google Maps is a reliable source of data (Birney, 2024), also it was more practical and less resource-heavy to gather the data from here, rather than measuring paths in person.

Another Excel sheet would be used to calculate the weighting in terms of time for each location based on accessibility factors (see Table B2). This data was gathered through physical surveying, which is accurate and reliable because it's first-party data (Swendeman, 2024). Information that couldn't be found online was also observed.

Finally, some calculations were performed to find the time taken for a manual wheelchair user to travel through that route in minutes, this required using time equals distance divided by speed, plus the accessibility time in the previously stated sheet (see Table B3). So, the edges between nodes use wheelchair-accessible time in minutes, by considering the average time a wheelchair user would take to traverse this route, whilst also considering the accessibility factors such as slope, narrow path, and terrain.

Implementation Overview

This implementation utilizes the code in practical 3 (SIT215_Prac_W3 (AFTER CLASS), 2024), with some changes. First, the necessary modules are imported.
Then an abstract class called '*Problem'* is defined. This is the parent template of all problem-solving scenarios in SIT215 so far. In this class, actions and results are always implemented in the subclass (child class), while the other methods are optionally implemented.

The subclass of '*Problem'*, which is '*GraphProblem'*, does implement the required methods, as well as _init_ and path_cost. This subclass is tailored to solve Graph problems, which is relevant to the problem that is being solved in this project since a map is a type of graph. Another important method it has is the *distance(self, a, b)* method which uses the Euclidean distance formula to find the distance between two points *a* and *b*. The method *h(self, node)* defines the heuristic function, which calculates the heuristic value for a given node in terms of its distance to the goal state.

But before that, is the *Node* class which represents a node in the search tree. The node is derived from a parent, the nodes that can be reached from this node, and a child_node can be generated by applying an action to the current node's state and calculating the resulting path cost for the next code.

The *Graph* class represents a data structure of type Graph. It connects the nodes with edges.

Then a *local_map* is defined, it's a dictionary of locations that are connected to each other, and their costs. *local_map.locations* defines the coordinates of the locations in this map.

The colours, positions and labels of the node are then defined; This involves using the local_map details that were developed previously.
show_map is used to define the features of the graph, its size, its labels, and its legend. Then this graph is shown using local_graph_data which was defined as narrated one line above the above line.

final_path_colors is used to colour the solution to the problem in green, to improve visibility for the user once the algorithm has found a solution.

display_visual is used to control the elements that are used to display the final visual. The widgets used to select the start and end values are defined here.
It also includes the slider, which communicates how many iterations of states through nodes that the algorithm has gone through so far for a problem. The A* algorithm is also called here, and then the solution is given its final colors. There is a latency of 0.5 seconds so that the user can easily visually see the iteration as a solution is found.

memoize is used to implement memoization, which is a technique where the results of expensive function calls are cached and reused when the same inputs occur again. This enables efficiency to improve since redundant calculations are avoided. When finding the best route it may be necessary to traverse back to a node, so this method is useful in that case.

best_first_graph_search_for_vis is used to enable the algorithm to search the nodes with the lowest f scores in the search space (map) first.

PriorityQueue enables elements to be ordered based on their priority, with the highest priority (so lowest cost) nodes being explored first.

Then A* Search algorithm method is defined. In that method, first, the heuristic is defined, and then best_first_graph_search_for_vis is used to explore the lowest cost nodes using both g(n) and h(n).

Finally, the visual map can be displayed using the relevant data, algorithm and problem. First, the user needs to select the start and end location values. Then the visual displays the progress of the search until finally a path where the nodes are green is used to display the generated optimal solution using the A* search algorithm.

Results, Evaluation and Analysis

A* Search Algorithm with Euclidean Distance Heuristic

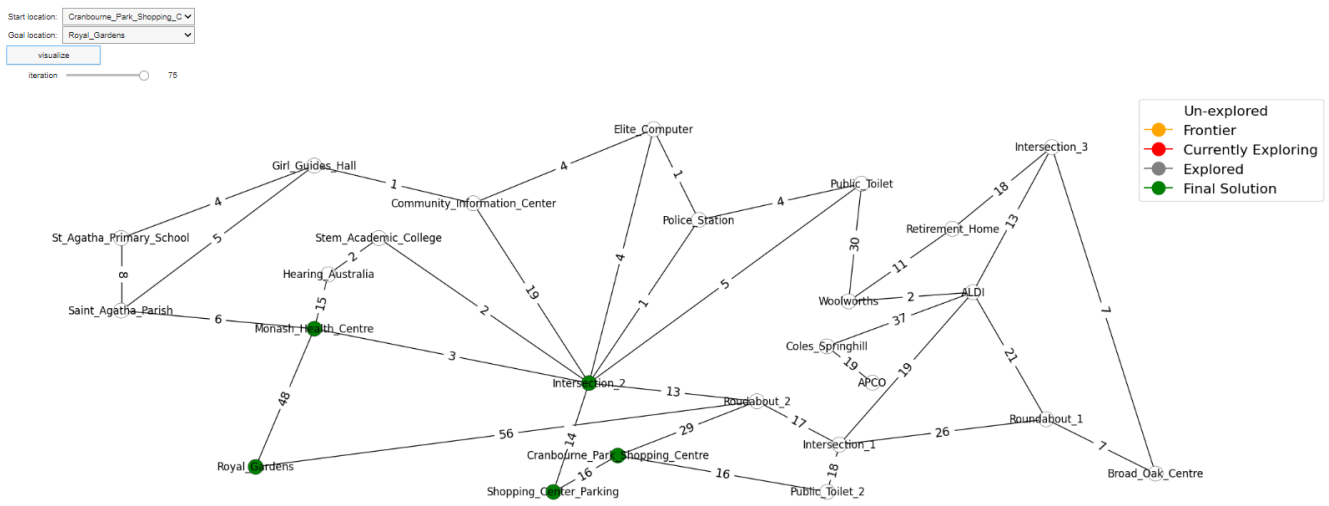
The first heuristic chosen is the Euclidean distance because in a map you can move in any direction. This heuristic determines the straight-line distance between two coordinates in n-dimensional space. (Eskander, 2023)
This feature is one of the reasons why this heuristic was justified for this solution. Through this, the straight-line distance between the coordinates of the current node and the goal node was determined when searching for the optimal path for wheelchair users.

Other common heuristics such as Manhattan Distance, which is when we are only allowed to move in 4 directions, were not suitable for this problem type. This is an intuitive and simple heuristic to use, which is a positive characteristic of this heuristic, which is another one of the reasons why this heuristic was selected for this solution (A* Search Algorithm, 2024).

This heuristic does have some limitations though. It ignores the correlation between dimensions. Since this map is placed on a cartesian plane, which is a 2D space this limitation is relevant. For example, even if there is a correlation between x and y (e.g. moving in one direction increases the other), the algorithm might expand on nodes that are shorter in one dimension but longer in another. As the dimensionality increases, Euclidean distance becomes less effective. Also, variables need to have the same unit or the calculation will be skewed. (What is Euclidean Distance?, n.d.)

Case 1:

In the case of finding a route from Cranbourne_Park_Shopping_Center to the Royal_Gardens, the graph was able to determine the route with the least wheelchair-accessible time to traverse.



Highlighted route:

Nodes traversed	Time
Cranbourne_Park_Shopping_Centre -> Shopping_Center_Parking	16
Shopping_Center_Parking -> Intersection_2	14
Intersection_2 -> Monash_Health_Centre	3
Monash_Health_Centre -> Royal_Botanic_Gardens	48

The total wheelchair-accessible time is **81** minutes.

An alternative route could have been:

Nodes traversed	Time
Cranbourne_Park_Shopping_Centre -> Roudabout_2	29
Roudabout_2 -> Intersection_2	13
Intersection_2 -> Monash_Health_Centre	3
Monash_Health_Centre -> Royal_Botanic_Gardens	48

Where the total wheelchair-accessible time would've been **93** minutes.

This alternative route time is higher than the route chosen by the solution, since 92 is greater than 81. Therefore, the solution was able to find the best route despite alternatives. The path returned is valid and optimal. This makes it an effective map for wheelchair users.

A heuristic needs to be admissible to guarantee optimal A* solutions, otherwise, the algorithm may explore paths that it should ignore.

$h(n) \leq h^*(n)$ (Admissibility of A* Algorithm, 2023)

In this case, the values in this formula would be,

81 ≤ 81.

h(n) is the cost indicated by h to reach a goal from n, which this algorithm found was 81 wheelchair-accessible minutes.

h*(n), the optimal cost to reach a goal from n is 81. This can be visually analysed by looking at the map.

This means that this heuristic is admissible since it doesn't overestimate the actual cost and avoids suboptimal solutions, such as the alternative route in the table for case 1. (SIT215 Week 4 - Lecture - Problem Solving and Evaluation AFTER CLASS, 2024)

Case 2

This is the selected route for a different set of inputs:

Start Location	End Location	Nodes traversed	Time (min)
Public_Toilet	Shopping_Center_Parking	Public_Toilet -> Intersection_2	5
		Intersection_2 -> Shopping_Center_Parking	14

Legend: Blue text means this is a node chosen in the path.

node	f(n)	g(n)	h(n) (to whole numbers)
Public_Toilet	17	0	= np.hypot((145.2942854- 145.28152), ((-21.1144225) - (-38.11188))) = 17
Police_Station	19	4	= np.hypot((145.2875418 - 145.28152), (-23.1133387 - (-38.11188)))

			= 15
Intersection_2	11	5	= np.hypot((145.2829935 - 145.28152), ((-32.1130201) - (-38.11188))) = 6
Shopping_Center_Parking	19	5 + 14 = 19	0

f(n) = g(n) + h(n), so when path cost is referred to, it means the f(n) value.

The Police_Station is a node that had a higher cost value than Intersection_2, since 19 > 11. So, the algorithm expanded on the Intersection_2 node to determine the path to the goal node. This resulted in a valid and optimal path since the route from the starting node (Public_Toilet) to the end node (Shopping_Center_Parking) was determined, with the route that had the lowest cost. It is also optimal since it found the route with the least wheelchair-accessible time to traverse.

Now the monotonicity of this solution will be analysed.
Formula used:

$$h(n) \leq \text{cost}(n, a, n') + h(n')$$

(SIT215 Week 4 - Lecture - Problem Solving and Evaluation AFTER CLASS, 2024)

Public_Toilet's h(n) should be less than or equal to its adjacent node Police_Station, and 17 ≤ 4 + 15. So, this is true, thus in this case the heuristic was monotonic.
Note: some fictional data was used for this assignment due to issues, as approved by the unit chair, so in all cases the heuristic may not be monotonic. This is not ideal, as it could lead to suboptimal paths or inaccurate solutions, but this is a limitation of this assignment.

Dijkstra's Algorithm, Alternative to the A* Algorithm Search

Dijkstra's algorithm was chosen as the alternative pathfinding algorithm because it's effective in finding the shortest path between points. The local_map used also meets the criteria of having non-negative edge weights and being connected. (itsadityash, 2024)

Compared to other algorithms such as Depth-First Search, Dijkstra's algorithm suits this problem better because Depth-First Search doesn't consider the weights between edges in the graph when determining a solution, whereas Dijkstra's algorithm does. (Mann, 2014)
Since Dijkstra's algorithm considers weights, its solutions may be more accurate and reliable. This means that the path found will likely be optimal.

Implementation:

The code for this part of the solution utilizes code from a user called nolfonzo.
The *shortestDijkstraPath* method takes in it's the parameters of the specified graph and the start and end nodes of the path to be found. If it's the program's first time through the function, **start** is set to 0. **end** is the goal test, so when **start** equals **end**, this means the path can be returned. But usually before that (is **start** and **end** are different), the section that is executed is where the tentative distance between the current node and its neighbours is compared. The **visited** list keeps track of which nodes have already been visited and compared.
This way, the neighbouring node with the shortest distance from the current note is stored as **start**. Then the closest unvisited node to this new **start** note is determined and *shortestDijkstraPath* recursively calls upon itself until the **end** node is reached. (nolfonzo, 2010)

A different implementation of the *local_map* and the coordinates of its locations had to be used to integrate with this solution. This *local_map* has the same locations and connections as the local_map in the A* method solution, but this is directed, and not an object of the Graph class. Remembering the main problem of this project, although

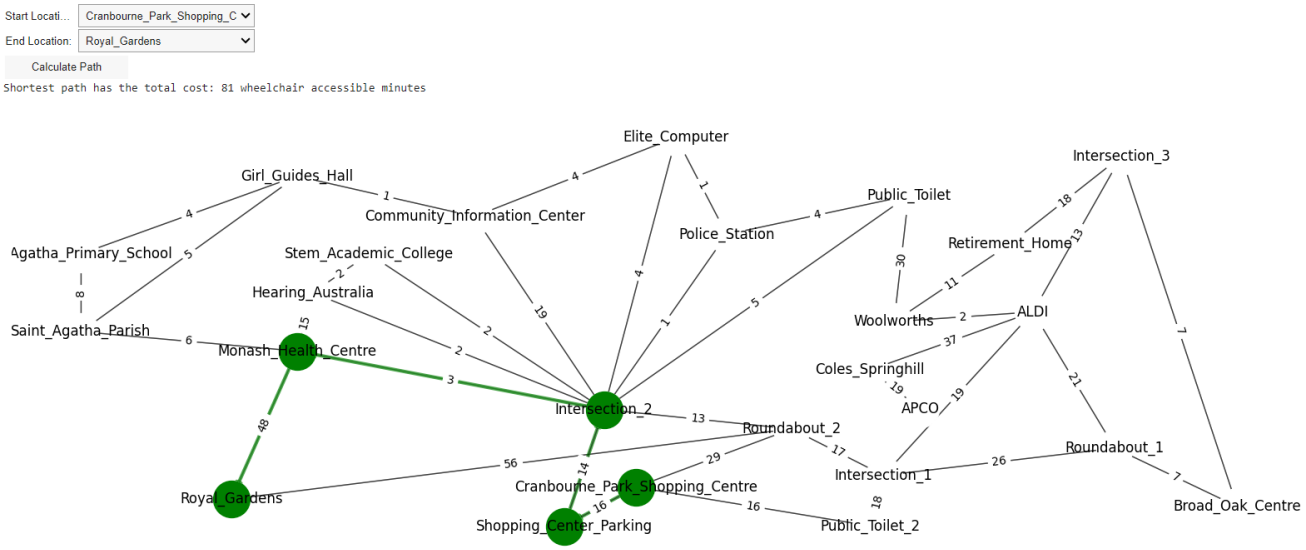
this algorithm doesn't use the coordinates data, it's still important to display a consistent visual, so that wheelchair users can navigate effectively.

The *display_visual_2* method enables the graph visual to be displayed and calls the *shortestDijkstraPath* method.

Compared to A* algorithm:

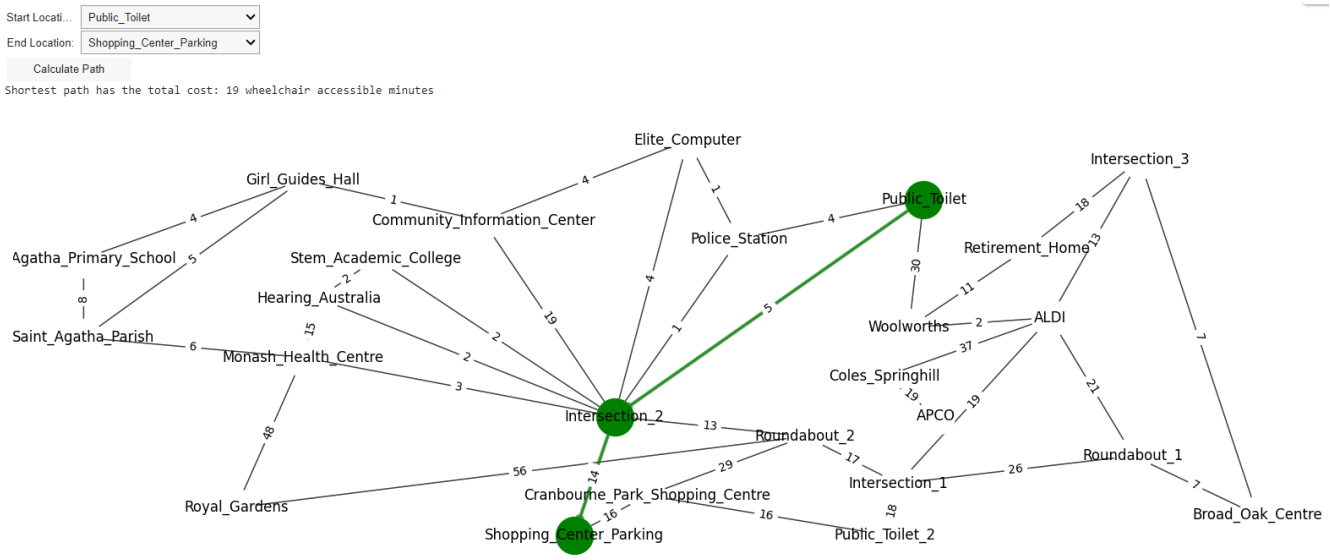
Compared to A* algorithm, this alternative algorithm provided the same optimal path solutions, so it is an effective technique. We can see this when observing paths determined from the same start and end points as in Case 1 and Case 2 from previous analyses.

Case 1:



Route: Cranbourne_Park_Shopping_Centre → Shopping_Center_Parking → Intersection_2 → Monash_Health_Centre → Royal_Gardens

Case 2:



Route: Public_Toilet → Intersection_2 → Shopping_Center_Parking

Case 3:

Some additional cases that seemed interesting were analysed, but the same optimal paths were produced by all start and end input combinations, in both the A* and Dijkstra algorithms.

Start Location	End Location	Nodes traversed	Time (min)
Police_Station	Woolworths	Police_Station -> Public_Toilet	4
		Public_Toilet -> Woolworths	30
Girl_Guides_Hall	Intersection_2	Girl_Guides_Hall -> Community_Information_Center	1
		Community_Information_Center -> Elite_Computer	4
		Elite_Computer -> Police_Station	1
		Police_Station -> Intersection_2	1
Broad_Oak_Centre	ALDI	Broad_Oak_Centre -> Intersection_3	7
		Intersection_3 -> ALDI	13

In professional practice, the A* algorithm is faster than Dijkstra's algorithm, if a good heuristic is designed. Dijkstra's algorithm could also be considered a type of A* algorithm, where the heuristic is equal to zero. (Simic & Aibin, 2024)
So, the A* algorithm is a more informed version of Dijkstra's algorithm, and this can lead to better outcomes. If working with a large graph, A* would be the better choice. Since Dijkstra's algorithm is less informed it can be more computationally intensive, and thus take more time, compared to its well-designed A* counterpart. If it's more computationally intensive that means that it expanded on more nodes than the A* search algorithm.

Conclusions and Lessons Learned

A key finding was that a map solution for wheelchair users can be created by using different heuristics in the A* algorithm. All the heuristics, Euclidean, Chebyshev and Max, contributed to the generation of the same optimal route for each respective starting and end location combination. The alternative algorithm, Dijkstra's Algorithm, also generated the same optimal routes as the previous A* search heuristic combinations.

This project enabled a deeper understanding of computational intelligence since programming was used to determine the best route for wheelchair users through computational intelligence. This required a proper understanding of the code, as well as computational intelligence, for the solution to be constructed in the necessary way to solve the problem. Different heuristic functions were implemented using the A* algorithm. A goal test was used to determine the correct node was reached, states were traversed through, actions were undertaken, and the path cost was determined.

This assignment used concepts from SIT215 to solve a real-world scenario, and during the preparation and design stages, it became evident that there are more elements to consider in real-world scenarios that increase the complexity of solutions compared to an in-class implementation. For example, the different elements that can affect the wheelchair-accessible time of users, such as slopes and uneven terrain. The importance of an accurate and structured dataset was realised since it can affect the accuracy of the solution. Additionally, it was realised that gathering data takes a long time too since the research for this data took four hours. In a real-world scenario, it may take a great number of resources to use computational intelligence to solve problems.

References

A* Search Algorithm. (2024, March 7). Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/a-search-algorithm/

Admissibility of A* Algorithm. (2023, April 23). Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/a-is-admissible/

Birney, A. (2024, April 7). ***How to measure distance on Google Maps***. Retrieved from Android Authority: https://www.androidauthority.com/measure-distance-google-maps-3104255/

brazofuerte. (2020 , April 25). ***What is non-Euclidean data?*** Retrieved from Artificial Intelligence Stack Exchange: https://ai.stackexchange.com/questions/11226/what-is-non-euclidean-data

Eskander, S. (2023, March 22). ***Exploring Common Distance Measures for Machine Learning and Data Science: A Comparative Analysis***. Retrieved from Medium: https://medium.com/@eskandar.sahel/exploring-common-distance-measures-for-machine-learning-and-data-science-a-comparative-analysis-ea0216c93ba3#:~:text=Chebyshev%20distance%20is%20a%20measure,data%20or%20in%20optimization%20problems.

itsadityash. (2024, March 8). ***What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm***. Retrieved from Geeks For Geeks: https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/

Lee, G. (2024). SIT215 Week 1 - Lecture - Intro CI and intelligent agents (AFTER CLASS) [PowerPoint slides]. Deakin Sync. https://d2l.deakin.edu.au/d2l/le/content/1424166/viewContent/7351249/View

Lee, G. (2024). SIT215 Week 2 - Lecture - Search Part 1 (AFTER - Recording (Public holiday)) [PowerPoint slides]. Deakin Sync. https://d2l.deakin.edu.au/d2l/le/content/1424166/viewContent/7360675/View

Lee, G. (2024). SIT215 Week 3 - Lecture - Search Part 2 (AFTER CLASS) [PowerPoint slides]. Deakin Sync. https://d2l.deakin.edu.au/d2l/le/content/1424166/viewContent/7363606/View

Lee, G. (2024). SIT215 Week 4 - Lecture - Problem Solving and Evaluation (AFTER CLASS) [PowerPoint slides]. Deakin Sync. https://d2l.deakin.edu.au/d2l/le/content/1424166/viewContent/7369727/View

Mann, E. (2014, March 5). *Depth-First Search and Breadth-First Search in Python*. Retrieved from Edd Mann: https://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

nolfonzo. (2010, August 24). *Dijkstra Shortest Path using Python*. Retrieved from Rebrained: https://rebrained.com/?p=392

***R Project*. (n.d.). Retrieved from Chebyshev distance: https://search.r-project.org/CRAN/refmans/abdiv/html/chebyshev.html**

Simic, M., & Aibin, M. (2024, March 18). *Dijkstra vs. A* – Pathfinding*. Retrieved from Baeldung: https://www.baeldung.com/cs/dijkstra-vs-a-pathfinding

***SIT215_Prac_W3 (AFTER CLASS).* (2024, March 21). Retrieved from Deakin Sync: https://d2l.deakin.edu.au/d2l/le/content/1424166/viewContent/7367933/View**

***SIT215_Prac_W4 (AFTER CLASS).* (2024, March 28). Retrieved from Deakin Sync: https://d2l.deakin.edu.au/d2l/le/content/1424166/viewContent/7373162/View**

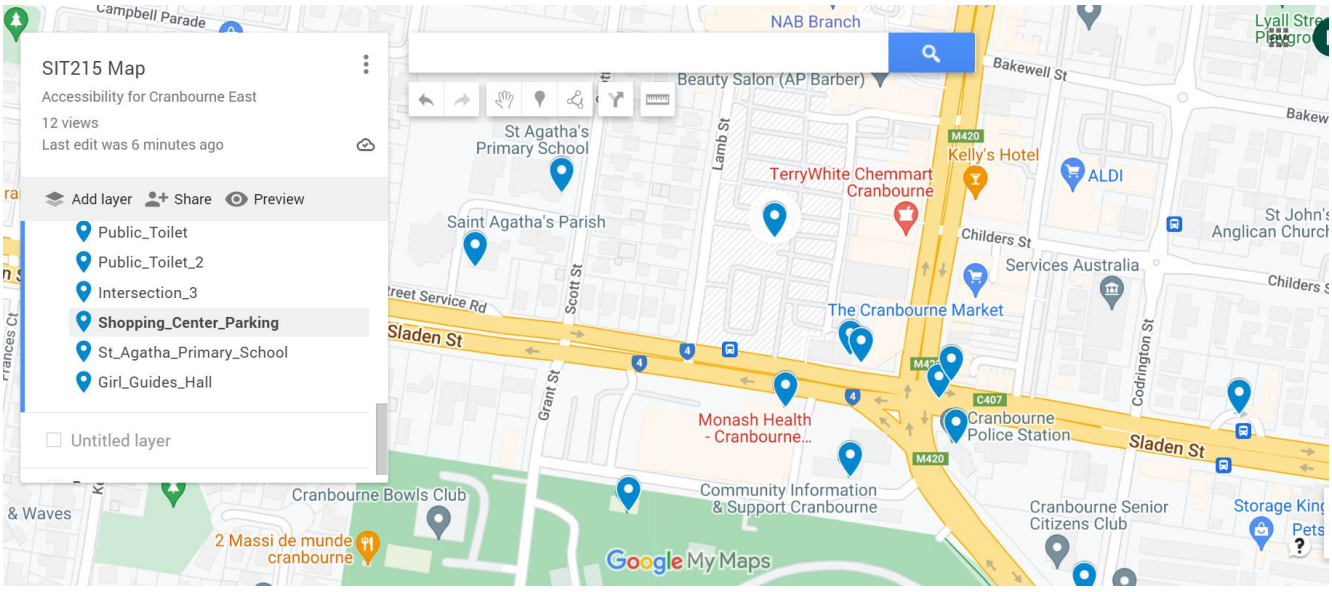
Swendeman, D. (2024, February 20). *First-Party Data [2023] | Definition, Examples, Segmentation*. Retrieved from daasity: https://www.daasity.com/post/first-party-data#:~:text=First%2Dparty%20data%20(aka%201st,comes%20directly%20from%20your%20customers.

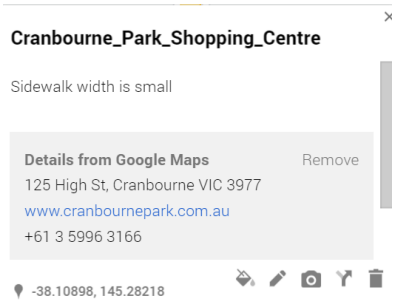
***What is Euclidean Distance?* . (n.d.). Retrieved from Bot Penguin: https://botpenguin.com/glossary/euclidean-distance**

***Wheelchair Usage*. (n.d.). Retrieved from SCIRE Professional: https://scireproject.com/evidence/wheeled-mobility-and-seating-equipment/manual-wheelchairs/wheelchair-use/wheelchair-usage/**

Appendix

Appendix A:





Appendix B:

Table B1:

	Royal_Botanic_Gardens	Cranbourne_Park_Shopping_Centre	Intersection_1	Broad_Oak_Medical_Centre	Roundabout_1	Roudabout_2	Intersection_2	Saint_Agatha_Parish
Royal_Botanic_Gardens						2.60		2.20
Cranbourne_Park_Shopping_Centre						0.90		
Intersection_1					1.00			
Broad_Oak_Medical_Centre					0.20			
Roundabout_1								
Roudabout_2			0.80					
Intersection_2						0.60		
Saint_Agatha_Parish								
APCO								
Coles_Springhill								
Monash_Health_Centre	2.22							
Woolworths_Cranbourne								
ALDI								
Hearing_Australia								
Stem_Academic_College								
Elite_Computer							0.20	
Police_Station								
Community_Information_Center								
Public_Toilet							0.20	
Public_Toilet_2			0.77					

Original Matrix | Location Weighting | Sheet1 | Speed Matrix

Table B2:

	Slope	Narrow Path	Uneven Terrain	Total Extra Time
Royal_Botanic_Gardens				0
Cranbourne_Park_Shopping_Centre		10		10
Intersection_1	4			4
Broad_Oak_Medical_Centre	3			3
Roundabout_1		4		4
Roudabout_2				0
Intersection_2				0
Saint_Agatha_Parish			3	3
APCO				0
Coles_Springhill				0
Monash_Health_Centre				0
Woolworths_Cranbourne		1		1
ALDI		1		1
Hearing_Australia		2		2
Stem_Academic_College	5			5
Elite_Computer				0
Police_Station		1		1
Community_Information_Center		2		2

SIT215 Map- Facilities | Original Matrix | Location Weighting | Speed Matrix

Table B3:

H17 =IF(((((('Original Matrix'!$H17*1000)/0.77)/60)<>0), ((((('Original Matrix'!$H17*1000)/0.77)/60)+'Location Weighting'!$F18), "")

	A	B	C	D	E	F	G	H	I
1		Royal_Botanic_Gardens	Cranbourne_Park_Shopping_Centre	Intersection_1	Broad_Oak_Medical_Centre	Roundabout_1	Roudabout_2	Intersection_2	Saint_Agatha_Parish
2	Royal_Botanic_Gardens						56		48
3	Cranbourne_Park_Shopping_Centre						29		
4	Intersection_1					26			
5	Broad_Oak_Medical_Centre					7			
6	Roundabout_1								
7	Roudabout_2			17					
8	Intersection_2						13		
9	Saint_Agatha_Parish								
10	APCO								
11	Coles_Springhill								
12	Monash_Health_Centre	48							
13	Woolworths_Cranbourne								
14	ALDI								
15	Hearing_Australia								
16	Stem_Academic_College								
17	Elite_Computer							4	
18	Police_Station								
19	Community_Information_Center								
20	Public_Toilet							5	
21	Public_Toilet_2			18					

Original Matrix | Location Weighting | Sheet1 | Speed Matrix