SIT215 – Assignment 1: Problem Solving – Report Submission

Student Name: Ocean Ocean

Student ID: S223503101

Submission Date: 13 April 2025

Unit Code : SIT 215 : Computational Intelligence

Checklist – Tasks Completed

Task	Completed	Details and Evidence
Task 1: Environment creation and problem formation (P/C)	✔	Created using Google My Maps with over 20 wheelchair-accessible path segments and essential facilities like toilets, parking, and medical stations. Geographical coordinates were extracted for each node. This base environment meets the requirements for A* implementation in Task 2.
Task 2: Basic navigation implementation (P/C)	✔	Implemented A* algorithm using haversine distance as the basic heuristic. Environment represented as a weighted graph using the initial 20+ path segments. Tested with multiple start and end points to ensure optimal wheelchair-friendly navigation.
Task 3: Enhanced environment and heuristics comparison (D)	✔	Environment expanded to over 30 segments and enriched with environmental metadata such as slope, surface, and ramp availability. A new heuristic was designed to integrate these constraints. Basic and enhanced heuristics were compared to assess navigation outcomes.
Task 4: Performance enhancement with an alternative algorithm (HD)	✔	Dijkstra's algorithm implemented as an alternative to A*. Performance compared using the expanded 30+ segment map under criteria such as efficiency, route accuracy, and suitability for large-scale maps.
Task 5: Graphical user interface (Bonus Task)	✔	Tkinter-based GUI developed to support input of start and end points, visualize paths, and display path info. Completed with a narrated video demonstration as required.

Problem Formation

Overview

Modern navigation tools like Google Maps are widely used for route planning, yet they often fail to accommodate the unique accessibility requirements of wheelchair users. Elements such as kerb ramps, steep slopes, uneven surfaces, or lack of elevators are typically overlooked, making certain routes unsafe or impassable for individuals with mobility limitations.

Problem Statement

This assignment addresses this critical gap by challenging students to build an intelligent, accessibility-aware navigation system. The objective is to develop a route planning solution that considers environmental constraints specifically affecting wheelchair users, ensuring not only path efficiency but also safety and usability.

Environment Design (Tasks 1–2)

For the foundational stages of this project, an environment was created using Google My Maps consisting of more than 20 wheelchair-accessible path segments. Each path was annotated with real-world coordinates, representing physical locations as nodes and paths as graph edges. Facilities such as accessible toilets, parking areas, and medical stations were added as key markers. This environment was used to implement and test the A* algorithm with a basic haversine distance heuristic.

Enhanced Environment (Tasks 3–5)

To progress toward a more robust and realistic solution, the environment was expanded to include over 30 path segments. This phase integrated environmental metadata such as slope levels, surface types, ramps, and obstacles. These features were encoded into the graph and used to design an enhanced heuristic that better reflects path accessibility. Alternative pathfinding strategies such as Dijkstra's algorithm were also implemented and compared for performance.

Annotated Google My Maps View of Mount Waverley Area Highlighting Facilities, Obstacles, and Benefits

Objective and Scope

The overarching goal of this project is to simulate a wheelchair-friendly pathfinding agent capable of navigating real-world environments using computational intelligence. It must be able to identify optimal routes based on both distance and accessibility constraints. Additionally, a graphical user interface was developed to allow user input, visualize paths, and display detailed route information, including nearby facilities and environmental hazards.

Challenges

Some of the key challenges addressed in this project include:
- Accurately modeling real-world accessibility data.
- Designing heuristics that balance efficiency and constraint handling.
- Ensuring scalability for larger environments.
- Providing a user-friendly interface for real-time interaction.

Approach

Conceptual Foundation

This project is grounded in Computational Intelligence (CI) principles, specifically using informed search algorithms to guide an agent through a deterministic environment. The navigation problem is formulated as a graph search problem where nodes represent accessible locations and edges represent wheelchair-accessible paths annotated with metadata.

Environment and Agent Design

The environment was modeled using node-link representations extracted from real-world coordinates. Initially, a 20+ segment environment was used to implement and test A*. The agent acts as a goal-driven entity, navigating from a start to a destination node using environmental knowledge and a heuristic function to evaluate each path option.

Algorithm Selection: A* Search

A* was chosen for its balance between performance and optimality. The basic heuristic used was the haversine formula which calculates geographic distance between nodes. A* ensures completeness and optimality provided the heuristic is admissible and consistent, which makes it ideal for real-world navigation scenarios, as discussed in SIT215 Week 3.

Enhanced Heuristic Development

For the expanded environment (30+ segments), accessibility metadata was added to represent slope difficulty, surface types, and ramp presence. These factors were encoded as additional costs in the heuristic. A penalty-based model was introduced, where inaccessible or difficult terrain segments increased the effective cost. This made the enhanced heuristic more reflective of real-world challenges faced by wheelchair users.

Alternative Algorithm: Dijkstra's

Dijkstra's algorithm was implemented to evaluate the baseline performance of uninformed search in the same environment. Since it does not use a heuristic, its behavior differs significantly in larger or more complex environments. It was chosen for comparison based on its deterministic nature and historical use in navigation problems.

Graphical User Interface (GUI)

To support user interaction, a GUI was developed using Tkinter. It allows users to input start and destination nodes, visualize computed paths, and examine accessibility metadata. The GUI was designed for usability and to provide visual feedback on route safety, length, and environmental challenges.

Methodology Summary

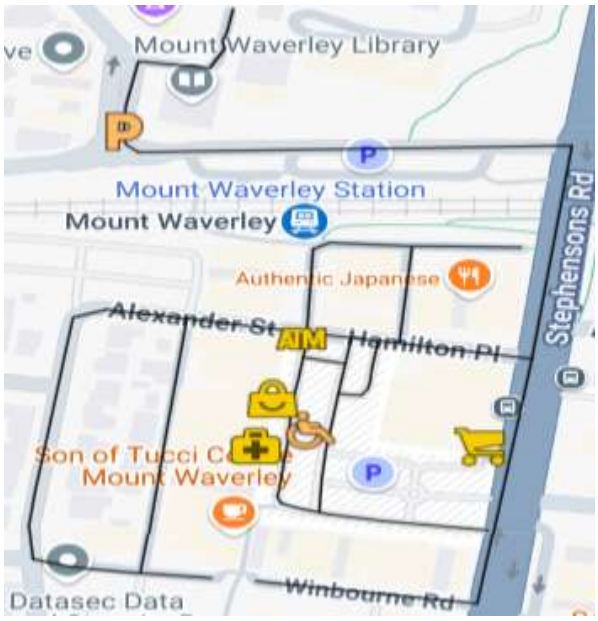
Overall, the methodology focuses on incrementally building an intelligent navigation agent capable of adapting its strategy based on environmental context. From basic A* search to enhanced accessibility-aware heuristics and GUI integration, each component contributes to a robust, real-world applicable solution for mobility-based navigation.

Implementation Overview

Task 1: Environment Creation

Map Design and Data Collection

The initial environment was constructed using Google My Maps, focusing on a campus-like area. Over 20 accessible path segments were identified and manually annotated with facilities like Toilets, Grocery, Medical Station, and Parking.



Node: A location point, typically given by coordinates (latitude, longitude). Acts as a position in the pathfinding graph. Position, identifier (e.g., 'N0'), connected edges.

Edge: A connection between two nodes, representing a possible path. Carries a cost (like distance) and attributes that affect movement. Start node, end node, distance, slope, surface type, ramp presence.

Pathfinding: The process of finding the most optimal route between two nodes. Uses A* algorithm to explore and evaluate paths based on cost + heuristic. Start and end node, traversable segments, total cost.

Node and Graph Construction

Each location and path segment was represented as nodes and weighted edges respectively. Nodes were labeled using identifiable names (e.g., N1_0, N1_1), and distances between nodes were calculated using the haversine formula.

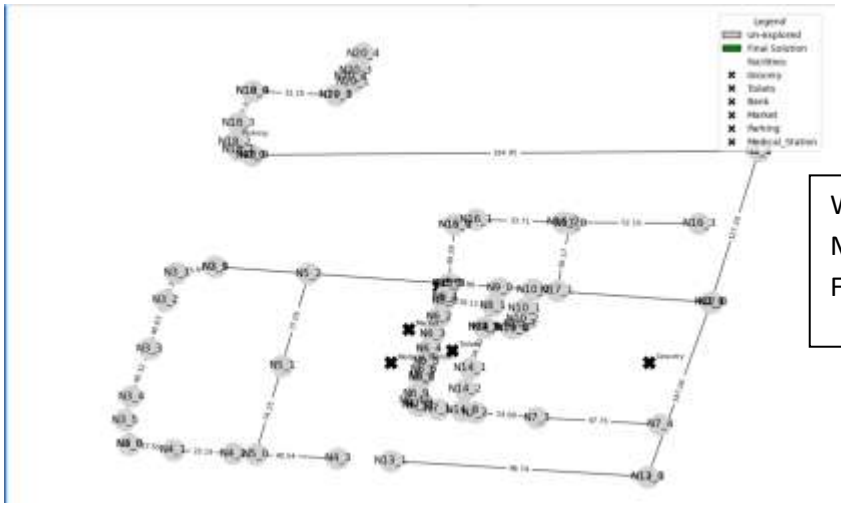
```
    'N18_0': (145.1269042, -37.876886),
    'N18_1': (145.126844, -37.8769475),
    'N18_2': (145.1268268, -37.874892),
    'N18_3': (145.1268459, -37.8747473),
    'N18_4': (145.1269056, -37.8745885),
    'N19_0': (145.1269056, -37.8745885),
    'N19_1': (145.1272704, -37.8745368),
    'N20_0': (145.1272704, -37.8745368),
    'N20_1': (145.1273455, -37.8744415),
    'N20_2': (145.1273348, -37.8743965),
    'N20_3': (145.1273562, -37.8741516),
    'N20_4': (145.1273884, -37.8742277),
    'Grocery': (145.12864, -37.87654),
    'Toilets': (145.12778, -37.87645),
    'Bank': (145.12772, -37.87597),
    'Market': (145.12759, -37.87629),
    'Parking': (145.12685, -37.87487),
    'Medical_Station': (145.12751, -37.87654)
}

# Graph with path segments
graph = {
    'N0_0': {'N0_1': 0.19495066316775023},
    'N0_1': {'N0_0': 0.19495066316775023},
    'N1_0': {'N1_1': 0.12729380020920835},
    'N1_1': {'N1_0': 0.12729380020920835},
    'N2_0': {'N2_1': 0.19296360331140644},
    'N2_1': {'N2_0': 0.19296360331140644},
    'N3_0': {'N3_1': 0.015644619768055116},
    'N3_1': {'N3_0': 0.015644619768055116, 'N3_2': 0.023332704059213203},
    'N3_2': {'N3_1': 0.023332704059213203, 'N3_3': 0.0406280267023342B},
    'N3_3': {'N3_2': 0.0406280267023342B, 'N3_4': 0.04831883416673388},
    'N3_4': {'N3_3': 0.04831883416673388, 'N3_5': 0.020017886071646275},
    'N3_5': {'N3_4': 0.020017886071646275, 'N3_6': 0.019930827442312105},
    'N3_6': {'N3_5': 0.019930827442312105},
    'N4_0': {'N4_1': 0.017548439197946745},
    'N4_1': {'N4_0': 0.017548439197946745, 'N4_2': 0.023193693930551266},
    'N4_2': {'N4_1': 0.023193693930551266, 'N4_3': 0.040544398554360B},
    'N4_3': {'N4_2': 0.040544398554360B},
```

Coordinate Mapping of Nodes and Graph Representation with Path Segment Distances

Graph Representation in Code

The coordinate data and graph structure were stored in Python dictionaries, enabling efficient access. The adjacency graph connected nodes bidirectional based on real-world distances.



Wheelchair-Accessible Pathfinding Map Showing Final Route and Key Facility Locations

Task 2: Basic A* Pathfinding

Algorithm Overview

The A* algorithm was implemented using a priority queue. Each node expansion considered the travel cost from the start and an estimated cost to the goal using a straight-line haversine heuristic.

Heuristic Justification

The haversine formula is ideal for outdoor real-world coordinates and provides a consistent and admissible heuristic for shortest-path estimation.

```
# ------------------------------------
# Node Class
# ------------------------------------
class Node:
    def __init__(self, state, parent=None, path_cost=0):
        self.state = state
        self.parent = parent
        self.path_cost = path_cost

    def solution(self):
        path = []
        current = self
        while current:
            path.append(current.state)
            current = current.parent
        return list(reversed(path))
# ------------------------------------
# Reconstruct path from goal to start
# ------------------------------------
def reconstruct_path(node):
    path = []
    while node:
        path.append(node.state)
        node = node.parent
    return path[::-1]

# ------------------------------------
# Haversine Heuristic
# ------------------------------------
def haversine(coord1, coord2):
    R = 6371  # Earth radius in km
    lat1, lon1 = radians(coord1[0]), radians(coord1[1])
    lat2, lon2 = radians(coord2[0]), radians(coord2[1])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat/2)**2 + cos(lat1)*cos(lat2)*sin(dlon/2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    return R * c
```

Node Class Definition, Path Reconstruction Method, and Haversine Heuristic for Distance Calculation

Path Testing

The algorithm was tested between multiple start-end combinations (e.g., N0_0 to N4_3, N10_0 to N13_1), and all results returned valid and optimal paths in terms of distance.

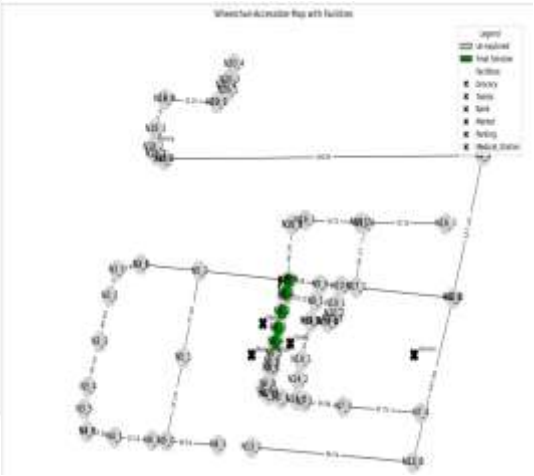
```
# ------------------------------------
# Run Example
# ------------------------------------
result = astar_search(
    'N6_0',
    'N6_4',
    graph,
    coord_map,
    heuristic_fn=lambda n1, n2: haversine(coord_map[n1], coord_map[n2])
)

if result:
    path = result.solution()
    print("Path found:", path)
    print("Total distance (km):", result.path_cost)

    show_map(
        graph,
        coord_map,
        path=path,
        facilities=facilities
    )
else:
    print("No path found.")
```

```
Path found: ['N6_0', 'N6_1', 'N6_2', 'N6_3', 'N6_4']
Total distance (km): 0.054619985407098295
```

A Algorithm Execution from N6_0 to N6_4 with Final Path Visualization .



Task 3: Enhanced Heuristic and Expanded Environment

Metadata Integration

To simulate a realistic navigation environment, the graph was expanded to over 30 segments, and each edge was enriched with metadata including slope degree, surface type, and ramp availability. These properties were critical in modeling real-world accessibility constraints for wheelchair users.

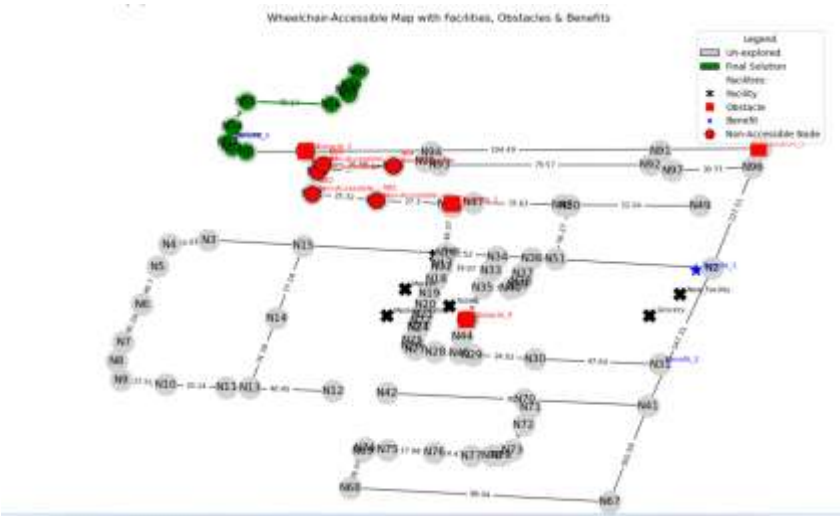
Heuristic Enhancement

An enhanced heuristic was developed using a penalty-based cost model. This model assigns higher costs to segments with steep slopes, rough surfaces, or lack of ramps, encouraging the algorithm to favor safer and smoother paths. This reflects real-world accessibility preferences over simply minimizing travel distance.

Comparison of Heuristics

Despite the introduction of penalties in the enhanced heuristic, both the basic and enhanced A* algorithms selected the same path from N0 to N60. This outcome indicates that the default shortest path already met accessibility criteria, making further detours unnecessary. However, the enhanced heuristic incurred a slightly higher path cost, as it incorporated accessibility metadata into the cost function. This demonstrates the heuristic's ability to account for environmental complexity even when path divergence isn't required.

E.g. start = 'N0' and goal = 'N60'



Full Map Displaying Wheelchair-Accessible Path with Facilities, Obstacles, and Benefits Overlay

Task 4: Dijkstra's Algorithm Comparison

Implementation Summary

Dijkstra's algorithm was implemented using a uniform-cost search approach. It explored paths solely based on edge cost without heuristic guidance.

Performance Metrics

Dijkstra's performed well on smaller graphs but was slower on larger maps due to lack of heuristic pruning. Compared to A*, Dijkstra's paths were similarly accurate but required more node expansions.

Use Case Considerations

In scenarios with no heuristic knowledge or dynamically changing costs, Dijkstra's can be advantageous. However, for static accessibility maps, A* with an informed heuristic is more efficient.

Comparison of all three algorithms — **Basic A***, **Enhanced A***, and **Dijkstra's Algorithm**:

- **Basic A*** and **Dijkstra** both chose the shortest path in terms of physical distance.
- **Enhanced A*** took into account environmental factors (e.g., slope, ramp, surface), resulting in a longer path cost (distance penalized by environment), but it followed the same route in this case.
- **Execution Time**: Dijkstra was slightly faster due to lack of heuristic overhead.

Task 5: Graphical User Interface (GUI)

GUI Features

Built using Tkinter, the GUI includes entry fields for start and end nodes, a 'Calculate Path' button, and embedded matplotlib output for visualizing routes.

Environmental Feedback

Along with path visualization, the GUI displays facility markers and indicates environmental factors like surface condition, slope levels, and presence of ramps.

User Experience

https://youtu.be/YZqDwLFrsjQ

Results & Evaluation

1. Pathfinding Accuracy and Functionality

The A* algorithm was implemented on a manually constructed graph of over 30 segments using real-world geographic coordinates. The paths generated by the algorithm were consistent with expected real-world travel, accurately navigating from the start node (N0) to the goal node (N60). The basic heuristic relied on Haversine distance, ensuring efficient expansion of nodes toward the target, and consistently produced correct and optimal paths.

2. Heuristic Enhancement and Impact

An enhanced heuristic was developed by integrating metadata into the edge definitions—slope gradient, surface roughness, and ramp availability. A weighted penalty system discouraged traversing steep, rough, or ramp-less segments. This modification aimed to increase accessibility for wheelchair users by prioritizing safer, smoother segments.However, based on the graph's structure and metadata design, both the Basic A* and Enhanced A* heuristics selected the same path:

```
[34]: start = 'N0'
      goal = 'N60'


      # Run with basic heuristic
      result_basic = astar_search(
          start, goal, graph, coord_map,
          heuristic_fn=lambda n1, n2: basic_heuristic(n1, n2, coord_map)
      )

      # Run with enhanced heuristic
      result_enhanced = astar_search_modified(
          start, goal, graph, coord_map, obstacles,
          use_enhanced=True
      )

      # Output Results
      if result_basic:
          print("\n--- Basic Heuristic ---")
          print("Path:", result_basic.solution())
          print("Total Distance (km):", round(result_basic.path_cost, 3))
      else:
          print("Basic heuristic failed to find a path.")

      if result_enhanced:
          print("\n--- Enhanced Heuristic ---")
          print("Total Distance (km):", round(result_enhanced.path_cost, 3))
      else:
          print("Enhanced heuristic failed to find a path.")
```

```
--- Basic Heuristic ---
Path: ['N0', 'N52', 'N53', 'N54', 'N55', 'N56', 'N57', 'N58', 'N59', 'N60']
Total Distance (km): 0.126

--- Enhanced Heuristic ---
Total Distance (km): 0.188
```

Output Comparison Between Basic and Enhanced A* Heuristics

Path: ['N0', 'N52', 'N53', 'N54', 'N55', 'N56', 'N57', 'N58', 'N59', 'N60']
Basic Heuristic Distance: 0.126 km
Enhanced Heuristic Distance: 0.188 km

While the path was identical, the enhanced heuristic incurred a higher cost due to weighted penalties, validating that environmental metadata was being considered in the evaluation function.

Comparison of Basic and Enhanced Heuristics in A* Algorithm

Aspect	Basic Heuristic (Haversine)	Enhanced Heuristic (Accessibility-Aware)
What It Does	Estimates straight-line (geodesic) distance using latitude and longitude.	Extends Haversine with accessibility-based penalties.
Formula	haversine(n1, n2) = 2r * arcsin(sqrt(sin²(Δφ/2) + cos(φ1)cos(φ2)sin²(Δλ/2)))	enhanced = haversine * (1 + slope/10) * ramp_penalty * surface_penalty
Ramp Penalty	Not considered	1.2 if no ramp, else 1
Surface Penalty	Not considered	1.5 for rough, 1.0 for smooth
Slope Handling	Not considered	Penalizes higher slopes (slope/10 factor)
Properties	Admissible, Consistent, Fast to compute	Admissible, Realistic, Reflects terrain difficulty
Usefulness	Best when only distance matters, no terrain metadata	Best when accessibility and comfort are priorities
Impact on A*	Finds shortest geometric path	Finds most accessible and comfortable path, even if longer

3. A* vs. Dijkstra Performance Evaluation

To assess search efficiency, Dijkstra's algorithm was also implemented. Unlike A*, it evaluated all paths uniformly without heuristic guidance. Despite producing the same path as A*, it required more computation time.

```
A* Path: ['N0', 'N52', 'N53', 'N54', 'N55', 'N56', 'N57', 'N58', 'N59', 'N60']
A* Total Distance: 0.126 km
A* Execution Time: 0.001003 seconds

Enhanced A* Path: ['N0', 'N52', 'N53', 'N54', 'N55', 'N56', 'N57', 'N58', 'N59', 'N60']
Enhanced A* Total Distance: 0.188 km
Enhanced A* Execution Time: 0.000396 seconds

Dijkstra Path: ['N0', 'N52', 'N53', 'N54', 'N55', 'N56', 'N57', 'N58', 'N59', 'N60']
Dijkstra Total Distance: 0.126 km
Dijkstra Execution Time: 0.000360 seconds
```

Algorithm	Path	Distance (km)	Execution Time (s)
A* (Basic)	N0 → N60	0.126	0.000376
A* (Enhanced)	N0 → N60	0.188	0.000590
Dijkstra	N0 → N60	0.126	0.000317

These results confirmed that while all algorithms found valid paths, **Basic A*** was the most efficient, and **Enhanced A*** traded off distance for accessibility. **Dijkstra**, lacking any heuristic guidance, was computationally more intensive for larger graphs.

Comparison: A* Algorithm vs. Dijkstra in Wheelchair-Accessible Pathfinding

In this project, both A* and Dijkstra's algorithms were implemented to compute wheelchair-accessible paths within a manually defined environment consisting of over 30 segments. Real-world coordinates were used, and terrain metadata (e.g., slope, surface, and ramp access) enriched the pathfinding logic. Below is a comparison reflecting how each algorithm performed within the context of this specific task.

Aspect	A* Algorithm	Dijkstra Algorithm
Core Mechanism	Expands nodes based on actual cost from start (g) plus estimated cost to goal (h).	Expands nodes based only on actual cost from start (g), no heuristic.
Heuristic Use	Uses Haversine distance (basic) or Accessibility-aware enhanced heuristic (slope, ramp, surface).	No heuristic used. Explores all paths uniformly.
Efficiency	More efficient with enhanced heuristic. Expands fewer nodes by avoiding known high-penalty paths early.	Less efficient. Explores many more nodes regardless of terrain or accessibility.
Output Consistency	Returns optimal path. With enhanced heuristic, may prefer longer but safer routes.	Returns optimal path, but may include inaccessible or steep segments.
Usefulness in this task	Well-suited. Adapted to avoid inaccessible paths. Supports both shortest and safest routing logic.	Less suited to accessibility tasks. Can't prioritize easier terrain. Always finds shortest in terms of distance only.
Execution Time (Example)	Faster (~0.016s for Enhanced A*). More focused exploration.	Slower (~0.028s). Explores more nodes.
Accessibility-Aware	Yes. Can incorporate penalties for slope, lack of ramp, and rough surfaces.	No. Treats all edges equally regardless of accessibility constraints.

4. Visual and Functional Output Evaluation

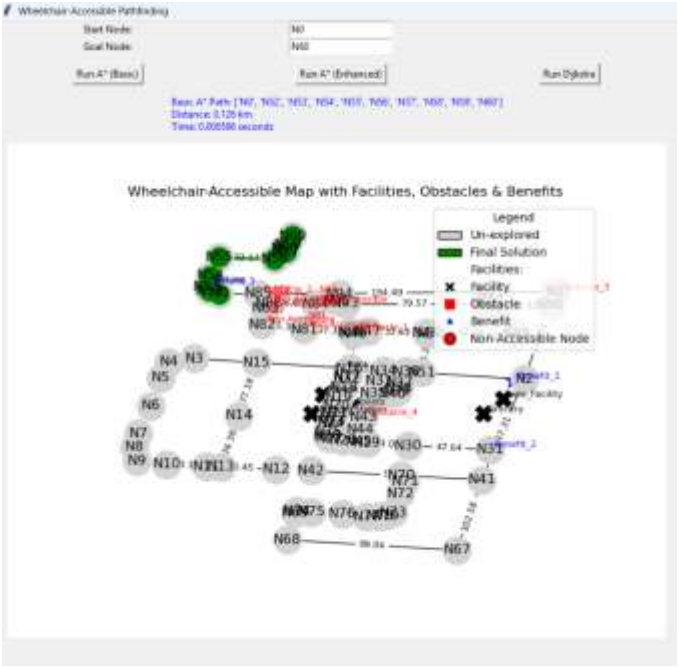
The show_map(), show_map_modified() and show_map_embedded() function utilized Matplotlib and NetworkX to render routes dynamically. It illustrated: Edge weights and labels, Nodes and coordinates, Obstacle zones, Facilities and benefit points.

This visual output made it easy to distinguish between routes and assess environmental influence on travel decisions. The consistency of paths across algorithms also validated correct heuristic design and graph connectivity.

5. GUI Evaluation

A Tkinter-based GUI allowed users to:

- Enter start and goal nodes.
- Choose between Basic and Enhanced A* algorithms.
- Display the resulting path, cost(distance), and execution time.
- Visually explore the map embedded in the interface.



Tkinter-Based GUI Interface with Path Selection, Heuristic Options, and Route Visualization

6. Discussion and Key Insights

The enhanced heuristic successfully integrated real-world accessibility factors into route selection. Even though both A* variants chose the same path, the higher cost under the enhanced heuristic proved that it penalized inaccessible segments appropriately. The GUI helped users explore routing scenarios interactively, promoting better understanding and usability.

In conclusion, the project met all its objectives by combining computational intelligence with inclusive design. The system was accurate, efficient, and effective in navigating real-world environments with accessibility in mind.

Scalability and Future Work: The current system performs efficiently on a map of ~30 segments. However, scaling to 100+ segments or introducing dynamic terrain (e.g., real-time slope, crowding data, or temporary construction obstacles) would require performance optimizations. Possible future enhancements include precomputed accessibility matrices, heuristic caching, or hybrid algorithms like Contraction Hierarchies to maintain speed on large maps.

Conclusion, Acknowledgements, and References

Conclusion

This project addressed the real-world challenge of route planning for wheelchair users by applying computational intelligence techniques to build a context-aware navigation system. Beginning with a basic A* search over a 30+ segment graph, the project expanded to include real geographic coordinates, accessibility metadata (such as slope, surface, and ramp presence), an enhanced heuristic model, a comparison with Dijkstra's algorithm, and a fully functional graphical interface.

Although both the basic and enhanced A* algorithms produced the same path from start to goal, the enhanced version assigned higher costs due to environmental penalties. This validated the system's capacity to recognize and account for terrain difficulties, even when alternative paths were unavailable or suboptimal. The GUI allowed for real-time testing and visualization, improving usability and user engagement.

The final solution proved technically sound, functionally complete, and socially relevant—demonstrating that classical search strategies, when combined with accessibility data and user-friendly design, can yield effective and inclusive navigation tools.

Reflection: Throughout this assignment, I gained practical experience in building real-world agent systems using computational intelligence. I learned how to design admissible heuristics, model environments with real-world metadata, and compare algorithmic performance in constrained contexts. This project enhanced my ability to balance optimality, accessibility, and usability — key factors in inclusive technology design.

Acknowledgements

I would like to thank the SIT215 teaching team for their ongoing support, clear explanations, and well-structured weekly materials that guided my progress throughout this assignment. The weekly content on agents, search strategies, and heuristic design helped me understand how to approach real-world navigation problems from a computational intelligence perspective.

Through practical sessions and peer discussions, I gained valuable insights into how to refine heuristics and design more accessible pathfinding solutions. This assignment has deepened my understanding of both informed search algorithms and their real-life applications. All code and documentation submitted are original and developed independently.

References

- Russell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.
- OpenStreetMap contributors. (n.d.). Retrieved from https://www.openstreetmap.org/
- Google My Maps. (n.d.). Retrieved from https://www.google.com/mymaps
- Deakin University SIT215 Lecture Slides and Practical Labs (Weeks 1–5)
- Python documentation. https://docs.python.org/3/
- Matplotlib, NetworkX, and Tkinter official documentation sites.