Artificial Intelligence Practice: Heuristic Search in Radars

Artificial Intelligence

Bachelor in Computer Science and Engineering

2024 - 2025

Completed By:

Lim Yong Kang - 100557942

Delicia R. Pryce - 100547379

Jody Ng Li Min - 100557936

May 15th 2025

Introduction

Our project explores the development of a path-finding search engine that incorporates A*search algorithm with heuristics to generate a path for the aircraft to minimise its detection probability while visiting the points of interest (POIs). The stealth aircraft should be able to navigate the grid space that has been filled with radars and visit every POI. We emphasise that heuristic search is important to guide decision-making by our search engine through its cost estimation so that the least-cost nodes can be selected to derive an optimal path. We explored the efficacy of the Manhattan Distance heuristic and the Chebyshev Distance heuristic across various scenarios and we evaluated the heuristics using path cost, time, and nodes visited as the evaluation metric. Additionally, we varied the number of radars and the number of POIs to see the effect these parameters have on the search engine, across both heuristics. We then conclude that the Manhattan Distance heuristic is well-suited for uniform, open environments, while the Chebyshev Distance heuristic is more adaptive in intricate or diagonal-heavy layouts. We also proposed possible areas of application and future refinements to our project. Through the project, we sought to better understand the heuristics we proposed and their impact on the A*star path finding algorithm.

Modelling the Search Problem

States and Operators

A state refers to a cell in a grid represented by coordinates (i, j), where i is the row (latitude index) and j is the column (longitude index). The set of all states refers to all grid cells that are below the threshold probability.

The operators are essentially the moves the plane can make. From any given state, the plane can move either up, down, left, or right to any adjacent cell so long as the cell is within bounds and has detection probability below the allowed threshold.

The cost of moving from one cell to another is equal to the detection probability of the destination cell.

Initial and Goal States

The current location of the aircraft on the grid will define a state. The state space S shall be defined as follows:

$S = A \times B = \{(a, b) a \in A, b \in B\}$

A will be the set of longitude values in the grid.

B will be the set of latitude values in the grid.

(a,b) will represent the position of the aircraft on the grid.

The aircraft can move in 4 directions - north, south, east, west.

Let **s** represent a state in the state space S.

Let **lonr** represent the grid resolution of the longitude.

Let **latr** represent the grid resolution of the latitude. Note that in a square grid, **lonr == latr**.

$Operations(s) = \{(a + (\Delta da * lonr), b + (\Delta db * latr)) | (\Delta da, \Delta db) \in \{(1, 0), (-1, 0), (0, 1), (0, -1)\}\}$

$a + (\Delta da * lonr)$ must be between 0 and maximum longitude.

$b + (\Delta db * latr)$ must be between 0 and maximum latitude.

Additionally, $P(a + (\Delta da * lonr), b + (\Delta db * latr)) \leq$ threshold where P(a,b) represents the detection probability.

The initial state

$s\,initial = \{\}$

The goal state

$s\,goal = \{(longitude, latitude)\ of\ the\ last\ point\ of\ interest\}$

The minimum cost of reaching **i** number of points of interest is the sum of the minimum cost of reaching the first point of interest to the (i-1)th point of interest. The final path should include the path transition between all POIs.

Heuristics Function

Two heuristic functions were implemented and passed to the A* algorithm:

- **h1:** Manhattan distance (adds up the vertical and horizontal steps between two points).

- **h2:** Chebyshev distance (takes the maximum of the horizontal or vertical distance).

These heuristics estimate how far it is from one point to another, helping the algorithm decide which paths to try first.

Admissibility

A* algorithm was applied to search for an optimal path in a radar detection grid. Each move from one cell to an adjacent cell incurs a cost equal to the radar detection value of the *destination* cell. These detection values are scaled to lie within the interval $[\varepsilon, 1]$, where ε is a small positive constant (e.g., $\varepsilon = 0.01$) introduced to ensure no cell has zero detection cost.

- $h_1(n)$: **Manhattan distance** between node n and the goal, multiplied by ε

- $h_2(n)$: **Chebyshev distance** between node n and the goal, multiplied by ε

These heuristics estimate the **minimum possible cost** of reaching the goal, assuming every step incurs the **lowest allowed cost**, ε .

$$h(n) = distance(n, goal) \times \varepsilon$$

The actual cost of moving from node n to the goal is computed as the sum of the detection values of all destination cells along the path. Since every such value is at least ε , the true cost will always be **greater than or equal to** the heuristic estimate.

Therefore, the following condition is always satisfied:

$$h(n) \leq actual\ cost(n, goal)\ \forall n$$

This confirms that both h_1 and h_2 are **admissible heuristics**, as they never overestimate the actual cost to reach the goal. As a result, A* will always return an optimal solution when using either heuristic.

System Architecture and Implementation

Map and Search Space Generation

To support UAV navigation through radar-surveilled airspace, we implemented a detection-aware search space based on a discretized grid map and its conversion into a graph structure suitable for heuristic pathfinding.

Detection Map Construction

The map is defined as a 2D grid of $H \times W$ cells, where each cell represents a specific geographic coordinate and stores a value indicating the probability of being detected by radar.

Radar Signal Model (Equation 1):
Each radar's detection range was estimated using the radar equation, which models signal attenuation with distance using an inverse-square law. This provides an upper bound on the radar's effective detection range.

Multivariate Gaussian Distribution (Equations 2–4):
Each radar was modeled as a 2D Gaussian distribution centered at its location. The detection value at any cell is computed based on its distance to the radar and falls off smoothly with increasing distance. The final detection field is generated by combining all radar distributions, resulting in overlapping detection zones and a smooth probability surface.

Min-Max Scaling (Equation 7 or 8):
After combining all detection contributions, the resulting values are normalized to fall within the range $[\varepsilon, 1]$ using min-max scaling:

$$\Psi scaled = ((\Psi max - \Psi min)/(\Psi - \Psi min))(1 - \varepsilon) + \varepsilon$$

This guarantees that all cells have non-zero cost, prevents zero-cost transitions, and facilitates consistent thresholding.

Graph Representation of the Search Space

To enable pathfinding, the detection map is converted into a directed graph:

- Each cell in the grid becomes a node in the graph, indexed by its (x, y) coordinate.

- Edges are added between each node and its four immediate neighbors (up, down, left, right).

- A connection is created **only if both the source and destination cells have detection values below a user-defined tolerance threshold**, representing areas deemed navigable by the UAV.

- The **cost of an edge** corresponds to the detection value of the **destination cell**. This models the idea that entering a high-risk area increases total radar exposure.

This representation preserves the structure of the physical environment while enforcing constraints on safe navigation through probabilistic radar zones.

A* Algorithm Integration

Pathfinding is performed using the A* algorithm provided by the networkx library. The algorithm operates on the graph representation described above and is guided by admissible heuristics.

- $h_1(n)$: **Manhattan distance** between node n and the goal, multiplied by ε

- $h_2(n)$: **Chebyshev distance** between node n and the goal, multiplied by ε

Each heuristic provides a lower bound estimate of the cost to the goal under the assumption that all steps incur the minimum possible detection cost. This guarantees admissibility and enables the A* algorithm to return optimal paths with respect to total radar exposure.

Threshold Filtering

To reflect operational constraints, a **tolerance threshold** is applied during graph generation. Cells with detection values above the threshold are considered unsafe and are excluded from the graph entirely. This prevents the aircraft from flying through regions where the likelihood of detection is too high.

The integration of the detection model, graph abstraction, and A* search enables the aircraft to plan efficient and safe routes between multiple points of interest. The system prioritizes paths that minimize radar exposure while respecting no-fly zones defined by a tunable threshold parameter.

Experimentation and Results

We have evaluated each scenario using varying tolerance values ranging from 0.2 to 1.0. For each scenario and tolerance level, we recorded the following metrics:

1. Number of nodes expanded
2. Total cost of the path
3. Execution time

We then selected the optimal configuration based on the lowest path cost, using node count and runtime as tie-breakers. The results were compiled into a comparative dataset and visualised for further analysis, as seen in *Figure 1*.

```
Scenario  Heuristic Tolerance Nodes     Cost        Time(s)
-------------------------------------------------------------
scenario_0h1        0.20      59        0.10        0.00065
scenario_0h2        0.20      68        0.10        0.00052
scenario_0h1        0.40      68        0.10        0.00047
scenario_0h2        0.40      212       0.43        0.00146
scenario_0h1        0.60      72        0.10        0.00050
scenario_0h2        0.60      144       0.18        0.00097
scenario_0h1        0.80      68        0.10        0.00054
scenario_0h2        0.80      68        0.10        0.00049
scenario_0h1        1.00      68        0.10        0.00047
scenario_0h2        1.00      82        0.15        0.00050
No path found for h1, scenario_1 with tolerance 0.2
```

Figure 1: Path cost across scenarios for the best performing heuristic at each tolerance level

Table 1 below shows our best performing heuristic for each scenario with its respective tolerance.

Scenario Number	**Heuristic**	**Tolerance**	**Nodes Visited**	**Cost**	**Time (in seconds)**
0	h1	0.2	59	0.1	0.00065
1	h1	0.8	53	0.1	0.00041
2	h2	0.2	819	0.51	0.00524
3	h1	1.0	1027	5.77	0.00644
4	h2	0.4	1159	3.48	0.00716
5	h2	1.0	5057	18.94	0.03104
6	h2	0.8	4727	26.12	0.02877
7	h1	1.0	29828	52.8	0.18911
8	h2	0.6	77138	93.2	0.51306
9	h1	0.6	2724816	799.25	24.69998
10	h1	0.6	89430	39.88	0.71511
11	h2	0.6	108084	113.37	0.72733

Table 1: Best-performing heuristic and corresponding tolerance across scenarios

Key Observations

Heuristic Effectiveness

We have observed that:

- The Manhattan heuristic (h1) consistently outperforms Chebyshev in simpler grid-based maps such as scenario 0, 1, 3, 7, 9, and 10. These maps tend to favor straight-line, axis-aligned paths.

- The Chebyshev heuristic (h2) shows superior performance in more complex or dense environments, particularly those with diagonal paths or higher obstacle densities (e.g., scenario 2, 4, 5, 6, 8, and 11).

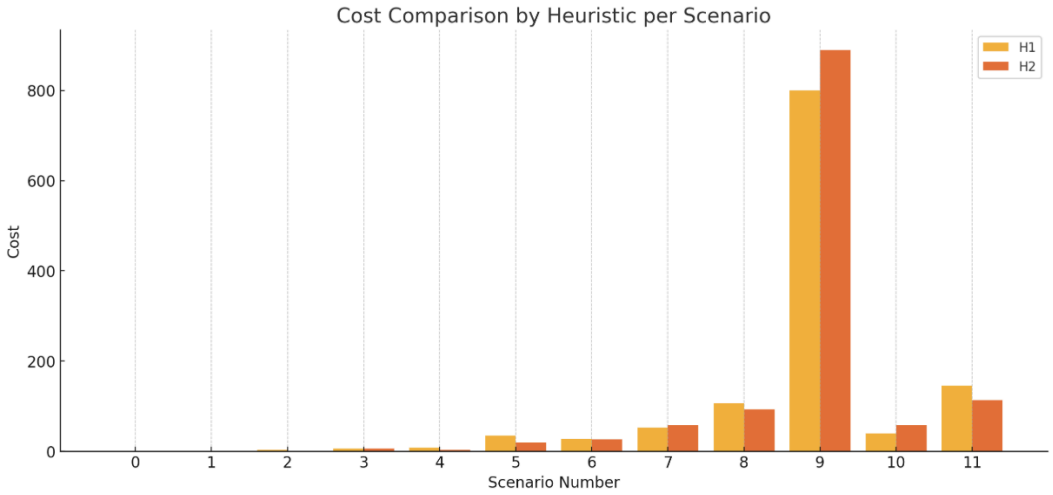
This suggests that h1 is well-suited for uniform, open environments, while h2 is more adaptive in intricate or diagonal-heavy layouts.

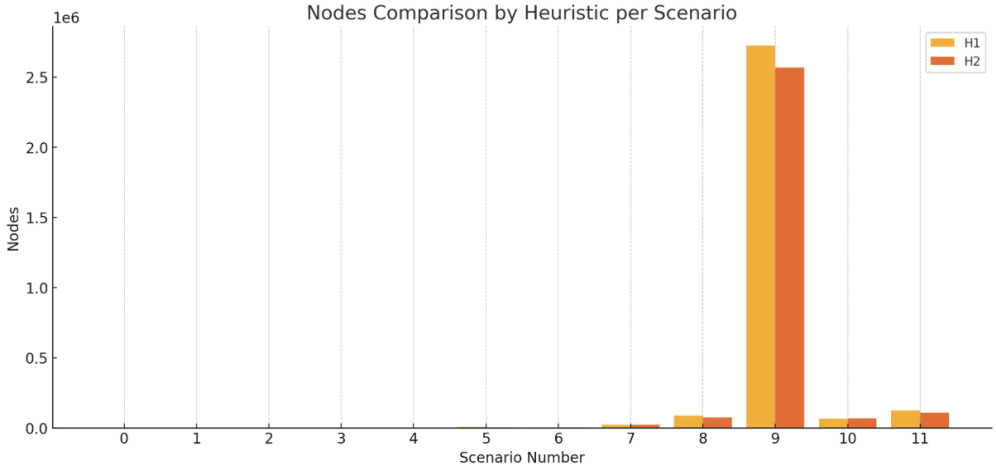
Tolerance Levels

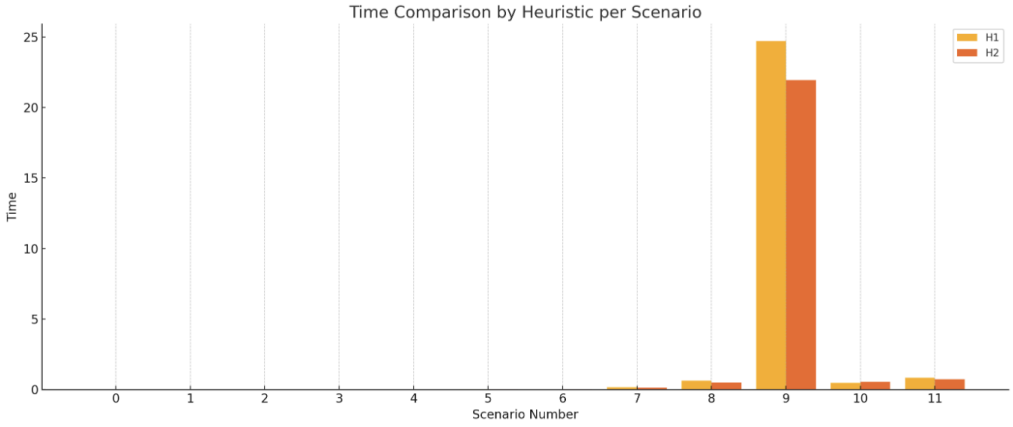
- Low tolerance values (e.g., 0.2) lead to highly optimal paths in terms of cost when paths exist, but often result in failure to find a path in complex maps.

- Moderate tolerance values (e.g., 0.4 to 0.6) offer a good balance between optimality and feasibility.

- High tolerance values (e.g., 0.8 to 1.0) are generally needed to resolve harder or more cluttered scenarios, though at the expense of higher costs and node expansions.







Using bar charts to compare cost, node expansions, and execution time across scenarios, we can clearly see:

- **Trade-offs** between low cost and computational expense.

- Certain scenarios like **scenario 5, 8, and 9** exhibit steep increases in node count and execution time, indicating high complexity and the need for more adaptive search strategies.

Custom Scenarios

Custom Scenario 1: Radar Density Adjustment

In this custom scenario, we modified scenario 8 by increasing the amount of radars in the map. This change was designed to simulate a more surveillance-intensive environment, where UAV must navigate through a denser detection field with fewer “safe zones”. All other parameters, including POIs and map size, remained unchanged.


```
{
    "scenario_10": {
        "max_lat": 37.29139325161781,
        "min_lat": 37.21979775354181,
        "max_lon": -115.78524417824534,
        "min_lon": -115.8885843284312,
        "W": 256,
        "H": 256,
        "n_radars": 10,
        "POIs": [[37.21979775, -115.83445377],
                 [37.21979775, -115.79016514],
                 [37.25048154, -115.79016514],
                 [37.28798394, -115.80164738],
                 [37.28798394, -115.83773441],
                 [37.28798394, -115.86397953]]
    }
},
```

Results & Analysis

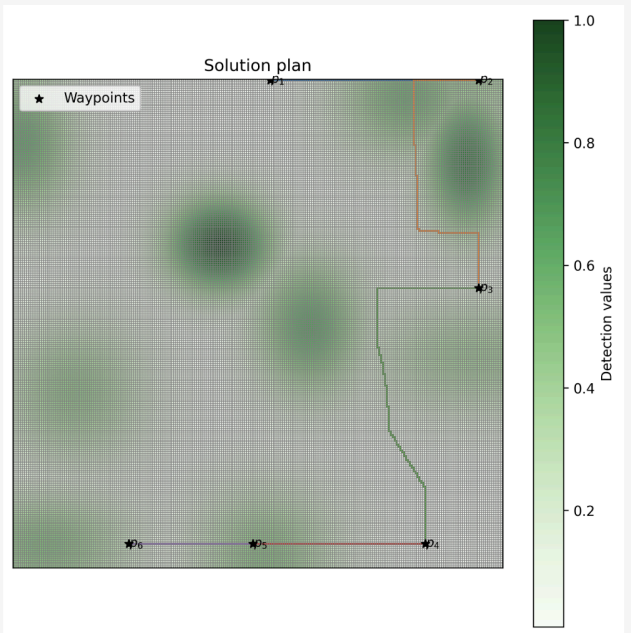


Figure 2: Pathfinding results from Custom Scenario 1 with increased radar density based on Scenario 8

As expected, the increased radar density led to significantly higher node expansions and overall path cost (*see figure 2*). The UAV had fewer viable routes to choose from, especially at lower tolerance levels. This scenario demonstrated the system's strong ability in constrained environments, while also showcasing a potential need for more adaptive heuristics or a dynamic tolerance strategy that is well suited to handle extremely cluttered airspace.

Custom Scenario 2: Increased Points of Interest (POIs)

This variation of Scenario 8 involved adding more POIs for the UAV to visit. While the radar configuration and tolerance levels remained unchanged, the route complexity increased as the UAV had to plan paths between a greater number of locations.

```
{
    "scenario_11": {
        "max_lat": 37.29139325161781,
        "min_lat": 37.21979775354181,
        "max_lon": -115.78524417824534,
        "min_lon": -115.8885843284312,
        "W": 256,
        "H": 256,
        "n_radars": 32,
        "POIs": [[37.21979775, -115.83445377],
                 [37.21979775, -115.79016514],
                 [37.25048154, -115.79016514],
                 [37.28798394, -115.80164738],
                 [37.28798394, -115.83773441],
                 [37.28798394, -115.86397953],
                 [37.23000000, -115.85000000],
                 [37.26000000, -115.81000000]]
    }
}
```


Results & Analysis

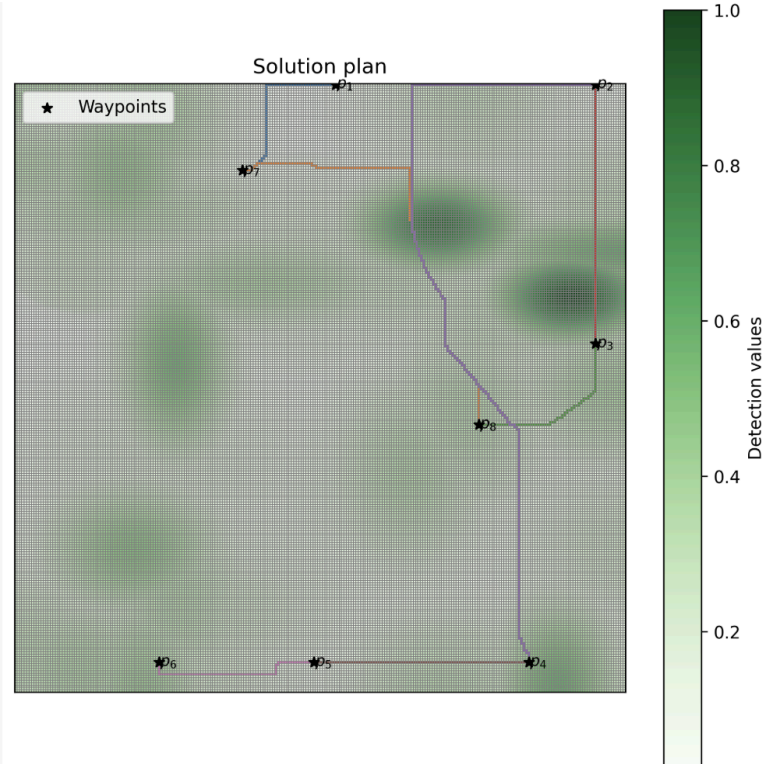


Figure 3: Performance results for Custom Scenario 2 with additional Points of Interest

Adding more POIs led to a more complex path planning challenge, as shown in *figure 3*. The total path cost increased severely, and execution time was greater as a

result of the additional A* computations between each POI pair. In spite of this, the system successfully generated valid paths using both heuristics, further proving its robustness in more demanding mission scenarios.

Proposed Improvements

Adaptive Heuristic Selection

We recommend implementing a **pre-processing phase** that analyses the map's properties, such as density, obstacle distribution, and potential for diagonal movement, to dynamically choose between h1 and h2. This would reduce trial-and-error and improve efficiency.

Tolerance Scheduling Strategy

Rather than running the algorithm with a fixed tolerance, a **progressive tolerance scheduling approach** should be adopted. The search can begin with a low tolerance (e.g., 0.2) and incrementally increase if no valid path is found. This maximises the likelihood of obtaining a low-cost path while minimising unnecessary computation.

Parallel Heuristic Execution

We propose executing **both heuristics in parallel**, particularly in time-sensitive applications. Once a valid path is found by either, the other can be terminated. This ensures that the faster heuristic is utilised without manually tuning for each scenario.

Cost-Efficiency Normalisation

Introduce a new metric such as **"Cost per Node Expanded"** to balance path quality against computation. This will be useful in scenarios where minimising resource usage is as important as the cost of the path itself.

Caching & Reuse of Subpaths

In larger maps (e.g., scenario 9), repeated sub-paths can be **memoised and reused** across runs, significantly cutting down search overhead. This is especially beneficial in environments with dynamic goals but static terrain.

Conclusion

From this project, we were successfully able to implement the search engine with our proposed heuristics and conclude that the heuristics' performance is dependent on the complexity of the search space. In the technical aspect, we deepened our understanding of graph search algorithms as we were given the opportunity to implement it in code. We also sharpened our teamwork skills as this project required close coordination from modelling the search problem to the heuristics and finally to the writing of code.

Our project faced the limitation of having limited computational power and as a result we were not able to test our search engine on highly complex scenarios such as having a large grid space.

Having developed this search engine in the course of this project, we are confident that it can be extended to incorporate 3-dimensional search space and radars have changing detection range. However, it must be noted that the current implementation can take some time due to the computational complexity required. Using the knowledge that we have learnt in this course, we researched online and found that genetic algorithms can be implemented together with the A*star search engine to optimise path-finding speeds (Mehmood *et al*, 2024), and it is something that can be looked at for future refinements.

In the real-world application of this type of search engine, it must be noted that the grid space can be very large, there may be environmental factors at play, and the radar detection can be dynamic, hence the search engine (if it is to be implemented for real-world use) will need to compute optimal paths in real-time and efficiently as aircraft fly at high speeds and points of interests can be of close proximity to one another. Our current implementation would be more appropriate for path planning,

provided that the radar detection capabilities are static and that the sky is clear without any cloud cover.

Use of Generative AI

Generative AI was used to generate our custom scenarios, scenario 10 and 11. The use of Generative AI helped to streamline our workflow as we could get the scenario data out quickly. The output was verified when we ran the scenarios into the code to see if there was any unexpected behaviour. Also, we checked the generated scenario data is of the proper JSON format and corrected any formatting errors prior to adding the scenario data into the scenarios.json file. The scenarios were generated using scenario 8 as the basis and we did a visual comparison of the generated scenario with scenario 8 to ensure that only the details we wanted changed were modified by generative AI.

References

Mehmood, D., Ali, A., Ali, S., Kulsoom, F., Chaudhry, H. N., & Haider, A. Z. U. (2024, January). A novel hybrid genetic and a-star algorithm for UAV path optimization. In *2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC)* (pp. 1-5). IEEE.