CS5011: P3 - Search - Flight route planner

Assignment: P3 - Practical 3

Deadline: 7th May 2024

Weighting: 34% of module mark

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

---

1 Objective

This practical aims to implement and evaluate a number of AI search algorithms applied to the task of a flight route planner.

2 Competencies

- Design, implement, and document AI search algorithms.
- Compare the performance of AI search algorithms.
- Test methods for optimising AI search.

3 Practical Requirements

3.1 Introduction

A flight plan is a document which indicates a flight planned route, including information such as departure, arrival points, choice of airways, weather forecasts etc. A flight route planner is a system that helps computing the best route for a flight, through the airspace system. There are numerous simulators for flight route planners that can be used for training or provided for aviation enthusiasts (e.g., RouteFinder[1], Flight Planner[2], etc...). A navigation route for a flight is normally given as a set of direction headings guiding the aircraft through the airspace[3].

3.2 The task

In this practical, we focus on a simplified version of a flight route planner to navigate in the constellation of planets Oedipus. In Oedipus, each planet is a 2d circular grid as shown in Figure 1. All planets have exactly 8 meridians (one every 45 degrees), which originates from a pole. Each planet has a fixed number of parallels N that indicates the size of that planet. For

[1] `http://rfinder.asalink.net/free/`

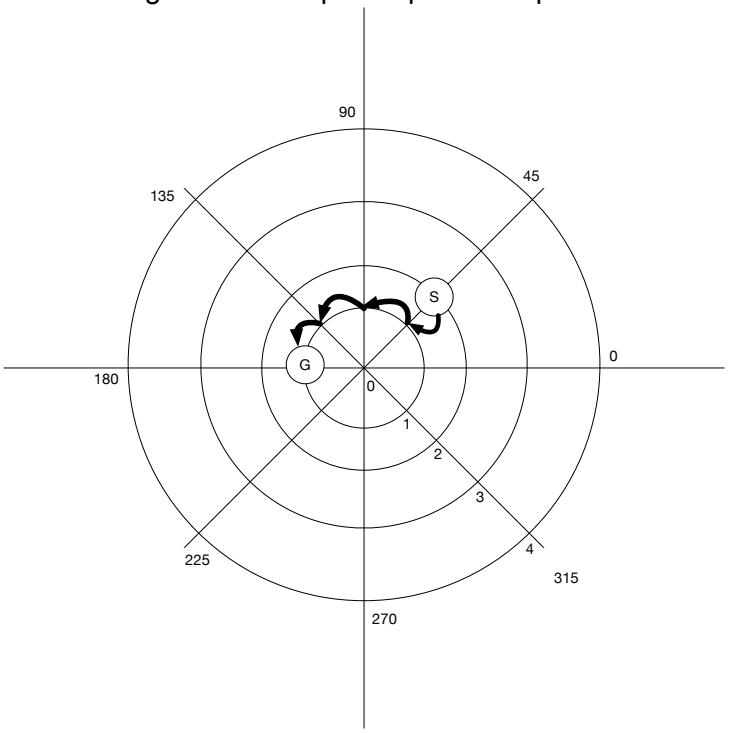
[2] `https://flightplandatabase.com/planner`

[3] Position and Navigation: `https://en.wikipedia.org/wiki/Polar_coordinate_system`

example, in Figure 1 we see Oedipus 5 which has 5 parallels (numbered from 0 at the pole to 4). Oedipus 8 has 8 parallels, Oedipus 10 has 10 parallels, etc.

Our route finder planner is to be operated by an agent whose aim is to find the best route from the departure airport (S) to the destination airport (G) for aircrafts navigating on the Oedipus airspace. Which algorithm should the agent use?

Figure 1: Example map for Oedipus 5



The flights navigating Oedipus airspace can move with these rules:

i. Polar coordinates are used to identify points in the grid, represented as a two-dimensional coordinate system in which each point on the 2d plane is identified by its distance from a centre/pole. A polar coordinate is a tuple $(d : angle)$, where d is the distance from the pole within $[0, N-1]$ and $angle$ is the angle of direction, one of $\{0, 45, 90, 135, 180, 225, 270, 315\}$

ii. Aircrafts start at the $S = (d_s : angle_s)$ point and terminate at the $G = (d_g : angle_g)$ point (e.g., in Figure 1 S=(2:45) and G=(1:180)).

iii. Aircrafts can only move along the grid, from point to point using the coordinate system, as shown by the marked path between S and G in Figure 1.

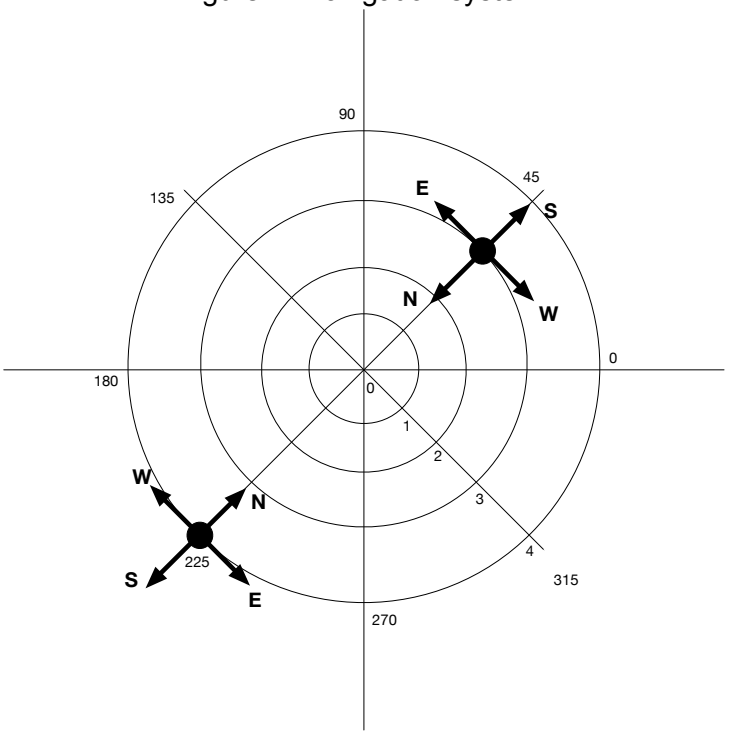
iv. Aircrafts can only move in one of the 4 directions along the grid, where going North means going to the centre (the Magnetic North), and we continue in a clockwise direction for East, South, and West as shown in Figure 2.

v. Coordinates at the pole, such as (0:0) are valid coordinates. Aircrafts, however, cannot reach or fly over the pole.

vi. Aircrafts cannot go beyond the last parallel.

vii. The distance between two parallels is 1, and the distance between two meridians is $1/8$ of the length of the circle at this parallel ($\frac{2*\pi*d}{8}$, please use `Math.PI` for π).

Figure 2: Navigation system



The aim of the practical is to implement and evaluate a set of AI search algorithms. There are two main criteria for evaluating search algorithms: the quality of the solution (e.g., the length/number of steps of the flight route), and efficiency represented here by the number of search states visited by the algorithm. In the appendix below, a number of start and end points, and planet types should be used to perform the evaluation and draw conclusions.

3.3 Part A: Uninformed Search

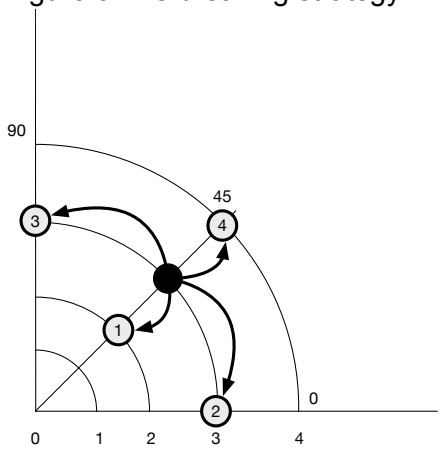
You are required to implement an agent that makes use of uninformed search to identify a route for an aircraft. Implement depth-first search (DFS) and breadth-first search (BFS) following the general algorithm for search. Please ensure that your implementation makes minimal changes between DFS and BFS. The first step in this part is to identify an adequate method to represent the polar grid system. For each algorithm, once the search has identified the goal, it must be able to construct the route it has found from the starting point to the goal, which should be represented by a set of polar coordinates. The algorithm should also avoid loops and redundant paths.

The tie-breaking strategy to be used for this part is the following: first we prioritise the lower d value, then the lower $angle$ value. For example see the priority for point $(3 : 45)$ indicated with a number inside of the neighbour points in Figure 3. Remember that the tie-breaking strategy is applied for all children of the same node (cf. Part B in the next section).

The output for this part must be the polar coordinates $(d : angle)$ in the nodes of the frontier at each step before polling a node, within squared brackets separated by a single comma, followed by three lines:

1. the path represented as a sequence of polar coordinates $(d : angle)$ with no spaces
2. the path cost (double) rounded at three decimal places (`String.format("%.3f",cost)`)
3. the number of nodes visited (int)

Figure 3: Tie breaking strategy



An example of the output for Figure 1 for BFS is formatted as follows:

```
[(2:45)]
[(1:45),(2:0),(2:90),(3:45)]
...
[(1:180),(2:225),(3:270),(4:315),(3:180),(4:135),(1:225)]
(2:45)(1:45)(1:90)(1:135)(1:180)
3.356
22
```

If there is no path, for example by trying to reach the pole, the output should be:

```
[(2:45)]
[(1:45),(2:0),(2:90),(3:45)]
...
[(4:225)]
fail
32
```

Notice that in this case, there should be no path printed but the path in step 1 above should be replaced by the word 'fail'. No other output should be printed.

3.4 Part B: Informed Search

You are required to implement a planner agent that makes use of informed search to identify a route for an aircraft. For many search problems, better results can be obtained by using heuristics to choose the state to explore next. Implement Best-first search (BestF), A*(AStar) and Simplified Memory-bounded A* (SMAStar) search, using the Euclidian distance as heuristic as discussed in lectures. Your implementation should follow the general algorithm for search where possible.

The same tie-breaking strategy is still used here. However, instead of applying the strategy on all children of the same node, this strategy is applied on all nodes with the same f_{cost} in the frontier. If there are two nodes with the same f_{cost}, first we prioritise the lower d value, then the lower $angle$ value. This extra constraint will ensure the frontier will store the nodes in the expected order. The output for this part must be formatted in the same way as part A with one modification to the frontier, where for each node we must print the coordinates followed by the f_{cost} rounded at 3 decimal places (`String.format("%.3f",f_cost)`). An example is provided below for A*.

```
[(2:45)2.798]
...
(2:45)(1:45)(1:90)(1:135)(1:180)
3.356
5
```

For SMA*, please use the size of the planet N as the default size for the memory. You may also investigate other sizes in your evaluation.

3.5 Part C: Advanced Search

It is strongly recommended that you ensure you have completed the requirements of other parts before attempting any of these requirements.

You may consider at most two additional functionalities for this part. Examples include:

1. Find out about another search algorithm or improvement for an existing algorithm, such as bidirectional search or search approximations for A*, implement it and evaluate it in comparison with the other approaches.

2. In addition to moving from a coordinate to the next, the agent might have to perform additional actions, such as changing altitude, speed etc. Extend the approaches in the previous parts to include the implementation of new types of actions and evaluate the algorithms under those conditions. Remember to consider well how to account for these actions in the heuristics for informed search.

3. The route finder agent is tasked with scheduling the route for the main aircraft but with two additional aircrafts where two wing-people are positioned. The wing-people's aircrafts are positioned on the east and west of the aircraft at the starting point and should arrive at the goal in the same position. The aircrafts move simultaneously of one position at a time but cannot be on the same coordinates at the same time. During the flight, the wing-people's aircrafts could also be heading ahead or behind the main aircraft but should stay at most one position in the grid away from the main aircraft. Adapt any of the previous algorithms developed to perform this task.

4. Suggest your own advanced requirements: for a significant advancement, you may apply your search algorithms to a different or an extended search problem, or identify some other search algorithm and propose its implementation to solve the route finder problem on an Oedipus planet.

4 Code Specification

4.1 Code Submission

The program must be written in Java and your implementation must compile and run without the use of an IDE. Your system should be compatible with the version of Java available on the School Lab Machines (Amazon Corretto 17 – JDK17). Please do *not* use libraries that implement search algorithms, but other libraries that are secondary to the objectives of the practical can be used. Your source code should be placed in a directory called **src/**. The code must run and produce outputs as described above.

Please note that code that does not adhere to these instructions may not be accepted.

[3] https://aws.amazon.com/corretto/

4.2 Starter Code

For this practical, you are given a single starter class (`P3main.java`) and a list of start and end points as shown in the appendix. `P3main.java` contains only example code for reading inputs and printing the required output. This class can be modified as required. Please ensure that the final format of input and output is as that specified.

4.3 Running

Your code should run using the following command:

```
java P3main <DFS|BFS|AStar|BestF|SMAStar|...> <N> <d_s:angle_s> <d_g:angle_g> [<param>]
```

For example for BFS in Figure 1 we will use:

```
java P3main BFS 5 2:45 1:180
```

Please include clear running instructions for any advanced functionalities of part C.

4.4 Automatic Testing

We provide a limited selection of stacscheck tests to help you check that your output is structured with the required format. They can be found at `/cs/studres/CS5011/Coursework/P3/Tests` and can be run with `stacscheck`.

In order to run the automated checker on your program, place your program in a directory in the School lab machines, ssh into one of the School computers running Linux if you are working remotely, then change the directory to your CS5011 P3 directory and execute the following command:

```
stacscheck /cs/studres/CS5011/Coursework/P3/Tests
```

Please note that these tests are not designed to check that your program works correctly. Please do test your system well and provide information on your own testing in the report.

5 Report

You are required to submit a report describing your submission in PDF with the structure and requirements presented in the additional document *CS5011_Report_Requirements* found on `studres`. The report includes 5 sections (Introduction, Design & Implementation, Testing, Evaluation, Bibliography) and has an advisory limit of 2000 words in total. Consider the following points in your report:

1. Describe the search problem components for the flight route finder.
2. Describe the implementation of the search strategy clearly, and the differences between the various algorithms in your implementation.
3. Evaluate the search algorithms. Comment on which algorithms are best in terms of the number of search states visited and which algorithms produce the best (shortest) routes on the basis of path cost.

6 Deliverables

A single ZIP file must be submitted electronically via MMS by the deadline. Submissions in any other format will be rejected.

Your ZIP file should contain:

1. A PDF report as discussed in Section 5
2. Your code as discussed in Section 4

7 Assessment Criteria

Marking will follow the guidelines given in the school student handbook. The following issues will be considered:

- Achieved requirements
- Quality of the solution provided
- Examples and testing
- Insights and analysis demonstrated in the report

Some guideline descriptors for this assignment are given below:

- For a mark of 8 to 10: the submission implements part of agent A able to find a path in at least one way, adequately documented or reported.
- For a mark of 11 to 13: the submission implements fully part A. The code submitted is of an acceptable standard and is tested appropriately, and the report describes clearly what was done, with good style.
- For a mark of 14 to 16: the submission implements parts A and two of the search algorithms in part B, with complete implementations of these at the higher end of this range. It contains clear and well-structured code that is tested well, and a clear report showing a good level of understanding of design and evaluation of search algorithms.
- For a mark of 17 to 18: the submission implements all agents of parts A and B, with complete implementations of these at the higher end of the range. It contains clear, well-designed code with comprehensive tests, and a clear and well-written report showing real insight into the design and evaluation of search algorithms.
- For a mark above 18: the submission completes all agents of parts A and B, and one or two (max) advanced agent functionalities of part C. The submission should demonstrate unusual clarity of implementation, together with an excellent and well-written report showing evidence of extensive understanding of design and evaluation of search algorithms.

8 Policies and Guidelines

Marking: See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

Lateness Penalty: The standard penalty for late submission applies (Scheme A: 1 mark per 24-hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

Good Academic Practice: The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Alice Toniolo, Nguyen Dang, Fahrurrozi Rahman

cs5011.staff@st-andrews.ac.uk

March 18, 2024

9 Appendix

Please evaluate your algorithm on the following paths:

ID	Size	Start	Goal
1	N=5	S=(2,0)	G=(2,135)
2	N=5	S=(1,180)	G=(4,180)
3	N=5	S=(1,315)	G=(4,45)
4	N=5	S=(3,90)	G=(0,0)
5	N=8	S=(1,135)	G=(5,315)
6	N=8	S=(4,0)	G=(7,90)
7	N=8	S=(6,270)	G=(0,45)
8	N=10	S=(9,225)	G=(2,45)
9	N=10	S=(2,45)	G=(9,225)
10	N=10	S=(7,270)	G=(7,90)