CS5011: Assignment 1 - Search - Robot-Assisted Evacuation

Assignment: A1 - Assignment 1

Deadline: 15th October 2018

Weighting: 20% 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 robot-assisted evacuation simulation in the event of a disaster.

2 Competencies

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

3 Practical Requirements

3.1 Introduction

When a country is hit by a natural disaster, such as an earthquake, timely response is critical. The first step involved in disaster response is rescue and evacuation. Among other tasks this involves evacuation of buildings where routes may be blocked by various obstacles and collapses. It is common during evacuations in emergency situations for crowds to increase speed and create bottlenecks and exit clogging. Autonomous guide robots have the potential to intervene in these situations by assisting crowds in evacuating a building, directing evacuees towards less crowded and unblocked exists. This also would avoid further danger to human rescue teams. Testbeds and simulators have been developed in recent years for robots and intelligent agents attempting to solve these problems. The RoboCup Rescue League[1] is a well known robot competitions for rescue operations. Other specific robot guides for evacuation have been developed for this purpose[2].

[1] `http://www.robocup.org/domains/2`

[2] See for example "E. Boukas, I. Kostavelis, A. Gasteratos, and G. C. Sirakoulis. *Robot guided crowd evacuation.* IEEE Transactions on Automation Science and Engineering, 12(2):739-751, 2015"

3.2 The task

For this practical, we consider the problem of an agent simulating a robot navigating a building badly affected by a wild fire. We refer to this agent as robot Alpha. The robot is equipped with a map of one building floor where the crowd is to be evacuated as shown in Figure 1. We assume that the robot can only move along the paths highlighted in the map, where intersection points between paths are identified with a lower case letter (e.g., *a,b,c,...*). The robot moves at once between two connected intersection points and can move on any direction on the path. For example, *(a,b)* indicates that Alpha moves from point *a* to point *b*. The robot has a starting point (e.g., Alpha starts in point *d*), which is the location of the crowd to be evacuated. The robot will guide the crowds towards the exit location (e.g., point *n*).

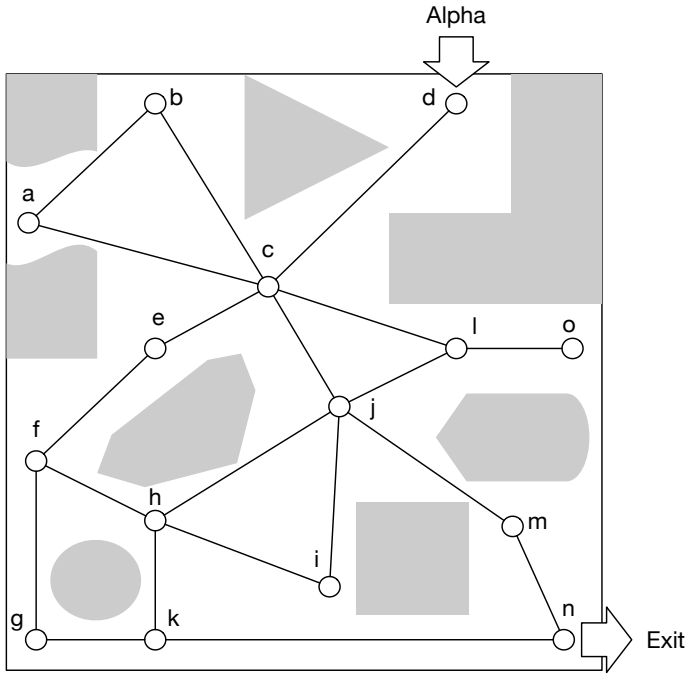

Figure 1: Example map for part 1

The task for robot Alpha is to find a route from the starting point to the exit. For convenience, we represent the map as an adjacency table, a square matrix with rows and columns representing points. Each cell contains an integer where:

- 10 represents no connection
- 5 represents that the two points are connected
- 0 represents the same point
- 2 represents the robot starting position (for robot Alpha at point *d*)
- 8 represents the exit position (e.g., *n*)

For example for the map in Figure 1, the table is:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
a	0	5	5	10	10	10	10	10	10	10	10	10	10	10	10
b	5	0	5	10	10	10	10	10	10	10	10	10	10	10	10
c	5	5	0	5	5	10	10	10	10	5	10	5	10	10	10
d	10	10	5	2	10	10	10	10	10	10	10	10	10	10	10
e	10	10	5	10	0	5	10	10	10	10	10	10	10	10	10
f	10	10	10	10	5	0	5	5	10	10	10	10	10	10	10
g	10	10	10	10	10	5	0	10	10	10	5	10	10	10	10
h	10	10	10	10	10	5	10	0	5	5	5	10	10	10	10
i	10	10	10	10	10	10	10	5	0	5	10	10	10	10	10
j	10	10	5	10	10	10	10	5	5	0	10	5	5	10	10
k	10	10	10	10	10	10	5	5	10	10	0	10	10	5	10
l	10	10	5	10	10	10	10	10	10	5	10	0	10	10	5
m	10	10	10	10	10	10	10	10	10	5	10	10	0	5	10
n	10	10	10	10	10	10	10	10	10	10	5	10	5	8	10
o	10	10	10	10	10	10	10	10	10	10	10	5	10	10	0

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 cost/length of the robot route), and efficiency represented here by the number of search states visited by the algorithm.

3.3 Part 1: Uninformed Search

Implement depth-first search (DFS), and breadth-first search (BFS) for the task described above. For each algorithm, once the robot Alpha reaches the exit, it must be able to construct the path it has found from the starting point. Assume that the distance between each connected point is 1 for this task. Your implementation should follow the general algorithm for search considering the order that new states are added to the list. The algorithm should also avoid loops and redundant paths. Please ensure that your implementation makes minimal changes between DFS and BFS. Your implementation should print out enough data to demonstrate that it works correctly. Make sure you print the current node, the list of states expanded, and the frontier at each step in the search, as well as the current solution details.

The file `maps1.txt` contains nine world configurations, use those for your initial implementation. These matrixes are represented as Java arrays filled with integers and represent the core of the adjacency table. Note that point labels are omitted (e.g., $a, b, ...$), you may use another array to map point labels to indexes for printing purposes. For example:

```
char[] points=new char[]{'a','b','c','d','e','f',
   'g','h','i','l','m','n','o'};
System.out.println(points[0]); //will print 'a'
```

Points to cover in your report:

1. Describe the state space, initial state and goal, successor functions, and actions in this search problem, path cost, and details of their respective implementation.

2. Describe the implementation of the search strategy clearly, and the differences between DFS and BFS in your implementation.

3. Describe your choice(s) of order to add new states to the list.

4. Describe how you tested your implementation.

5. Evaluate BFS and DFS for this problem by running them on the data provided. Which algorithm is best in terms of the number of search states visited? Which algorithm produces the best (shortest) routes on the basis of path cost?

6. Include example runs of both DFS and BFS on the given maps `maps1.txt`, and use these for comparing the two algorithms. You could include your own example maps, after testing with those given.

3.4 Part 2: Informed Search

For many search problems, better results can be obtained by using a heuristic to choose the state to explore next. For this task, we impose a 10x10 grid so that we know the location of each point in the grid, and we can establish how far each point is in reality from another point. Figure 2 represents this approach.

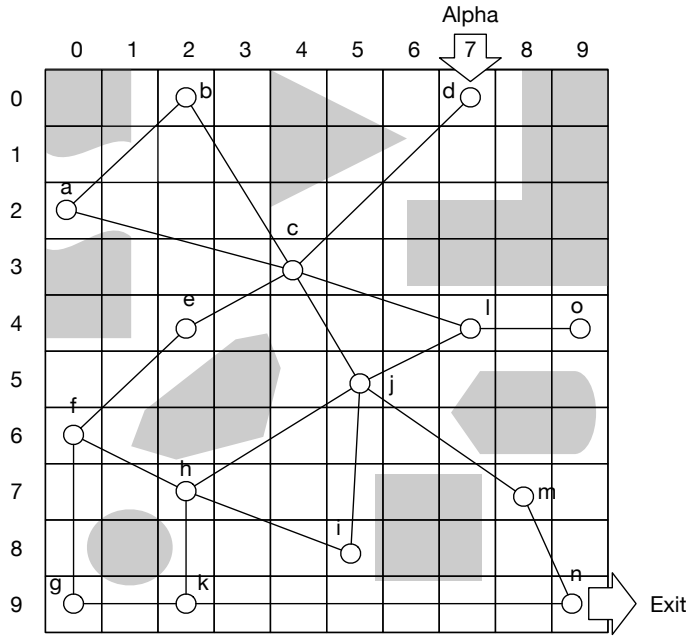

Figure 2: Example map for Part 2

In addition to `maps1.txt`, the file `maps2.txt` includes nine matrixes of integers corresponding to locations of the points in the worlds of maps1, for example $loc1$ corresponds to $world1$ etc. Each tuple in a matrix $loc1$ represents the location of the point at that index in its correspondent adjacency matrix. Assume that we have $loc1[q] = \{x, y\}$, q is the index of the point in the adjacency matrix, and x, y are the coordinates for this point. For example, point a in Figure 2 is represented as $\{0, 2\}$ and located at index 0, such that $loc1[0]$ will return an array $[0, 2]$, and similarly for all other points we obtain:

```
System.out.println(loc1[0][0]); //point 'a', coordinate 'x', prints 0
System.out.println(loc1[0][1]); //point 'a', coordinate 'y', prints 2
System.out.println(loc1[1][0]); //point 'b', coordinate 'x', prints 2
System.out.println(loc1[1][1]); //point 'b', coordinate 'y', prints 0
...
System.out.println(loc1[13][0]); //point 'n', coordinate 'x', prints 9
System.out.println(loc1[13][1]); //point 'n', coordinate 'y', prints 9
```

Implement best-first search (BestFS) and A* search, defining one heuristic to guide your search based on the distance of these points. As per Part 1, your implementation should follow the general algorithm for search.

Points to cover in your report:

1. Describe the heuristic chosen and its implementation.
2. Describe the search algorithms in your implementation.
3. Describe how you tested your implementation.
4. Evaluate BestFS and A* for this problem by running them on the data provided. Compare the two algorithms and discuss which of your algorithms is best in terms of the number of search states visited and path chosen. You could include your own example maps, after testing with those given.

3.5 Part 3: Extensions

It is strongly recommended that you ensure you have completed the Requirements in part 1 and part 2 before attempting any of these additional requirements. You may consider the following additional tasks:

1. Find out about bidirectional search, implement this approach and evaluate it in comparison with the results obtained in Part 1 or 2.
2. Select a different heuristic for the informed searches and evaluate your algorithms in comparison with the results obtained in Part 2.
3. Consider the following problem:
   Two agents simulating two robots navigate the building. We refer to these agents simply as robot Alpha and robot Bravo. Similar to the task above each robot is equipped with a map of one building floor to be evacuated as shown in Figure 1 and can only move along the paths given. Each robot has a starting point (e.g, Alpha starts in d represented by 2 in the adjacency table, Bravo starts in a represented by 4 in the adjacency table), where we assume that part of the crowd to be evacuated is located. Both robots will guide the crowds towards the same exit location (e.g., point n) as shown in Figure 3. The robots can only make a single move at a time simultaneously (e.g. Bravo moves (a, b) and Alpha (d, c) at the same time). However, for this task two robots cannot be on the same point at the same time with the exception of the exit point. This is so to avoid overcrowded locations.

   Adapt any of the algorithms developed in Part 1 or in Part 2 for this task.

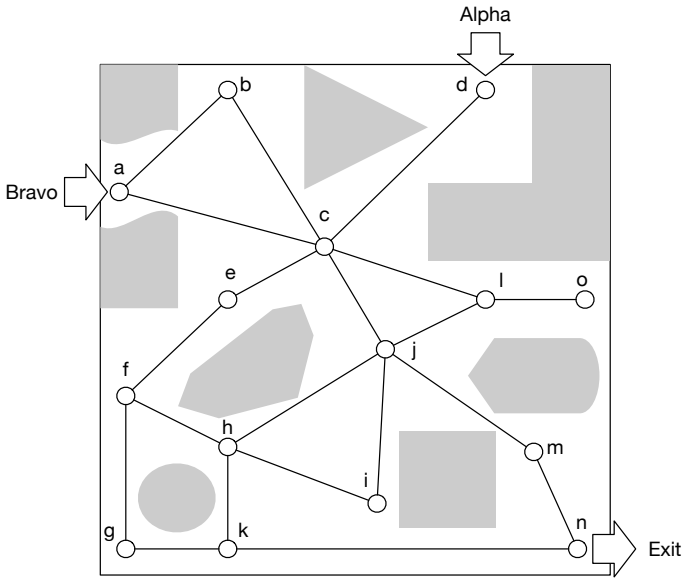

Figure 3: Example map for part 3

4. Suggest your own additional requirements: for a significant extension, 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 robot evacuation problem.

4 Code and Report Specifications

4.1 Code Submission and Running

The program must be written in Java and your implementation must compile and run on the School lab Machines. Please do not use libraries that implement search algorithms. A jar file is to be submitted for each part of the three parts attempted.
Name the jar file for Part 1 as "Search1.jar" and ensure it runs using:

```
java -jar Search1.jar [any param]
```

Name the jar file for Part 2 as "Search2.jar" and ensure it runs using:

```
java -jar Search2.jar [any param]
```

Name the jar file for Part 3 as "Search3.jar" and ensure it runs using:

```
java -jar Search3.jar [any param]
```

4.2 Report

You are required to submit a report describing your submission, with a limit of 3600 words excluding references. The report should include:

1. A list of the parts implemented and any extension attempted
2. If any of the functionalities is only partially working, ensure that this is discussed in your report
3. Literature Review: A short literature survey on applications of search algorithms
4. Design: A description and justification for the mechanisms you have implemented as part of your solution. Please discuss the items listed for each part of the practical that was attempted.
5. Examples and Testing: Examples of the main functionalities and your approach to testing the system. Make sure to include small working examples to show that your system is working correctly.
6. Evaluation: A critical analysis of the functionalities of your system and what can be improved.
7. Running: Include clear instructions on how to run your system
8. Bibliography: List all the references you cite in your report and code.

More details on points to be discussed under sections 4–6 are listed in the description of each part of this assignment. Please include a word count in your report and an estimate of the time taken to complete the assignment in hours.

5 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 4.2
2. One, two or three jars depending on the tasks achieved as discussed in Section 4.1
3. The source code of your implementation containing any non-standard libraries

6 Assessment Criteria

Marking will follow the guidelines given in the school student handbook (see link in the next section).

The following issues will be considered:

- Achieved requirements
- Quality of the solution provided

- Code quality
- Examples
- Insights and analysis demonstrated in the report

Some guidelines are as follows. For a mark up to the band 11-13 you must complete Part 1. For a mark up to the band 14-16 you must complete Part 1 and make an attempt to at least one informed search algorithm in Part 2. For a mark up to 17 you must complete Part 1 and Part 2 with a robust evaluation of the algorithms. To obtain marks above 17, in addition to Part 1 and 2, at least one extension should be completed as those suggested in Part 3. All parts are to be accompanied by a good insightful report covering all points and good design and implementation quality.

7 Policies and Guidelines

7.1 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`

7.2 Lateness Penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

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

7.3 Good Academic Practice

The University policy on Good Academic Practice applies:

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

Alice Toniolo
(a.toniolo@st-andrews.ac.uk)
September 25, 2018