COMP3411/9414 Artificial Intelligence
Summer session, 2019

Assignment 2 – Heuristics, Search and Learning

Due: Monday 04 February, 11:59pm

Marks: 20% of final assessment

Question 1: Search Algorithms for the 15-Puzzle (4 marks)

In this question you will construct a table showing the number of states expanded when the 15-puzzle is solved, from various starting positions, using four different searches:

(i) Uniform Cost Search (with Dijkstra's Algorithm)
(ii) Iterative Deepening Search
(iii) A*Search (using the Manhattan Distance heuristic)
(iv) Iterative Deepening A*Search

Go to the Open Learning Course Web Site, Module 3 Prolog Code: Path Search, scroll to the Activity at the bottom of the page and click on "`prolog search.zip`". The code is also available in WebCMS3 in Assignments /Assignment-2 - Prolog Code: Path Search. Unzip the file and change directory to prolog search, e.g.

```
unzip prolog_search.zip
cd prolog_search
```

Start prolog and load puzzle15.pl and ucsdijkstra.pl by typing

```
[puzzle15].
[ucsdijkstra].
```

Then invoke the search for the specified start10 position by typing

```
start10(Pos),solve(Pos,Sol,G,N),showsol(Sol).
```

When the answer comes back, just hit Enter/Return. This version of UCS uses Dijkstra's algorithm which is memory efficient, but is designed to return only one answer. Note that the length of the path is returned as G, and the total number of states expanded during the search is returned as N.

a) Draw up a table with four rows and five columns. Label the rows as UCS, IDS, A* and IDA*, and the columns as `start10, start12, start20, start30` and `start40.` Run each of the following algorithms on each of the 5 start states:

```
(i)     [ucsdijkstra]
(ii)    [ideepsearch]
(iii)   [astar]
(iv)    [idastar]
```

In each case, record in your table the number of nodes generated during the search. If the algorithm runs out of memory, just write "Mem" in your table. If the code runs for five minutes without producing output, terminate the process by typing Control-C and then "a", and write "Time" in your table. Note that you will need to re-start prolog each time you switch to a different search.

b) Briefly discuss the efficiency of these four algorithms (including both time and memory usage).

Question 2: Heuristic Path Search for 15-Puzzle (4 marks)

In this question you will be exploring an Iterative Deepening version of the Heuristic Path Search algorithm discussed in the Week 2 Tutorials (Module 4 - Tutorial: Heuristic Path Search). Draw up a table in the following format:

	start50		start60		start64	
IDA*	50	14642512	60	321252368	64	1209086782
1.2						
1.4						
1.6						
Greedy						

The top row of the table has been filled in for you (to save you from running some rather long computations).

a) Run `[greedy]` for `start50, start60` and `start64`, and record the values returned for `G` and `N` in the last row of your table (using the Manhattan Distance heuristic defined in `puzzle15.pl`).

b) Now copy `idastar.pl` to a new file heuristic.pl and modify the code of this new file so that it uses an Iterative Deepening version of the Heuristic Path Search algorithm discussed in the Week 2 Tutorial Exercise (Module 4 - Tutorial: Heuristic Path Search) with `w = 1.2` .

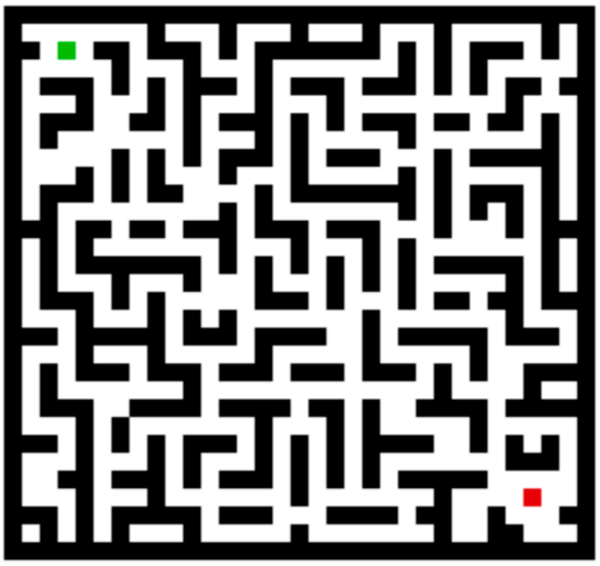
   In your submitted document, briefly show the section of code that was changed, and the replacement code.

c) Run `[heuristic]` on `start50, start60` and `start64` and record the values of `G` and `N` in your table. Now modify your code so that the value of `w is 1.4, 1.6` ; in each case, run the algorithm on the same three start states and record the values of G and N in your table.

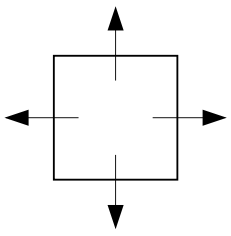
d) Briefly discuss the tradeoff between speed and quality of solution for these five algorithms.

Question 3: Maze Search Heuristics (4 marks)

Consider the problem of an agent moving around in a 2-dimensional maze, trying to get from its current position *(x,y)* to the Goal position (x_G,y_G) in as few moves as possible, avoiding obstacles along the way.



a) Assume that at each time step, the agent can move one unit either up, down, left or right, to the centre of an adjacent grid square:



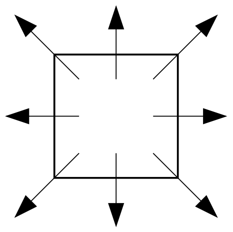
One admissible heuristic for this problem is the Straight-Line-Distance heuristic:

$$h_{\text{SLD}}(x, y, x_G, y_G) = \sqrt{(x - x_G)^2 + (y - y_G)^2}$$

However, this is not the best heuristic. Name another admissible heuristic which dominates the Straight-Line-Distance heuristic, and write the formula for it in the format:

$$h(x, y, x_G, y_G) = \quad \ldots$$

b) Now assume that at each time step, the agent can take one step either up, down, left, right or diagonally. When it moves **diagonally**, it travels to the centre of a diagonally neighboring grid square, but a diagonal step is still considered to have the same "cost" (i.e. one "move") as a horizontal or vertical step (like a King move in Chess).



(i) Assuming that the cost of a path is the total number of moves to reach the goal, is the Straight-Line-Distance heuristic still admissible? Explain why.

(ii) Is your heuristic from part (a) still admissible? Explain why.

(iii) Try to devise the best admissible heuristic you can for this problem, and write a formula for it in the format:

$$h(x, y, x_G, y_G) = \quad \ldots$$

Question 4: Decision trees (4 marks)

In this question you need to draw a decision tree for the problem of deciding whether to move forward at a road intersection, given that the light has just turned green. (Hint: think of real-world situations, e.g. pedestrians, cyclists, other cars…).

Briefly explain what you have presented with your decision tree. Can this knowledge be used by an agent? If yes, explain why and if no, why not.

Question 5: Decision trees (4 marks)

Suppose we generate a training set from a decision tree and then apply decision-tree learning to that training set.

a) Show working examples with 2 attributes, with 3 attributes and with 4 attributes.

b) Is it the case that the learning algorithm will eventually return the correct tree as the training-set size goes to infinity? Why or why not?

Submission

This assignment must be submitted electronically.
Put your zID and your name at the top of every page of your submission!

COMP3411 students should submit by typing

```
give cs3411 hw2 ...
```

(for example: `give cs3411 hw2 assignment2.pdf`)

COMP9414 students should submit by typing

```
give cs9414 hw2 ...
```

(for example: `give cs9414 hw2 assignment2.pdf`)

The give script will accept *.pdf *.txt *.doc *.rtf
Late submissions will incur a penalty of 15% per day, applied to the maximum mark.

Group submissions will not be allowed. By all means, discuss the assignment with your fellow students. But you must write (or type) your answers individually. Do NOT copy anyone else's assignment, or send your assignment to any other student.

Good luck!