CS 411 — Artificial Intelligence I — Spring 2017

Assignment 1

Department of Computer Science, University of Illinois at Chicago

Instructor: Xinhua Zhang

Due: Feb 13, 2017 by 5 PM (CST)
Out: Jan 27, 2017

Instructions

This is an **individual** assignment**.** Your mark will be out of 100, and it contributes 10% to your final course score. It consists of two parts: **written questions** and **programming**. The purpose of the programming part is to make you start experimenting with search algorithms. We will build on the implementation of the algorithms from the textbook (called the AIMA code at http://aima.cs.berkeley.edu/code.html).

What to submit: You should submit to **Blackboard** a single zipped file (.zip or .tar) which includes the following files:

1. For the written problems, provide your answers in a single Portable Document Format (PDF) file, and name it as Written.pdf. It can be either typed using WORD or latex, or scanned copies of handwritten submissions provided that the handwriting is neat and legible.

2. For the programming part, only typed submissions are accepted. Please submit:

a) **Code**: All the code you wrote – in such a form that the TA can run it. Please include plenty of comments. Put all code in one folder named Code/.
b) **ReadMe**: A ReadMe file that explains to your TA how to run your code. Name the file as ReadMe.
c) **Result files**: six plain text files that detail the solution found. See detailed requirement in Question 4.
d) **A Report** in PDF format with filename Program.pdf documenting your answers to the questions.

How to submit: You'll submit your homework online, via blackboard. Go to the class web site, and folder Assignments. The entry "Assignment 1" in this folder will have a link through which you can upload your homework. Submit a single zipped file named Surname_UIN.zip or Surname_UIN.tar, where Surname is your last name and UIN is your UIC UIN. Inside it, there should be two PDF files, one ReadMe, six result files, and one code folder as mentioned above.

Resubmission: The submission site will be open up to **5 PM of Feb 13, 2017**. You are allowed to resubmit (upload a new version) until the deadline. Grading will be conducted on the **last** version submitted **before** the deadline.

Programming Languages: You will need to invest a fair amount of time to understand how the AIMA code works, so that you can add to it and modify it. The book code comes in three languages: Java,

Python and LISP (see http://aima.cs.berkeley.edu/code.html). You can choose either **Java or Python**. Warning: the implementation that faithfully follows the current version of the book is the one in Java, the Python implementation is a bit less developed (but perfectly adequate for this homework).

Late Policy: A mark of 0 will be given if no submission has been made by the deadline (in the absence of medical evidence or other special permission).

Cheating and Plagiarism: All assignments must be done individually. Remember that plagiarism is a university offence and will be dealt with according to university procedures. Please refer to the corresponding UIC policies: http://dos.uic.edu/docs/Student%20Disciplinary%20Policy.pdf

Latex primer: http://ctan.mackichan.com/info/lshort/english/lshort.pdf

Part I. [50 points] Programming using search algorithms

The ring-shaped sliding tile puzzle consists of N red tiles (R), N green tiles (G), and two blank spaces (*). N > 1. They are placed on a ring with 2N+2 slots. The slots are numbered as in Figure 1. In the initial configuration, the tiles can be in any position, and an example for N=3 is shown in Figure 2.

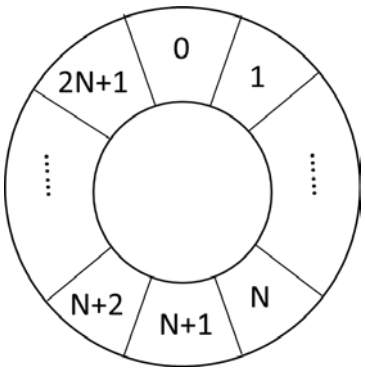


Figure 1: Numbering of slots

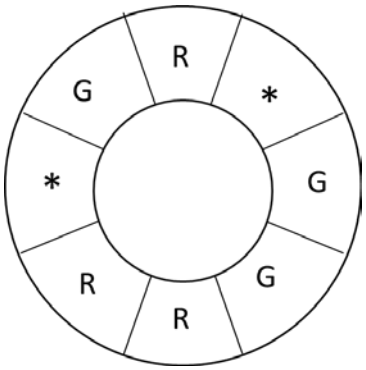
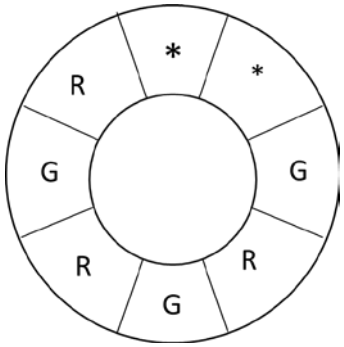


Figure 2: A random initial state for N=3

The goal state is any configuration where no two red and green tiles are next to each other. In other words, the state contains no consecutive pairs of red tiles or consecutive pairs of green tiles. See examples in Figure 3, and 4. Obviously rotating a goal state results in another goal state.



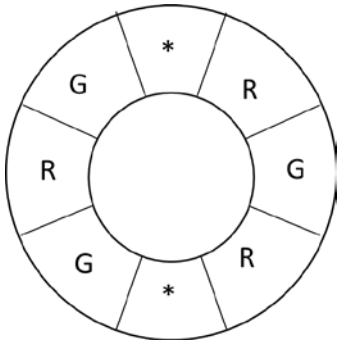


Figure 3: Two goal states for N=3

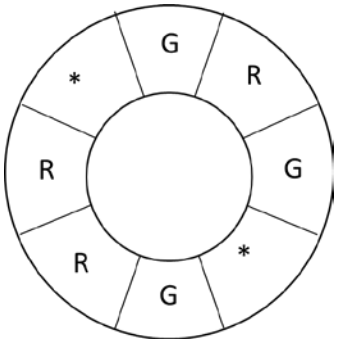


Figure 4: NOT A goal state for N=3

The puzzle has two kinds of legal moves with respective associated costs:

1. A tile may move into either of the blanks (*) provided that it is adjacent to the blank on the ring. This makes the tile's original location a blank. The cost of this move is 1.

2. A tile can jump over one other slot into either of the blanks (*), incurring a cost of 2. This also makes the tile's original location a blank. Note a tile can jump over a blank to get into another blank.

Questions

Unless otherwise specified, there is no assumption on N, except N > 1. Your answer should therefore be generically applicable for any value of N.

1. [6 pts] How are you going to represent the states and actions? You can write down the class definition in Java/Python, with comments on the meaning of each variable. For each action, explain how it maps an old state into a new one. Try to be concise and use symbols rather than 1,2,3… The **mod** operator might be useful here (the modulo operation that finds the remainder after division of one number by another).

2. [6 pts] Suppose we want to randomly initialize the configuration. What do you think is a good way to do so? Implement it as a function that takes a seed of random number generator. This will allow random experiments to be replicated.

3. [8 pts] Define an admissible heuristic $h(n)$ that can be used by the A* algorithm to solve this problem. Try to be as smart as possible (what does this mean?). For example, $h(n) = 0$ for all n is a bad heuristic. For grading, 4 points are given if it is admissible. The rest 4 points are computed by $\max\{0, 5 - x\}$, where x is the number of different heuristics proposed by your classmates that strictly dominate your heuristic. h_1 strictly dominates h_2 if and only if $h_1(n) \geq h_2(n)$ for all n, and $h_1(n) > h_2(n)$ for at least one n.

4. [15 pts] Fix N=3. Pick one uninformed search algorithm among the following: breadth-first (graph-search version), uniform-cost (graph-search version), depth-first (tree-search), iterative-deepening. Randomly generate **three** initial states and run both A* algorithm (graph-search version) and your picked algorithm on the three problems. In principle, you should use seeds 1, 2, 3, ... and skip a value if and only if it results in a state that is 0~2 steps away from a goal state.

(a) In your PDF report, show all the following results (3*2=6 sets of results), **and comment on them**.

i. the number of nodes expanded;

ii. the cost of the solution.

(b) Compare the results of A* and your picked algorithm. Also mention the value of the three seeds used to invoke the random initializer in Question 2, so that the TA can reproduce your results.

(c) For each of the 6 sets of test, dump to a plain text file the sequence of actions that lead to the returned solution. The filenames should be

Astar_1.dat, X_1.dat, Astar_2.dat, X_2.dat, Astar_3.dat, X_3.dat,

where X ∈{breadth, uniform, depth, iterative}, depending on the algorithm you pick.

The file content should employ the following format, exemplified with the initial state given by Figure 2:

R * G G R R * G
2 1 1
4 6 2
….

The first line encodes the initial configuration: the slots #0, #1, #2, #3, … have (respectively) a red tile, a blank, a green tile, a green tile, etc. From line 2, each line

x y z

means moving a tile from slot x to slot y, with the cost z. Slot y must be a blank before the move. In the above snippet based on Figure 2, it first moves the green tile from slot #2 to slot #1 with cost 1, then moves a red tile from slot #4 to slot #6 with cost 2 (jumping over a red tile in slot #5). There is a single space between all characters and numbers.

5. [15 pts] Conduct the following for each value of N from 3 to 10. Randomly generate 50 initial states. Run A* algorithm to find an optimal solution for all of them, and record the average number of (averaged over 100 initializations):

a) the path cost of the optimal solution (i.e. the smallest number of moves leading to a goal state);

b) the number of nodes generated by A* (not just the number of expanded node). This number might not be readily available from the functions of AIMA code base. If necessary, you can approximate it by the number of times that your heuristic function is invoked.

Plot your results in two figures like Figure 5 and 6 (curves are made up). The horizontal axis is N, and the vertical axis is the average numbers above. Comment on the result.

If it is too costly (in computation or space) to run for some values of N, mention it in your report and say how long you've waited or how many nodes are being kept in memory.

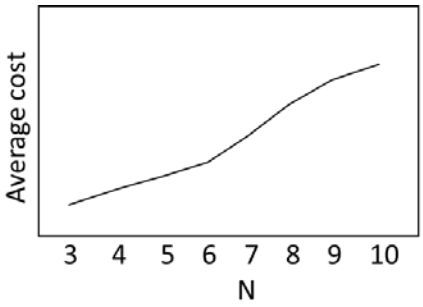


Figure 5: Avg path cost v.s. N

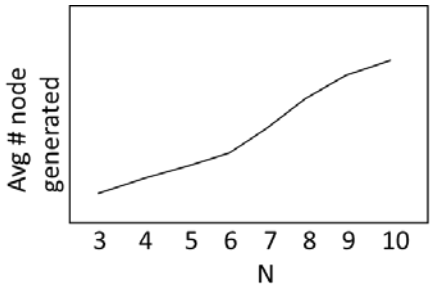


Figure 6: Avg # node generated v.s. N

Part II. [50 points] Written questions

6. [10 pts] A Mars rover has to leave the lander, collect rock samples from three places (in any order) and return to the lander every day. Assume that it has a navigation module that can take it directly to places of interest. So it has primitive actions

go-to-lander, go-to-rock-1, go-to-rock-2, and go-to-rock-3.

We know the time it takes to traverse between each pair of special locations. Our goal is to find a sequence of actions that will perform this task in the shortest amount of time.

(a) [3 pts] Formulate this problem as a problem-solving problem by specifying the search space, initial state, path-cost function, and goal test. Assume that the world dynamics are deterministic.

(b) [3 pts] Say what search technique would be most appropriate, and why. If your search technique requires a heuristic, give an appropriate one.

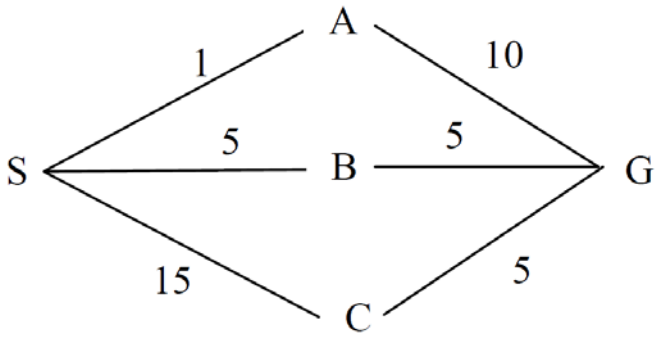
(c) [4 pts] Now assume that, in addition, the rover needs to be at a special communications location at 3PM. Any plan that misses a communications rendezvouz is unsatisfactory. So it now has primitive actions

go-to-lander, go-to-comm-location, communicate, go-to-rock-1, go-to-rock-2, and go-to-rock-3.

communicate can only be executed when the rover is at the communications location at 3 PM and executes a communications sequence that takes an hour.

How would you modify the search space, path-cost function, and/or goal-test to handle this additional requirement?

7. [10 pts] Consider the following route-finding problem:



Let S be the initial state and G be the goal state. The cost of each action is as indicated.

(a) [6 pts] Consider uniform-cost search implemented with graph-search. Give a trace of uniform-cost search, by showing the nodes in frontier and *explored set* at each iteration of the main loop.

(b) [4 pts] When A generates G which is a goal state with a path cost of 11, why doesn't the algorithm halt and return the search result now that the goal has been found? With your observation, discuss how uniform-cost search ensures that the shortest path solution is selected.

8. [10 pts]

(a) [7 pts] (AIMA Ex 3.23) Trace the operation of A* search (graph-search version) applied to the problem of getting to Bucharest from Lugoj using the straight-line distance heuristic (see Fig 3.22 of the textbook). Note A* instantiates the Uniform-cost search in Figure 3.14 with its own cost function. It can update the path-cost of nodes in the frontier if a lower path-cost is found. A* terminates only when a goal state is expanded (not generated).

What to write down: the sequence of nodes that the algorithm will consider, the update of frontier and explored set, and the f, g, and h score for each node on the frontier. The first three steps are done in the following table, and you just need to continue and complete the table. Add **notes** to explain

- When a node is generated, why is it not added to the frontier?
- When a node is generated, why is it used or not used to update the *f* value of another node in the frontier which has the same state?
- Why A* does not return immediately after finding a goal state?

You do not necessarily meet with all these scenarios in this particular problem.

Step & expanded node	Frontier (nodes are sorted by its *f* value)	Explored set	Notes
1.	L[0+244=244]		
2. L	M[70+241=311], T[111+329=440]	L	
3. M	D[145+242=387], T[111+329=440]	L, M	M has another child L, but L is not added to frontier because it's already in the explored set

(b) [3 pts] Figure 3.24 of AIMA showed a step-by-step breakdown of A* algorithm for Arad to Bucharest. Rewrite the steps in the form of the table above. Remember to add the abovementioned notes.

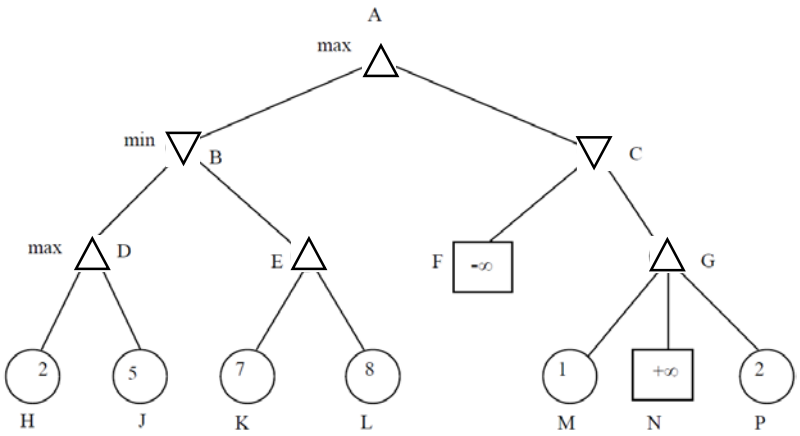
9. [10 pts]

(a) [5 pts] The following minimax search tree has heuristic evaluation function values with respect to the max player for all the leaf nodes, where square leaf nodes denote end of game with $+\infty$ representing that the max player wins the game and $-\infty$ representing that the min player is the winner.

i. Calculate the MINIMAX values of nodes A, B, C, D, E, and G.

ii. What is the next move of the max player from node A?

iii. Which is the target leaf node (H to P) that the max player hopes to reach?



(b) [5 pts] Suppose we use $\alpha - \beta$ pruning in the direction from left to right to prune the search tree in (a).

i. Which nodes are pruned by the procedure (don't forget leaf nodes H to P)? For example, in Figure 5.5 on page 168, the two dashed nodes (with dashed arcs) in (f) are pruned. If a whole subtree is pruned, e.g. the one rooted at B, then just mention node B, with no need of enumerating its descendants.

ii. With pruning, do we get the same answer in terms of the max player's next move and target leaf node?

10. [10 pts]
(a) [2 pts] Suppose we are running the AC-3(csp) algorithm (Figure 6.3 on page 209). Prove that if an arc (X_i, X_j) is **not** currently in the queue, then the current D_i and D_j must satisfy that X_i is arc-consistent with respect to X_j.

(b) [3 pts] In the AC-3(csp) algorithm, there is a line

for each X_k **in** X_i.NEIGHBORS - {X_j} **do**

Why can X_j be skipped from being added to the queue (noting that the algorithm has just checked the arc consistency of (X_i, X_j), not (X_j, X_i))? Justify your answer. That is, if you answer yes, then give a proof. If you answer no, then give an example where adding this "if condition" will cause some problem.

(c) [5 pts] Consider the 4-queens problem on a 4 x 4 chess board. Suppose the leftmost column is column 1, and the topmost row is row 1. Let Q_i denote the row number of the queen in column i, i = 1, 2, 3, 4. Assume that variables are assigned in the order Q_1, Q_2, Q_3, Q_4, and the domain values of Q_i are tried in the order 1, 2, 3, 4. Show a trace of the backtracking algorithm with forward checking to solve the 4-queens problem. The first step is done below, with a cross representing the positions in the other columns ruled out by the first queen. You just need to draw the next steps using the same representation.