CS 188 Fall 2020 Introduction to Artificial Intelligence **Exam Prep 1 Solutions**

Q1. Search

For this problem, assume that all of our search algorithms use tree search, unless specified otherwise.

(a) For each algorithm below, indicate whether the path returned after the modification to the search tree is guaranteed to be identical to the unmodified algorithm. Assume all edge weights are non-negative before modifications.

(i) Adding additional cost $c > 0$ to every edge weight.

	Yes	No
BFS	●	○
DFS	●	○
UCS	○	●

(ii) Multiplying a constant $w > 0$ to every edge weight.

	Yes	No
BFS	●	○
DFS	●	○
UCS	●	○

(b) **For part (b)**, two search algorithms are defined to be **equivalent** if and only if they expand the same states in the same order and return the same path. **Assume all graphs are directed and acyclic.**

(i) Assume we have access to costs c_{ij} that make running UCS algorithm with these costs c_{ij} equivalent to running BFS. How can we construct new costs c'_{ij} such that running UCS with these costs is equivalent to running DFS?

○ $c'_{ij} = 0$ ○ $c'_{ij} = 1$ ○ $c'_{ij} = c_{ij}$

● $c'_{ij} = -c_{ij}$ ○ $c'_{ij} = c_{ij} + \alpha$ ○ Not possible

Breadth-First Search expands the node at the shallowest depth first. Assigning a constant positive weight to all edges allows to weigh the nodes by their depth in the search tree. Depth-First Search expands the nodes which were most recently added to the fringe first. Assigning a constant negative weight to all edges essentially allows to reduce the value of the most recently nodes by that constant, making them the nodes with the minimum value in the fringe when using uniform cost search. Hence, we can construct new costs c'_{ij} by flipping the sign of the original costs c_{ij}.

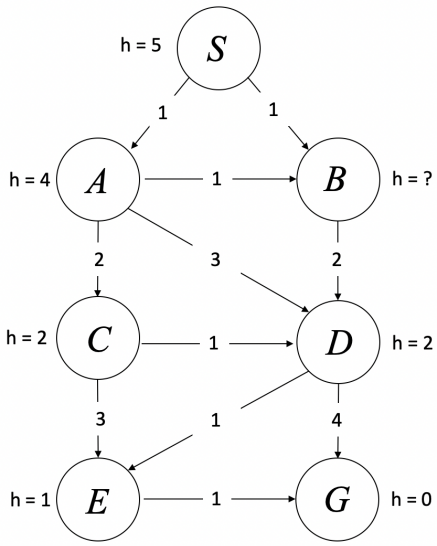
(ii) Given edge weight $c_{ij} = h(j) - h(i)$, where $h(n)$ is the value of the heuristic function at node n, running UCS on this graph is equivalent to running which of the following algorithm on the same graph?

○ DFS ○ BFS ○ Iterative Deepening

● Greedy ○ A* ○ None of the above.

Greedy Search expands the node with the lowest heuristic function value $h(n)$. If $c_{ij} = h(j) - h(i)$, then the cost of a node n on the fringe when running uniform-cost search will be $\sum_{ij} c_{ij} = h(1) - h(start) + h(2) - h(1) + ... + h(n) - h(n-1) = h(n) - h(\text{start})$. As $h(\text{start})$ is a common constant subtracted from the cost of all nodes on the fringe, the relative ordering of the nodes on the fringe is still determined by $h(n)$, i.e. their heuristic values.

(c) Consider the following graph. $h(n)$ denotes the heuristic function evaluated at node n.



(i) Given that G is the goal node, and heuristic values are fixed for all nodes other than B, for which values of $h(B)$ will A* tree search be guaranteed to return the optimal path? Fill in the lower and upper bounds or select "impossible."

0 $\leq h(B) \leq$ 4 ○ Impossible

An admissible heuristic is sufficient for A* tree search to be optimal. Recall the constraint for an admissible heuristic: $\forall n, 0 \leq h(n) \leq h^*(n)$. Since all other nodes satisfy the constraint above, we just need to enforce the constraint on node B, so that $0 \leq h(B) \leq h^*(B) = 4$.

(ii) With the heuristic values fixed for all nodes other than B, for which values of $h(B)$ will A* graph search be guaranteed to return the optimal path? Either fill in the lower and upper bound or select "impossible."

4 $\leq h(B) \leq$ 4 ○ Impossible

We need a consistent heuristic for A* tree search to be optimal. Recall the constraint for a consistent heuristic: $\forall A, C \quad h(A) - h(C) \leq \text{cost}(A, C)$. Since all other nodes satisfy the constraint above, we just need to enforce the constraint on node B. Enforcing consistency along $S - B$, gives $h(B) \geq 4$, $A - B$ gives $h(B) \geq 3$, $B - D$ gives $h(B) \leq 4$.

Q2. SpongeBob and Pacman (Search Formulation)

Recall that in Midterm 1, Pacman bought a car, was speeding in Pac-City, and the SpongeBob wasn't able to catch him. Now Pacman has run out of gas, his car has stopped, and he is currently hiding out at an undisclosed location.

In this problem, you are on the SpongeBob side, tryin' to catch Pacman!

There are still p SpongeBob cars in the Pac-city of dimension m by n. In this problem, **all SpongeBob cars can move, with two distinct integer controls: throttle and steering, but Pacman has to stay stationary**. Once one SpongeBob car takes an action which lands him in the same grid as Pacman, Pacman will be arrested and the game ends.

Throttle: $t_i \in \{1, 0, -1\}$, corresponding to {Gas, Coast, Brake}. This controls the **speed** of the car by determining its acceleration. The integer chosen here will be added to his velocity for the next state. For example, if a SpongeBob car is currently driving at 5 grid/s and chooses Gas (1) he will be traveling at 6 grid/s in the next turn.
Steering: $s_i \in \{1, 0, -1\}$, corresponding to {Turn Left, Go Straight, Turn Right}. This controls the **direction** of the car. For example, if a SpongeBob car is facing North and chooses Turn Left, it will be facing West in the next turn.

(a) Suppose you can **only control 1 SpongeBob car**, and have absolutely no information about the remainder of $p - 1$ SpongeBob cars, or where Pacman stopped to hide. Also, the SpongeBob cars can travel up to 6 grid/s so $0 \leq v \leq 6$ at all times.

(i) What is the **tightest upper bound** on the size of state space, if your goal is to use search to plan a sequence of actions that guarantees Pacman is caught, no matter where Pacman is hiding, or what actions other SpongeBob cars take. Please note that your state space representation must be able to represent **all** states in the search space.

$28mn * 2^{mn}$

There are mn positions in total. At each legal position, there are 7 possible speeds (0, 1, 2, 3, 4, 5, 6), so a factor of 7 is multiplied. In addition, since change of direction depends on orientation of the car, another factor of 4 is multiplied.
The only sequence of actions which guarantees that Pacman is caught is a sequence of actions which visits every location. Thus, we also need to a list of $m * n$ boolean to keep track of whether we have visited a specific grid location, and that is another factor of 2^{mn}

(ii) What is the maximum branching factor? Your answer may contain integers, m, n.

9

3 possible throttle inputs, and 3 possible steering inputs. The list of boolean does not affect the branching factor.

(iii) Which algorithm(s) is/are guaranteed to return a path passing through all grid locations on the grid, if one exists?

- ☐ Depth First Tree Search
- ■ Breadth First Tree Search
- ■ Depth First Graph Search
- ■ Breadth First Graph Search

Please note the list of boolean is in the state space representation, so we can revisit the same grid position if we have to.

(iv) Is Breadth First Graph Search guaranteed to return the path with the shortest number of **time steps**, if one exists?

- ● Yes
- ○ No

The Breadth First Graph Search is guranteed to return the path with the shortest amount of time, because each edge here represent moving for 1 unit of time.

(b) Now let's suppose you can control **all** p SpongeBob cars at the same time (and know all their locations), but you still have no information about where Pacman stopped to hide

(i) Now, you still want to search a sequence of actions such that the paths of p SpongeBob car combined **pass through all** $m * n$ **grid locations**. Suppose the size of the state space in part (a) was N_1, and the size of the state space in this part is N_p. Please select the correct relationship between N_p and N_1

○ $N_p = p * N_1$ ○ $N_p = p^{N_1}$ ○ $N_p = (N_1)^p$ ● None of the above

In this question, we only need one boolean list of size mn to keep track of whether we have visited a specific grid location. So the size of the state space is bounded by $N_p = (28mn)^p 2^{mn}$, which is none of the above.

(ii) Suppose the maximum branching factor in part (a) was b_1, and the maximum branching factor in this part is b_p. Please select the correct relationship between b_p and b_1

○ $b_p = p * b_1$ ○ $b_p = p^{b_1}$ ● $b_p = (b_1)^p$ ○ None of the above

For example, the case of $p = 2$ means two cars can do all 9 options, so the branching factor is $9^2 = 81$. In general, the branching factor is then b_1^p.