CS 188 Fall 2020 — Introduction to Artificial Intelligence — Written HW 1

Due: Wednesday 09/16/2020 at 10:59pm (submit via Gradescope).

Policy: Can be solved in groups (acknowledge collaborators) but must be written up individually

Submission: Your submission should be a PDF that matches this template. Each page of the PDF should align with the corresponding page of the template (page 1 has name/collaborators, question 1 begins on page 2, etc.). **Do not reorder, split, combine, or add extra pages.** The intention is that you print out the template, write on the page in pen/pencil, and then scan or take pictures of the pages to make your submission. You may also fill out this template digitally (e.g. using a tablet.)

First name	
Last name	
SID	
Collaborators	

For staff use only:

Q1.	Probability Review	/20
Q2.	Uninformed Search and Heuristics	/50
Q3.	The Camping Scheduling Problem	/30
	Total	/100

Q1. [20 pts] Probability Review

This question is meant to review the part of probability prerequisite. It might be helpful to look into resources under **General Recourses** at https://piazza.com/berkeley/fall2020/cs188/resources.

Let A, B, C, D be four random variables.

(a) What is the smallest set of independence or conditional independence relationships we need to assume for the following scenarios?

(i) [1 pt] $P(A, B) = P(A|B)P(B)$

(ii) [1 pt] $P(A, B) = P(A)P(B)$

(iii) [1 pt] $P(A, B, C) = P(A|B)P(B|C)P(C)$

(iv) [2 pts] $P(A, B, C) = P(A)P(B|C)P(C)$

(v) [2 pts] $P(A, B, C) = P(A)P(B)P(C)$

(b) Simplify the following expressions to one probability expression. Please show your work.

(i) [2 pts] $\frac{P(A,B)}{\sum_a P(a,B)}$

(ii) [2 pts] $\frac{P(A,B,C,D)}{\sum_a \sum_b P(a,b,C,D)}$

(iii) [2 pts] $\frac{P(A,C,D|B)}{P(C,D|B)}$

(iv) [3 pts] $\frac{P(A|B)}{\sum_c P(c|B)}$

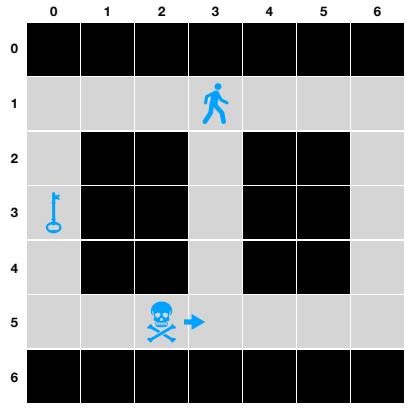
(v) [4 pts] $\frac{\sum_b P(A,b|C)P(D|A,b,C)}{P(A|B,C)}$, given $A \perp\!\!\!\perp B|C$

Q2. [50 pts] Uninformed Search and Heuristics

Consider the following simplified version of the classic Atari video game, *Montezuma's Revenge*: It is played on the board illustrated below. An agent (represented by the person icon in cell (1,3)) wishes to grab the key (in cell (3,0)). A skull starts in cell (5,2) and moves to the right by one cell after each action is executed until it ends up in the rightmost cell, at which point it starts moving to the left, and repeats this pattern back and forth.

The agent can be facing either left or right. There are 10 possible actions for the agent: 2 turning actions ($turn\_left, turn\_right$) and 8 moving actions ($left, right, up, down, left\_up, left\_down$, $right\_up, right\_down$). The agent can move up or down while facing either direction, but can move sideways or diagonally only if facing in that direction. For example, if the agent is facing right but tries to move $left\_up$, the agent will not move and nothing will happen. Furthermore, if the agent is already facing $left$ and a $turn\_left$ action is taken, nothing happens.

Lastly, the agent cannot move into a cell **currently occupied** by the skull, or a wall.



(a) Answer the following questions for the Montezuma's revenge board above:

(i) [4 pts] Let N be the number of possible cell locations that the agent can be in, and let M be the number of possible cell locations that the skull can be in. Recall that for "pacman pathing", the representation of the state was (x, y) where x was the row and y was the column of pacman's position.

Describe a representation of a state in the state space for this game and give an expression for the size of the state space.

Representation of the state space:

Size of the state space:

Explanation of each term in the size of the state space:

(ii) [4 pts] Please fill in the following pseudocode for the **getSuccessor** function for this game:

---

procedure GETSUCCESSOR(*state*)
    successors ← empty *list*
    for *action* ∈ [ ] **do**
        if *action* ∈ {*left*, *left_up*, *left_down*} and *state.facing_direction* == [ ] **then**
            continue
        if *action* ∈ {*right*, *right_up*, *right_down*} and *state.facing_direction* == [ ] **then**
            continue
        if *action* ∈ {*turn_right*} and *state.facing_direction* == [ ] **then**
            continue
        if *action* ∈ {*turn_left*} and *state.facing_direction* == [ ] **then**
            continue
        next_state ← *state.apply_action*(*action*)
        if *next_state* not out of bound and *next_state* [ ]
and *next_state* [ ] **then**
            successors.append(*next_state*)
    return *successors*

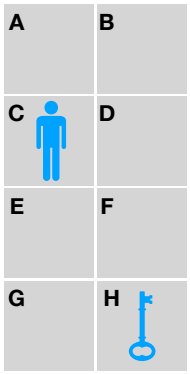
(iii) [2 pts] What is the goal test?

(b) Now, consider a simplified version of the board below, which has **no skull** and **no facing-direction for the agent** (i.e., the agent can move in any of the 8 directions as long as it remains in the board). For the three following graph search algorithms, perform the search procedure yourself (**please show your work**) and provide answers to the questions below regarding the nodes expanded during the search as well as the final path found by the algorithm.

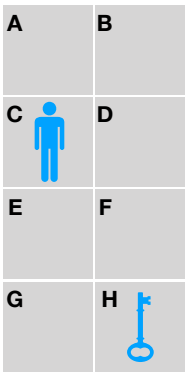
On this board, assume that a diagonal move has a cost of 3, whereas moving left, right, up, or down has a cost of 1. Do notice the difference in costs, and recall which algorithms use this cost information and which algorithms do not.

Remember that the search procedure should begin at the agent's starting position (C). To break ties when adding nodes of equal cost to the fringe, follow alphabetical order.

Finally, when listing the order/number of nodes expanded, do not include nodes which are taken off the fringe but discarded immediately due to already having been visited.



(i) [3 pts] **Breadth first graph search**
Fringe Data structure: queue
Recall that BFS computes the smallest number of steps, $b(v)$, taken to get to a node v from the start node.



Order of nodes expanded:

Number of nodes expanded:

Path returned:

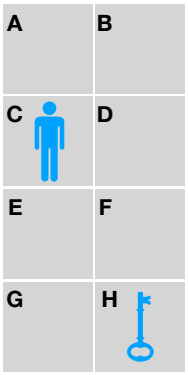
Length of path:

Cost of path:

What is $b(A), b(B), \ldots b(H)$?

State s	A	B	C	D	E	F	G	H
$b(s)$								

(ii) [3 pts] **Uniform cost graph search**
Fringe data structure: priority queue (make sure you update/reorder the whole PQ after each addition)
Recall that UCS keeps track of the lowest cost, $c(v)$, to get from the start node to the node v.



Order of nodes expanded:

Number of nodes expanded:

Path returned:

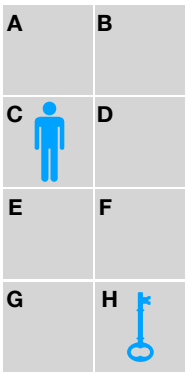
Length of path:

Cost of path:

What is $c(A), c(B), \ldots, c(H)$?

State s	A	B	C	D	E	F	G	H
$c(s)$								

(iii) [4 pts] **A* graph search (with Manhattan distance to the goal as the heuristic)**
Fringe data structure: priority queue (make sure you update/reorder the whole PQ after each addition)
Recall that A* computes $f(v)$ for the nodes v that it expands, with $f(v) = c(v) + h(v)$ where $c(v)$ is the lowest cost to reach v from the start node and $h(v)$ is an estimate of the distance from v to the goal.



Order of nodes expanded during the search:

Number of nodes expanded during the search:

Path returned by the search:

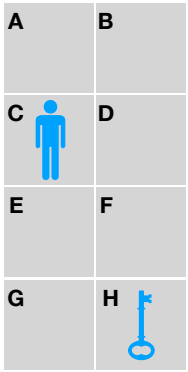
Length of path returned by the search:

Cost of path returned by the search:

What is $f(A), f(B), \ldots, f(H)$? Note that here, we are asking for the true $f(v)$ values as dictated by the definition, which is the value populated by the search algorithm only if it were to expand every node. This particular search problem doesn't end up expanding all nodes, so the $f(v)$ estimate maintained by the algorithm on the queue is not the true $f(v)$ value that we're asking for. Hint: you can fill out these values directly by looking at the board.

State s	A	B	C	D	E	F	G	H
$f(s)$								

(c) [10 pts] Given your answers above, what are the qualitative differences between the results achieved by BFS, UCS, and A*? Which one finds the shortest path (in number of steps)? Which one finds the optimal path (in cost)?



(d) For the same board and setting as part (b), give an example for each of the following types of heuristics. Please briefly explain why the example you chose satisfies the requested properties.

(i) [3 pts] Admissible and consistent.
Note: You can use a heuristic that we have frequently used in this class, or you can just assign any numbers that qualify as an admissible and consistent heuristic.

State s	A	B	C	D	E	F	G	H
Heuristic $h(s)$								

Explanation:

(ii) [4 pts] Admissible but inconsistent

State s	A	B	C	D	E	F	G	H
Heuristic $h(s)$								

Explanation:

(iii) [3 pts] Inadmissible and inconsistent

State s	A	B	C	D	E	F	G	H
Heuristic $h(s)$								

Explanation:

(e) [10 pts]

In the previous questions, perhaps you used "relaxed" heuristics; that is, you estimated the true cost of something by evaluating the cost for a simpler version of the problem (which is easier to compute). For example, using euclidean distance would be a "relaxed" heuristic to estimate the length of the shortest path from Arad to Bucharest in Romania.

Formally, we will define a **relaxed heuristic** as a function on a state that returns a value that is always admissible and consistent. In this problem, we will consider two changes ("skull" and "teleporation") to the board/game above, and we will reason about the effect of these changes on the consistency of heuristics.

(i) [5 pts] For this new version of the game, your friend Nancy suggests taking the old game setting from part (b) and now adding the ability for the agent to perform a maximum of 1 "teleportation" action during the game. That is, on one of the agent's moves, it can choose to jump from its current state to any other non-goal state on the board.

1. How does this new teleportation ability change the state space of the game from part (b), which was (x, y)? Does anything need to be added or removed?

2. Nancy argues that in this new game, at least one previously consistent heuristic can become inconsistent. Is Nancy right?

◯ Yes, and I will give an example below.
◯ No, and I will provide a proof below.

Note: we define heuristics for this problem as being a function of **only** the cell location: They cannot incorporate anything that did not exist in the old version of the game that we are comparing to.

If you believe Nancy is right, give an example of a heuristic that used to be consistent in the old game but is no longer consistent in this new game. Make sure to explain why it is no longer consistent (perhaps with a drawing of a board state and an explanation).

State	A	B	C	D	E	F	G	H
$h(s)$								

If you believe Nancy is wrong, provide an argument for why a heuristic that was consistent in the old game must also remain consistent in this new game. Be specific about your reasoning and use mathematical quantities such as heuristic costs of states $h(a)$ and true costs of transitions $c(ab)$.

(ii) [5 pts] For this new version of the board, your friend Ethan suggests adding the skull back to the old board setting from part (b), and having the skull move back and forth between the cells E and F.

1. How does the presence of this skull change the state space of the game from part (b), which was (x, y)? Does anything need to be added or removed?

2. Ethan argues that in this new board, at least one previously consistent heuristic can become inconsistent. Is Ethan right?

◯ Yes, and I will give an example below.
◯ No, and I will provide a proof below.

Note: we define heuristics for this problem as being a function of **only** the cell location: They cannot incorporate the location of the skull, since that did not exist in the old version of the board that we are comparing to.

If you believe Ethan is right, give an example of a heuristic that used to be consistent on the old board but is no longer consistent on this new board. Make sure to explain why it is no longer consistent (perhaps with a drawing of a board state and an explanation.

State	A	B	C	D	E	F	G	H
$h(s)$								

If you believe Ethan is wrong, provide an argument for why a heuristic that was consistent in the old board must also remain consistent in this new board. Be specific about your reasoning and use mathematical quantities such as heuristic costs of states $h(a)$ and true costs of transitions $c(ab)$.

Q3. [30 pts] The Camping Scheduling Problem

Eight camping groups would like to camp at a campground with eight time slots per week. Each group will take only one slot and each slot can only be taken by one group. Before the assignment is conducted, the camping ground manager has kindly asked every group to indicate their preferences toward the time slots (e.g. group 1 cannot take slot 8, or group 2 prefers to take the slot earlier than group 3, etc.). The manager would then assign the time slots according to the preferences provided by all the groups.

(a) During the week before school starts, all the groups are available for all the slots and there are no preferences indicated by any group. The manager realizes there is a striking similarity between this instance of the scheduling task and the **N-queens problem** from *lecture 5: CSP II*.

(i) [7 pts] Formulate the camping scheduling problem into a CSP with regard to *Variables*, *Domain*, and *Constraints*, in a way similar to the N-queens problem.

Variables: _____. A total of 64 variables.

Domain: _____. A total of _____ elements in the domain.

Constraints (please use mathematical notations, and express in terms of the variables):

(ii) [3 pts] Is this CSP **identical** to the eight-queens problem?

◯ Yes

◯ No

Explanation:

(b) During a long weekend, the availability and preferences of the groups become more complicated, and the manager proposed to formulate the CSP in the following way: if we represent the 8 groups numbered from 1 to 8, and the 8 slots numbered from 1 to 8, we have

- Variables: $\{X_j\}_{j=1}^{8}$ where X_j represents the slot ID of the j-th group.
- Domain: $\{1, 2, 3, 4, 5, 6, 7, 8\}$.
- Constraints:
  1. $X_i \neq X_j \quad \forall i \neq j$
  2. $X_{3k} = 3X_k \quad \forall k \in \{1, 2\}$
  3. $X_{4k} = 4X_k \quad \forall k \in \{1, 2\}$
  4. $X_5 > X_6 > X_7$

The constraints above are all binary, and reflect the preferences of all the TAs towards all the slots.

(Feel free to use the table below for scratch work)

X_1	1	2	3	4	5	6	7	8
X_2	1	2	3	4	5	6	7	8
X_3	1	2	3	4	5	6	7	8
X_4	1	2	3	4	5	6	7	8
X_5	1	2	3	4	5	6	7	8
X_6	1	2	3	4	5	6	7	8
X_7	1	2	3	4	5	6	7	8
X_8	1	2	3	4	5	6	7	8

(i) [8 pts] What are the remaining values in each domain after **only enforcing arc consistency**?

X_1 : _____, X_2 : _____, X_3 : _____, X_4 : _____,

X_5 : _____, X_6 : _____, X_7 : _____, X_8 : _____.

(ii) [2 pts] John observed that all domains in (b.i) are non-empty after arc consistency is enforced, and he claims that his observation guarantees that at least one solution exist. Is John correct?

◯ Yes

◯ No

Explanation:

(c) Your friend Catalina decides to solve the CSP formulated in part (b) without enforcing arc consistency. To speed up the backtracking search, she would like to use variable ordering like MRV (Minimum Remaining Value) and LCV (Least Constrained Value).

(i) [4 pts] Which variable ordering(s) improve the worst-case runtime of backtracking search in big O notation?

☐ MRV

☐ LCV

☐ Neither

Explanation:

(ii) [2 pts] Catalina's friend Frank claims that "using LCV means assigning variables in the ascending order of number of binary constraints associated with the variables". Is Frank correct?

◯ Yes

◯ No

Explanation:

(d) [4 pts] If we wish to use local search to solve the CSP formulated in part (b), with the initial value assignments to all the variables as $X_k = k, \forall k = 1, ..., 8$ Which variables have conflicting constraints?

☐ X_1 ☐ X_2 ☐ X_3 ☐ X_4 ☐ X_5 ☐ X_6 ☐ X_7 ☐ X_8