CS-3600-A Midterm

Vidit Dharmendra Pokharna

TOTAL POINTS

13 / 15

QUESTION 1

1 1a **0 / 1**

+ 1 pts 16/9 = 1.77 (not counting leaves), or 16/17 = .94 (counting leaves

✓ **+ 0 pts** *Incorrect*

QUESTION 2

2 1b **0 / 1**

+ 1 pts Value for 1a is greater than value for 1b

✓ **+ 0 pts** *Incorrect*

QUESTION 3

3 1c **1 / 1**

✓ **+ 1 pts** *Backward because:*

1. The branching factor is smaller.

2. The number of expansions is smaller.

+ 0 pts Incorrect

QUESTION 4

4 1d **1 / 1**

✓ **+ 1 pts** *Anything involving branching factor is smaller causing the number of expansions to be smaller.*

+ 0 pts Incorrect

QUESTION 5

5 2a **1 / 1**

✓ **+ 1 pts** *18*

+ 0 pts Incorrect

QUESTION 6

6 2b **1 / 1**

✓ **+ 1 pts** *50 or 49 (not including the goal)*

+ 0 pts Incorrect

QUESTION 7

7 2c **2 / 2**

✓ **+ 2 pts** *Option 1: Use R(state) = 18-Manhattan(state) or 100-Manhattan(state)*

Option 2: some sort of spreading activation from the goal which should give the same as 100-manhattan(state)

+ 1.5 pts Not clear enough

+ 1 pts Unclear reward description.

+ 0.5 pts Incorrect

+ 0 pts Click here to replace this description.

QUESTION 8

8 2d **1 / 1**

✓ **+ 1 pts** *Will not get stuck.*

Exploration (or epsilon-greedy)

+ 0 pts Incorrect

QUESTION 9

9 3a **1 / 1**

✓ **+ 1 pts** *Value-iteration*

+ 0 pts Incorrect

QUESTION 10

10 3b **1 / 1**

✓ **+ 1 pts** *Deep Q-Learning*

+ 0 pts Incorrect

QUESTION 11

11 3c **1 / 1**

✓ **+ 1 pts** *A**

+ 0 pts Incorrect

QUESTION 12

12 3d **1 / 1**

✓ **+ 1 pts** *Deep Q-Learning*

+ 0 pts Incorrect

QUESTION 13

13 3e **1 / 1**

✓ **+ 1 pts** *Tabular Q-Learning*

+ 0 pts Incorrect

QUESTION 14

14 3f **1 / 1**

✓ **+ 1 pts** *Deep Q-Learning*

+ 0 pts Incorrect

QUESTION 15

15 Late Days **0 / 0**

✓ **+ 0 pts** *Within late day budget*

- 3 pts 1 late day beyond allowed

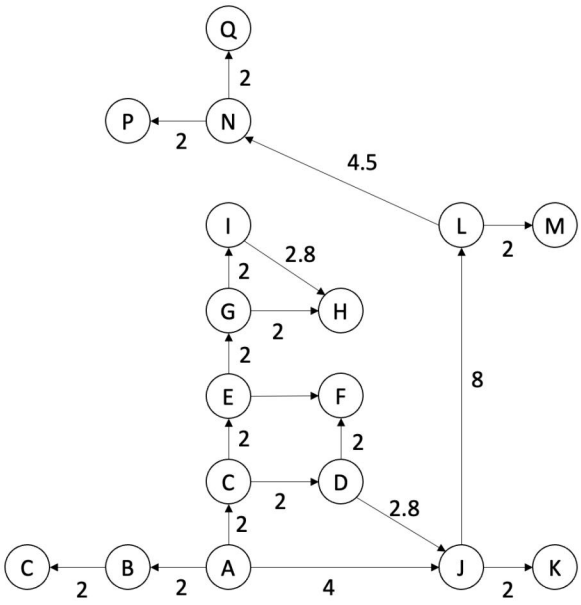
- 6 pts 2 late days beyond allowed

- 9 pts 3 late day beyond allowed

- 12 pts 4 late day beyond allowed

- 15 pts 5 late days beyond allowed

Question 1. Consider the following layout of your secret underground lair built into the side of a mountain. The circles are rooms and the edges are corridors. The numbers are the distance in decameters. You have cleverly built a robot to deliver your meals. Unfortunately, because of the geology of the mountain all the corridors are sloped and the robot can only traverse the corridors going downhill (minions must carry the robot back uphill when it has finished making its delivery). The direction of the arcs in the graph indicate the downhill direction. The robot always starts in room A (the kitchen).



1.a (1 pt.) Compute the average number of successors: 19/17

1.b (1 pt.) Compute the average number of predecessors. The predecessors are found by reversing the directionality of the arcs: 19/17

1.c (1 pt.) Suppose you are in your science lab in room P when you get hungry. The agent could run the breadth-first search *forward* from the kitchen to the delivery destination, or run breadth-first search *backward* by reversing the directionality of the arcs and starting from the destination and working back to the kitchen.

Should the search be conducted forward or backward? Backward

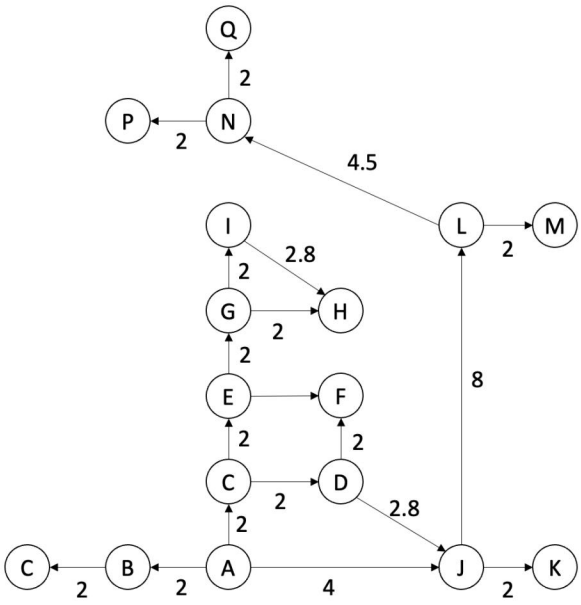
Explain why: It would be better to use the backward search as there are less obstacles or branches in that direction, and thus less nodes to check. Below are examples of BFS in both directions, with priority to alphabetical order:
Backwards: P, N, L, J, {A,D}

1 1a **0 / 1**

+ 1 pts 16/9 = 1.77 (not counting leaves), or 16/17 = .94 (counting leaves

✓ **+ 0 pts** *Incorrect*

Question 1. Consider the following layout of your secret underground lair built into the side of a mountain. The circles are rooms and the edges are corridors. The numbers are the distance in decameters. You have cleverly built a robot to deliver your meals. Unfortunately, because of the geology of the mountain all the corridors are sloped and the robot can only traverse the corridors going downhill (minions must carry the robot back uphill when it has finished making its delivery). The direction of the arcs in the graph indicate the downhill direction. The robot always starts in room A (the kitchen).



1.a (1 pt.) Compute the average number of successors: 19/17

1.b (1 pt.) Compute the average number of predecessors. The predecessors are found by reversing the directionality of the arcs: 19/17

1.c (1 pt.) Suppose you are in your science lab in room P when you get hungry. The agent could run the breadth-first search *forward* from the kitchen to the delivery destination, or run breadth-first search *backward* by reversing the directionality of the arcs and starting from the destination and working back to the kitchen.

Should the search be conducted forward or backward? Backward

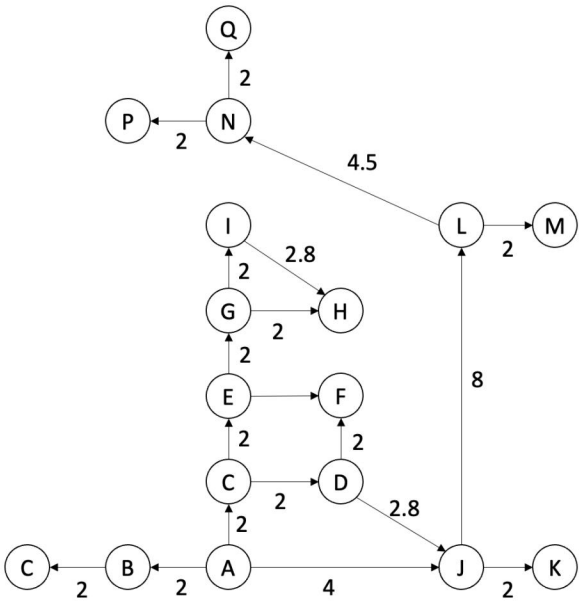
Explain why: It would be better to use the backward search as there are less obstacles or branches in that direction, and thus less nodes to check. Below are examples of BFS in both directions, with priority to alphabetical order:
Backwards: P, N, L, J, {A,D}

2 1b **0 / 1**

+ 1 pts Value for 1a is greater than value for 1b

✓ **+ 0 pts** *Incorrect*

Question 1. Consider the following layout of your secret underground lair built into the side of a mountain. The circles are rooms and the edges are corridors. The numbers are the distance in decameters. You have cleverly built a robot to deliver your meals. Unfortunately, because of the geology of the mountain all the corridors are sloped and the robot can only traverse the corridors going downhill (minions must carry the robot back uphill when it has finished making its delivery). The direction of the arcs in the graph indicate the downhill direction. The robot always starts in room A (the kitchen).



1.a (1 pt.) Compute the average number of successors: 19/17

1.b (1 pt.) Compute the average number of predecessors. The predecessors are found by reversing the directionality of the arcs: 19/17

1.c (1 pt.) Suppose you are in your science lab in room P when you get hungry. The agent could run the breadth-first search *forward* from the kitchen to the delivery destination, or run breadth-first search *backward* by reversing the directionality of the arcs and starting from the destination and working back to the kitchen.

Should the search be conducted forward or backward? Backward

Explain why: It would be better to use the backward search as there are less obstacles or branches in that direction, and thus less nodes to check. Below are examples of BFS in both directions, with priority to alphabetical order:
Backwards: P, N, L, J, {A,D}

Forwards: A, {B,C,J}, {C,E,D,K,L}, …
The forward direction would take much more than 5 nodes to find P, whereas the backwards method found A in 5 nodes and is much more efficient in this case.

1.d (1 pts.) Is it always more efficient to run the search in the direction you specified in 1.c regardless of what the robot's destination is? Explain why.
In this graph, it is always more efficient to run backwards for a path from A. The way the graph is structured, there are more outgoing nodes from A than any other node. When you have more outgoing nodes from A, it implies that there are more possible paths or options to explore when starting the search from A. By initiating the search from the destination and moving backward, you explore the options and potential paths more efficiently. This method allows you to traverse through the graph in a way that potentially requires fewer steps compared to a forward search.

3 1c **1 / 1**

✓ **+ 1 pts** *Backward because:*

1. The branching factor is smaller.

2. The number of expansions is smaller.

+ 0 pts Incorrect

Forwards: A, {B,C,J}, {C,E,D,K,L}, …
The forward direction would take much more than 5 nodes to find P, whereas the backwards method found A in 5 nodes and is much more efficient in this case.

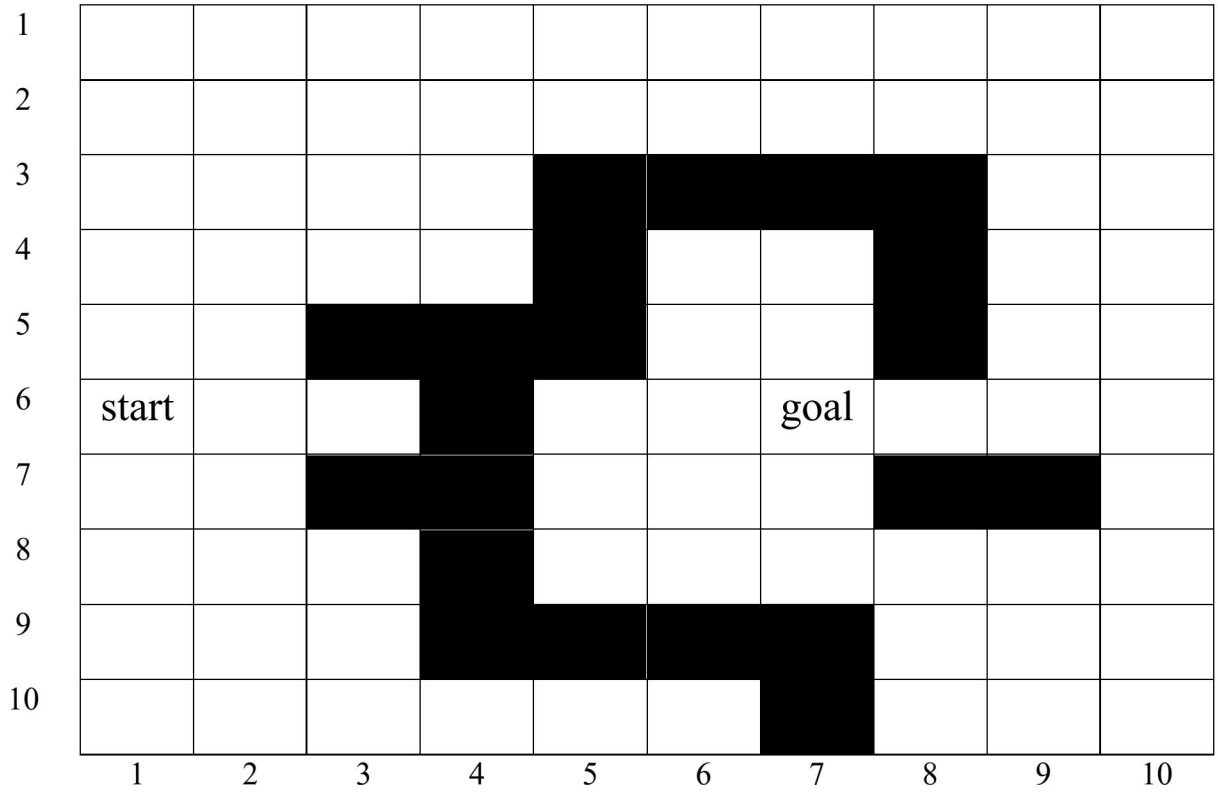
1.d (1 pts.) Is it always more efficient to run the search in the direction you specified in 1.c regardless of what the robot's destination is? Explain why.
In this graph, it is always more efficient to run backwards for a path from A. The way the graph is structured, there are more outgoing nodes from A than any other node. When you have more outgoing nodes from A, it implies that there are more possible paths or options to explore when starting the search from A. By initiating the search from the destination and moving backward, you explore the options and potential paths more efficiently. This method allows you to traverse through the graph in a way that potentially requires fewer steps compared to a forward search.

4 1d **1 / 1**

✓ **+ 1 pts** *Anything involving branching factor is smaller causing the number of expansions to be smaller.*

+ 0 pts Incorrect

Question 2. Consider the following grid world. The agent can move up, down, left, and right. There is a cost of 1 for every action. The agent starts in the designated cell and must reach the cell marked "goal". The goal is a terminal state and also has a reward value of 100.



2.a. (1 pt.) If the transition function is deterministic, one would use A* with a heuristic h(state) = manhattan_distance(state, goal). What is the length of the optimal path from the start to the goal?

18

2.b. (1 pt.) How many states must be visited by A* using the above heuristic? Hint: the maximum f-value of any state that can be part of the solution path is 18.

50, including the start and goal state

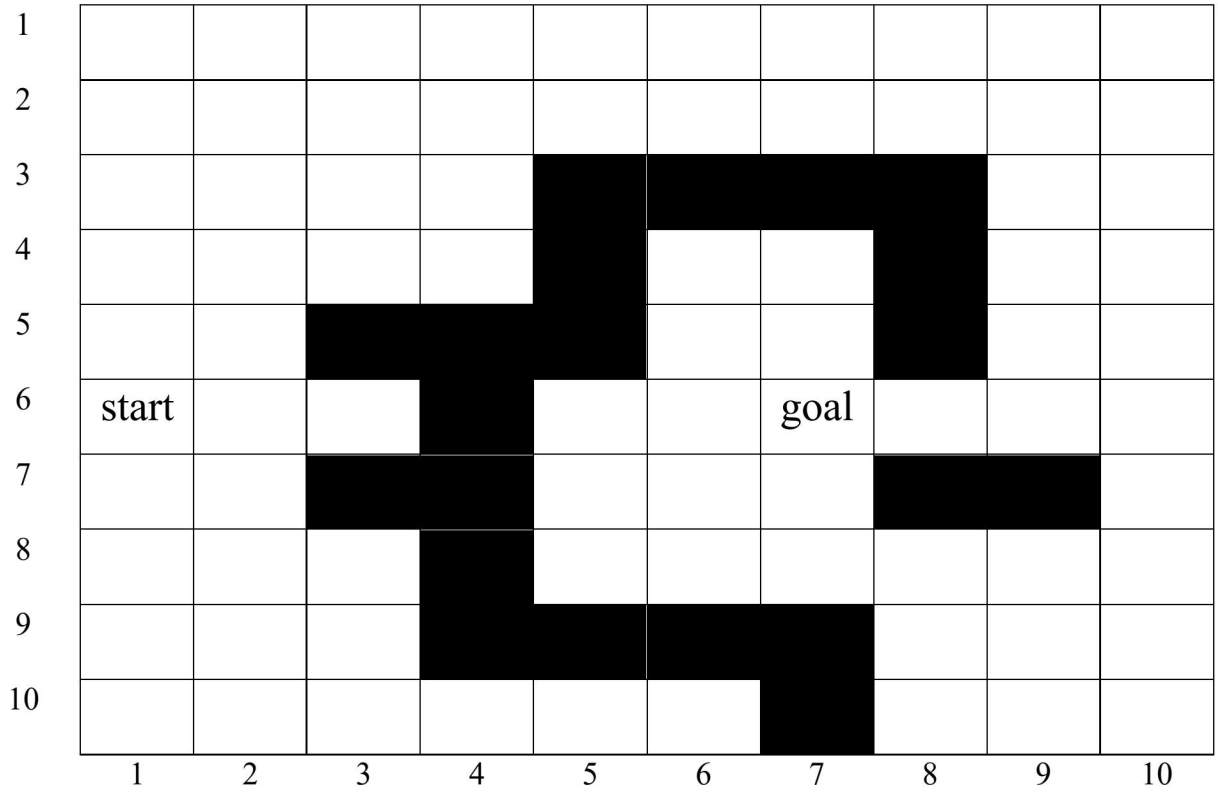
2.c. (2 pts.) Suppose we didn't have the transition function (and thus didn't know if it was deterministic or not) and thus used Q-learning instead. We refer to the fact that there is only a single state, the "goal", that provides a reward as a "sparse" reward function. With an empty Q-table and with most states returning zero reward, it can take quite a while for the agent to find the goal randomly and to start propagating q-values to neighboring states. If we had a "dense" reward function, then every state would provide a reward and the agent would learn much faster.

5 2a **1 / 1**

✓ **+ 1 pts** *18*

+ 0 pts Incorrect

Question 2. Consider the following grid world. The agent can move up, down, left, and right. There is a cost of 1 for every action. The agent starts in the designated cell and must reach the cell marked "goal". The goal is a terminal state and also has a reward value of 100.



2.a. (1 pt.) If the transition function is deterministic, one would use A* with a heuristic h(state) = manhattan_distance(state, goal). What is the length of the optimal path from the start to the goal?

18

2.b. (1 pt.) How many states must be visited by A* using the above heuristic? Hint: the maximum f-value of any state that can be part of the solution path is 18.

50, including the start and goal state

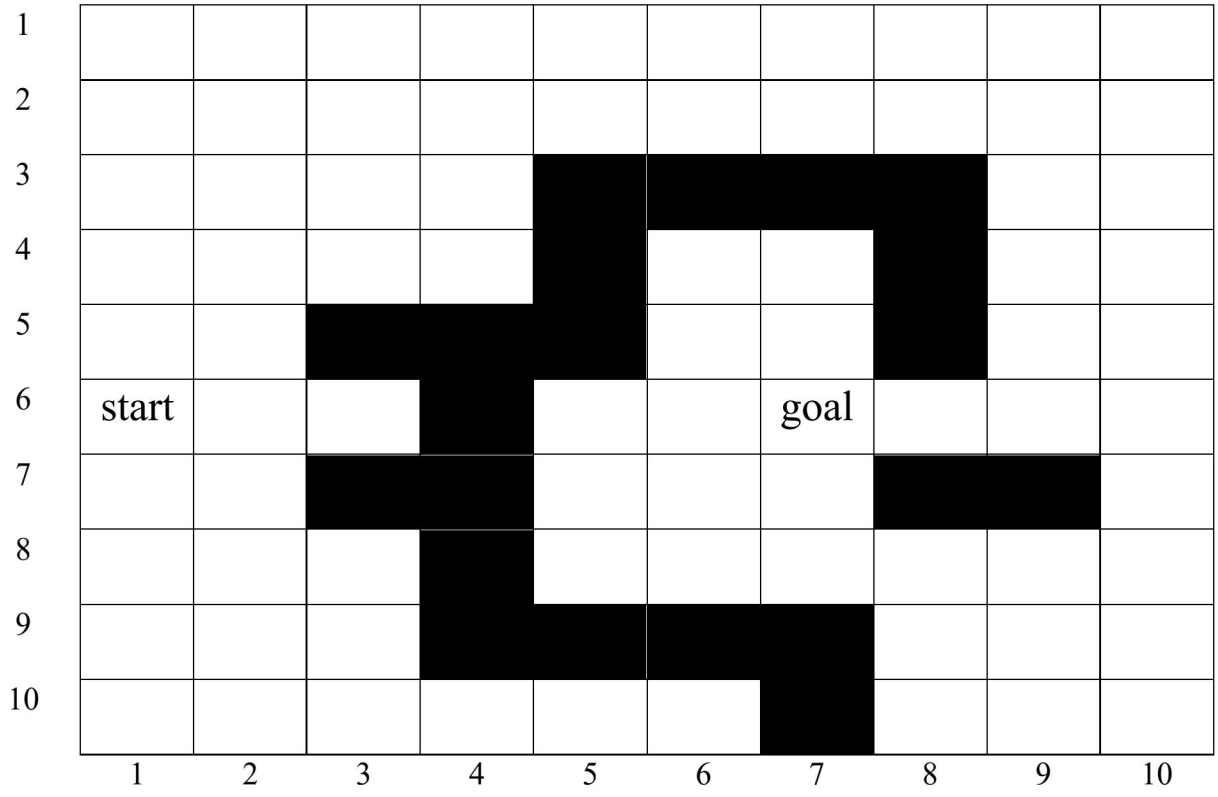
2.c. (2 pts.) Suppose we didn't have the transition function (and thus didn't know if it was deterministic or not) and thus used Q-learning instead. We refer to the fact that there is only a single state, the "goal", that provides a reward as a "sparse" reward function. With an empty Q-table and with most states returning zero reward, it can take quite a while for the agent to find the goal randomly and to start propagating q-values to neighboring states. If we had a "dense" reward function, then every state would provide a reward and the agent would learn much faster.

6 2b **1 / 1**

✓ **+ 1 pts** *50 or 49 (not including the goal)*

+ 0 pts Incorrect

Question 2. Consider the following grid world. The agent can move up, down, left, and right. There is a cost of 1 for every action. The agent starts in the designated cell and must reach the cell marked "goal". The goal is a terminal state and also has a reward value of 100.



2.a. (1 pt.) If the transition function is deterministic, one would use A* with a heuristic h(state) = manhattan_distance(state, goal). What is the length of the optimal path from the start to the goal?

18

2.b. (1 pt.) How many states must be visited by A* using the above heuristic? Hint: the maximum f-value of any state that can be part of the solution path is 18.

50, including the start and goal state

2.c. (2 pts.) Suppose we didn't have the transition function (and thus didn't know if it was deterministic or not) and thus used Q-learning instead. We refer to the fact that there is only a single state, the "goal", that provides a reward as a "sparse" reward function. With an empty Q-table and with most states returning zero reward, it can take quite a while for the agent to find the goal randomly and to start propagating q-values to neighboring states. If we had a "dense" reward function, then every state would provide a reward and the agent would learn much faster.

A perfect dense reward function would be one in which each state gave out reward proportional to its true utility (i.e. imagine we had a transition function and used value iteration until convergence then copied the utility values into the reward function).

Given that we cannot know the true state utilities ahead of time, perhaps we can instead compute in O(n) time (where n is the number of states) a dense reward function that approximates the true utility values for each state. That is, we would need to create a reward function by performing a O(1) operation on each state. Describe in a few sentences how to compute such a reward function.

Hint: think about how admissible heuristics are created.

To derive a dense reward function approximating the true utility values in the absence of the transition function, we can employ an admissible heuristic. The reciprocal of the heuristic value can be allocated as the reward for each state. In this context, utilizing the Manhattan distance would be effective for creating the heuristic function.

2.d. (1 pt.) Suppose the q-learning agent is using an epsilon-greedy exploration strategy using the dense rewards you computed above. Is it a problem if the reward values for states might be misleading? For example, the state at (x=3, y=6) might end up with a reward that is higher than the rewards given at (x=3, y=5) or (x=2, y=5) or (x=2, y=6). That is, will the agent get stuck and never learn a policy that reaches the goal? Why or why not?

If the reward values for states might be misleading due to the dense reward function, it could negatively affect the Q-learning process. However, it will not necessarily prevent the agent from learning a policy that reaches the goal. This is because the Q-learning algorithm is designed to explore the environment to find an optimal policy based on the rewards received. Initially the agent might be misled by the misleading reward values and focus on states that appear to have higher rewards, but the strategy provided by epsilon-greedy ensures that the agent continues to explore other states, preventing it from getting stuck in a suboptimal policy.

7 2c **2 / 2**

✓ **+ 2 pts** *Option 1: Use R(state) = 18-Manhattan(state) or 100-Manhattan(state)*

Option 2: some sort of spreading activation from the goal which should give the same as 100-manhattan(state)

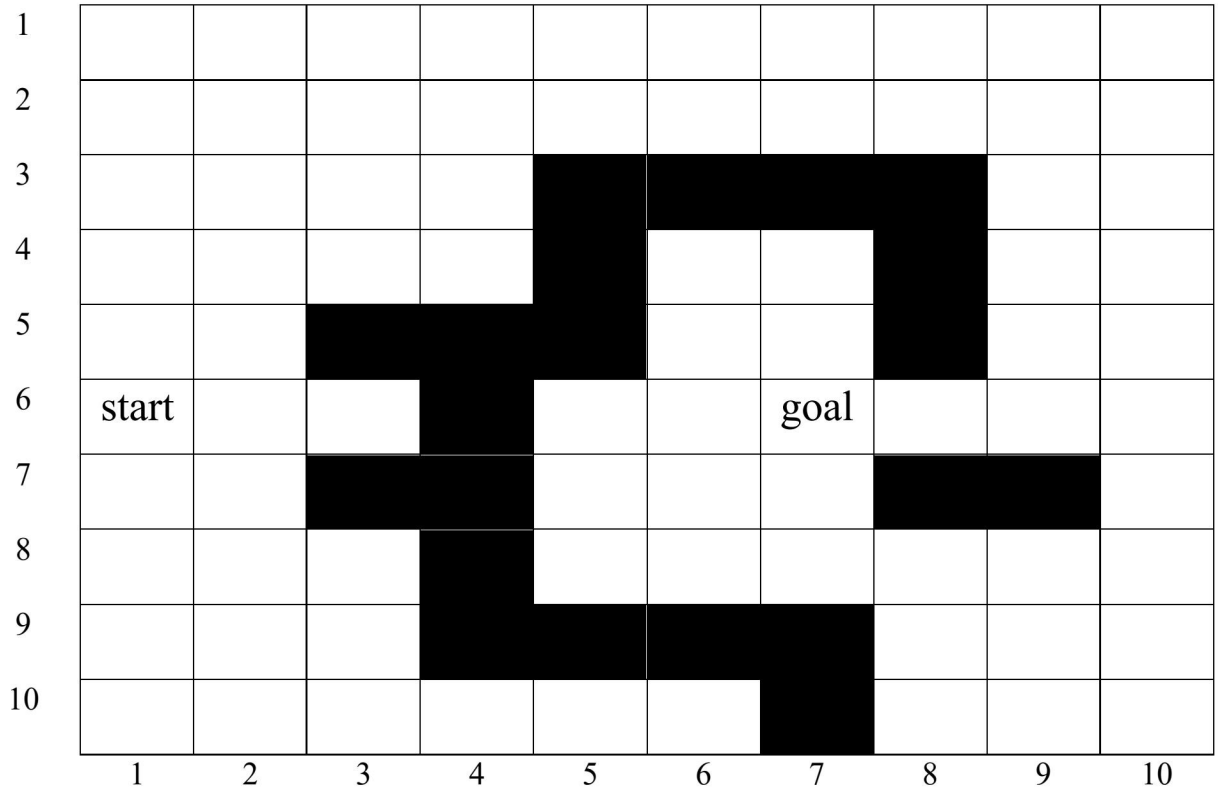
+ 1.5 pts Not clear enough

+ 1 pts Unclear reward description.

+ 0.5 pts Incorrect

+ 0 pts Click here to replace this description.

Question 2. Consider the following grid world. The agent can move up, down, left, and right. There is a cost of 1 for every action. The agent starts in the designated cell and must reach the cell marked "goal". The goal is a terminal state and also has a reward value of 100.



2.a. (1 pt.) If the transition function is deterministic, one would use A* with a heuristic h(state) = manhattan_distance(state, goal). What is the length of the optimal path from the start to the goal?

18

2.b. (1 pt.) How many states must be visited by A* using the above heuristic? Hint: the maximum f-value of any state that can be part of the solution path is 18.

50, including the start and goal state

2.c. (2 pts.) Suppose we didn't have the transition function (and thus didn't know if it was deterministic or not) and thus used Q-learning instead. We refer to the fact that there is only a single state, the "goal", that provides a reward as a "sparse" reward function. With an empty Q-table and with most states returning zero reward, it can take quite a while for the agent to find the goal randomly and to start propagating q-values to neighboring states. If we had a "dense" reward function, then every state would provide a reward and the agent would learn much faster.

8 2d **1 / 1**

✓ **+ 1 pts** *Will not get stuck.*

Exploration (or epsilon-greedy)

+ 0 pts Incorrect

Question 3: For each of the following scenarios indicate whether it would be best to use A*, value-iteration, tabular Q-learning, or deep Q-learning. Justify your answer for each.

a. (1 pt). An MDP has thousands of states, four actions per state, the transition function is known, and there are several states that give reward.

   Technique: Value Iteration

   Justification: A* is used for deterministic transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

b. (1 pt.) An MDP has billions of states, 10 actions per state, the transition function is unknown, and there are many states that give reward.

   Technique: Deep Q-Learning

   Justification: A*, Value Iteration, and Tabular Q-Learning can all only handle smaller state spaces, whereas Deep Q-Learning can manage large state and action spaces.

c. (1 pt.) An MDP has tens of thousands of states, each state has different actions associated with it (though never more than five), the transition function is deterministic, and there is only one state that gives reward which is also terminal.

   Technique: A*

   Justification: Value Iteration is used for known transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

d. (1 pt.) An MDP has thousands of states, and each state has 700^4 possible actions, the transition function is deterministic, and there are several states that give reward.

   Technique: Deep Q-Learning

   Justification: Deep Q-Learning can handle large action spaces where the transition function is not fully known.

e. (1 pt.) An MDP has thousands of states, ten actions per state, the transition function is not known, there and there are several states that give reward.

   Technique: Tabular Q-Learning

   Justification: Tabular Q-Learning can be used for scenarios where the transition function is not known. It can efficiently explore and update Q-values for a not large MDP, and is efficient with a manageable number of states, which is why Deep Q-Learning is not needed.

9 3a **1 / 1**

✓ **+ 1 pts** *Value-iteration*

+ 0 pts Incorrect

Question 3: For each of the following scenarios indicate whether it would be best to use A*, value-iteration, tabular Q-learning, or deep Q-learning. Justify your answer for each.

a. (1 pt). An MDP has thousands of states, four actions per state, the transition function is known, and there are several states that give reward.

   Technique: Value Iteration

   Justification: A* is used for deterministic transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

b. (1 pt.) An MDP has billions of states, 10 actions per state, the transition function is unknown, and there are many states that give reward.

   Technique: Deep Q-Learning

   Justification: A*, Value Iteration, and Tabular Q-Learning can all only handle smaller state spaces, whereas Deep Q-Learning can manage large state and action spaces.

c. (1 pt.) An MDP has tens of thousands of states, each state has different actions associated with it (though never more than five), the transition function is deterministic, and there is only one state that gives reward which is also terminal.

   Technique: A*

   Justification: Value Iteration is used for known transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

d. (1 pt.) An MDP has thousands of states, and each state has 700^4 possible actions, the transition function is deterministic, and there are several states that give reward.

   Technique: Deep Q-Learning

   Justification: Deep Q-Learning can handle large action spaces where the transition function is not fully known.

e. (1 pt.) An MDP has thousands of states, ten actions per state, the transition function is not known, there and there are several states that give reward.

   Technique: Tabular Q-Learning

   Justification: Tabular Q-Learning can be used for scenarios where the transition function is not known. It can efficiently explore and update Q-values for a not large MDP, and is efficient with a manageable number of states, which is why Deep Q-Learning is not needed.

10 3b **1 / 1**

✓ **+ 1 pts** *Deep Q-Learning*

+ 0 pts Incorrect

Question 3: For each of the following scenarios indicate whether it would be best to use A*, value-iteration, tabular Q-learning, or deep Q-learning. Justify your answer for each.

a. (1 pt). An MDP has thousands of states, four actions per state, the transition function is known, and there are several states that give reward.

   Technique: Value Iteration

   Justification: A* is used for deterministic transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

b. (1 pt.) An MDP has billions of states, 10 actions per state, the transition function is unknown, and there are many states that give reward.

   Technique: Deep Q-Learning

   Justification: A*, Value Iteration, and Tabular Q-Learning can all only handle smaller state spaces, whereas Deep Q-Learning can manage large state and action spaces.

c. (1 pt.) An MDP has tens of thousands of states, each state has different actions associated with it (though never more than five), the transition function is deterministic, and there is only one state that gives reward which is also terminal.

   Technique: A*

   Justification: Value Iteration is used for known transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

d. (1 pt.) An MDP has thousands of states, and each state has 700^4 possible actions, the transition function is deterministic, and there are several states that give reward.

   Technique: Deep Q-Learning

   Justification: Deep Q-Learning can handle large action spaces where the transition function is not fully known.

e. (1 pt.) An MDP has thousands of states, ten actions per state, the transition function is not known, there and there are several states that give reward.

   Technique: Tabular Q-Learning

   Justification: Tabular Q-Learning can be used for scenarios where the transition function is not known. It can efficiently explore and update Q-values for a not large MDP, and is efficient with a manageable number of states, which is why Deep Q-Learning is not needed.

11 3c **1 / 1**

✓ **+ 1 pts** *A**

+ 0 pts Incorrect

Question 3: For each of the following scenarios indicate whether it would be best to use A*, value-iteration, tabular Q-learning, or deep Q-learning. Justify your answer for each.

a. (1 pt). An MDP has thousands of states, four actions per state, the transition function is known, and there are several states that give reward.

   Technique: Value Iteration

   Justification: A* is used for deterministic transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

b. (1 pt.) An MDP has billions of states, 10 actions per state, the transition function is unknown, and there are many states that give reward.

   Technique: Deep Q-Learning

   Justification: A*, Value Iteration, and Tabular Q-Learning can all only handle smaller state spaces, whereas Deep Q-Learning can manage large state and action spaces.

c. (1 pt.) An MDP has tens of thousands of states, each state has different actions associated with it (though never more than five), the transition function is deterministic, and there is only one state that gives reward which is also terminal.

   Technique: A*

   Justification: Value Iteration is used for known transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

d. (1 pt.) An MDP has thousands of states, and each state has 700^4 possible actions, the transition function is deterministic, and there are several states that give reward.

   Technique: Deep Q-Learning

   Justification: Deep Q-Learning can handle large action spaces where the transition function is not fully known.

e. (1 pt.) An MDP has thousands of states, ten actions per state, the transition function is not known, there and there are several states that give reward.

   Technique: Tabular Q-Learning

   Justification: Tabular Q-Learning can be used for scenarios where the transition function is not known. It can efficiently explore and update Q-values for a not large MDP, and is efficient with a manageable number of states, which is why Deep Q-Learning is not needed.

12 3d **1 / 1**

✓ **+ 1 pts** *Deep Q-Learning*

+ 0 pts Incorrect

Question 3: For each of the following scenarios indicate whether it would be best to use A*, value-iteration, tabular Q-learning, or deep Q-learning. Justify your answer for each.

a. (1 pt). An MDP has thousands of states, four actions per state, the transition function is known, and there are several states that give reward.

   Technique: Value Iteration

   Justification: A* is used for deterministic transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

b. (1 pt.) An MDP has billions of states, 10 actions per state, the transition function is unknown, and there are many states that give reward.

   Technique: Deep Q-Learning

   Justification: A*, Value Iteration, and Tabular Q-Learning can all only handle smaller state spaces, whereas Deep Q-Learning can manage large state and action spaces.

c. (1 pt.) An MDP has tens of thousands of states, each state has different actions associated with it (though never more than five), the transition function is deterministic, and there is only one state that gives reward which is also terminal.

   Technique: A*

   Justification: Value Iteration is used for known transition functions, and Tabular/Deep Q-Learning is used for unknown transition functions.

d. (1 pt.) An MDP has thousands of states, and each state has 700^4 possible actions, the transition function is deterministic, and there are several states that give reward.

   Technique: Deep Q-Learning

   Justification: Deep Q-Learning can handle large action spaces where the transition function is not fully known.

e. (1 pt.) An MDP has thousands of states, ten actions per state, the transition function is not known, there and there are several states that give reward.

   Technique: Tabular Q-Learning

   Justification: Tabular Q-Learning can be used for scenarios where the transition function is not known. It can efficiently explore and update Q-values for a not large MDP, and is efficient with a manageable number of states, which is why Deep Q-Learning is not needed.

13 3e **1 / 1**

✓ **+ 1 pts** *Tabular Q-Learning*

+ 0 pts Incorrect

f. (1 pt.) An MDP has thousands of states, four actions per state, the transition function is unknown, there are many states that give reward, and the state space is partially-observable.

Technique: Deep Q-Learning

Justification: Since the transition is unknown, the answer will be Q-Learning. Deep Q-Learning is better suited for learning in partially observable environments, as it can effectively handle the uncertainties associated with such environments.

14 3f **1 / 1**

✓ **+ 1 pts** *Deep Q-Learning*

+ 0 pts Incorrect

15 Late Days **0 / 0**

✓ **+ 0 pts** *Within late day budget*

- 3 pts 1 late day beyond allowed

- 6 pts 2 late days beyond allowed

- 9 pts 3 late day beyond allowed

- 12 pts 4 late day beyond allowed

- 15 pts 5 late days beyond allowed

CS 3600 Midterm (section A)

We expect this exam to take ~1.5 hours, however you may use your time in any way you see fit. The exam is open notes, open book, and open lecture videos. The course collaboration policy and late day policy are in effect.

You may edit the file to insert your answers or print, write by hand, and scan back in. You may add extra white space between questions if you need it.

Submit your final version as a PDF to Gradescope.

List any collaborations here: _____