COMP5600/6600 Artificial Intelligence HW1

INSTRUCTIONS

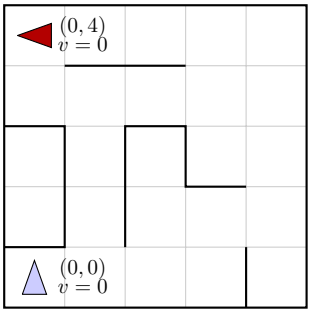
- **Due: Wednesday, 12 February 2025 at 11:59 PM CST.**
- **Format:** Write your answers in the `yoursolution.tex` file and latex a pdf (preferred) or you can type directly on the blank pdf. Make sure that your answers are within the dedicated regions for each question/part. If you do not follow this format, we may deduct points.
- **Images:** To insert pictures, we recommend drawing it on PowerPoint or Google Drawings, saving it as an image and including it in your latex source.
- **How to submit:** Submit a pdf with your answers on Canvas. Log in and click on our class 5600/6600 and click on the submission titled Homework 1 and upload your pdf containing your answers.
- **Policy:** See the course website for homework policies and Academic Integrity.

Name	Will Gasser
Auburn ID	904256760 — wbg0023
Hours to complete?	◯ (0, 2] hours ◯ (2, 4] hours ● (4, 6] hours ◯ (6, 8] hours ◯ > 8 hours

For TA use only

Q1	Q2	Q3	Q4	Q5	Total
/23	/10	/25	/17	/25	/100

Q1. [23 pts] Search and Heuristics



Consider a car-like agent trying to navigate out of a maze, similar to the one depicted above. The agent is directional and always faces a specific direction $d \in (N, S, E, W)$. With each action, the agent has the option to either move forward at a variable speed v or to turn.

The movement actions include *faster*, *maintain*, and *slower*. When the agent performs these actions, it moves a number of squares corresponding to its **new** adjusted velocity. Let v represent the agent's current velocity, and v' denote the new adjusted velocity.

- *Faster*: $v' = v + 1$
- *Slower*: $v' = v$ - 1
- *Maintain*: $v' = v$

The turning actions are *left* and *right*, which change the agent's direction by 90 degrees. **Turning is only permitted when the velocity is zero.** Turning leaves the speed at zero.

- *Left*: change the agent's direction by 90 degrees counterclockwise
- *Right*: change the agent's direction by 90 degrees clockwise

For example, if the agent is currently on (0, 0) facing north with velocity 0 (as pictured) and wants to get to (2, 0) facing east with velocity 0, the sequence of actions will be: *right, faster, maintain, slower*.

Illegal actions include

- Any action that would result in a collision with a wall (i.e. there is a wall between the current position and the position you would be in if you took said action)
- Any action that would reduce v below 0 (slowing when v=0) or above a maximum speed V_{max}
- Maintaining a velocity of 0
- Turning when velocity $\neq 0$

The agent's goal is to find a plan which parks it ($v = 0$) in the goal direction on the exit square using as few actions (time steps) as possible. Note that the cost of a path is defined by the number of actions the agent takes.

(a) [3 pts] If the grid is M by N and the maximum speed is V_{max}, what is the size of the state space? You should assume that all configurations are reachable from the start state.

State Space Size:
4 * M * N * (V_{max} + 1)
The constant 4 is multiplied to account for all directions NSEW.
The M * N factor is included to account for each state of the grid position.
The factor V_{max} includes all velocities of the actor and +1 to account for the state of velocity 0.

(b) [3 pts] A "child" of a state s is any other state s' reachable via a legal action from state s. Is it possible that a state in the state space has no children? If so, give an example of such a state. If not, briefly explain why every state must have at least one child.

● Yes ○ No

Example State or Explanation:
A state can occur where the actor is at a speed and cannot turn. If the actor approaches the wall and reaches the wall space with a speed greater than 1, the actor will not be able to slow to 0. Thus, when the actor is facing the wall, they cannot turn because their speed is too high. All actions become illegal and there is no child state.

(c) [4 pts] What is the maximum branching factor of this problem? Draw an example state (x, y, orientation, velocity) that has this branching factor, and list the set of available actions. For example, in the above picture, if the agent was in (0, 0) facing North with a velocity of $v = 0$, the branching factor would be 2. The agent could turn left or right (but not go faster since it would hit a wall).

Illegal actions are simply not returned by the problem model and therefore not counted in the branching factor. You do not necessarily have to use the example grid above. If you need to include a drawing of your own, label properly and **make sure it fits in the solution box**.

Maximum Branching Factor:
3

Maximum Branching Example State and Available Actions:
There are two primary cases where the branching factor can be up to three: when v equals 0 and v is greater than 0. For instance, if the actor is at (4,2,S,0) in the grid, it can choose from three actions: turn left, turn right, or accelerate. Alternatively, if the actor is at (4,2,S,1), its options are to accelerate, maintain its speed, or decelerate.

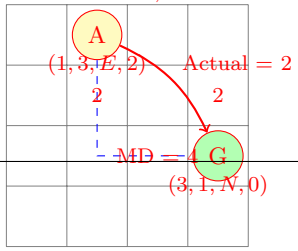
(d) [4 pts] Is the Manhattan distance from the agent's location to the exit's location admissible?

If not, draw an example state (x, y, orientation, velocity) where this heuristic overestimates at that state, and specify: 1) the heuristic value at that state and 2) the actual cost from that state to the goal.

You do not necessarily have to use the example grid above. Make sure to label your drawing, including the goal state (location, orientation, speed) and action sequence, and fit it into the solution box.

○ Yes ● No

Example State, Heuristic Value, Actual Cost:
If the velocity of the moving agent is greater than one, the agent can reach the goal faster than the manhattan heuristic because the agent can move faster than 1 unit/time. Therefore the manhattan distance can overestimate because it will display a larger heuristic than the actual outcome. Therefore, the manhattan heuristic is not admissable.



Manhattan (4) > Speed (2 units/time)

(e) [4 pts] Is the following heuristic admissible? *Manhattan distance* / V_{max}.

If yes, state why. If not, draw an example state (x, y, orientation, velocity) where this heuristic overestimates at that state, and specify: 1) the heuristic value at that state and 2) the actual cost from that state to the goal.

You do not necessarily have to use the example grid above. Make sure to label your drawing, including the goal state (location, orientation, speed) and action sequence, and fit it into the solution box.

● Yes ○ No

Example State, Heuristic Value, Actual Cost:
Normalizing the Manhattan distance by dividing by V_{max} assumes the car is always at its maximum speed which is unrealistic because turns require deceleration and subsequent acceleration. As a result, this adjusted heuristic almost always underestimates the actual cost, making it admissible since it will never overestimate the true cost to the goal.

(f) [1 pt] If we used an inadmissible heuristic in A* Tree search, could it change the completeness of the search? Assume the graph is finite and the heuristic is non-negative.

○ Yes ● No

(g) [1 pt] If we used an inadmissible heuristic in A* Tree search, could it change the optimality of the search? Assume the graph is finite and the heuristic is non-negative.

● Yes ○ No

(h) [3 pts] Which of the following may be a good reason to use an inadmissible heuristic over an admissible one? Select all that apply.

☐ An inadmissible heuristic may be easier to compute, leading to a faster state heuristic computation time.

☒ An inadmissible heuristic can be a closer estimate to the actual cost (even if it's an overestimate) than an admissible heuristic, thus exploring fewer nodes.

☐ An inadmissible heuristic will still find optimal paths when the actual costs are non-negative.

☐ An inadmissible heuristic may be used to completely block off searching part of a graph in a search algorithm.

Q2. [10 pts] All Searches Lead to the Same Destination

For all the questions below assume :

- All search algorithms are *graph* search (as opposed to tree search).
- $c_{ij} > 0$ is the cost to go from node i to node j.
- There is only one goal node (as opposed to a set of goal nodes).
- All ties are broken alphabetically.
- Assume heuristics are consistent.

Definition: Two search algorithms are defined to be ***equivalent*** if and only if they expand the same nodes in the same order and return the same path.

In this question we study what happens if we run uniform cost search with action costs d_{ij} that are potentially different from the search problem's actual action costs c_{ij}. Concretely, we will study how this might, or might not, result in running uniform cost search (with these new choices of action costs) being equivalent to another search algorithm.

(a) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Breadth-First Search**.

- ☐ $d_{ij} = 0$
- ☒ $d_{ij} = \alpha$, $\alpha > 0$
- ☐ $d_{ij} = \alpha$, $\alpha < 0$
- ☒ $d_{ij} = 1$
- ☐ $d_{ij} = -1$
- ☐ None of the above

(b) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Depth-First Search**.

- ☐ $d_{ij} = 0$
- ☐ $d_{ij} = \alpha$, $\alpha > 0$
- ☒ $d_{ij} = \alpha$, $\alpha < 0$
- ☐ $d_{ij} = 1$
- ☒ $d_{ij} = -1$
- ☐ None of the above

(c) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Uniform Cost Search** with the original costs c_{ij}.

- ☐ $d_{ij} = c_{ij}^2$
- ☐ $d_{ij} = 1/c_{ij}$
- ☒ $d_{ij} = \alpha\ c_{ij}, \qquad \alpha > 0$
- ☐ $d_{ij} = c_{ij} + \alpha, \qquad \alpha > 0$
- ☐ $d_{ij} = \alpha\ c_{ij} + \beta, \quad \alpha > 0,\ \beta > 0$
- ☐ None of the above

(d) Let $h(n)$ be the value of the heuristic function at node n.

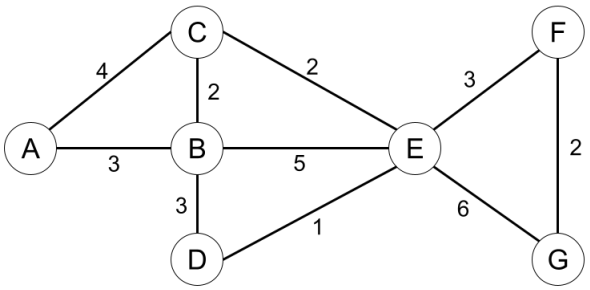
(i) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **Greedy Search** with the original costs c_{ij} and heuristic function h.

- ☐ $d_{ij} = h(i) - h(j)$
- ☒ $d_{ij} = h(j) - h(i)$
- ☐ $d_{ij} = \alpha\ h(i), \qquad \alpha > 0$
- ☐ $d_{ij} = \alpha\ h(j), \qquad \alpha > 0$
- ☐ $d_{ij} = c_{ij} + h(j) + h(i)$
- ☐ None of the above

(ii) [2 pts] Mark *all* choices for costs d_{ij} that make running **Uniform Cost Search** algorithm with these costs d_{ij} *equivalent* to running **A* Search** with the original costs c_{ij} and heuristic function h.

- ☐ $d_{ij} = \alpha\ h(i), \quad \alpha > 0$
- ☐ $d_{ij} = \alpha\ h(j), \quad \alpha > 0$
- ☐ $d_{ij} = c_{ij} + h(i)$
- ☐ $d_{ij} = c_{ij} + h(j)$
- ☐ $d_{ij} = c_{ij} + h(i) - h(j)$
- ☒ $d_{ij} = c_{ij} + h(j) - h(i)$
- ☐ None of the above

Q3. [25 pts] Searching a Graph



Node	h_1	h_2
A	12	11
B	6	7
C	9	6
D	3	4
E	3	5
F	2	1
G	0	0

Consider the graph shown above. A is the start state and G is the goal state. The costs for each edge are shown on the graph. The graph is bi-directional so each edge can be traversed from either direction. Please refer to the search algorithms **exactly as presented on the lecture slides** as the ordering of the actions matters.

(a) [15 pts] For each of the following **graph search** strategies, mark with an X which (if any) of the listed paths it could return. Note that for some search strategies the specific path returned might depend on tie-breaking behavior. In any such cases, make sure to mark **all** paths that could be returned under some tie-breaking scheme. If a graph search strategy returns a path not listed, **write out the correct path** in the *Other* column.

Algorithm	A-C-E-G	A-C-E-F-G	A-B-D-E-F-G	Other
UCS	(i)	(ii) X	(iii)	(iv)
Greedy with heuristic h_1	(v)	(vi)	(vii)	(viii) X ABDEG
Greedy with heuristic h_2	(ix) X	(x)	(xi)	(xii)
A* with heuristic h_1	(xiii)	(xiv)	(xv) X	(xvi)
A* with heuristic h_2	(xvii)	(xviii) X	(xix)	(xx)

(b) [2 pts] What is the cost of the optimal path for uniform cost search from A to G?

Answer:
The cost was 11.

(c) [4 pts] Is h_1 admissible? Is it consistent?

Admissible: ○ Yes ● No

Consistent: ○ Yes ● No

(d) [4 pts] Is h_2 admissible? Is it consistent?

Admissible: ● Yes ○ No

Consistent: ○ Yes ● No

Q4. [17 pts] Search: Multiple Choice and Short Answer Questions

(a) [12 pts] Consider the following true/false questions with each question worth 2 points. For the following search problems, assume every action has a cost of at least ϵ, with $\epsilon > 0$. Assume any heuristics used are consistent.

Depth-first tree-search on a finite graph is guaranteed to be complete.
○ **True** ● **False**

Breadth-first tree-search on a finite graph is guaranteed to be complete.
○ **True** ● **False**

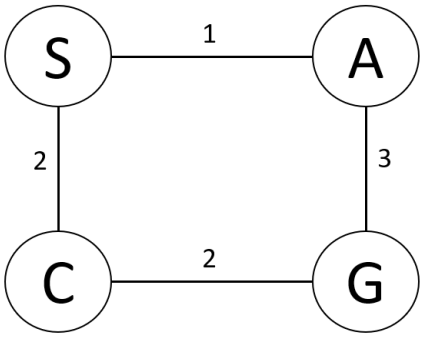
Iterative deepening tree-search on a finite graph is guaranteed to be complete.
● **True** ○ **False**

For all graphs without cycles, graph-search contains a larger frontier than tree-search.
○ **True** ● **False**

Iterative deepening graph-search has the time complexity of BFS and the space complexity of DFS.
○ **True** ● **False**

If h_1(s) is a consistent heuristic and h_2(s) is a consistent heuristic, then min(h_1(s), h_2(s)) must be consistent.
● **True** ○ **False**

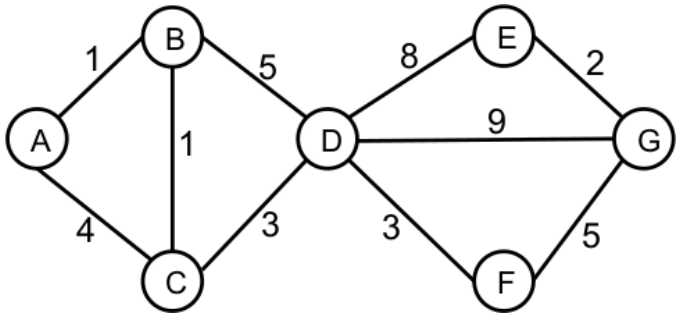
(b) [5 pts] Consider the state space graph shown below. S is the start state and G is the goal state. The costs for each edge are shown on the graph. For the following table below, fill in potential heuristic values such that the heuristic is admissible but not consistent.



Heuristic Function	
State	$h(s)$
S	4
A	1
C	3
G	0

Q5. [25 pts] Search Again

Consider the state space graph shown below. A is the start state and G is the goal state. The costs for each edge are shown on the graph. Each edge can be traversed in both directions.



Use the Graph Search Algorithm (v2) discussed in class. Execute the following search algorithms using priority queues, by filling in the search table for each part. Write nodes as a tuple containing a state sequence and cost (e.g. (A-B-C, 2)). Note that for Breadth first and Depth first the algorithms ignore the true "cost" so you can just use the depth of the node as the second part of the tuple and then expand nodes with either the highest or lowest depth. Break ties alphabetically. Note that all steps in the table below will not necessarily be used. You may skip any steps where a node is removed from the frontier but not expanded. Note that nodes are only expanded after they are removed from the frontier, after checking the goal test, and after checking if not in the closed set.

(a) [3 pts] Breadth First Graph Search

Step	**Priority Queue**	**Expand**
1	(A,0)	A
2	(A-B,1), (A-C,1)	B
3	(A-C,1), (A-B-D,2)	C
4	(A-B-D,2)	D
5	(A-B-D-E,2), (A-B-D-F,2), (A-B-D-G,2) - Expand E	
6	(A-B-D-F,2), (A-B-D-G,2)	F
7	(A-B-D-G,2)	
8		

Solution: A-B-D-G, c=3

(b) [3 pts] Depth First Graph Search.

Step	**Priority Queue**	**Expand**
1	(A,0)	A
2	(A-B,1), (A-C,1)	B
3	(A-B-D,2), (A-C,1)	D
4	(A-B-D-E,2), (A-B-D-F,2), (A-B-D-G,2), (A-C,1)	E
5	(A-B-D-F,2), (A-B-D-G,2), (A-C,1) - Expand F	
6	(A-B-D-G,2), (A-C,1)	
7		
8		

Solution:A-B-D-G, c=3

(c) [3 pts] Uniform Cost Graph Search.

Step	**Priority Queue**	**Expand**
1	(A,0)	A
2	(A-B,1), (A-C,4)	B
3	(A-B-C,2), (A-B-D,6)	C
4	(A-B-C-D,5)	D
5	(A-B-C-D-F,8), (A-B-C-D-E,13), (A-B-C-D-G,14) - Expand F	
6	(A-B-C-D-E,13), (A-B-C-D-F-G,13)	E
7	(A-B-C-D-F-G,13)	
8		

Solution:A-B-C-D-F-G, c=13

(d) Heuristic Search

(i) [4 pts] Consider the two heurisitics h_1 and h_2, only one of which is consistent. Which one is consistent?
h_1

Node	A	B	C	D	E	F	G
h_1	9.5	9	8	7	1.5	4	0
h_2	10	12	10	8	1	4.5	0

(ii) [3 pts] Then do A* search with that heuristic.

Step	**Priority Queue**	**Expand**
1	(A,9.5)	A
2	(A-B,10), (A-C,12)	B
3	(A-B-C,10), (A-B-D,13)	C
4	(A-B-C-D,12)	D
5	(A-B-C-D-F,12), (A-B-C-D-G,14), (A-B-C-D-E,14.5) - Expand F	
6	(A-B-C-D-F-G,13), (A-B-C-D-E,14.5)	
7		
8		

Solution: A-B-C-D-F-G, c=13

(iii) [9 pts] Suppose you are completing the new heuristic function h_3 shown below. All the values are fixed except $h_3(B)$.

Node	A	B	C	D	E	F	G
h_3	10	?	9	7	1.5	4.5	0

For each of the following conditions, write the set of values that are possible for $h_3(B)$. For example, to denote all non-negative numbers, write [0, ∞], to denote the empty set, write $\emptyset$, and so on.

1. What values of $h_3(B)$ make h_3 admissible?
   $[-\infty, 12]$
2. What values of $h_3(B)$ make h_3 consistent?
   $[9, 10]$
3. What values of $h_3(B)$ will cause A* graph search to expand from node A to C, then node A to B, then node A to B to D in that order?
   $[13, 14]$