CS 495 & 540 Spring 2016 Artificial Intelligence — Midterm

- The exam is closed book, closed notes except a one-page crib sheet.
- The total score is 120pts and you have approximately 110 minutes. Use your time wisely. If you get stuck, it is advised that you should try those questions that are most rewarding with respect to the time it takes you to solve.
- I leave plenty of space for each problem. Please write your solution on the exam itself. Two blank sheets are attached at the back of the exam to serve as scratch paper for you. DO NOT detach the sheets.
- If you finish the exam early, please leave the exam on the desk and I will collect it.

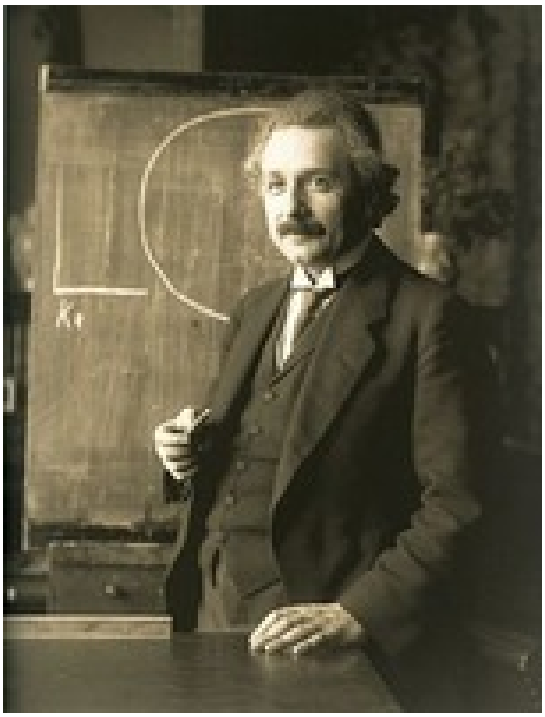
First name	
Last name	
First name of student to your left	
First name of student to your right	

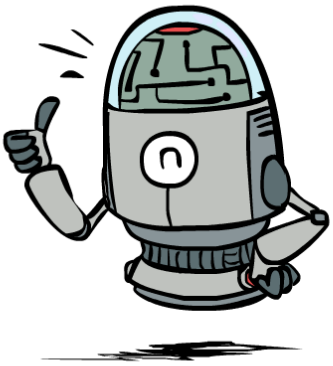
For staff use only:

Q1.	Warm-Up	/2
Q2.	BFS, DFS, Iterative Deepning Search	/10
Q3.	CSPs: Properties	/12
Q4.	X Values	/10
Q5.	MDPs and RL: Mini-Grids	/28
Q6.	CSPs: Job Assignments	/21
Q7.	Dynamic A* Search	/12
Q8.	Theory: Offline MDP: Value Iteration, Policy Evaluation and Policy Iteration	/25
Q9.	Scratch paper: Do Not Detach	/0
Q10.	Scratch paper: Do Not Detach	/0
	Total	/120

Q1. [2 pts] Warm-Up

Circle the AI mascot that has been used in the slides





Q2. [10 pts] BFS, DFS, Iterative Deepning Search

Given a balanced tree with branching factor = b and height = m. Let suppose the goal is hidden at level k (leave node is at level 0). Please draw a simple graph (along with some brief description) for each sub question:

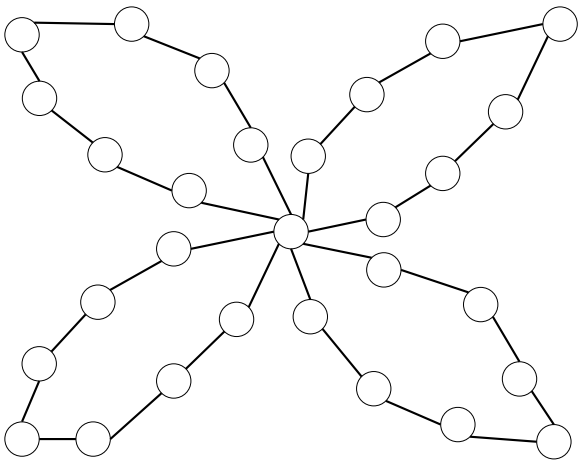
(a) [5 pts] Iterative Deepening gets defeated by DFS in terms of time complexity. Please brief describe the complexity of each and explain the condition among parameters m, b, k to guarantee this occurs.

(b) [5 pts] Iterative Deepening outperforms DFS in terms of time complexity. Please brief describe the complexity of each and explain the condition among parameters m, b, k to guarantee this occurs.

Q3. [12 pts] CSPs: Properties

(a) [4 pts] What is the maximum number of times a backtracking search algorithm might have to backtrack in a *tree-structured* CSP, if it is running arc consistency and using an optimal variable ordering?

(b) [7 pts] **Constraint Graph** Consider the following constraint graph:



(1)[5] In two sentences or less, describe a strategy for efficiently solving a CSP with this constraint structure.

(2) [3] We say this graph is k-colorable. What is the minimal number that k can be?

Q4. [10 pts] X Values

Instead of the Bellman update equation, consider an alternative update equation, which learns the X value function. The update equation, assuming a discount factor $\gamma = 1$, is shown below:

$$X_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \max_{a'} \sum_{s''} T(s',a',s'') \left[R(s',a',s'') + X_k(s'') \right] \right]$$

(a) [6 pts] Assuming we have an MDP with two states, S_1, S_2, and two actions, a_1, a_2, draw the expectimax tree rooted at S_1 that corresponds to the alternative update equation.

(b) [4 pts] Write the mathematical relationship between the X_k-values learned using the alternative update equation and the V_k-values learned using a Bellman update equation, or write *None* if there is no relationship.

Q5. [28 pts] MDPs and RL: Mini-Grids

The following problems take place in various scenarios of the gridworld MDP (as in Project 3). In all cases, A is the start state and double-rectangle states are exit states. From an exit state, the only action available is *Exit*, which results in the listed reward and ends the game (by moving into a terminal state X, not shown).

From non-exit states, the agent can choose either *Left* or *Right* actions, which move the agent in the corresponding direction. There are no living rewards; the only non-zero rewards come from exiting the grid.

Throughout this problem, assume that value iteration begins with initial values $V_0(s) = 0$ for all states s.

First, consider the following mini-grid. For now, the discount is $\gamma = 1$ and legal movement actions will always succeed (and so the state transition function is deterministic).

(a) [2 pts] What is the optimal value $V^*(A)$?

(b) [2 pts] When running value iteration, remember that we start with $V_0(s) = 0$ for all s. What is the first iteration k for which $V_k(A)$ will be non-zero?

(c) [2 pts] What will $V_k(A)$ be when it is first non-zero?

(d) [2 pts] After how many iterations k will we have $V_k(A) = V^*(A)$? If they will never become equal, write *never*.

Now the situation is as before, but the discount γ is less than 1.

(e) [2 pts] If $\gamma = 0.5$, what is the optimal value $V^*(A)$?

(f) [2 pts] For what range of values γ of the discount will it be optimal to go *Right* from A? Remember that $0 \leq \gamma \leq 1$. Write *all* or *none* if all or no legal values of γ have this property.

Let's kick it up a notch! The *Left* and *Right* movement actions are now stochastic and fail with probability f. When an action fails, the agent moves *up* or *down* with probability $f/2$ each. When there is no square to move *up* or *down* into (as in the one-dimensional case), the agent stays in place. The *Exit* action does not fail.

For the following mini-grid, the failure probability is $f = 0.5$. The discount is back to $\gamma = 1$.

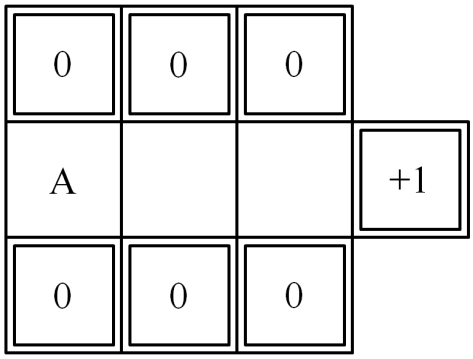
(g) [2 pts] What is the optimal value $V^*(A)$?

(h) [2 pts] When running value iteration, what is the smallest value of k for which $V_k(A)$ will be non-zero?

(i) [2 pts] What will $V_k(A)$ be when it is first non-zero?

(j) [2 pts] After how many iterations k will we have $V_k(A) = V^*(A)$? If they will never become equal, write *never*.

Now consider the following mini-grid. Again, the failure probability is $f = 0.5$ and $\gamma = 1$. Remember that failure results in a shift *up* or *down*, and that the only action available from the double-walled exit states is *Exit*.



(k) [1 pt] What is the optimal value $V^*(A)$?

(l) [2 pts] When running value iteration, what is the smallest value of k for which $V_k(A)$ will be non-zero?

(m) [2 pts] What will $V_k(A)$ be when it is first non-zero?

(n) [2 pts] After how many iterations k will we have $V_k(A) = V^*(A)$? If they will never become equal, write *never*.

Q6. [21 pts] CSPs: Job Assignments

In some exam, there are a total of 6 questions on the exam and each question will cover a topic. Here is the format of the exam:

- q1. Search
- q2. Games
- q3. CSPs
- q4. MDPs
- q5. True/False
- q6. Short Answer

There are 7 people on the course staff: Brad, Donahue, Ferguson, Judy, Kyle, Michael, and Nick. Each of them is responsible to work with Dr. Chiang on one question. (But a question could end up having more than one staff person, or potentially zero staff assigned to it.) However, the staff are pretty quirky and want the following constraints to be satisfied:

(i) Donahue (D) will not work on a question together with Judy (J).

(ii) Kyle (K) must work on either Search, Games or CSPs

(iii) Michael (M) is very odd, so he can only contribute to an odd-numbered question.

(iv) Nick (N) must work on a question that's before Michael (M)'s question.

(v) Kyle (K) must work on a question that's before Donahue (D)'s question

(vi) Brad (B) does not like grading exams, so he must work on True/False.

(vii) Judy (J) must work on a question that's after Nick (N)'s question.

(viii) If Brad (B) is to work with someone, it cannot be with Nick (N).

(ix) Nick (N) cannot work on question 6.

(x) Ferguson (F) cannot work on questions 4, 5, or 6

(xi) Donahue (D) cannot work on question 5.

(xii) Donahue (D) must work on a question before Ferguson (F)'s question.

(a) [2 pts] We will model this problem as a constraint satisfaction problem (CSP). Our variables correspond to each of the staff members, J, F, N, D, M, B, K, and the domains are the questions 1, 2, 3, 4, 5, 6. After applying the unary constraints, what are the resulting domains of each variable? (The second grid with variables and domains is provided as a back-up in case you mess up on the first one.)

B	1	2	3	4	5	6
D	1	2	3	4	5	6
F	1	2	3	4	5	6
J	1	2	3	4	5	6
K	1	2	3	4	5	6
N	1	2	3	4	5	6
M	1	2	3	4	5	6

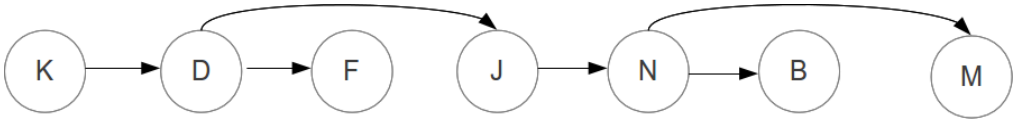
B	1	2	3	4	5	6
D	1	2	3	4	5	6
F	1	2	3	4	5	6
J	1	2	3	4	5	6
K	1	2	3	4	5	6
N	1	2	3	4	5	6
M	1	2	3	4	5	6

(b) [6 pts] If we apply the Minimum Remaining Value (MRV) heuristic, please list the first three variables that will be chosen and their associated value. And if the variable has mulitple options, choose the highest value.

(c) [3 pts] Normally we would now proceed with the variable you found in (b), but to decouple this question from the previous one (and prevent potential errors from propagating), let's proceed with assigning Michael first. For value ordering we use the Least Constraining Value (LCV) heuristic, where we use *Forward Checking* to compute the number of remaining values in other variables domains. What ordering of values is prescribed by the LCV heuristic? Include your work—i.e., include the resulting filtered domains that are different for the different values.

(d) Realizing this is a tree-structured CSP, we decide not to run backtracking search, and instead use the efficient two-pass algorithm to solve tree-structured CSPs. We will run this two-pass algorithm after applying the unary constraints from part (a). Below is the linearized version of the tree-structured CSP graph for you to work with.

(i) [6 pts] **First Pass: Domain Pruning.** Pass from *right to left* to perform Domain Pruning. Write the values that remain in each domain below each node in the figure above.



K	D	F	J	N	B	M
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	5	5	5	5	5
6	6	6	6	6	6	6

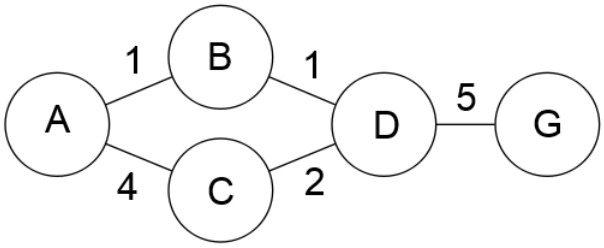
(ii) [4 pts] **Second Pass: Find Solution.** Pass from *left to right*, assigning values for the solution. If there is more than one possible assignment, choose the **lowest** value.

Q7. [12 pts] Dynamic A* Search

After running A* graph search and finding an optimal path from start to goal, the cost of one of the edges, $X \to Y$, in the graph changes. Rather than re-running the entire search, you want to find a more efficient way of finding the optimal path for this new search problem.

You have access to the fringe, the closed set and the search tree as they were at the completion of the initial search. In addition, you have a *closed node map* that maps a state, s from the closed set to a list of nodes in the search tree ending in s which were not expanded because s was already in the closed set.

For example, after running A* search with the null heuristic on the following graph, the data structures would be as follows:



Fringe: {}

Closed Node Map: {A:[ ], B:[ ], C:[ ], D:[(A-C-D, 6)]}

Closed Set: {A, B, C, D}

Search Tree:
{A : [(A-B, 1), (A-C, 4)],
A-B : [(A-B-D, 2)],
A-C : [ ],
A-B-D : [(A-B-D-G, 7)],
A-B-D-E : [ ]}

For a general graph, for each of the following scenarios, select the choice that finds the correct optimal path and cost *while expanding the fewest nodes*. Note that if you select the 4^{th} choice, you must fill in the change, and if you select the last choice, you must describe the set of nodes to add to the fringe.

In the answer choices below, if an option states some nodes will be added to the fringe, this also implies that the final state of each node gets cleared out of the closed set (indeed, otherwise it'd be rather useless to add something back into the fringe). You may assume that there are no ties in terms of path costs.

Following is a set of eight choices you should use to answer the questions on the following page: (the order is changed, please read carefully)

i. The optimal path does not change, but the cost decreases by n
ii. The optimal path does not change, and the cost remains the same.
iii. The optimal path does not change, but the cost increases by n
iv. The optimal path for the new search problem can be found by adding the subtree rooted at Y that was expanded in the original search back onto the fringe and re-starting the search.
v. The optimal path does not change, but the cost changes by ?_____
vi. The optimal path for the new search problem can be found by adding the subtree rooted at X that was expanded in the original search back onto the fringe and re-starting the search.
vii. The optimal path for the new search problem can be found by adding all nodes for each state in the *closed node map* back onto the fringe and re-starting the search.
viii. The optimal path for the new search problem can be found by adding some other set of nodes back onto the fringe and re-starting the search. Describe the set below.

(a) [2 pts] Cost of $X \to Y$ is increased by $n, n > 0$, the edge is on the optimal path, and was explored by the first search.

(b) [2 pts] Cost of $X \to Y$ is decreased by $n, n > 0$, the edge is on the optimal path, and was explored by the first search.

(c) [2 pts] Cost of $X \to Y$ is increased by $n, n > 0$, the edge is not on the optimal path, and was explored by the first search.

(d) [2 pts] Cost of $X \to Y$ is decreased by $n, n > 0$, the edge is not on the optimal path, and was explored by the first search.

(e) [2 pts] Cost of $X \to Y$ is increased by $n, n > 0$, the edge is not on the optimal path, and was not explored by the first search.

(f) [2 pts] Cost of $X \to Y$ is decreased by $n, n > 0$, the edge is not on the optimal path, and was not explored by the first search.

Q8. [25 pts] Theory: Offline MDP: Value Iteration, Policy Evaluation and Policy Iteration

We are given the following parameters: $S = \{s_1, ..., s_m\}$: the set of possible states. $A = \{a_1, \cdots a_n\}$: the set of actions. Let $p(s_j|s_i, a_t)$ be defined as the probability of moving from state s_i to state s_j when action a_t is taken. It is clear that $|S| = m$ and $|A| = n$. Suppose the height of the tree is k (i.e. we hav level 0, ..., level k) and you are also given a policy $\pi_1 = \{a_i|a_i \in A\}$ and $|\pi_1| = k$. [If explanation is required, answers without justification will not be given partial credits]

(a)[4] What is the complexity of **one level iteration** for value iteration? Why?

(b)[4] What is the complexity of **one level iteration** for policy iteration? Why?

(c)[4] What is the complexity of **evaluating the whole tree** if your code is simply recursive for the value iteration ? Why?

(d)[4] What is the complexity of **evaluating the whole tree** if your code is simply recursive for the **policy evaluation**? Why?

(e) We are morphing (moving) by using **policy iteration**, from π_i then π_2, and so on, in order to find the best policy to achieve optimal, which value iteration does in one shot. Please describe the the complexity of this approach by describing

(1)[3] How many possible moves (π_i policies) are there?

(2)[6] What is the lower bound and upper bound of this policy iteration approach? Why?

Q9. [0 pts] Scratch paper: Do Not Detach

(a) [0 pts]

Q10. [0 pts] Scratch paper: Do Not Detach

(a) [0 pts]