CS 188 Fall 2006 Introduction to Artificial Intelligence Midterm

You have 80 minutes. The exam is closed book, one page cheat sheet allowed, non-programmable basic calculators allowed. 80 points total. Pace yourself, and don't panic!

Mark your answers ON THE EXAM ITSELF. Write your name, SID, login, and section number at the top of each page.

If you are not sure of your answer you may wish to provide a *brief* explanation. All short answers can be successfully answered in a few sentences *at most.*

For staff use only

Q. 1	Q. 2	Q. 3	Q. 4	Q. 5	Q. 6	Total
/17	/10	/15	/12	/16	/10	/80

1. (17 points.) Short Answer

Answers should fit in a single sentence.

(a) **(3 pts):** Give a set of broad conditions under which A* search reduces to BFS.

If the cost of all actions are equal and the heuristic function returns a constant value.

(b) **(3 pts):** When would DFS be a better choice than A* search? Describe a broad qualitative condition rather than a specific example.

If you don't care about finding an optimal solution and if the solutions are deep and dense then DFS may be a better choice than A* because it will find a solution by going deeper into the search tree more quickly than A*. DFS also has a memory advantage over A*. Note that although we were lenient on the grading of this question, if you don't care about optimality, but solutions aren't very dense, *greedy* search may be a good idea, but DFS will still to be much much worse than A*.

(c) **(3 pts):** Why is temporal difference learning of q-values (Q-learning) superior to temporal difference learning of values?

Because if you use temporal difference learning on the values, it is hard to extract a policy from your learned values (you would need to know the transition model, T, to compute the policy from the values). For temporal difference learning of Q-values, a policy can be extracted directly by taking

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

(d) **(3 pts):** Write a Bellman update for *q-value iteration*, which is like value iteration except q-values rather than values are learned from previous q-value estimates, using a one-step lookahead (i.e. you should express Q_{i+1} estimates in terms of Q_i estimates).

$$Q_{i+1}(s,a) \longleftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

(e) **(5 pts):** In a *Markov game*, or adversarial MDP, two players, MAX and MIN alternate actions in an MDP. Assume the game is zero-sum, so that when a transition (s,a,s') occurs, MAX receives $R(s,a,s')$, while MIN receives $-R(s,a,s')$, regardless of who initiated the transition. Assume that both players have the same set of actions available in any state and that both players use the same discount per time step. Let $V_{MAX}(s)$ and $V_{MIN}(s)$ be the expected future discounted rewards for each player. Write *two* Bellman equations, expressing each of V_{MAX} and V_{MIN} in terms of adjacent lookahead values of V_{MAX} and / or V_{MIN}.

$$\begin{aligned} Vmax_{i+1}(s) &= \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma Vmin_i(s')] \\ Vmin_{i+1}(s) &= \min_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma Vmax_i(s')] \end{aligned}$$

Note the lack of an inversion of the rewards. In a single time step, max maxes R, while min mins it.

2. (10 points.) Search and Heuristics

Consider the following variant of Mazeworld, shown below, where a *single planning agent* must move four robots from a configuration on one side of the grid to a goal configuration on the other side, shown below. In any given time step, each robot can move one square North, South, East, West, or stay where it is. After the moves occur, no two robots can occupy the same result square. However, for example, two adjacent robots *can* swap positions in a single time step. A possible successor of the start state is shown below in which B moved North, D moved West, C moved North, and A stood still. The cost of each time step is 1, regardless of how many robots actually moved.

```
##############     ##############     ##############
#            #     #            #     #            #
# # ## ##### #     # # ## ##### #     # # ## ##### #
# #    #     #     # #    #     #     # #    #     #
# ## ##### ###     #B## ##### ###     # ## ##### ###
#BD#     #   #     #DC#     #   #     #  #     # CD#
#AC# ###     #     #A # ###     #     #  # ###   AB#
##############     ##############     ##############

    Start            Successor             Goal
```

(1) (2 pts): What is the *branching factor* for this search problem (this kind of maze in general, not necessarily this particular configuration of walls). Give the tightest (smallest) upper bound you can.

5^4. Each of the 4 robots has 5 options.

(2) (2 pts): If the grid is n by n, what is the size of the *state space*? Give the tightest upper bound you can.

$n^2 * (n^2 - 1) * (n^2 - 2) * (n^2 - 3)$, which is $O((n * n)^{numRobots})$

(3) (4 pts): Propose an efficient, simple admissible heuristic for use in A* search. The number of computations made by your heuristic should not depend on the size of the grid. Trivial and nearly trivial answers (e.g. "1" unless at a goal state) will not receive credit.

One possible heuristic might be the maximum manhattan distance over all robots to their respective goal location.

(4) (2 pts): Propose an admissible heuristic that takes advantage of the functionality of your Mazeworld code. Your heuristic should return the correct forward cost (i.e. be the perfect heuristic) in the case of a single robot, but need not be constant time.

For each robot we could find the cost of the shortest path from the robot's position to its goal position using A*. We could then return the max of these costs over all the robots.

3. (15 points.) CSPs

You are in charge of scheduling for computer science classes that meet Mondays, Wednesdays and Fridays. There are 5 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

The classes are:

- Class 1 - Intro to Programming: meets from 8:00-9:00am
- Class 2 - Intro to Artificial Intelligence: meets from 8:30-9:30am
- Class 3 - Natural Language Processing: meets from 9:00-10:00am
- Class 4 - Computer Vision: meets from 9:00-10:00am
- Class 5 - Machine Learning: meets from 9:30-10:30am

The professors are:

- Professor A, who is available to teach Classes 3 and 4.
- Professor B, who is available to teach Classes 2, 3, 4, and 5.
- Professor C, who is available to teach Classes 1, 2, 3, 4, 5.

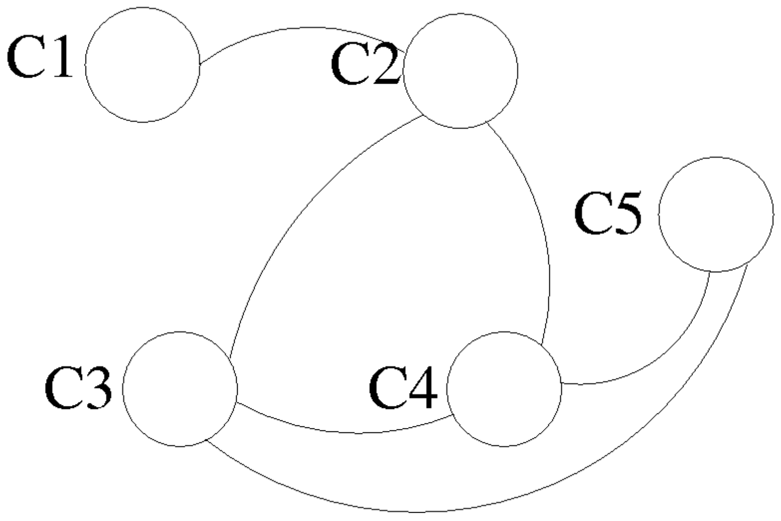
(1) (4 pts): Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

$Variables$	$Domains(orunaryconstraints)$
$C1$	C
$C2$	B, C
$C3$	A, B, C
$C4$	A, B, C
$C5$	B, C

Constraints:

$C1 \neq C2$
$C2 \neq C3$
$C3 \neq C4$
$C4 \neq C5$
$C2 \neq C4$
$C3 \neq C5$

(2) (2 pts): Draw the constraint graph associated with your CSP.



(3) (4 pts): Show the domains of the variables after running arc-consistency on this initial graph (after having already enforced any unary constraints).

$Variable$	$Domain$
$C1$	C
$C2$	B
$C3$	A, C
$C4$	A, C
$C5$	B, C

Note that C5 cannot possibly be C, but arc consistency does not rule it out.

(4) (1 pt): Give one solution to this CSP.

C1 = C, C2 = B, C3 = C, C4 = A, C5 = B. One other solution is possible (where C3 and C4 are switched).

(5) (2 pts): Your CSP should look nearly tree-structured. Briefly explain (one sentence or less) why we might prefer to solve tree-structures CSPs.

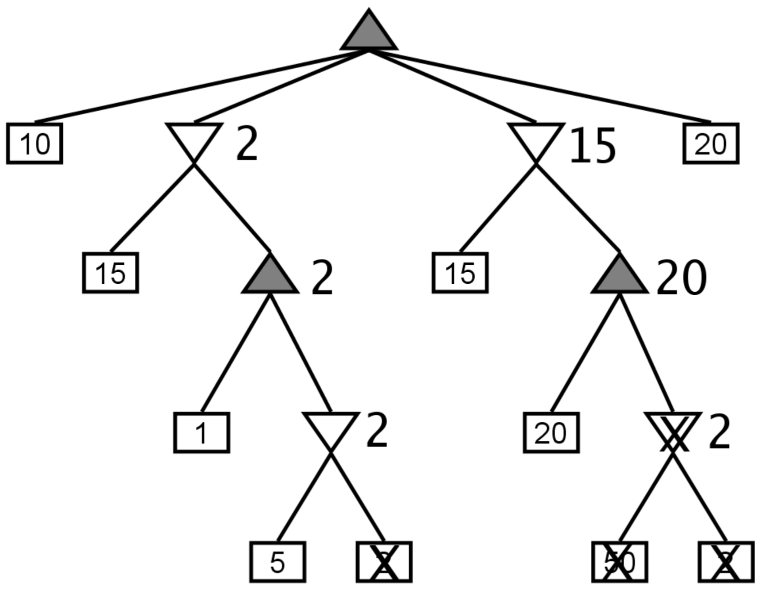
Minimal answer: Can solve them in polynomial time. If a graph is tree structured (i.e. has no loops), then the CSP can be solved in $O(nd^2)$ time as compared to general CSPs, where worst-case time is $O(d^n)$. For tree-structured CSPs you can choose an ordering such that every node's parent precedes it in the ordering. Then you can greedily assign the nodes in order and will find a consistent assignment without backtracking.

(6) (2 pts): Name (or briefly describe) a standard technique for turning these kinds of nearly tree-structured problems into tree-structured ones.

Minimal answer: cutset conditioning. One standard technique is to instantiate cutset, a variable (or set of variables) whose removal turns the problem into a tree structured CSP. To instantiate the cutset you set its values in each possible way, prune neighbors, then solve the reduced tree structured problem (which is fast).

4. (12 points.) Minimax Search

Consider the following minimax tree.



(1) (2 pts): What is the minimax value for the root?

20

(2) (5 pts): Draw an X through any nodes which will not be visited by alpha-beta pruning, assuming children are visited in left-to-right order.

See pic above.

(3) (2 pts): Is there another ordering for the *children of the root* for which more pruning would result? If so, state the order.

Yes, if we had the children ordered as 20, 15, 10, 2.

(4) (3 pts): Propose a general, practical method for ordering children of nodes which will tend to increase the opportunities for pruning. You should be concise, but clearly state both what to do about min nodes and max nodes.

In general we would want to use an evaluation function to estimate the values of the children and then order them so that at max nodes we order the children with larger estimated values first and at min nodes we order the children with larger estimated values first. Most of you did not mention the evaluation function; ordering nodes by their true values is not practical.

5. (16 points.) Multi-Agent Search

(1) (4 pts): For the case where ghosts move randomly, state Pacman (with Project 1 rules) as a single-agent MDP where Pacman is the agent. Each component should be answered in *at most* one sentence:

States: Configurations of the board, as given in a GameState object, including food locations, agent positions, etc.

Actions: A subset of {*north, south, east, west, stop*}, varying by state.

Transitions: For each s, a, s' triple, $T(s, a, s')$ is the probability that, if Pacman chooses action a, the ghosts' actions will result in s'.

Rewards: The score received for the Pacman and Ghost actions which lead from s to s'.

Discount: 1 (i.e. no discount in 1.3 Pacman rules).

Answer the following questions precisely, but in around one sentence.

```
%%%%%%%%%              %%%%%%%%%
% G%   .%              %G %   .%
%%%% %% %              %%%% %% %
%       %      ⇔       %       %
% %% %% %              % %% %% %
% %   <.%              % %  > .%
%%%%%%%%%              %%%%%%%%%
```

(2) (3 pts): Pacman thrashes, alternating between the two configurations shown (with some irrelevant motion from the ghost), making no progress. If Pacman is using minimax search, depth 3, with the state score as the evaluation function, why would he exhibit this behavior?

When using minimax search with depth 3, Pacman is indifferent between the states in which he eats the dot sooner versus later. His search is not deep enough to register that earlier eating allows him to make progress towards ending the game sooner.

(3) (2 pts): Would an increase to the depth fix this? If so, what depth would be needed for Pacman to eat the dot and move on? If not, why not?

Yes, if you increased the depth to 5 pacman would eat the dot and move on. At a depth of 5, pacman will be able to see a path to both of the remaining dots and will prefer it to the paths where he gets only one dot in the same time.

Some of you pointed out that with very deep search, he might still be happy to wander around for a while. This is strictly not true, since he loses a point each round, but was considered correct if you stated it clearly.

(4) (2 pts): Keeping the depth at 3, how could we fix this thrashing with a basic idea from MDPs?

If you add a discount factor for future rewards it would make eating dots immediately more appealing and would solve the problem of the two states appearing to have equal utility.

Some of you mentioned *reward shaping*, where Pacman would get rewards for making progress towards a dot. This would indeed fix the problem by making him prefer the future where he ate the dot and made more progress to the one where he made less progress.

Some of you mentioned moving randomly, with exploration, which doesn't directly address this issue, but might help by accident.

(5) (5 pts): If Pacman knows that the ghosts are using 2-ply minimax, with a known evaluation function, what is his optimal search process? You may assume that there is only one ghost. Clearly specify an algorithm which describes the computation you recommend.

Pacman's optimal action is always expectimax (in some specific cases this reduces to minimax). In this case, Pacman can simulate the ghosts in any state and figure out what their optimal actions are according to 2-ply minimax search with their known evaluation function. He should then use expectimax where those actions receive all the probability (with equal weight unless he knows something about how they break ties). If you did not realize there could be multiple ghost-equivalent actions, that was acceptable.

6. (10 points.) MDPs and Reinforcement Learning

Consider an autonomous robot which can either move FAST or SLOW in any time step. Moving FAST generally gives a reward of +2, while moving SLOW gives a reward of only +1. However, the robot must also take into account its internal temperature, which can be either HOT or OK. Driving SLOW tends to lower the temperature, while driving FAST tends to raise it. If the robot is HOT, there is a danger if it overheating, at which point it must stop, cool down, and make repairs. The MDP transitions and rewards are specified as follows:

s	a	s'	$T(s,a,s')$	$R(s,a,s')$
OK	SLOW	OK	1.0	+1
OK	FAST	OK	0.5	+2
OK	FAST	HOT	0.5	+2
HOT	SLOW	OK	1.0	+1
HOT	FAST	HOT	0.5	+2
HOT	FAST	OK	0.5	-10

Note that while repairs are costly, the robot is OK afterwards (the last row in the table).

(1) (5 pts): Run two rounds of value iteration in the table below, using a discount of 0.8. You may skip the greyed-out square.

s	V_0	V_1	V_2
OK	0	2	3.2
HOT	0	1	▨

$$V_{i+1}(s) \longleftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$$

$$V_1(ok) \longleftarrow \max[(1(1+\gamma * 0)), (0.5(2+\gamma * 0) + 0.5(2+\gamma * 0))] = max(1,2) = 2$$

$$V_1(hot) \longleftarrow \max[(1(1+\gamma * 0)), (0.5(2+\gamma * 0) + 0.5(-10+\gamma * 0))] = max(1,-4) = 1$$

$$V_2(ok) \longleftarrow \max[(1(1+\gamma * 2)), (0.5(2+\gamma * 2) + 0.5(2+\gamma * 1))] = max(2.6,3.2) = 3.2$$

(1) (5 pts): Run Q-learning with a discount of 0.8 and a learning rate of 0.5, using the transition samples below. Do not copy over q-values which have not changed in a given step.

Assume the agent experiences the samples:

OK, FAST, HOT, reward +2, calculate Q_1
HOT, FAST OK, reward -10, calculate Q_2
OK, SLOW, OK, reward +1, calculate Q_3

s	a	Q_0	Q_1	Q_2	Q_3
OK	SLOW	0			0.9
OK	FAST	0	1.0		
HOT	SLOW	0			
HOT	FAST	0		-4.6	

$$\begin{aligned}
Q(s,a) &\longleftarrow Q(s,a) + 0.5[R(s,a,s') + 0.8\max_{a'} Q(s',a') - Q(s,a)] \\
Q_1(ok, fast) &\longleftarrow 0.0 + 0.5[2.0 + 0.8\max_{a'} Q(hot,a') - 0.0] = 1 \\
Q_2(hot, fast) &\longleftarrow 0 + 0.5[-10 + 0.8\max_{a'} Q(ok,a') - 0] = -4.6 \\
Q_3(ok, slow) &\longleftarrow 0 + 0.5[1 + 0.8\max_{a'} Q(ok,a') - 0] = 0.9
\end{aligned}$$

End of Exam