CS221 Practice Midterm

Autumn 2012

1 Other Midterms

The following pages are excerpts from similar classes' midterms. The content is similar to what we've been covering this quarter, so that it should be useful for practicing. Note that the topics and terminology differ slightly, so feel free to ignore the questions that we did not cover (e.g., greedy search, Bayes nets). The real midterm will lie somewhere between these problems and the homework problems in style and difficulty.

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.

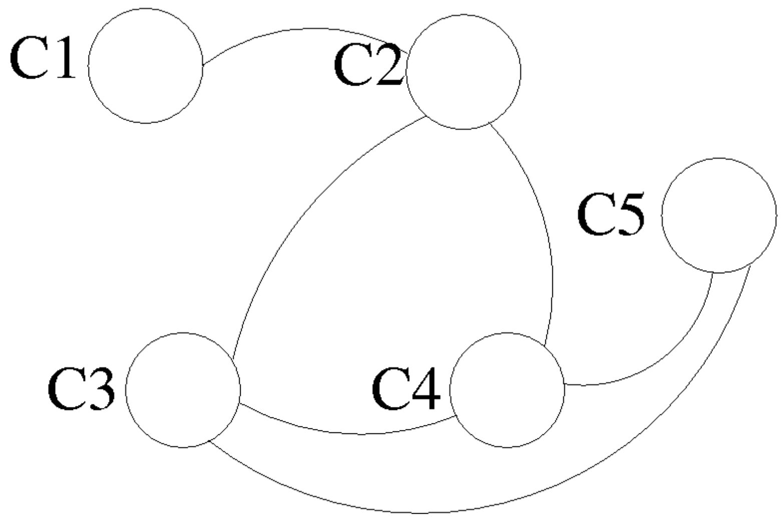
(1) (4 pts): Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

$Variables$	$Domains(orunaryconstraints)$
$C1$	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.

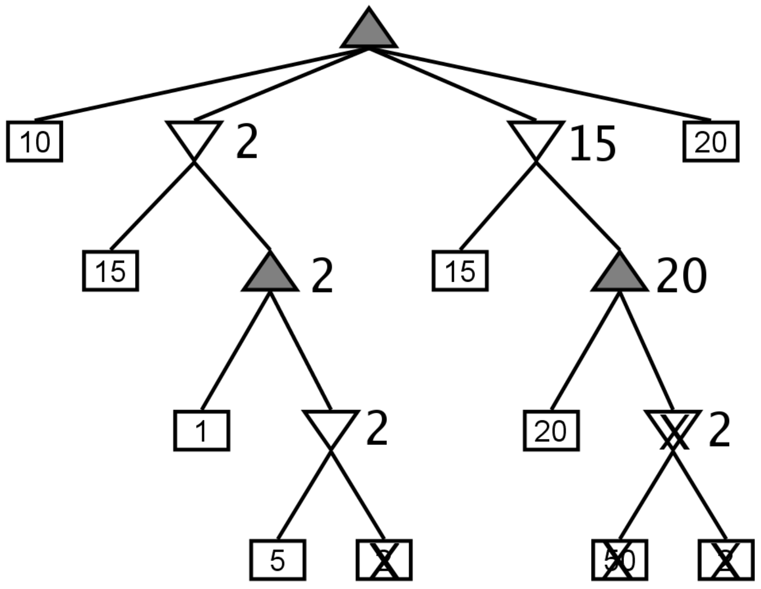
Minimal answer: Can solve them in polynomial time. If a graph is tree structured (i.e. has no loops), then the CSP can be solved in $O(nd^2)$ time as compared to general CSPs, where worst-case time is $O(d^n)$. For tree-structured CSPs you can choose an ordering such that every node's parent precedes 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.

1. (12 points.) Search: Mr. and Ms. Pacman

Pacman and Ms. Pacman are lost in an NxN maze and would like to meet; *they don't care where.* In each time step, *both* simultaneously move in one of the following directions: {NORTH, SOUTH, EAST, WEST, STOP}. They do *not* alternate turns. You must devise a plan which positions them together, somewhere, in as few time steps as possible. Passing each other does not count as meeting; they must occupy the same square at the same time.

(a) (4 points) Formally state this problem as a *single-agent* state-space search problem.

States:

Answer: The set of pairs of positions for Pacman and Ms. Pacman:
$\{((x_1, y_1), (x_2, y_2)) \mid x_1, x_2, y_1, y_2 \in \{1, 2, \ldots, N\}\}$

Maximum size of state space:

Answer: N^2 for both pacmen, hence N^4 total

Maximum branching factor:

Answer: Each pacman has a choice of 5 actions, hence $5^2 = 25$ total

Goal test:

Answer: $isGoal((x_1, y_1), (x_2, y_2)) := (x_1 = x_2) \wedge (y_1 = y_2)$

(b) (3 points) Give a non-trivial admissible heuristic for this problem.

Answer: Manhattan distance between Pacman and Ms. Pacman DIVIDED BY 2 (since both take a step simultaneously)

(c) (3 points) Circle all of the following graph search methods which are guaranteed to output optimal solutions to *this problem*:

(i) DFS
(ii) BFS
(iii) UCS
(iv) A* (with a consistent and admissible heuristic)
(v) A* (with heuristic that returns zero for each state)
(vi) Greedy search (with a consistent and admissible heuristic)

Answer: BFS, UCS, A* (with a consistent and admissible heuristic), A* (with heuristic that returns zero for each state)

(d) (2 points) If h_1 and h_2 are admissible, which of the following are also guaranteed to be admissible? Circle all that apply:

(i) $h_1 + h_2$
(ii) $h_1 * h_2$
(iii) $max(h_1, h_2)$
(iv) $min(h_1, h_2)$
(v) $(\alpha)h_1 + (1 - \alpha)h_2$, for $\alpha \in [0, 1]$

Answer: $max(h_1, h_2)$, $min(h_1, h_2)$, $(\alpha)h_1 + (1 - \alpha)h_2$, for $\alpha \in [0, 1]$

2. (11 points.) CSPs: Finicky Feast

You are designing a menu for a special event. There are several choices, each represented as a variable: (A)ppetizer, (B)everage, main (C)ourse, and (D)essert. The domains of the variables are as follows:

A: (v)eggies, (e)scargot
B: (w)ater, (s)oda, (m)ilk
C: (f)ish, (b)eef, (p)asta
D: (a)pple pie, (i)ce cream, (ch)eese

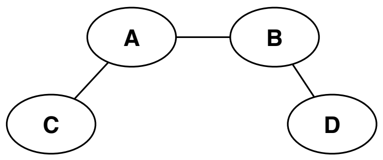
Because all of your guests get the same menu, it must obey the following dietary constraints:

(i) Vegetarian options: The appetizer must be veggies or the main course must be pasta or fish (or both).

(ii) Total budget: If you serve the escargot, you cannot afford any beverage other than water.

(iii) Calcium requirement: You must serve at least one of milk, ice cream, or cheese.

(a) (3 points) Draw the constraint graph over the variables A, B, C, and D.



(b) (2 points) Imagine we first assign A=e. Cross out eliminated values to show the domains of the variables after forward checking.

A	[		e		]
B	[	w	s	m	]
C	[	f	b	p	]
D	[	a	i	ch	]

Answer: The values s, m, and b should be crossed off. “s” and “m” are eliminated due to being incompatible with “e” based on constraint (ii). “b” is eliminated due to constraint (i).

(c) (3 points) Again imagine we first assign A=e. Cross out eliminated values to show the domains of the variables after arc consistency has been enforced.

A	[		e		]
B	[	w	s	m	]
C	[	f	b	p	]
D	[	a	i	ch	]

Answer: The values s, m, b, and a should be eliminated. The first three are crossed off for the reasons above, and “a” is eliminated because there is no value for (B) that is compatible with “a” (based on constraint (iii)).

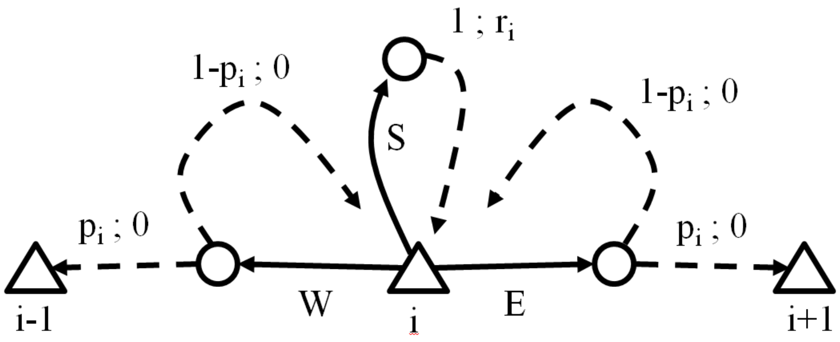
(d) (1 point) Give a solution for this CSP or state that none exists.
Answer: Multiple solutions exist. One is A=e, B=w, C=f, and D=i.

(e) (2 points) For general CSPs, will enforcing arc consistency after an assignment *always* prune at least as many domain values as forward checking? Briefly explain why or why not.
Answer: Two answers are possible:
Yes. The first step of arc consistency is equivalent to forward checking, so arc consistency removes all values that forward checking does.
No. While forward checking is a subset of arc consistency, after any assignment, arc consistency may have already eliminated values in a previous step that are eliminated in that step by forward checking. Thus, enforcing arc consistency will never leave more domain values than enforcing forward checking, but on a given

4. (17 points.) MDPs and RL: Wandering Merchant

There are N cities along a major highway numbered 1 through N. You are a merchant from city 1 (that's where you start). Each day, you can either travel to a neighboring city (actions *East* or *West*) or stay and do business in the current city (action *Stay*). If you choose to travel from city i, you successfully reach the next city with probability p_i, but there is probability $1 - p_i$ that you hit a storm, in which case you waste the day and do not go anywhere. If you stay to do business in city i, you get $r_i > 0$ in reward; a travel day has reward 0 regardless of whether or not you succeed in changing cities.

The diagram below shows the actions and transitions from city i. Solid arrows are actions; dashed arrows are resulting transitions labeled with their probability and reward, in that order.



(a) (2 points) If for all i, $r_i = 1$, $p_i = 1$, and there is a discount $\gamma = 0.5$, what is the value $V^{stay}(1)$ of being in city 1 under the policy that always chooses *stay*? Your answer should be a real number.

Answer: for all cities (states) $i = 1, \ldots, N$, we have that the optimal value behaves as follows:

$$V^{stay}(i) = r_i + \gamma V^{stay}(i)$$

(remember, this is like the Bellman equation for a fixed policy). Plugging in values, we get $V^{stay}(i) = 1 + 0.5V^{stay}(i)$. Now we can just solve for $V^{stay}(i)$ using algebra to obtain $V^{stay}(i) = 2$. In particular, $V^{stay}(1) = 2$.

(b) (2 points) If for all i, $r_i = 1$, $p_i = 1$, and there is a discount $\gamma = 0.5$, what is the optimal value $V^*(1)$ of being in city 1?

Intuitive Answer: since all the cities offer the same reward ($r_i = 1$), there is no incentive to move to another city to do business, so the optimal policy is to always stay, yielding $V^*(1) = 2$.

More Formal Answer:

For all cities (states) $i = 1, \ldots, N$, writing out the Bellman equations:[1]

$$V^*(i) = \max\{\underbrace{r_i + \gamma V^*(i)}_{\text{stay}}, \underbrace{p_i\gamma V^*(i-1) + (1-p_i)\gamma V^*(i)}_{\text{left}}, \underbrace{p_i\gamma V^*(i+1) + (1-p_i)\gamma V^*(i)}_{\text{right}}\}$$

Since $p_i = 1$, this drastically simplifies:

$$V^*(i) = \max\{\underbrace{r_i + \gamma V^*(i)}_{\text{stay}}, \underbrace{\gamma V^*(i-1)}_{\text{left}}, \underbrace{\gamma V^*(i+1)}_{\text{right}}\}$$

From this, we see that $V^*(i)$ is the same for all i, so the max is obtained always with the stay action.

(c) (2 points) If the r_i's and p_i's are known positive numbers and there is almost no discount, i.e. $\gamma \approx 1$, describe the optimal policy. You may define it formally or in words, e.g. "always go east," but your answer

[1]For $i = 1$, omit the left action; for $i = N$, omit the right action.

should precisely define how an agent should act in any given state. *Hint:* You should not need to do any computation to answer this question.

Basically Right Answer: the optimal policy is to always move towards the city with the highest reward. Once there, stay there and do business forever.

Technical Answer: The only complication is due to possible ties. Let $r^* = \max_{1 \le i \le n} r_i$ be the maximum reward out of all the cities. The optimal policy from city i is as follows: if $r_i = r^*$, stay; otherwise, move towards the closest city j that has $r_j = r^*$, where distance between i and $j > i$ is the the expected number of moves to get there ($\sum_{k=i}^{j-1} 1/p_k$).

Suppose we run value iteration. Recall that $V_k(s)$ is the value of state s after k rounds of value iteration and all the values are initialized to zero.

(d) (2 points) If the optimal value of being in city 1 is positive, i.e. $V^*(1) > 0$, what is the largest k for which $V_k(1)$ could still be zero? Be careful of off-by-one errors.

Answer: Assuming $r_i > 0$, then the largest k is 0, because $V_1(s) = \max\{r_i + 0, \cdots\} > 0$.

(Intended) Answer: If we don't assume $r_i > 0$, then the largest k is $N - 1$. Proof: since $V^*(1) > 0$, at least one of the r_i's must be strictly positive. After one iteration, $V_1(i) > 0$; after two iterations, $V_2(i-1) > 0$; finally after i iterations, $V_i(1) > 0$. In the meantime, if all $r_j = 0$ for $j < i$, then $V_j(1) = 0$ for all $j < i$. In the worst case, $i = N$, so $V_{N-1}(1) = 0$ is possible, but $V_N(1) > 0$.

(e) (2 points) If all of the r_i and p_i are positive, what is the largest k for which $V_k(s)$ could still be zero for some state s? Be careful of off-by-one errors.

Answer: Since $r_i > 0$, the largest k is 0, because $V_1(s) = \max\{r_i + 0, \cdots\} > 0$.

Suppose we don't know the r_i's or the p_i's, so we decide to do Q-learning.

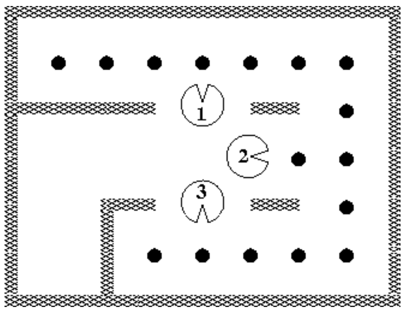
(f) (3 points) Suppose we experience the following sequence of states, actions, and rewards: (s=1, a=*stay*, r=4), (s=1, a=*east*, r=0), (s=2, a=*stay*, r=6), (s=2, a=*west*, r=0), (s=1, a=*stay*, r=4, s=1). What are the resulting $Q(s, a)$ values if the learning rate is 0.5, the discount is 1, and we start with all $Q(s, a) = 0$? Fill in the table below; each row should hold the q-values after the transition specified in its first column. You may leave unchanged values blank.

(s,a,r,s')	Q(1,S)	Q(1,E)	Q(2,W)	Q(2,S)
initial	0	0	0	0
(1,S,4,1)				
(1,E,0,2)				
(2,S,6,2)				
(2,W,0,1)				
(1,S,4,1)				

After $(1, S, 4, 1)$, we update $Q(1, S) \leftarrow 0.5[4 + 1 \cdot 0] + 0.5(0) = 2$.
After $(1, E, 0, 2)$, we update $Q(1, E) \leftarrow 0.5[0 + 1 \cdot 0] + 0.5(0) = 0$.
After $(2, S, 6, 2)$, we update $Q(2, S) \leftarrow 0.5[6 + 1 \cdot 0] + 0.5(0) = 3$.
After $(2, W, 0, 1)$, we update $Q(2, W) \leftarrow 0.5[0 + 1 \cdot 2] + 0.5(0) = 1$.

2. (10 points) Cooperative Pac-Family

Pacman is trying eat all the dots, but he now has the help of his family! There are initially k dots, at positions $(f_1, \ldots f_k)$. There are also n Pac-People, at positions $(p_1, \ldots, p_n)$; initially, all the Pac-People start in the bottom left corner of the maze. Consider a search problem in which all Pac-People move *simultaneously*; that is, in each step each Pac-Person moves into some adjacent position (N, S, E, or W, no STOP). Note that any number of Pac-People may occupy the same position.



(a) (3 pt) Define the state space of the search problem.

Solution:

A state must contain the location of each Pac-Person and a bit for whether each food is eaten $\{(p_1, \ldots, p_n), (b_1, \ldots, b_j)$, for $j <= k\}$

(b) (1 pt) Give a reasonable upper bound on the size of the state space for a general r by c grid.

Solution:

Each Pac-Person can be in rc positions and each food can be eaten or not: $(rc)^n 2^k$

(c) (1 pt) What is the goal test?

Solution:

Number of uneaten food is 0.

(d) (1 pt) What is the maximum branching factor of the successor function in a general grid?

Solution:

4^n. Each Pac-Person has up to 4 possible moves.

(e) (4 pt) Circle the admissible heuristics below (-1/2 point for each mistake.)

i. $h_1(s) = 0$
Solution:
Yes. Trivial, but always admissible.

ii. $h_2(s) = 1$
Solution:
No. Not admissible at the goal.

iii. $h_3(s) =$ number of remaining food / n
Solution:
Yes. Relaxed problem: each Pac-Person eats a food on each move.

iv. $h_4(s) = \max_i \max_j \text{manhattan}(p_i, \text{food}_j)$
Solution:
No. `[<. <.]`

v. $h_5(s) = \max_i \min_j \text{manhattan}(p_i, \text{food}_j)$
Solution:
No. `[<  <.]`

CS 188 Spring 2009 Introduction to Artificial Intelligence Midterm Solutions

1. (16 points) True/False

For the following questions, a correct answer is worth 2 points, no answer is worth 1 point, and an incorrect answer is worth 0 points. Circle *true* or *false* to indicate your answer.

a) (*true* or *false*) If $g(s)$ and $h(s)$ are two admissible A* heuristics, then their average $f(s) = \frac{1}{2}g(s) + \frac{1}{2}h(s)$ must also be admissible.

True. Let $h^*(s)$ be the true distance from s. We know that $g(s) \leq h^*(s)$ and $h(s) \leq h^*(s)$, thus $f(s) = \frac{1}{2}g(s) + \frac{1}{2}h(s) \leq \frac{1}{2}h^*(s) + \frac{1}{2}h^*(s) = h^*(s)$

b) (*true* or *false*) For a search problem, the path returned by uniform cost search may change if we add a positive constant C to every step cost.

True. Consider that there are two paths from the start state (S) to the goal (G), $S \rightarrow A \rightarrow G$ and $S \rightarrow G$. $cost(S, A) = 1$, $cost(A, G) = 1$, and $cost(S, G) = 3$. So the optimal path is through A. Now, if we add 2 to each of the costs, the optimal path is directly from S to G. Since uniform cost search finds the optimal path, its path will change.

c) (*true* or *false*) The running-time of an efficient solver for tree-structured constraint satisfaction problems is linear in the number of variables.

True. The running time of the algorithm for tree-structured CSPs is $O(n \cdot d^2)$, where n is the number of variables and d is the maximum size of any variable's domain.

d) (*true* or *false*) If $h_1(s)$ is a consistent heuristic and $h_2(s)$ is an admissible heuristic, then $\min(h_1(s), h_2(s))$ must be consistent.

False. For instance, if $h_2(s)$ be admissible but inconsistent, and $h_1(s)$ dominate $h_2(s)$, then $\min(h_1(s), h_2(s)) = h_1(s)$, which is inconsistent.

e) (*true* or *false*) The amount of memory required to run minimax with alpha-beta pruning is $O(b^d)$ for branching factor b and depth limit d.

True and **False** (everyone wins). The memory required is only $O(bd)$, so we accepted **False**. However, by definition an algorithm that is $O(bd)$ is also $O(b^d)$, because O denotes upper bounds that may or may not be tight, so technically this statement is *True* (but not very useful).

f) (*true* or *false*) In a Markov decision process with discount $\gamma = 1$, the difference in values for two adjacent states is bounded by the reward between them: $|V(s) - V(s')| \leq \max_a R(s, a, s')$.

False. Let $V(s') = 0$, and $R(s, a, s') = 0 \ \forall a$, but there is an action a' which takes s to the terminal state T and gives a reward 100. Thus $V(s) \geq 100$, but the inequality above says that $|V(s) - 0| \leq 0$.

g) (*true* or *false*) Value iteration and policy iteration must always converge to the same policy.

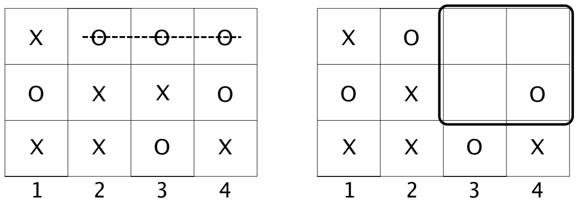
True and **False** (everyone wins). Both algorithms are guaranteed to converge to the optimal policy, so we accepted **True**. If there are multiple policies that are optimal (meaning they yield the same maximal values for every state), then the algorithms might diverge. Both value iteration and policy iteration will always lead to the same optimal values.

h) (*true* or *false*) In a Bayes' net, if $A \perp\!\!\!\perp B$, then $A \perp\!\!\!\perp B \mid C$ for some variable C other than A or B.

False. Consider the Bayes' net $A \rightarrow C \leftarrow B$. Clearly, $A \perp\!\!\!\perp B$, but $A \not\perp\!\!\!\perp B \mid C$.

4. (10 points) Multi-agent Search: Connect-3

In the game of Connect-3, players X and O alternate moves, dropping their symbols into columns 1, 2, 3, or 4. Three-in-a-row wins the game, horizontally, vertically or diagonally. X plays first.



(a) (1 pt) What is the maximum branching factor of minimax search for Connect-3?
4.

(b) (1 pt) What is the maximum tree depth in plies?
6. According to our terminology, a search ply in a multi-agent search problem includes one move for every agent. Because the textbook differs in its definition, we also accepted 12.

(c) (1 pt) Give a reasonably tight upper bound on the number of terminal states. Several answers were accepted:

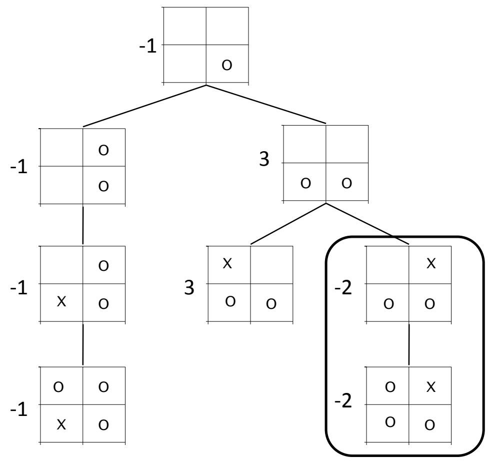
4^{12} is a reasonable bound on the number of leaves of the search tree.

3^{12} is a reasonable bound on the number of total states in the game, because each square can be empty, contain an X or contain an O.

A tighter bound is 15^4, which follows from the observation that there are only 15 ways to legally fill a column from the bottom up with X's and O's.

Bounds based on the assumption that the entire board would be filled with X's and O's were not accepted, even if they in fact overestimated the true number of terminal states.

(d) (2 pt) Draw the game tree starting from the board shown above right (with O to play next). You may abbreviate states by drawing only the upper-right region circled. The root node is drawn for you.



(e) (2 pt) X is the maximizer, while O is the minimizer. X's utility for terminal states is k when X wins and $-k$ when O wins, where k is 1 for a horizontal 3-in-a-row win (as in above left), 2 for a vertical win, and 3 for a diagonal win. A tie has value 0. Label each node of your tree with its minimax value.

(f) (3 pt) Circle all nodes of your tree that will *not* be explored when using alpha-beta pruning and a move ordering that maximizes the number of nodes pruned.