Written Assignment 2

Deadline: Feb 03, 2020

Instruction: You may discuss these problems with classmates, but please complete the write-ups individually. (This applies to BOTH undergraduates and graduate students.) Remember the collaboration guidelines set forth in class: you may meet to discuss problems with classmates, but you may not take any written notes (or electronic notes, or photos, etc.) away from the meeting. Your answers must be **typewritten**, except for figures or diagrams, which may be hand-drawn. Please submit your answers (pdf format only) on **Canvas**.

Q1. Solving CSPs (25 points)

In a combined 3^{rd} and 5^{th} grade class, students can be 8, 9, 10, and 11 years old. We are trying to solve for the ages of **A**nn, **B**ob, **C**laire, and **D**oug (abbreviations: **A**, **B**, **C**, **D**). Consider the following constraints:

- No students is older in years than Clair (but may be the same age)
- Bob is two years older than Ann
- Bob is younger in years than Doug.

Complete the following questions:

1. (5 pts) Draw the constraint graph.
2. (3 pts) Suppose we are using the AC-3 algorithm for arc consistency. How many total arcs will be en-queued when the algorithm begins execution?
3. (9 pts) Assuming all ages $\{8, 9, 10, 11\}$ are possible for each student before running arc consistency, manually run arc consistency on **only** arc from **A** to **B**.
   (a) What values on A remain viable after this operation?
   (b) What values on B remain viable after this operation?
   (c) Assuming there are no arcs left in the list of arcs to be processed, which arc(s) would be added to the queue for processing after this operation?
4. (8 pts) Suppose we enforce arc consistency on all arcs. What ages remain in each person's domain?

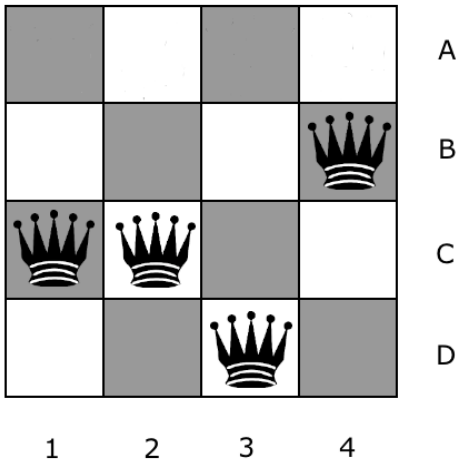
Q2. CSP Properties (True or False Answer) (13 points)

1. Even when using arc consistency, backtracking might be needed to solve a CSP.
2. Even when using forward checking, backtracking might be needed to solve a CSP.
3. When using backtracking search with the same rules to select unassigned variables and to order value assignments (in our case, usually Minimum Remaining Values and Least-Constraining Value, with alphabetical tiebreaking), arc consistency will always give the same solution as forward checking, if the CSP has a solution.
4. For a CSP with binary constraints that has no solution, some initial values may still pass arc consistency before any variable is assigned.
5. If the CSP has no solution, it is guaranteed that enforcement of arc consistency resulted in at least one domain being empty.
6. If the CSP has a solution, then after enforcing arc consistency, you can directly read off the solution from resulting domains.
7. In general, to determine whether the CSP has a solution, enforcing arc consistency alone is not sufficient; backtracking may be required.
8. If after a run of arc consistency during the backtracking search we end up with the filtered domains of *all* of the not yet assigned variables being empty, this means the CSP has no solution.
9. If after a run of arc consistency during the backtracking search we end up with the filtered domains of **all** of the not yet assigned variables being empty, this means the search should backtrack because this particular branch in the search tree has no solution.
10. If after a run of arc consistency during the backtracking search we end up with the filtered domain of *one* of the not yet assigned variables being empty, this means the CSP has no solution.
11. If after a run of arc consistency during the backtracking search we end up with the filtered domains of all of the not yet assigned variables each having exactly one value left, this means we have found a solution.
12. If after a run of arc consistency during the backtracking search we end up with the filtered domains of all of the not yet assigned variables each having more than one value left, this means we have found a whole space of solutions and we can just pick any combination of values still left in the domains and that will be a solution.
13. If after a run of arc consistency during the backtracking search we end up with the filtered domains of all of the not yet assigned variables each having more than one value left, this means we can't know yet whether there is a solution somewhere further down this branch of the tree, and search has to continue down this branch to determine this.

Q3. 4-Queens (12 points)

The min-conflicts algorithm attempts to solve CSPs iteratively. It starts by assigning some value to each of the variables, ignoring the constraints when doing so. Then, while at least one constraint is violated, it repeats the following: (1) randomly choose a variable that is currenly violating a constraint, (2) assign to it the value in its domain such that after the assignment the total number of constraints violated is minimized (among all possible selections of values in its domain).

In this question, you are asked to execute the min-conflicts algorithm on a simple problem: the 4-queens problem in the figure shown below. Each queen is dedicated to its own column (i.e. we have variables Q_1, Q_2, Q_3, and Q_4 and the domain for each one of them is $\{A, B, C, D\}$). In the configuration shown below, we have $Q_1 = C$, $Q_2 = C$, $Q_3 = D$, $Q_4 = B$. Two queens are in conflict if they share the same row, diagonal, or column (though in this setting, they can never share the same column).



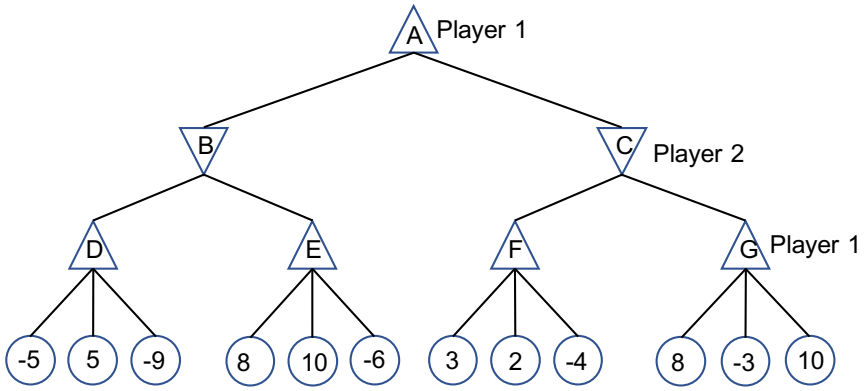
You will execute min-conflicts for this problem three times, starting with the state shown in the figure above. When selecting a variable to reassign, min-conflicts chooses a conflicted variable at random. For this problem, assume that your random number generator always chooses the leftmost conflicted queen. When moving a queen, move it to the square in its column that leads to the fewest conflicts with other queens. If there are ties, choose the topmost square among them.

1. Starting with the queens in the configuration shown in the above figure, which queen will be moved, and where will it be moved to?
2. Continuing off of Part 1, which queen will be moved, and where will it be moved to?
3. Continuing off of Part 2, which queen will be moved, and where will it be moved to?

Q4. Adversarial Search (25 points)

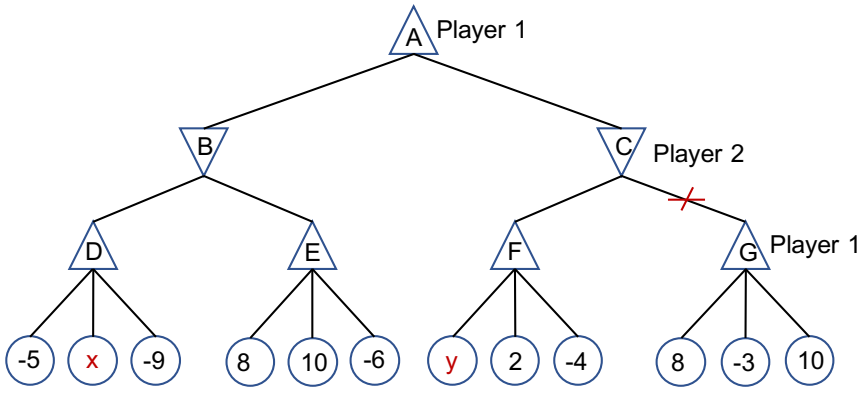
Q4.1. Search Algorithms. (15 points) Consider the zero-sum game tree shown below. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing player. Outcome values for

the maximizing player are listed for each leaf node, represented by the values in circles at the bottom of the tree.



1. (5 points) Assuming both players act optimally, carry out the minimax search algorithm. Enter the values for the letter nodes.
2. (10 points) Assuming both players act optimally, use alpha-beta pruning to find the value of the root node. The search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child.
   - Enter the values of the letter nodes.
   - Select the leaf nodes that don't get visited due to pruning.

Q4.2. Unknown Leaf Nodes. (10 points) Consider the following game tree with the same setting as Q4.1 except two of the leaves has an unknown payoff, x and y.



For what values of x and y such that the pruning shown in the figure will be achieved? The search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child. Please specify your answer in one of the following forms:

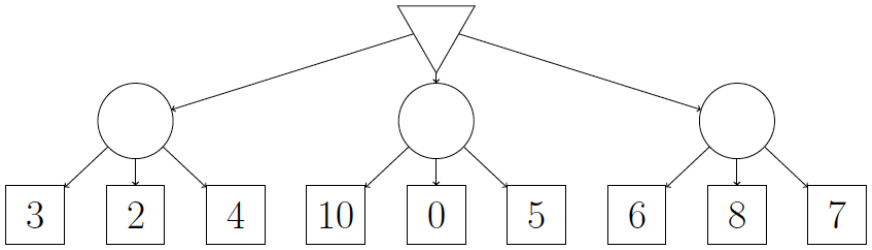
- Write All if x can take on all values
- Write None if x has no possible values

- Use an inequality in the form $x < \{value\}$, $x > \{value\}$, or $\{value1\} < x < \{value2\}$ to specify an interval of values. As an example, if you think x can take on all values larger than 16, you should enter $x > 16$.

Q5. Expectimin (25 points)

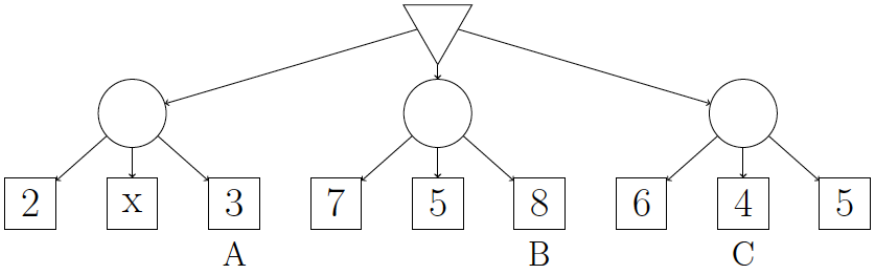
In this problem we model a game with a minimizing player and a random player. We call this combination "expectimin" to contrast it with expectimax with a maximizing and random player. Assume all children of expectation nodes have equal probability and sibling nodes are visited left to right for all parts of this question.

(a) **[4 pts]** Fill out the "expectimin" tree below.



(b) **[9 pts]** Suppose that before solving the game we are given additional information that all values are non-negative and all nodes have exactly 3 children. Which leaf nodes in the tree above can be pruned with this information?

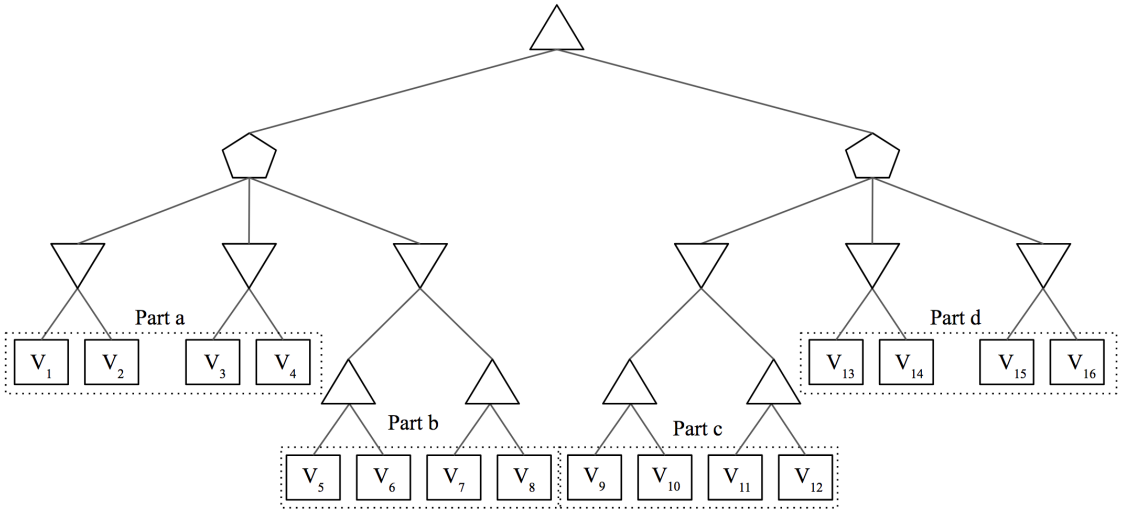
(c) **[12 pts]** For each of the leaves labeled A, B, and C in the tree below, determine which values of x will cause the leaf to be pruned, given the information that all values are non-negative and all nodes have 3 children. Assume we do not prune on equality.



- A:
- B:
- C:

Q6. (Grads Only) (25 points)

You're living in utopia! Despite living in utopia, you still believe that you need to maximize your utility in life, other people want to minimize your utility, and the world is a zero-sum game. But because you live in utopia, a benevolent social planner occasionally steps in and chooses an option that is a compromise. Essentially, the social planner (represented as the pentagon) is a median node that chooses the successor with median utility. Your struggle with your fellow citizens can be modelled as follows:



There are some nodes that we are sometimes able to prune. In each part, mark all of the terminal nodes such that **there exists a possible situation** for which the node **can be pruned**. In other words, you must consider **all** possible pruning situations. Assume that evaluation order is **left to right** and all Vi's are **distinct**. Provide explanations of your choices.

Note that as long as there exists ANY pruning situation (does not have to be the same situation for every node), you should mark the node as prunable. Also, alpha-beta pruning does not apply here, simply prune a sub-tree when you can reason that its value will not affect your final utility.

Part a.	(1) V_1	(2) V_2	(3) V_3	(4) V_4	(5) None
Part b.	(1) V_5	(2) V_6	(3) V_7	(4) V_8	(5) None
Part c.	(1) V_9	(2) V_{10}	(3) V_{11}	(4) V_{12}	(5) None
Part d.	(1) V_{13}	(2) V_{14}	(3) V_{15}	(4) V_{16}	(5) None