CS188 Fall 2018 Section 3: CSPs + Games

1 CSP: Air Traffic Control

We have five planes: A, B, C, D, and E and two runways: international and domestic. We would like to schedule a time slot and runway for each aircraft to **either** land or take off. We have four time slots: $\{1, 2, 3, 4\}$ for each runway, during which we can schedule a landing or take off of a plane. We must find an assignment that meets the following constraints:

- Plane B has lost an engine and must land in time slot 1.
- Plane D can only arrive at the airport to land during or after time slot 3.
- Plane A is running low on fuel but can last until at most time slot 2.
- Plane D must land before plane C takes off, because some passengers must transfer from D to C.
- No two aircrafts can reserve the same time slot for the same runway.

a) Complete the formulation of this problem as a CSP in terms of variables, domains, and constraints (both unary and binary). Constraints should be expressed implicitly using mathematical or logical notation rather than with words.

Variables: A, B, C, D, E for each plane.

Domains: a tuple *(runway type, time slot)* for runway type $\in \{international, domestic\}$ and time slot $\in \{1, 2, 3, 4\}$.

Constraints:

$$B[1] = 1$$

$$D[1] \geq 3$$

$$A[1] \leq 2$$

$$D[1] < C[1]$$

$$A \neq B \neq C \neq D \neq E$$

b) For the following subparts, we add the following two constraints:

- Planes A, B, and C cater to international flights and can only use the international runway.
- Planes D and E cater to domestic flights and can only use the domestic runway.

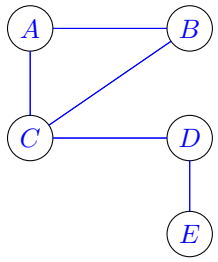
i With the addition of the two constraints above, we completely reformulate the CSP. You are given the variables and domains of the new formulation. Complete the constraint graph for this problem given the original constraints and the two added ones.

Variables: A, B, C, D, E for each plane.

Domains: $\{1, 2, 3, 4\}$

Explanation of Constraint Graph:
We can now encode the runway information into the identity of the variable, since each runway has more than enough time slots for the planes it serves. We represent the non-colliding time slot constraint as a binary constraint between the planes that use the same runways.

Constraint Graph:



ii What are the domains of the variables after enforcing arc-consistency? Begin by enforcing unary constraints. (Cross out values that are no longer in the domain.)

A	~~1~~	2	~~3~~	~~4~~
B	1	~~2~~	~~3~~	~~4~~
C	~~1~~	~~2~~	~~3~~	4
D	~~1~~	~~2~~	3	~~4~~
E	1	2	~~3~~	4

iii Arc-consistency can be rather expensive to enforce, and we believe that we can obtain faster solutions using only **forward-checking** on our variable assignments. Using the Minimum Remaining Values heuristic, perform backtracking search on the graph, breaking ties by picking lower values and characters first. List the $(variable, assignment)$ pairs in the order they occur (including the assignments that are reverted upon reaching a dead end). Enforce unary constraints before starting the search.

(You don't have to use this table, it won't be graded.)

A	1	2	3	4
B	1	2	3	4
C	1	2	3	4
D	1	2	3	4
E	1	2	3	4

Answer: (B, 1), (A, 2), (C, 3), (C, 4), (D, 3), (E, 1)

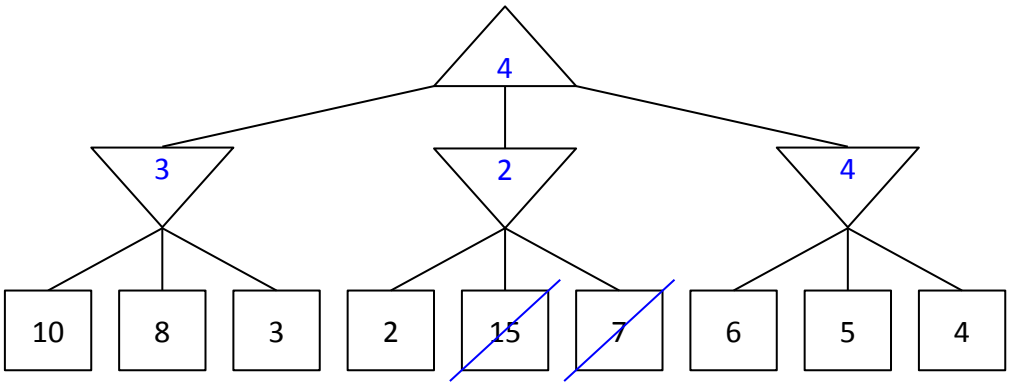
c) Suppose we have just one runway and n planes, where no two planes can use the runway at once. We are assured that the constraint graph will always be tree-structured and that a solution exists. What is the runtime complexity in terms of the number of planes, n, of a CSP solver that runs arc-consistency and then assigns variables in a topological ordering?

$O(n^3)$. Modified AC-3 for tree-structured CSPs runs arc-consistency backwards and then assigns variables in forward topological (linearized) ordering so we that we don't have to backtrack. The runtime complexity of modified AC-3 for tree-structured CSPs is $O(nd^2)$, but note that the domain of each variable must have a domain of size at least n since a solution exists

2 Games

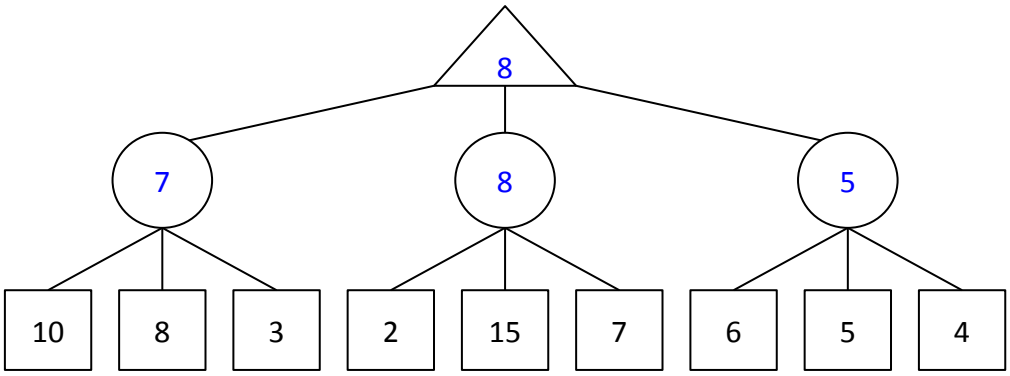
1. Consider the zero-sum game tree shown below. Triangles that point up, such as at the top node (root), represent choices for the maximizing player; triangles that point down represent choices for the minimizing

player. Assuming both players act optimally, fill in the minimax value of each node.



2. Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. Assume the search goes from left to right; when choosing which child to visit first, choose the left-most unvisited child.

3. (optional) Again, consider the same zero-sum game tree, except that now, instead of a minimizing player, we have a chance node that will select one of the three values uniformly at random. Fill in the expectimax value of each node. The game tree is redrawn below for your convenience.

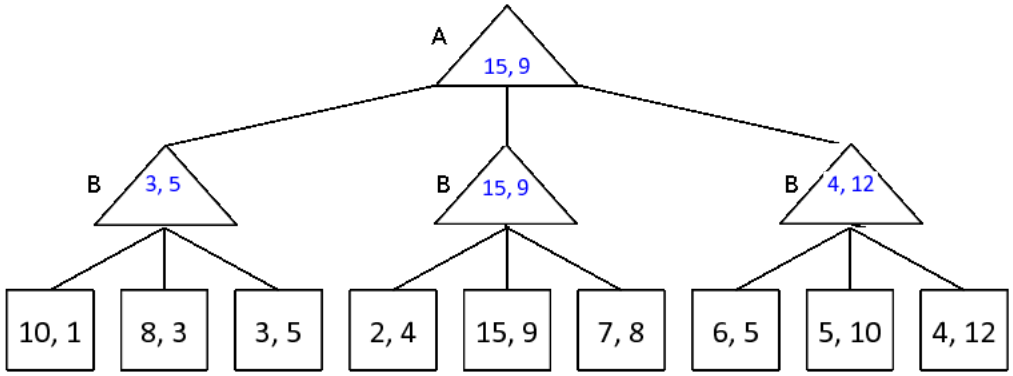


4. (optional) Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. No nodes can be pruned. There will always be the possibility that some leaf further down the branch will have a very high value, which increases the overall average value.

3 (Optional) Nonzero-sum Games

1. Let's look at a non-zero-sum version of a game. In this formulation, player A's utility will be represented as the first of the two leaf numbers, and player B's utility will be represented as the second of the two leaf

numbers. Fill in this non-zero game tree assuming each player is acting optimally.



2. Which nodes can be pruned from the game tree above through alpha-beta pruning? If no nodes can be pruned, explain why not. No nodes can be pruned. Because this game is non-zero-sum, there can exist a leaf node anywhere in the tree that is good for both player A and player B.