Problem 1) Consider the tic-tac-toe board state shown in Figure 1. Draw the full minimax search tree starting from this state, and ending in terminal nodes. Show the utility value for each terminal and non-terminal node. Also show which move the Minimax algorithm decides to play. Utility values are +1 if X wins, 0 for a tie, and -1 if O wins. Assume that X makes the next move (X is the MAX player).

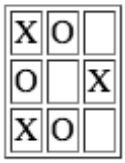
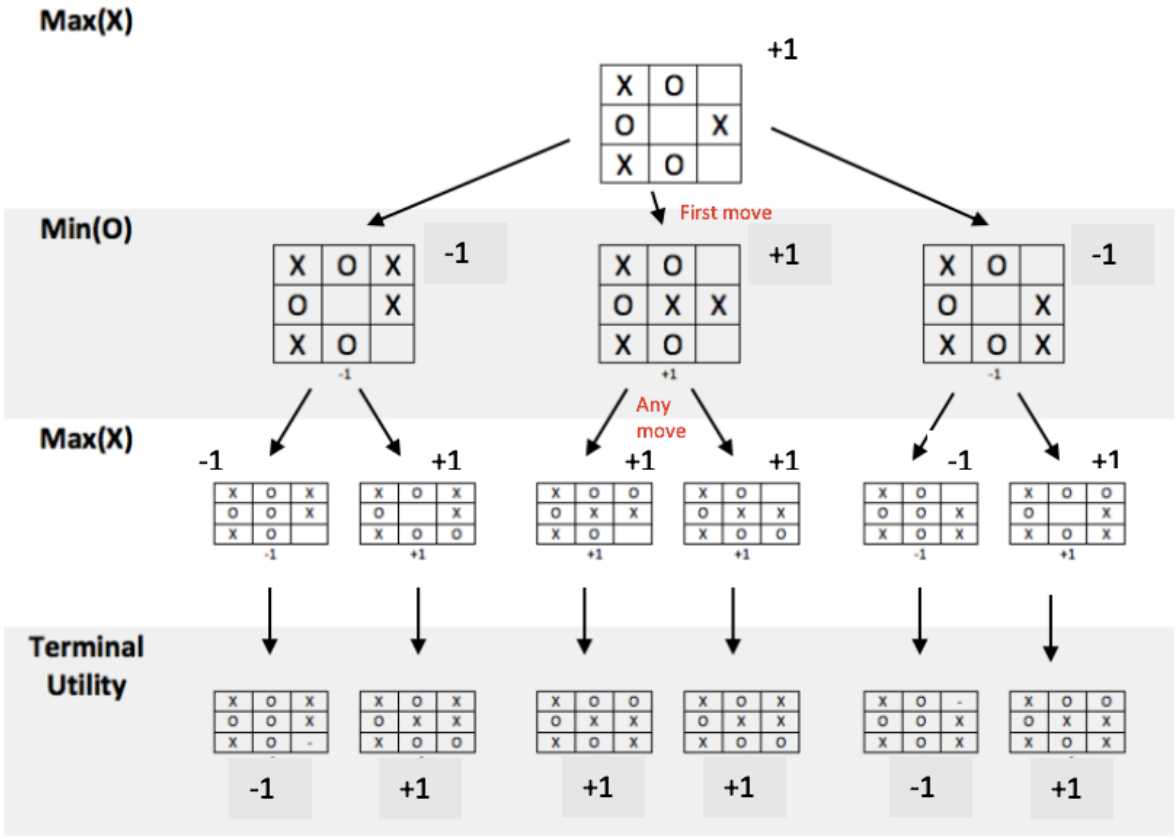
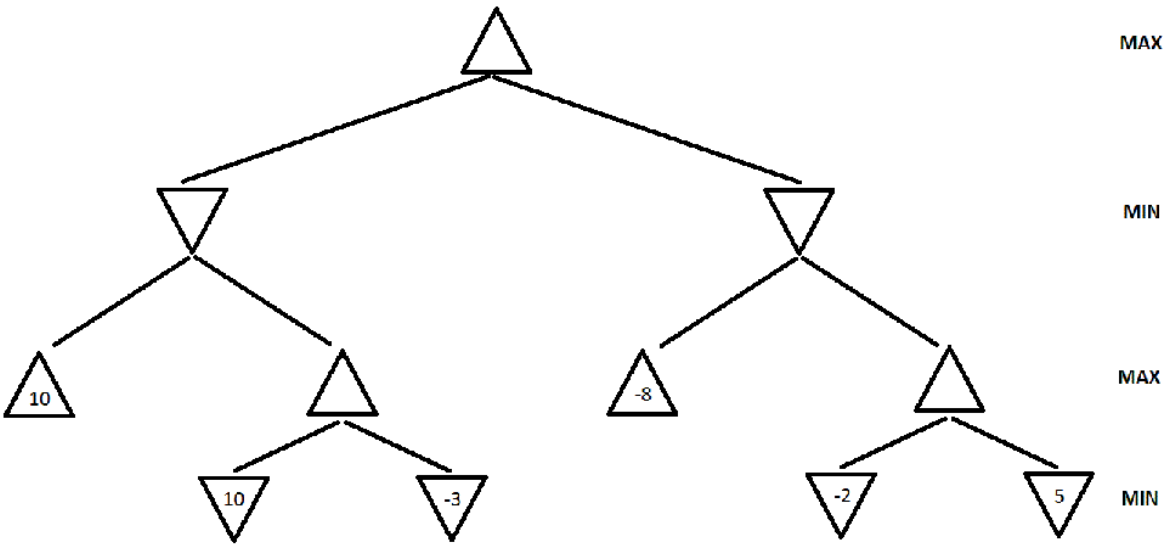


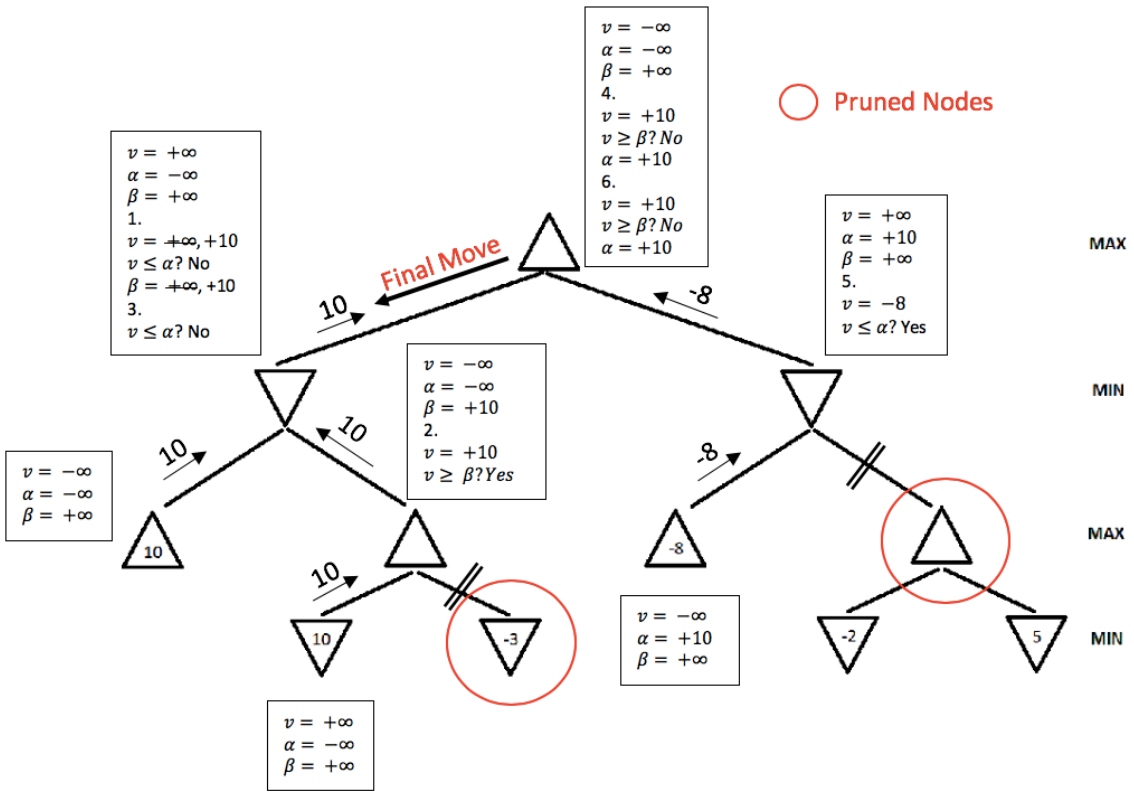
Figure 1. A tic-tac-toe board state.



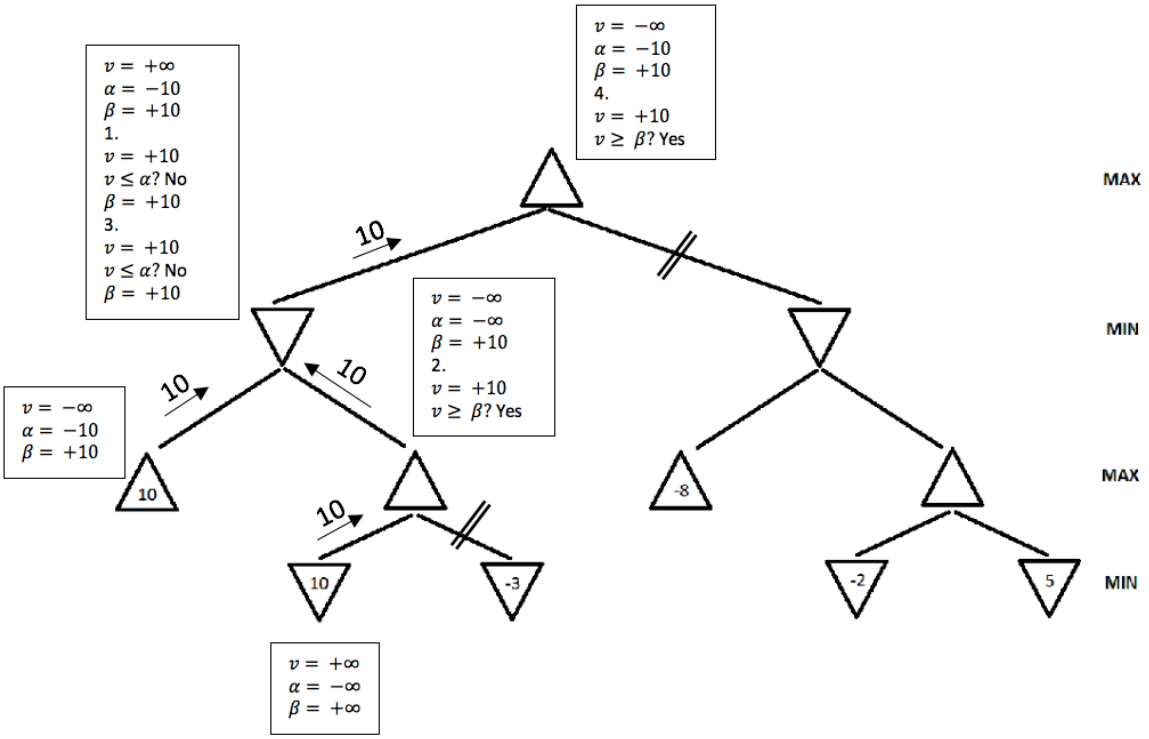
Problem 2)



A) In the game search tree of Figure 2, indicate what nodes will be pruned using alpha-beta search, and what the estimated utility values are for the rest of the nodes. Assume that, when given a choice, alpha-beta search expands nodes in a left-to-right order. Also, assume the MAX player plays first. Finally indicate which action the Minmax algorithm will pick to execute.



B) This question is also on the game search tree of Figure 2. Suppose we are given some additional knowledge about the game: the maximum utility value is 10, i.e., it is not mathematically possible for the MAX player to get an outcome greater than 10. How can this knowledge be used to further improve the efficiency of alpha-beta search? Indicate the nodes that will be pruned using this improvement. Again, assume that, when given a choice, alpha-beta search expands nodes in a left-to-right order, and that the MAX player plays first.



Problem 3)

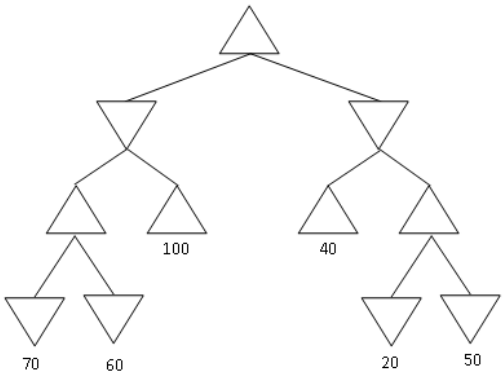


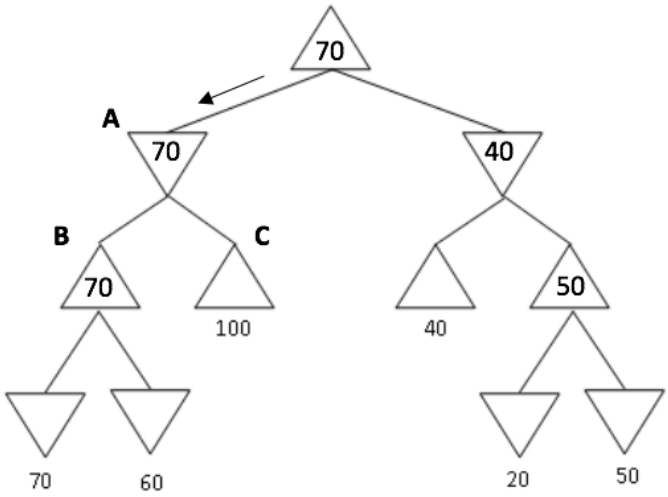
Figure 3: Yet another game search tree

Consider the MINIMAX tree above. Suppose that we are the MAX player, and we follow the MINIMAX algorithm to play a full game against an opponent. However, *we do not know what algorithm the opponent uses.*

Under these conditions, what is the best possible outcome of playing the full game for the MAX player? What is the worst possible outcome for the MAX player? Justify your answer.

NOTE: the question is not asking you about what MINIMAX will compute for the start node. It is asking you what the best and worst outcome of a complete game under the assumptions is stated above.

Applying MINMAX algorithm for the given graph:



- This shows that the MAX player would make a move towards 70 points based on MINMAX algorithm, as shown in the figure.
- The best possible outcome is MAX winning 100 points. This happens only when MIN plays a move towards its right child, that is from Node A to Node C.
- The worst possible outcome is MAX winning 70 points. This happens only then MIN plays a move towards its left child, that is from Node A to Node B. At Node B, the algorithm would select the best route, thus winning 70 points.

Problem 4) Find the value of every non-terminal node in the expectiminmax tree given above. Also indicate which action will be performed by the algorithm.

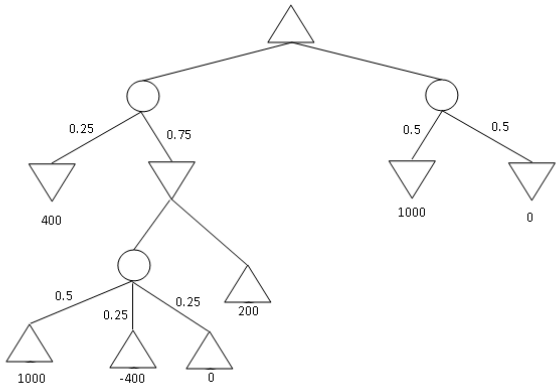
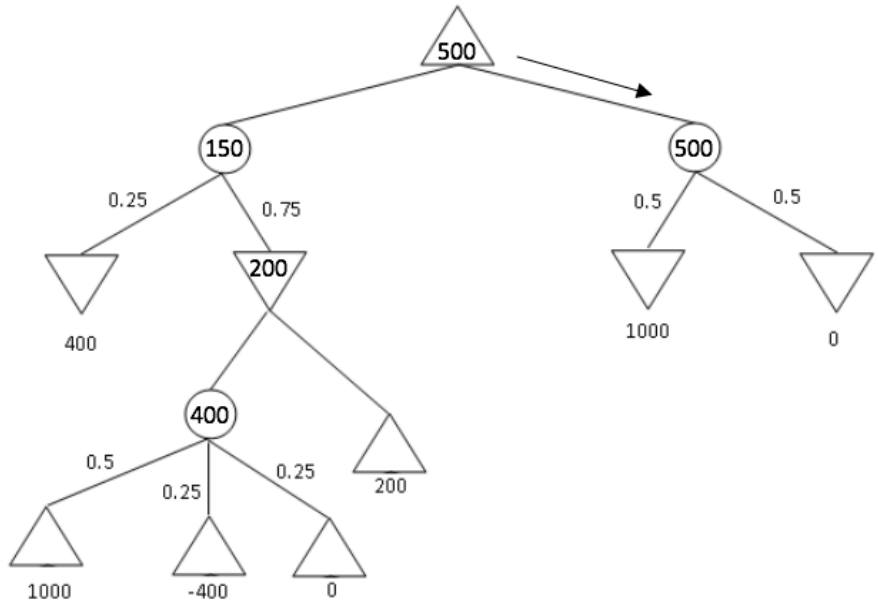


Figure 4: An Expectiminmax tree.



Problem 5) The following outline map needs to be colored. Your job is to color the various sections such that no two sections sharing a border have the same color. You are allowed to use the colors (Red, Green, Blue).

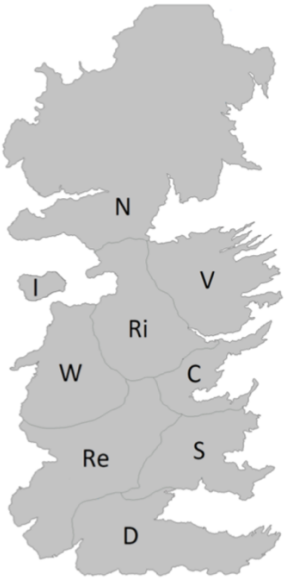
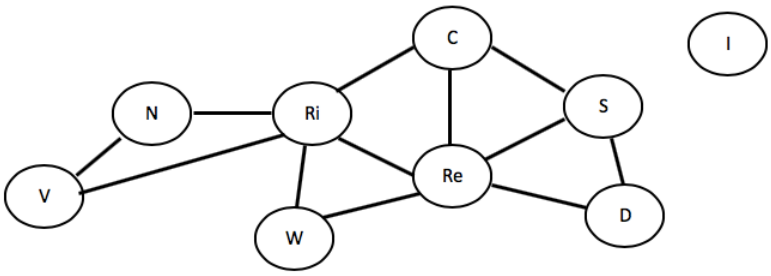


Figure 5: Map to be colored.

A. Draw the Constraint Graph for this problem. Can you use this information to simplify the problem?



B. ***Assuming you are using Backtracking search to solve this problem and that you are using both MRV and Degree heuristic to select the variable, Which variable will be selected at each level of the search tree [You do not need to draw the tree. Just let me know which variable will be selected and why (MRV and degree values)]. Note: Multiple possible answers. You only have to give one.***

From the graph above we can infer the following:

State	*Number of Constrains*
N	2
Ri	5
V	2
C	3
W	2
Re	5
S	3
D	2
I	0

In Minimum Remaining Value (MRV), the variable with least remaining value is selected. And in Degree Heuristic, we select the variable based on maximum number of constrains.

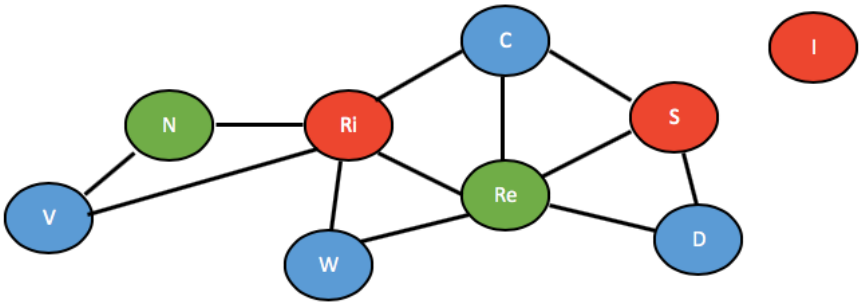
Sorting the above table based on number of constrains.

State	*Number of Constrains*
Ri	5
Re	5
C	3
S	3
N	2
V	2
W	2
D	2
I	0

State	*Ri*	*Re*	*C*	*S*	*N*	*V*	*W*	*D*	*I*
Initial	R G B	R G B	R G B	R G B	R G B	R G B	R G B	R G B	R G B
Level 0	R	G B	G B	R G B	G B	G B	G B	R G B	R G B
Level 1	R	G	B	R B	G B	G B	B	R B	R G B
Level 2	R	G	B	R	G B	G B	B	R B	R G B
Level 3	R	G	B	R	G B	G B	B	B	R G B
Level 4	R	G	B	R	G B	G B	B	B	R G B
Level 5	R	G	B	R	G B	G B	B	B	R G B
Level 6	R	G	B	R	G	B	B	B	R G B
Level 7	R	G	B	R	G	B	B	B	R G B
Level 8	R	G	B	R	G	B	B	B	R
Final	R	G	B	R	G	B	B	B	R

Notes:
Level 0 – Selecting based on maximum number of constrains
Level 1 – There are 5 variables (Re, C, N, V, W) with MRV. Selecting variable Re based on number of constrains.
Level 2 – There are 3 variables (C, S, W) with MRV. Selecting variable C based on number of constrains.
Level 3 – There are 2 variables (S, W) with MRV. Selecting variable S based on number of constrains.
Level 4 – There are 2 variables (W, D) with MRV. Selecting variable W based on number of constrains.
Level 5 – There is a single variable D with MRV. Selecting variable D.
Level 6 – There are 2 variables (N, V) with MRV. Selecting variable N based on number of constrains.
Level 7 - There is a single variable V with MRV. Selecting variable V.
Level 8 - There is a single variable I with MRV. Selecting variable I.

C. ***Give one valid solution to this problem.***
We have got a valid solution by solving the above problem. Representing the same in constrain graph.



Problem 6) Suppose that you want to implement an algorithm that will compete on a two-player deterministic game of perfect information. Your opponent is a supercomputer called DeepGreen. DeepGreen does not use Minimax. You are given a library function DeepGreenMove(S), that takes any state S as an argument, and returns the move that DeepGreen will choose for that state S (more precisely, DeepGreenMove (S) returns the state resulting from the opponent's move).

Write an algorithm in pseudocode (following the style of the Minimax pseudocode) that will always make an optimal decision given the knowledge we have about DeepGreen. You are free to use the library function DeepGreenMove(S) in your pseudocode. How does this compare to Minimax wrt optimality of solution and the number of states explored.

```
function min-max-decision (state) returns an action
    inputs: state, the current state in the game.
    returns an action 'a' from actions[state] based on max-value (state)

function max-value (state) returns a utility value
    inputs: state, the current state in the game.
    If Terminal-Test(state) then return Utility(state)
    v <- -∞
```

for a, s in Successors(state) do v <- Max(v, min-max-decision (DeepGreenMove(s)))
return v

Here we do not have a function to compute the Min-value as the same is being done by the opponent function DeepGreenMove(s). Therefore, for the MAX node we always need to find the maximum value returned by DeepGreenMove() for all its successors.

Here our Bot would still be running on MinMax algorithm. The only thing changes is the Min moves are decided by the function DeepGreenMove.

In the Minmax algorithm we used to explore all the nodes to find the best actions for the parent. But here, we only need to expand the action nodes and their first successors, i.e the MIN nodes.