CS 340 Home Work # 3 Solution
CLO 3

Total marks: 70

Q 1. (15 Marks) For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples where appropriate.

(a) A hill-climbing algorithm that never visits states with lower value (or higher cost) is guaranteed to find the optimal solution if given enough time to find a solution.
(b) Suppose the temperature schedule for simulated annealing is set to be constant up to time N and zero thereafter. For any finite problem, we can set N large enough so that the algorithm is returns an optimal solution with probability 1.
(c) For any local-search problem, hill-climbing will return a global optimum if the algorithm is run starting at any state that is a neighbor of a neighbor of a globally optimal state.
(d) Stochastic hill climbing is guaranteed to arrive at a global optimum.

Solution.

(a) False. Such an algorithm will reach a local optimum and stop (or wander on a plateau).
(b) False. If the temperature is fixed the algorithm always has a certain amount of randomness, so when it stops there is no guarantee it will be in an optimal state. Cooling slowly towards zero is the key to optimality.
(c) False. The intervening neighbor could be a local minimum and the current state is on another slope leading to a local maximum. Consider, for example, starting at the state valued 5 in the linear sequence 7,6,5,0,9,0,0.
(d) False. Stochastic hill climbing chooses from the uphill moves at random, but, unlike simulated annealing, it stops if there are none, so it still gets trapped in local optima.

Q2. (20 marks) Generate a large number of 8-puzzle and 8-queens instances and solve them (where possible) by **hill climbing (steepest-ascent and first-choice variants), hill climbing with random restart, and simulated annealing**. Measure the search cost and percentage of solved problems and graph these against the optimal solution cost. Submit your results and comments on the differences observed by you between the used local search methods.

Q3. (10 Marks) Consider the problem of solving two 8-puzzles: there are two 8-puzzle boards, you can make a move on either one, and the goal is to solve both.
(a) Give a complete problem formulation.
(b) How large is the reachable state space? Give an exact numerical expression.

Solution

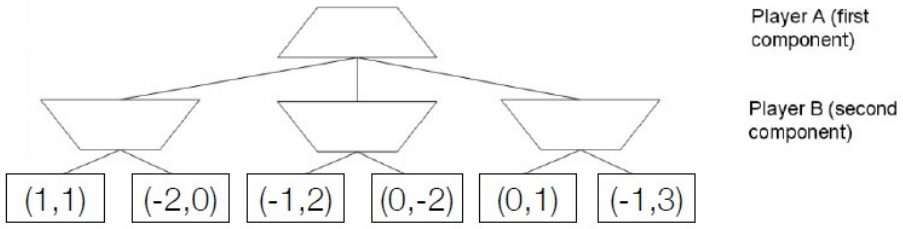
a. Initial state: two arbitrary 8-puzzle states. Successor function: one move on an unsolved puzzle. (You could also have actions that change both puzzles at the same time; this is OK but technically you have to say what happens when one is solved but not the other.) Goal test: both puzzles in goal state. Path cost: 1 per move.

b. Each puzzle has $9!/2$ reachable states (remember that half the states are unreachable). The joint state space has $(9!)^2/4$ states.

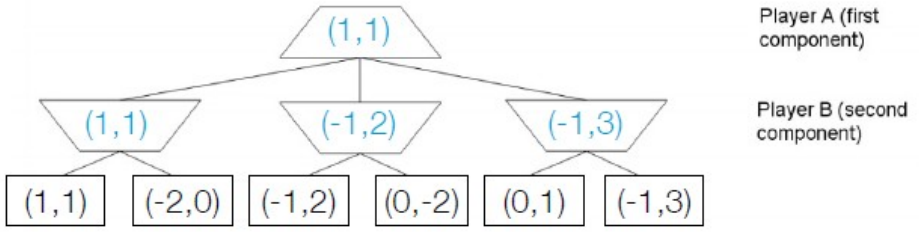
Q4. (10 Marks)

The standard Minimax algorithm calculates worst-case values in a *zero-sum* two player game, i.e. a game in which for all terminal states s, the utilities for players A (MAX) and B (MIN) obey $U_A(s) + U_B(s) = 0$. In the zero sum case, we know that $U_A(s) = -U_B(s)$ and so we can think of player B as simply minimizing $U_A(s)$.

In this problem, you will consider the *non zero-sum* generalization in which the sum of the two players' utilities are not necessarily zero. Because player A's utility no longer determines player B's utility exactly, the leaf utilities are written as pairs $(U_A; U_B)$, with the first and second component indicating the utility of that leaf to A and B respectively. In this generalized setting, A seeks to maximize U_A, the first component, while B seeks to maximize U_B, the second component.



1. Propagate the terminal utility pairs up the tree using the appropriate generalization of the minimax algorithm on this game tree. Fill in the values (as pairs) at each of the internal node. Assume that each player maximizes their own utility.



2. Briefly explain why no alpha-beta style pruning is possible in the general non-zero sum case. *Hint*: think first about the case where $U_A(s) = U_B(s)$ for all nodes.

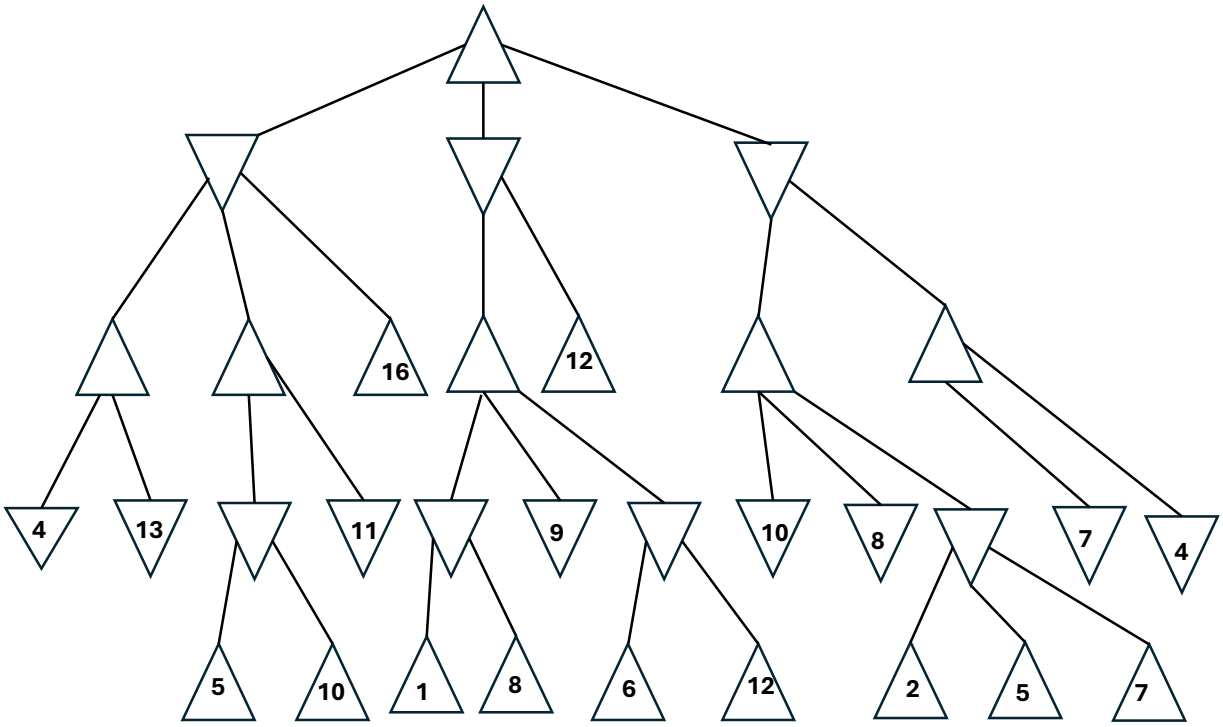
The values that the first and second player are trying to maximize are independent, so we no longer have situations where we know that one player will never let the other player down a particular branch of the game tree.

For instance, in the case where $U_A = U_B$, the problem reduces to searching for the max-valued leaf, which could appear anywhere in the tree.

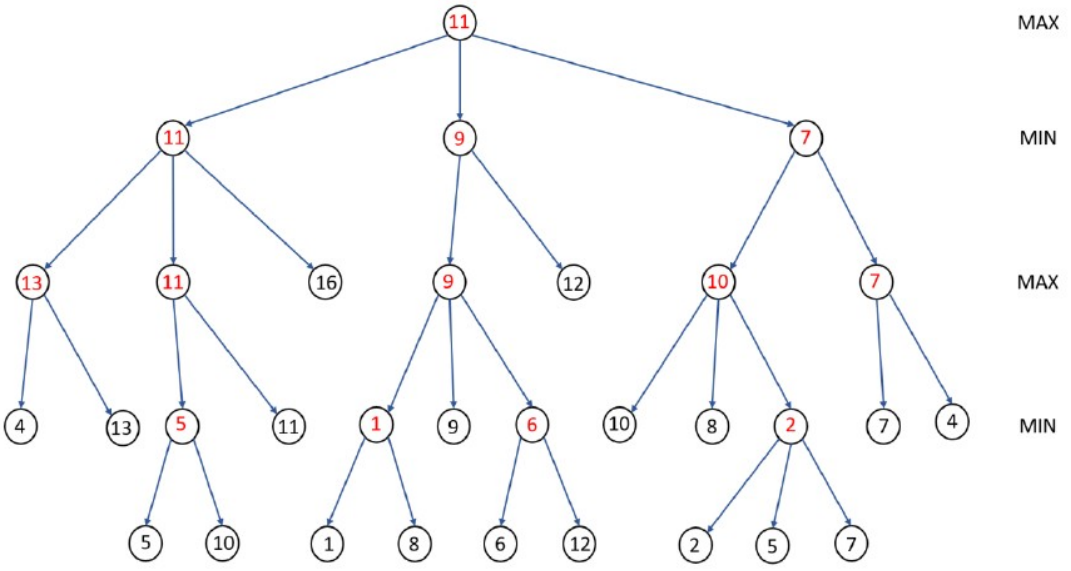
Q5. (20 Marks)

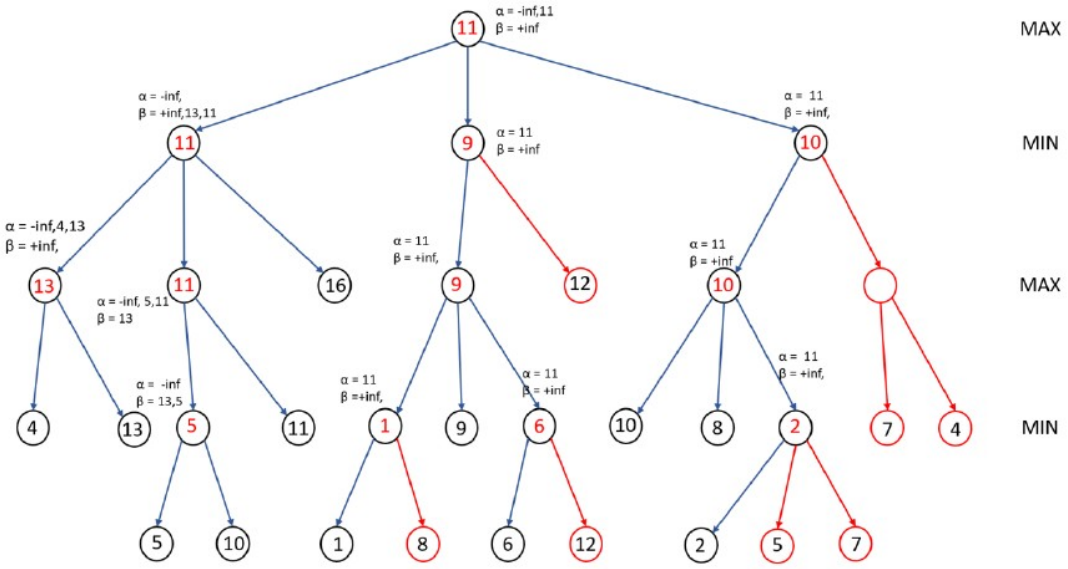
(a) Use the Minimax algorithm to compute the minimax value at each node for the game tree given below. (5 marks)

(b) Use the Alpha-Beta pruning algorithm to prune the game tree assuming child nodes are visited from left to right. Show all final alpha and beta values computed at root, each internal node explored, and at the top of pruned branches. Also show the pruned branches.



[Ans]:




11
α = -inf,11
β = +inf
MAX
α = -inf,
β = +inf,13,11
11
9
α = 11
β = +inf
α = 11
β = +inf,
10
MIN
α = -inf,4,13
β = +inf,
13
α = -inf, 5,11
β = 13
11
16
α = 11
β = +inf,
9
12
α = 11
β = +inf,
10
MAX
α = -inf
β = 13,5
α = 11
β =+inf,
α = 11
β = +inf
α = 11
β = +inf,
4
13
5
11
1
9
6
10
8
2
7
4
MIN
5
10
1
8
6
12
2
5
7