National University of Singapore
School of Computing
CS3243: Foundations of Artificial Intelligence
Solutions for Tutorial 3

Readings: AIMA Chapters 4 & 6

1. Consider the 8-puzzle that we discussed in class. Suppose we define a new heuristic function h_3 which is the average of h_1 and h_2, and another heuristic function h_4 which is the sum of h_1 and h_2. That is,

$$h_3 = \frac{h_1 + h_2}{2}$$
$$h_4 = h_1 + h_2$$

where h_1 and h_2 are defined as "the number of misplaced tiles", and "the sum of the distances of the tiles from their goal positions", respectively. Are h_3 and h_4 admissible? If admissible, compare their dominance with respect to h_1 and h_2.

Since $h_1 \leq h_2$*,*

$$h_3 = \frac{h_1 + h_2}{2} \leq \frac{h_2 + h_2}{2} = h_2 \leq h^*$$

hence h_3 *is admissible. Since* $h_1 = \frac{h_1+h_1}{2} \leq \frac{h_1+h_2}{2} = h_3$ *we have* $h_1 \leq h3 \leq h2$*. That is,* h_2 *dominates* h_3*, and* h_3 *dominates* h_1*.*

On the other hand, h_4 *is not admissible. Consider a board in which moving one tile will reach the goal. In this case,* $h_1 = h_2 = h^* = 1$*, and*

$$h4 = h1 + h2 = 1 + 1 > h^*$$

2. Refer to the Figure 1 below. Apply the best-first search algorithm to find a path from Fagaras to Craiova, using the following evaluation function $f(n)$:

$$f(n) = g(n) + h(n)$$

where $h(n) = |h_{SLD}(\text{Craiov}) - h_{SLD}(n)|$ and $h_{SLD}(n)$ is the straight-line distance from any city n to Bucharest given in Figure 4.1. Trace the best-first search algorithm by showing the series of search trees as each node is expanded, based on the `TREE-SEARCH` algorithm below. Prove that $h(n)$ is an admissible heuristic.

function TREE-SEARCH(*problem, fringe*) **returns** a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  loop do
    if *fringe* is empty **then return** failure
    node ← REMOVE-FRONT(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **return** *node*
    fringe ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

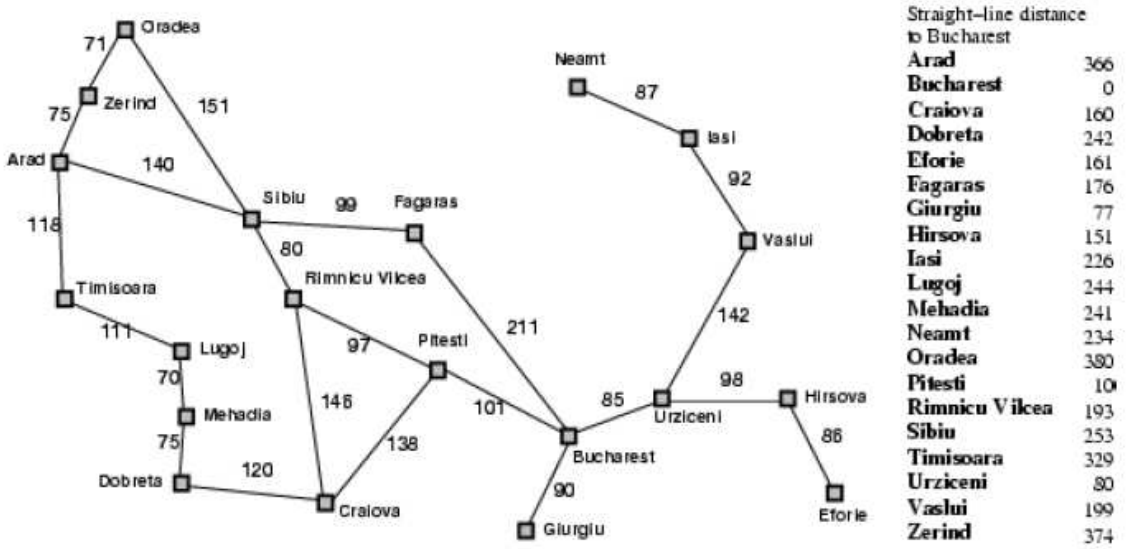
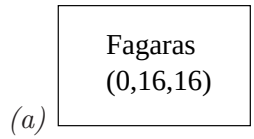


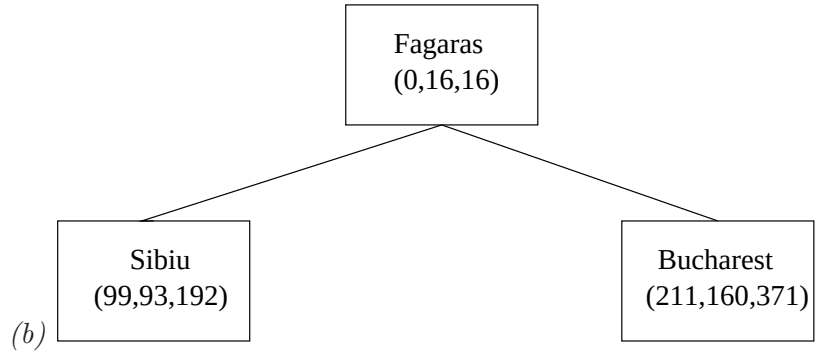
Figure 1: Graph of Romania.

The series of search trees, where the 3-tuple in each node denotes (g, h, f)*:*

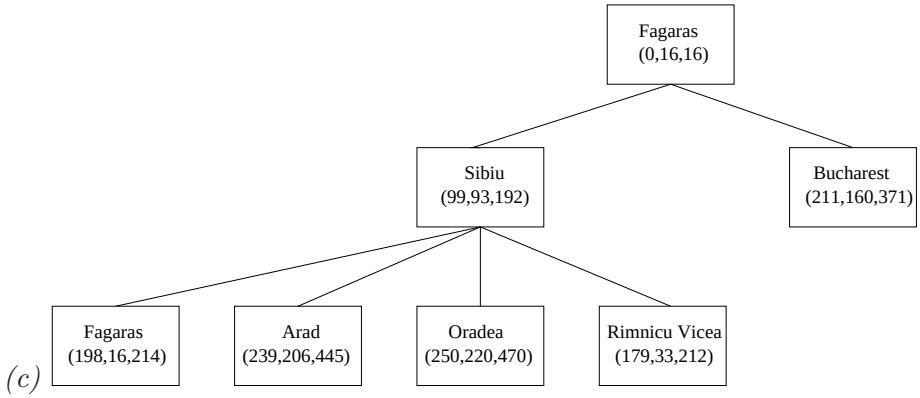
(a)

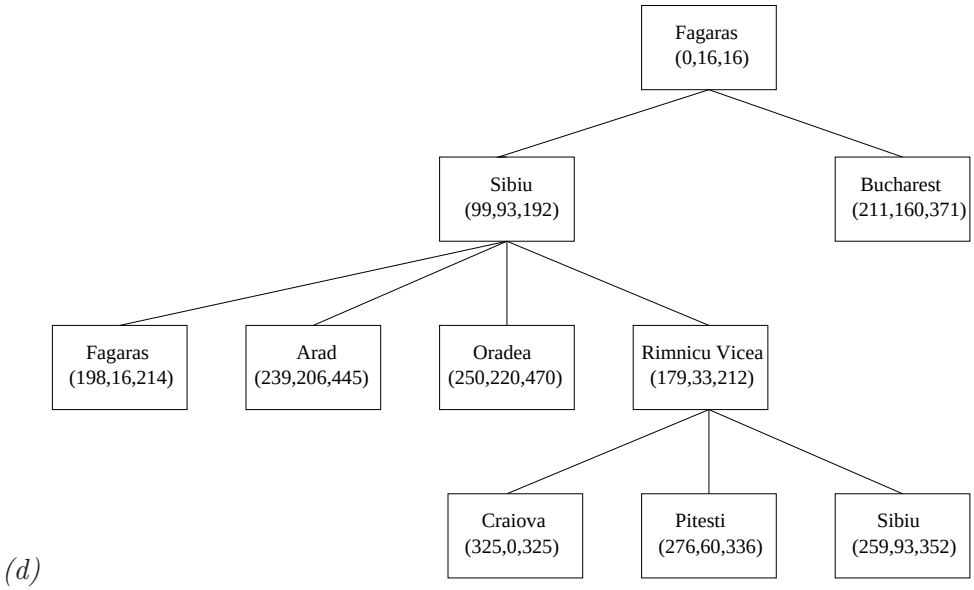


(b)

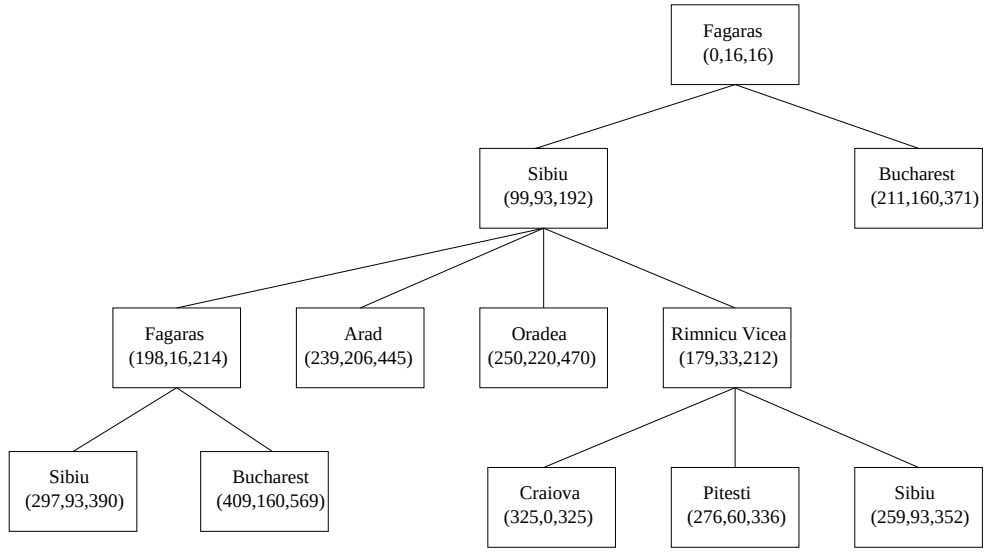


(c)





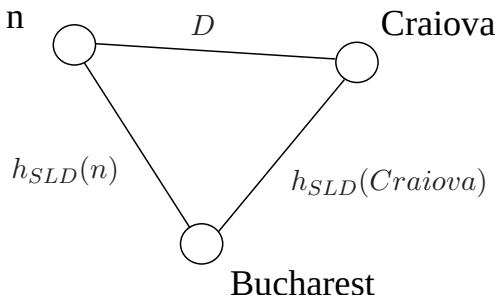
(d)



(e)

Note the we need to expand Faragas even though we had reached the destination the third step because getting to the destination is not sufficient. A finds the optimal solution.*

Now consider the following triangle:



Let D be the straight-line distance between n and Craiova. From the above triangle, we can see by the Triangle Inequality that

$$D + h_{SLD}(n) \geq h_{SLD}(Craiova)$$

and

$$D + h_{SLD}(Craiova) \geq h_{SLD}(n)$$

Hence,

$$D \geq h_{SLD}(Craiova) - h_{SLD}(n)$$

and

$$D \geq -(h_{SLD}(Craiova) - h_{SLD}(n))$$

which is equivalent to $D \geq |h_{SLD}(Craiova) - h_{SLD}(n)|$. Hence, $D \geq h(n)$. But we also know that $h^*(n) \geq D$, so $h^*(n) \geq h(n)$, and $h(n)$ is admissible.

3. (a) Given that a heuristic h is such that $h(G) = 0$, where G is any goal state, prove that if h is consistent, then it must be admissible.

The proof is by induction on k, the number of actions from a node n to the goal node G.

Base step: *$k = 1$, i.e., node n is one step from G. Since the heuristic function h is consistent,*

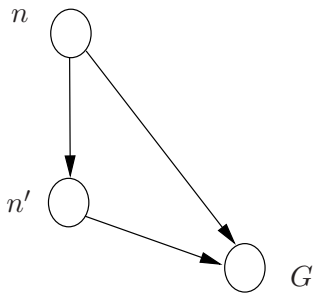
$$h(n) \leq c(n, a, G) + h(G)$$

Since $h(G) = 0$,

$$h(n) \leq c(n, a, G) = h^*(n)$$

Therefore, h is admissible.

Induction step:



Assume: If $h(n')$ is consistent, then $h(n')$ is admissible, for all nodes n' that are k steps from G.

Now consider node n which is $k + 1$ steps from G. If h is a consistent heuristic,

$$h(n) \leq c(n, a, n') + h(n')$$

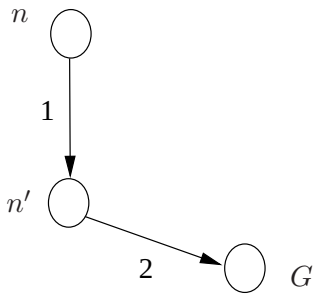
Since n' is k steps from G and h is admissible for node n' that is k steps from G,

$$\begin{aligned} h(n') &\leq h^*(n') \\ \Rightarrow c(n, a, n') + h(n') &\leq c(n, a, n') + h^*(n') \\ \Rightarrow h(n) &\leq c(n, a, n') + h^*(n') = h^*(n) \end{aligned}$$

Hence h is admissible for node n.

(b) Give an example of an admissible heuristic function that is not consistent.

An example of an admissible heuristic function that is not consistent is as follows: $h(n) = 3, h(n') = 1$. *h is admissible since* $h(n) \leq h^*(n) = 1 + 2 = 3$, *and* $h(n') \leq h^*(n') = 2$. *h is not consistent since* $3 = h(n) > c(n, a, n') + h(n') = 1 + 1 = 2$.



(c) Is it possible for a heuristic to be consistent and yet not admissible? If not, prove it. If it is, define one such heuristic.

Yes. Take a heuristic h_1 that is both consistent and admissible, such that $h_1(G) = 0$. This means that following condition is satisfied for all nodes n and n':

$$h_1(n) \leq c(n, a, n') + h_1(n')$$

Consider the heuristic $h_2 = h_1 + 1$. Clearly,

$$h_2(n) \leq c(n, a, n') + h_2(n')$$

is satisfied for all nodes n and n' as well and h_2 is consistent. However, $h_2(G) = 1 > 0$ and h_2 is not admissible.

4. Assume that we have the following initial state and goal state for the 8-puzzle game. We will use h_1 defined as "the number of misplaced tiles" to evaluate each state.

1	2	8
	4	3
7	6	5

initial state

1	2	3
8		4
7	6	5

goal state

(a) Apply the hill-climbing search algorithm in Figure 4.11 (reproduced below). Can the algorithm reach the goal state?

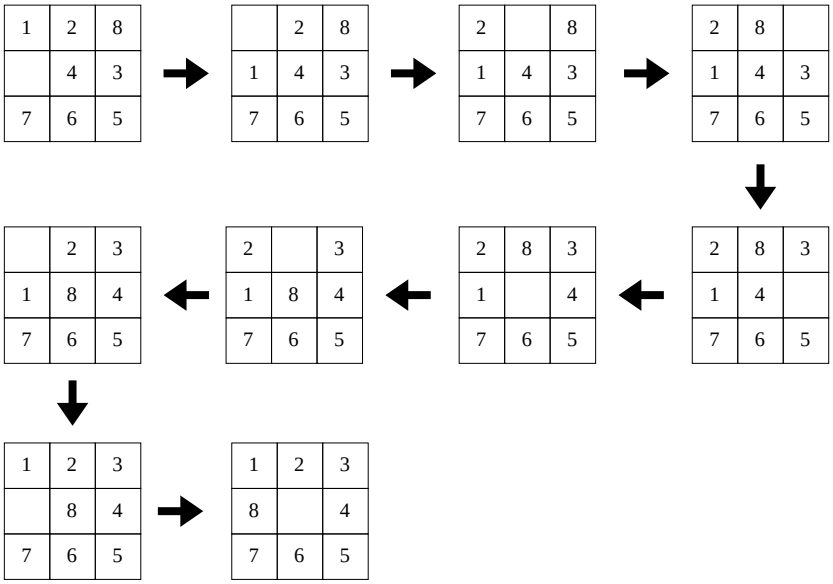
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
       neighbor ← a highest-valued successor of current
       if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
       current ← neighbor
   end
```

According to Figure 4.11, the hill-climbing search algorithm will move to a neighboring state only if the neighboring state is better. Since no successor of the initial state has a better h_1 value, the algorithm is stuck and is not able to reach the goal state.

(b) Identify a sequence of actions leading from the initial state to the goal state. Is it possible for simulated annealing to find such a solution?

The following sequence of actions leads from the initial state to the goal state:



Simulated annealing allows an action that leads to a worse value to be taken with some probability. Thus it is possible for simulated annealing to find the above solution.

5. Consider Figure 6.1 in the textbook (reproduced in Figure 2). The Tic-Tac-Toe search space can actually be reduced by means of symmetry. This is done by eliminating those states which become identical with an earlier state after a symmetry operation (e.g. rotation). The following diagram shows a reduced state space for the first three levels with the player making the first move using "x" and the opponent making the next move with "o". Assume that the following heuristic evaluation function is used at each leaf node n:

$$Eval(n) = P(n) - O(n)$$

where $P(n)$ is the number of winning lines for the player while $O(n)$ is the number of winning lines for the opponent. A winning line for the player is a line (horizontal, vertical or diagonal)

that either contains nothing or "x". For the opponent, it is either nothing or "o" in the winning line. Thus, for the leftmost leaf node in Figure 3, $Eval(n) = 6 - 5 = 1$.

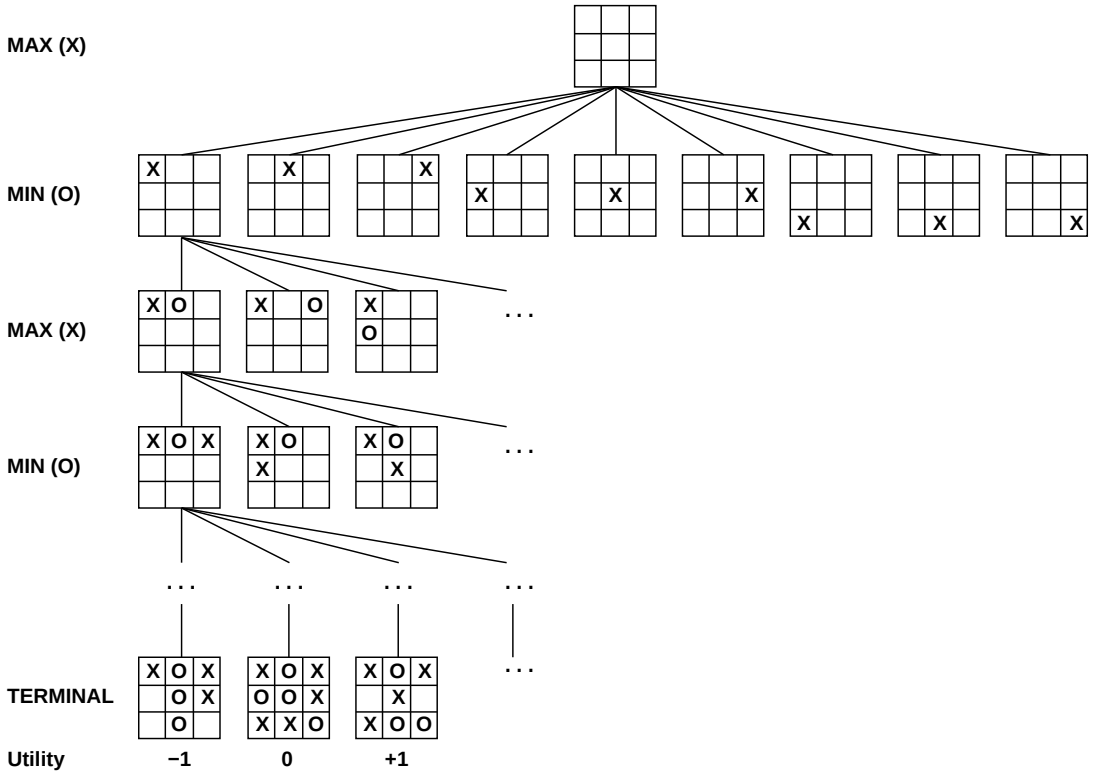


Figure 2: Search space for Tic-Tac-Toe.

(a) Use the minimax algorithm to determine the first move of the player, searching 2-ply deep search space shown in Figure 3.

(b) Assume that the "x" player will now make his second move after his opponent has placed an "o". Complete the following minimax tree in Figure 4 by filling the remaining blank boards at the leaf nodes. Compute the evaluation function for each of the filled leaf nodes and determine the second move of the "x" player (searching 2-ply deep).

(c) The minimax search tree in Figure 5 has heuristic evaluation function values with respect to the max player for all the leaf nodes, where square leaf nodes denote end of game with $+\infty$ representing that the max player wins the game and $-\infty$ representing that the min player is the winner. Do a minimax search and determine the next move of the max player from node A. Which is the target leaf node that the max player hopes to reach?

As shown in Figure 5, the max player will move from A to B. He hopes to reach J.

(d) Suppose we use alpha-beta pruning in the direction from left to right to prune the search tree in question 3. Indicate which arcs are pruned by the procedure. Do you get the same answer in terms of the max player's next move and target leaf node?

The arcs pruned away by alpha-beta pruning are indicated as "X" above in Figure 6.

The arc EL can be pruned since whatever value that branch returns, node E will have a minimax value of at least 7, which does not influence the minimax value at node B since

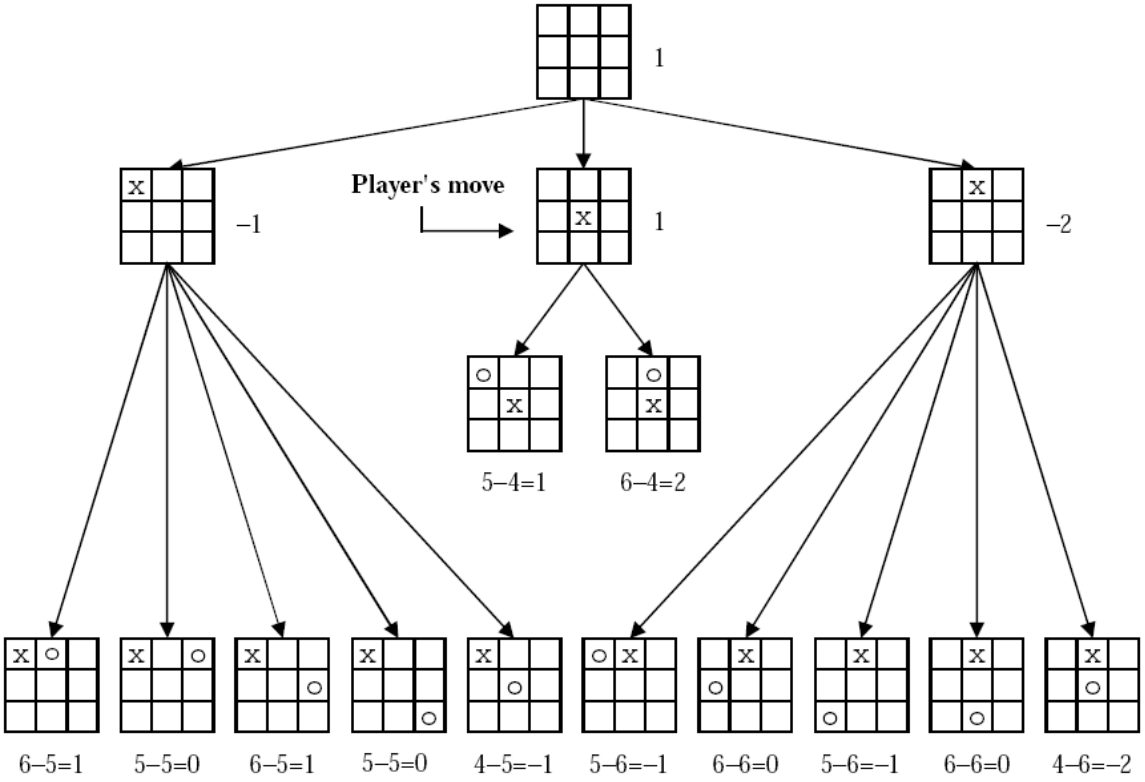


Figure 3: 2-ply deep search space

node B will always have the minimax value of 5 from node D. Similarly, the arc CG can be pruned, since node C will always have the minimax value of from node F regardless of what value is returned from the branch CG.

The max player's next move (B) and target leaf node (J) are still the same. That is, alpha-beta pruning does not alter the outcome.

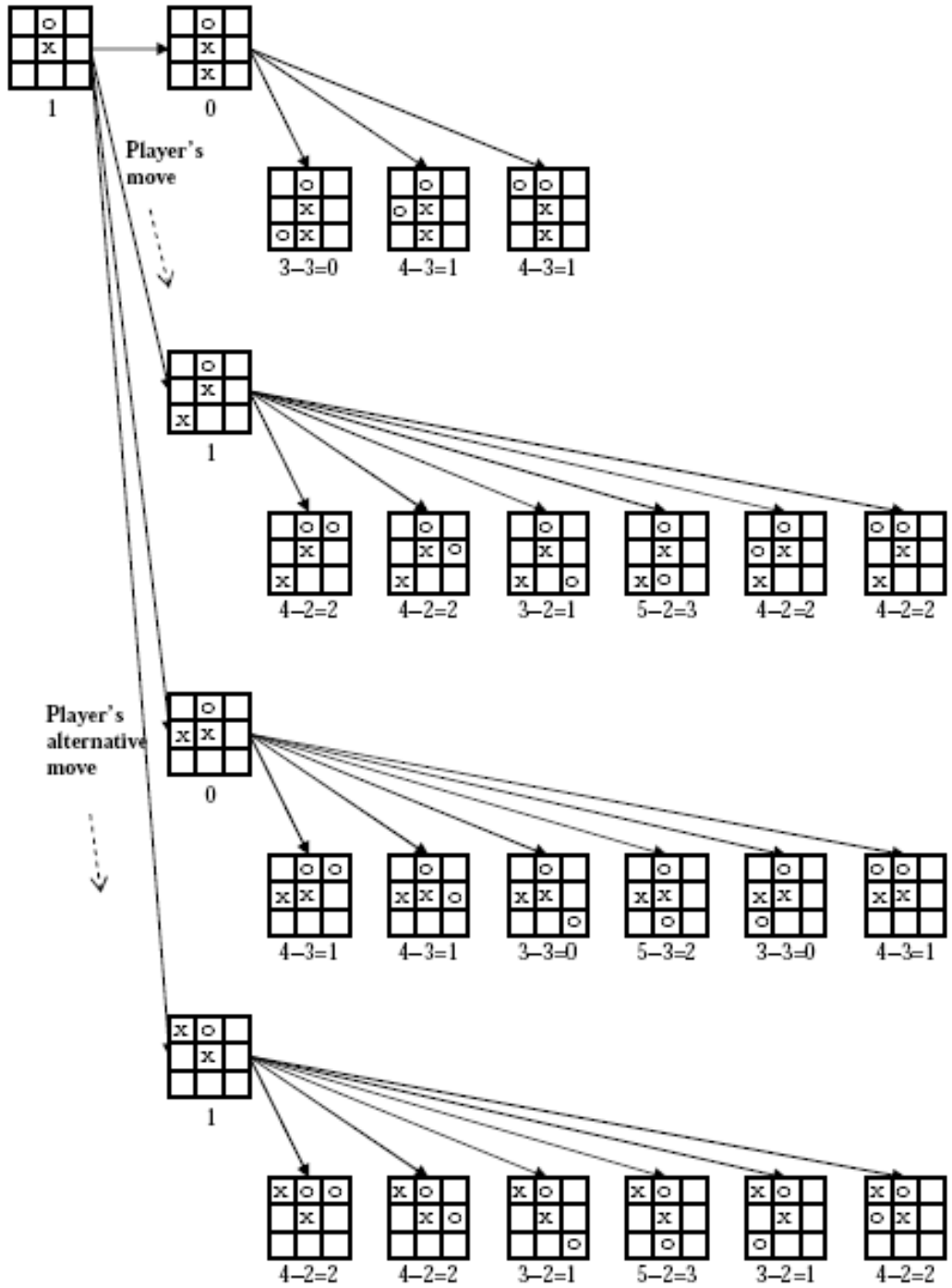


Figure 4: 3-ply deep search space

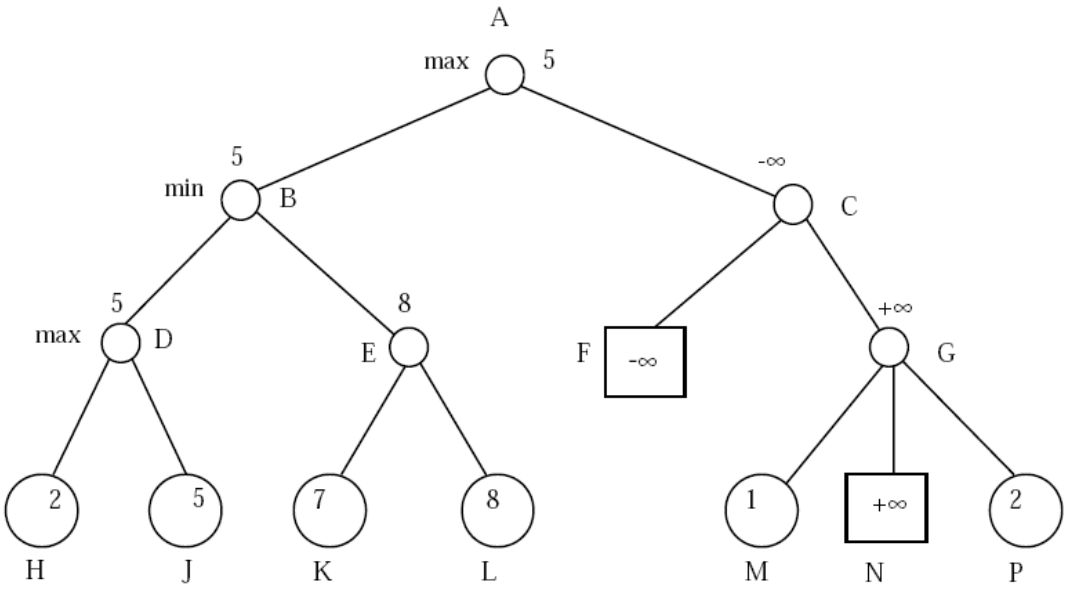


Figure 5: minimax search tree

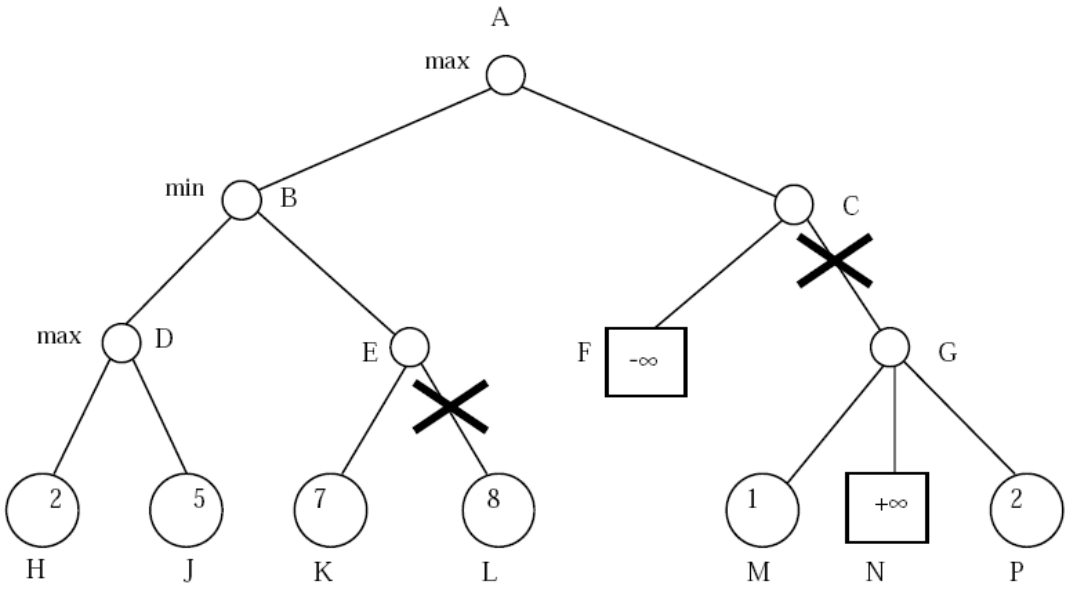


Figure 6: minimax search tree with alpha-beta pruning