Artificial Intelligence (AI)

CCS-3880 – 3rd Semester 2023

CO6: Games (Adversarial) as Search

Dr. Abdullah Alshanqiti

Games (Adversarial) as Search

CO6: Investigate the adversarial (game) search and its algorithms.

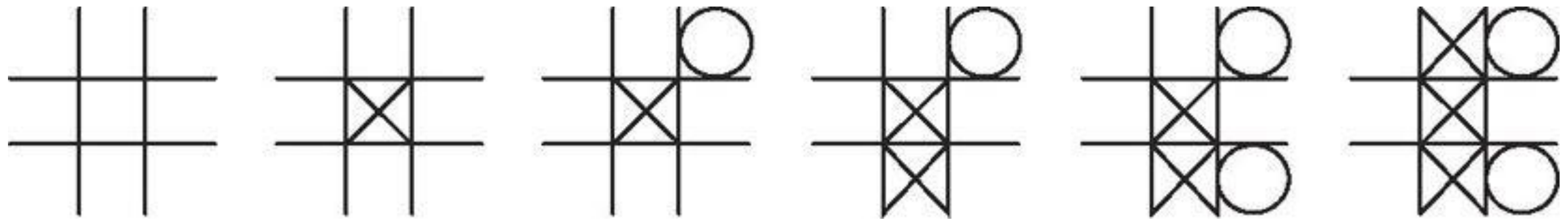
Outline

- Games as a Search Problem
- Optimal Decisions in Games
- Game Trees and Minimax Evaluation
- Minimax with Alpha-Beta Pruning

Adversarial Search

- We examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.
- In this lecture we will cover competitive environments, in which the agents' goals are in conflict, giving rise to adversarial search problems—often known as **games**.

Game playing introduces an additional challenge: an **adversary** who is trying to impede your advancement.



Games as a Search Problem

A game can be defined as a search problem with the following elements:

- S_0: The **initial state**, which specifies how the game is set up at the start.
- PLAYER(s): Defines which player has the move in a state.
- ACTIONS(s): Returns the set of legal moves in a state.
- RESULT(s,a): The **transition model**, which defines the result of a move.
- TERMINAL-TEST(s): A **terminal test**, which is true when the game is over and false otherwise.
- U T I L I T Y (s, p): A **utility function** (also called an objective function), defines the final numeric value for a game that ends in terminal *state s* for a player *p*.

Games

The initial state, ACTIONS function, and RESULT function define the **game tree**—a tree where the nodes are game states, and the edges are moves.

Figure 5.1 shows part of the game tree for tic-tac-toe.

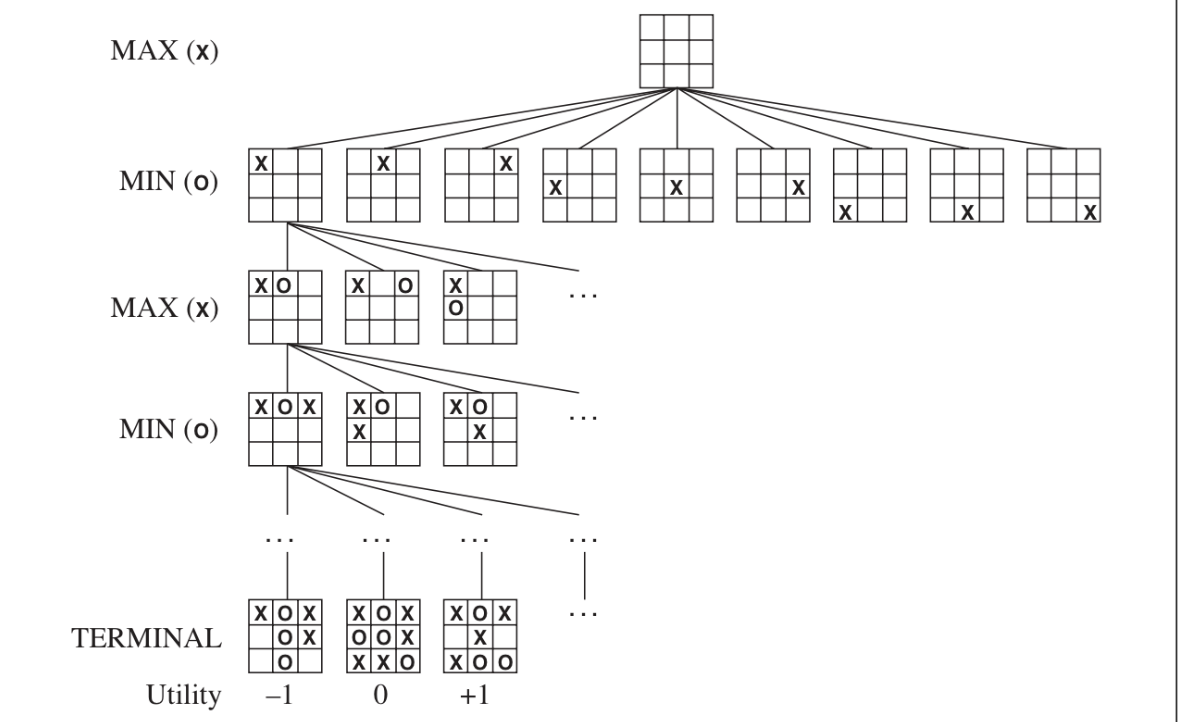
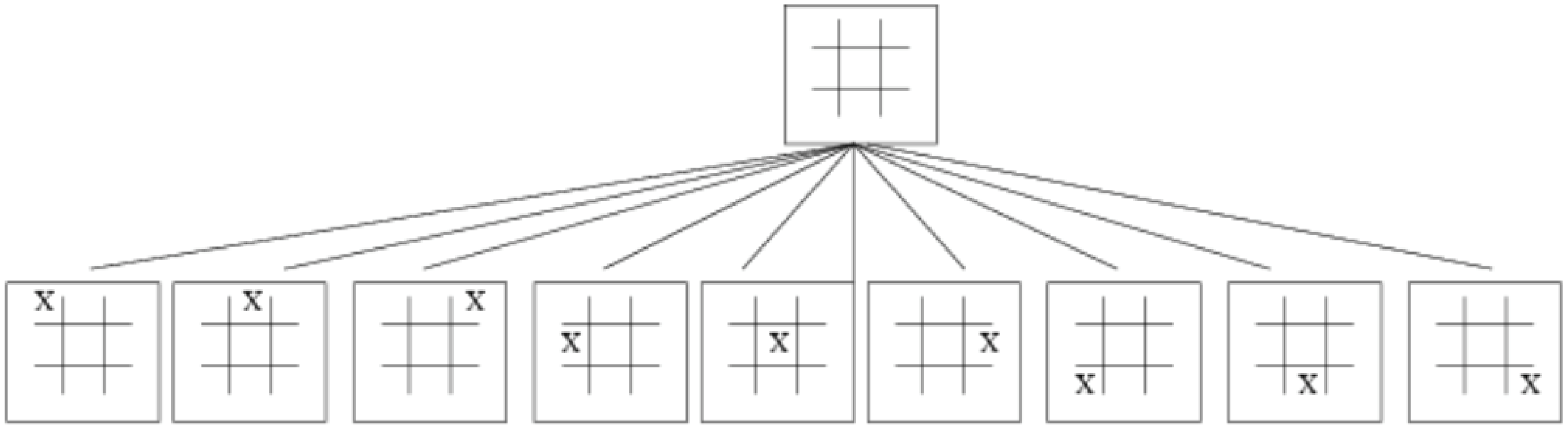


Figure 5.1 A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

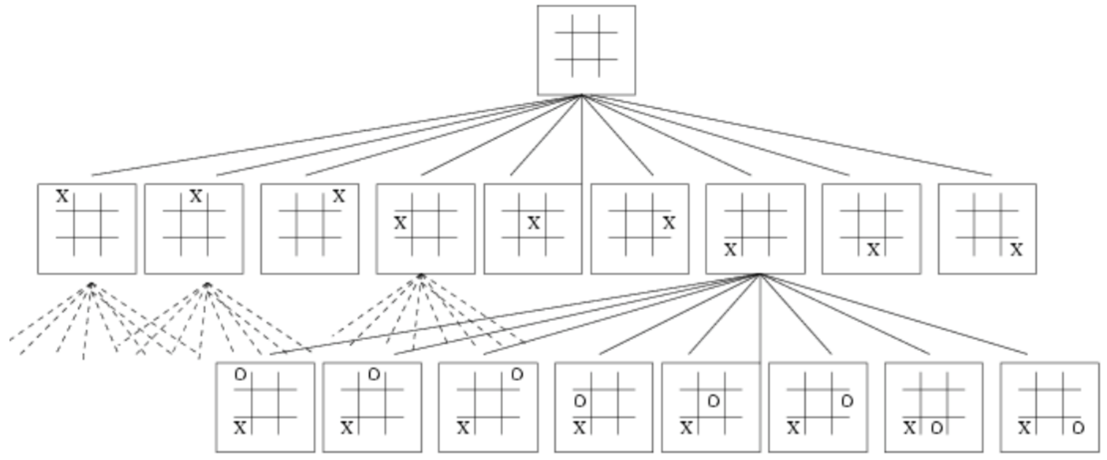
Simple Approach in Tic-Tac-Toe

- Here is a simple algorithm that will not work to calculate the winning move in a game:
- From the current position, calculate all the possible moves that could be made from that position.
- Expand each of these new positions so that they include the possible moves for the other player:



Simple Approach in Tic-Tac-Toe

- Expand each of these new positions so that they include the possible moves for the other player.
- Continue this expansion until a winning position for the required player is found.
- This algorithm will work, in the sense that it will locate a series of winning moves for the game, but the cost of performing that calculation is enormous (**large**). At the first level of the graph there will be one board. At the next there will be 9, at the next there will be 9∗8, at the level past that there will be 9∗8∗7 … ➔ 9! = 362880
- This is not so big that we cannot calculate it, but it is alarming since this is such a simple game.



Game’s Complexity

- It has been estimated that the total number of distinct (good and bad) chess games is about 10^{120}.
- In comparison, experts say there are 10^{63} molecules in the universe.

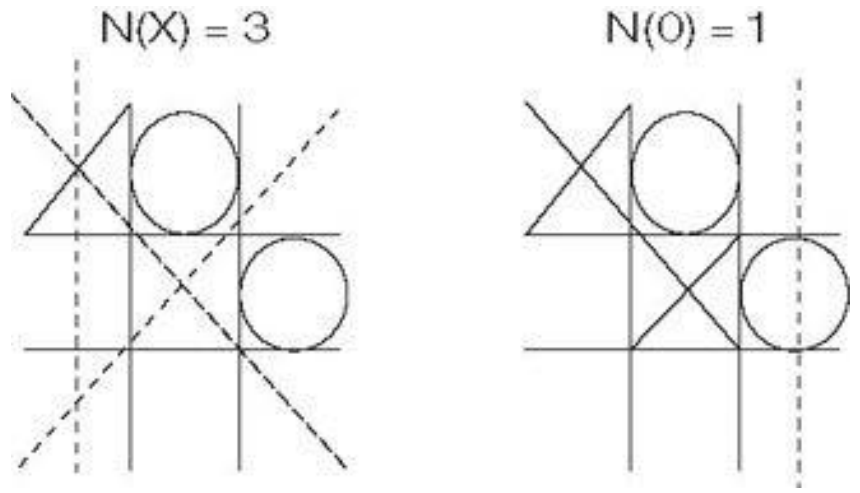
With games the main challenge in evaluating moves is the ability to look ahead as far as possible and then apply a **heuristic evaluation** of game positions.

Heuristic Evaluation

- When a game tree has been expanded to the end of a game, measuring the goodness of a move is trivial.
- For complicated games, you need to use heuristic evaluation.

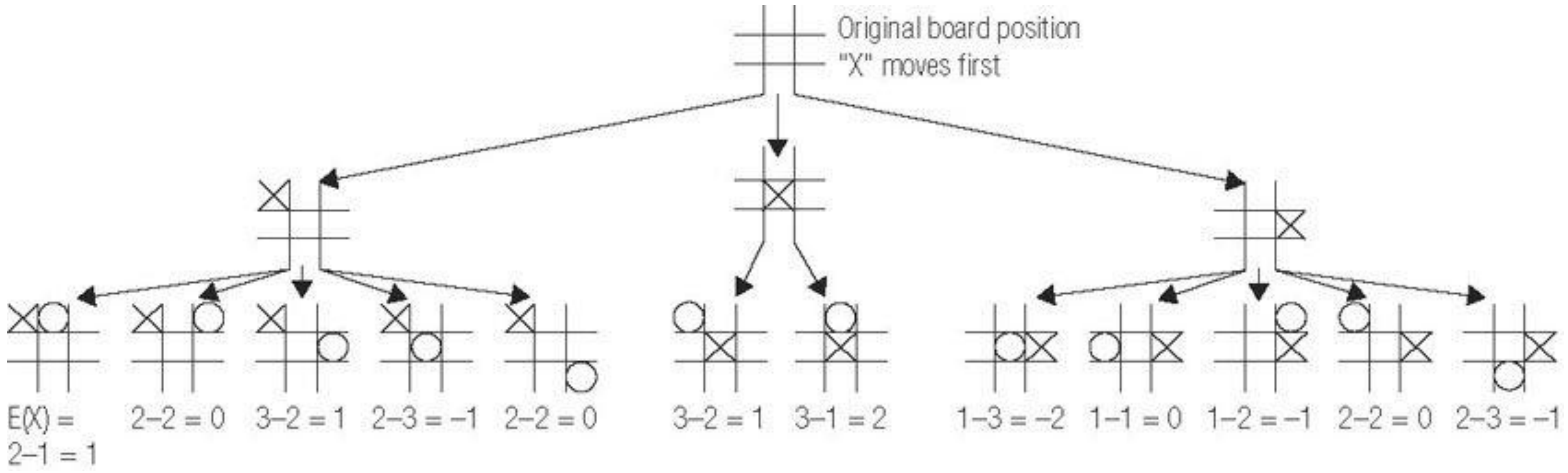
For example, you can use heuristic evaluation to solve the game of tic-tac-toe.

- Let N(X) equal the number of rows, columns, and diagonals that X could possibly complete,
- The heuristic evaluation of a game position h(X) is defined as N(X) – N(O).



Heuristic Evaluation

Heuristic evaluation provides a tool to assign values to leaf nodes in the game tree



Optimal Decisions in Games

Two Player Games:

- These are also called zero-sum games. A zero-sum games is one in which gain of one player is balanced exactly by the loss of another player.
- Static Evaluation Function *f(n)* = Win Positions – Loss Positions

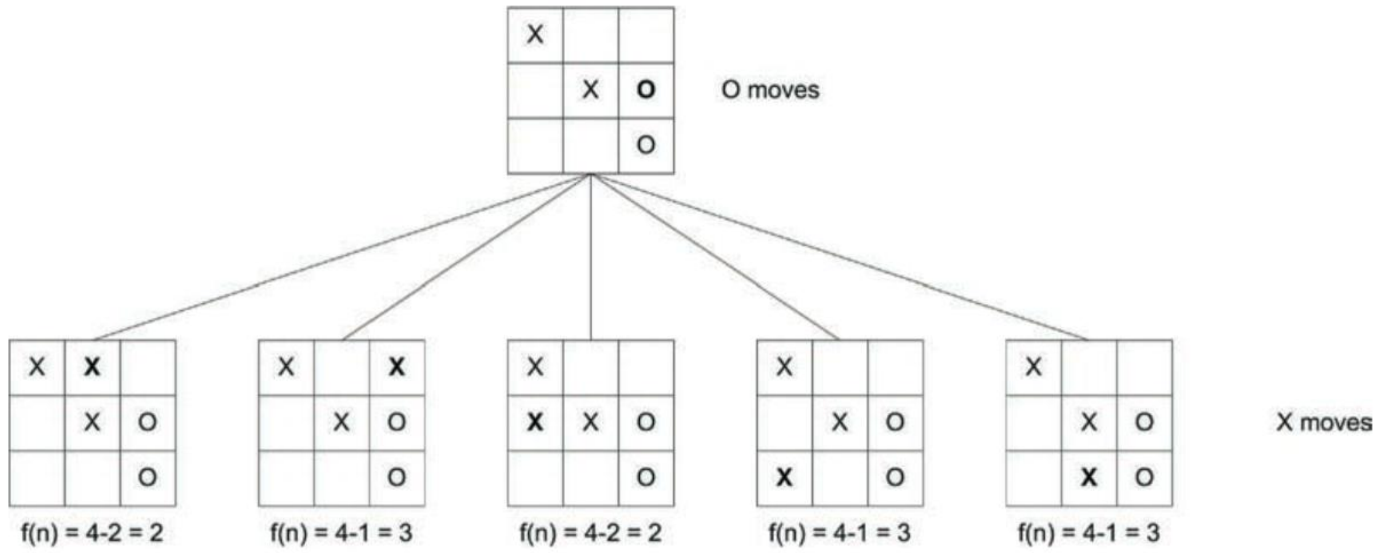


FIGURE 4.2: **Tic-Tac-Toe game tree with static evaluation function.**

Optimal Decisions in Games

- Higher the result of f(n), the closer the move brings the player towards a win. Three moves result in 3 but only one move results a win for X while the rest are the cases of win for O in *Figure 4.2*.

- Static evaluation function f(n) is useful but another heuristics is necessary to pick the move with highest f(n) while protecting against a loss in the next move.

- For this purpose, a Minimax algorithm is given next. The theory behind minimax is that the algorithm's opponent will be trying to minimize whatever value the algorithm is trying to maximize (hence, "*minimax*").

Minimax Algorithm

1. Minimax uses one of the two basic strategies:
    - Entire game tree is searched to the leaf nodes
    - Tree is searched to a predefined depth and then evaluated.
2. In order to pursue the tree further we will have to make guesses as to how the opponent will play.
3. The cost function can be used again to evaluate how the opponent is likely to play.
4. After evaluating some number of moves ahead we examine the total value of the cost to each player.
5. The goal is to find a move which will maximize the value of our move and will minimize the value of the opponents moves.
6. The algorithm used is the Minimax search procedure.

The Minimax algorithm

```
function MINIMAX(N) is
begin
if N is deep enough then
        return the estimated score of this leaf
else
        Let N1, N2, .., Nm be the successors of N;
        if N is a Min node then
                return min{MINIMAX(N1), .., MINIMAX(Nm)}
        else
                return max{MINIMAX(N1), .., MINIMAX(Nm)}
end MINIMAX;
```

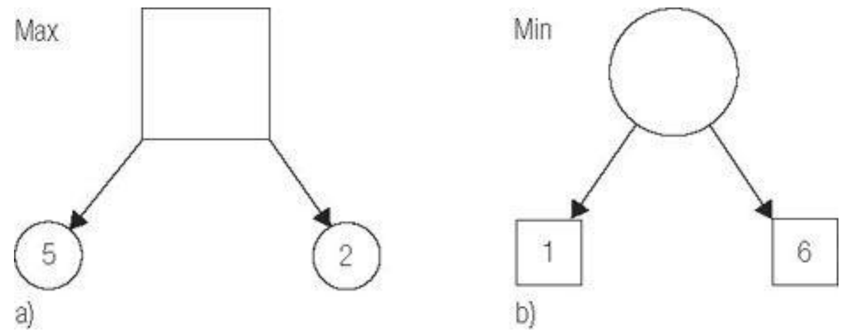
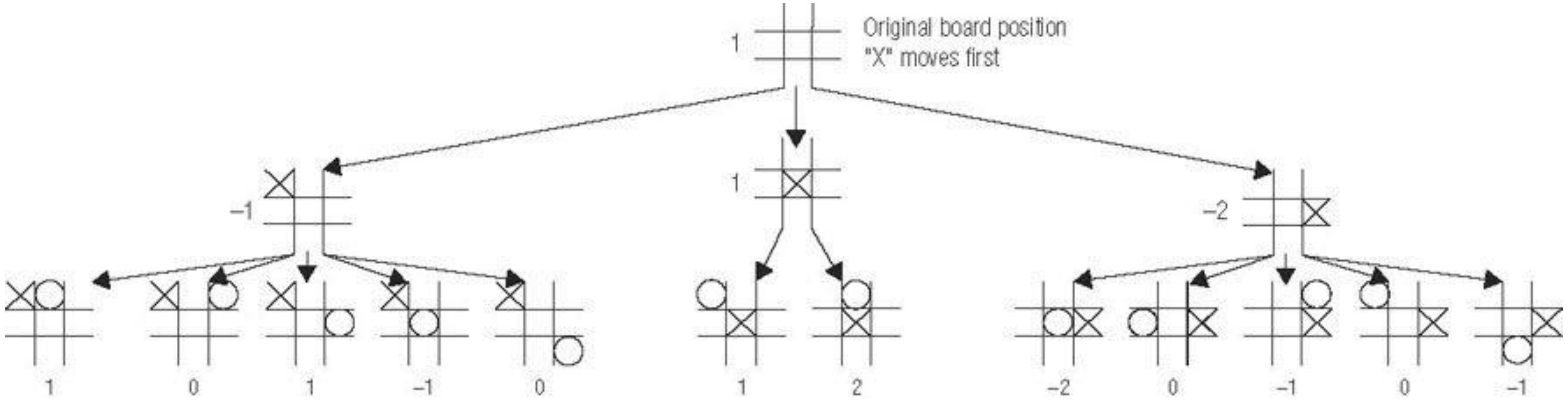


Figure 1: Minimax algorithm. Two player, perfect information zero-sum game

Example: Tic Tac Toe



Partial Example Tree of Tic-Tac-Toe with cost or utility function

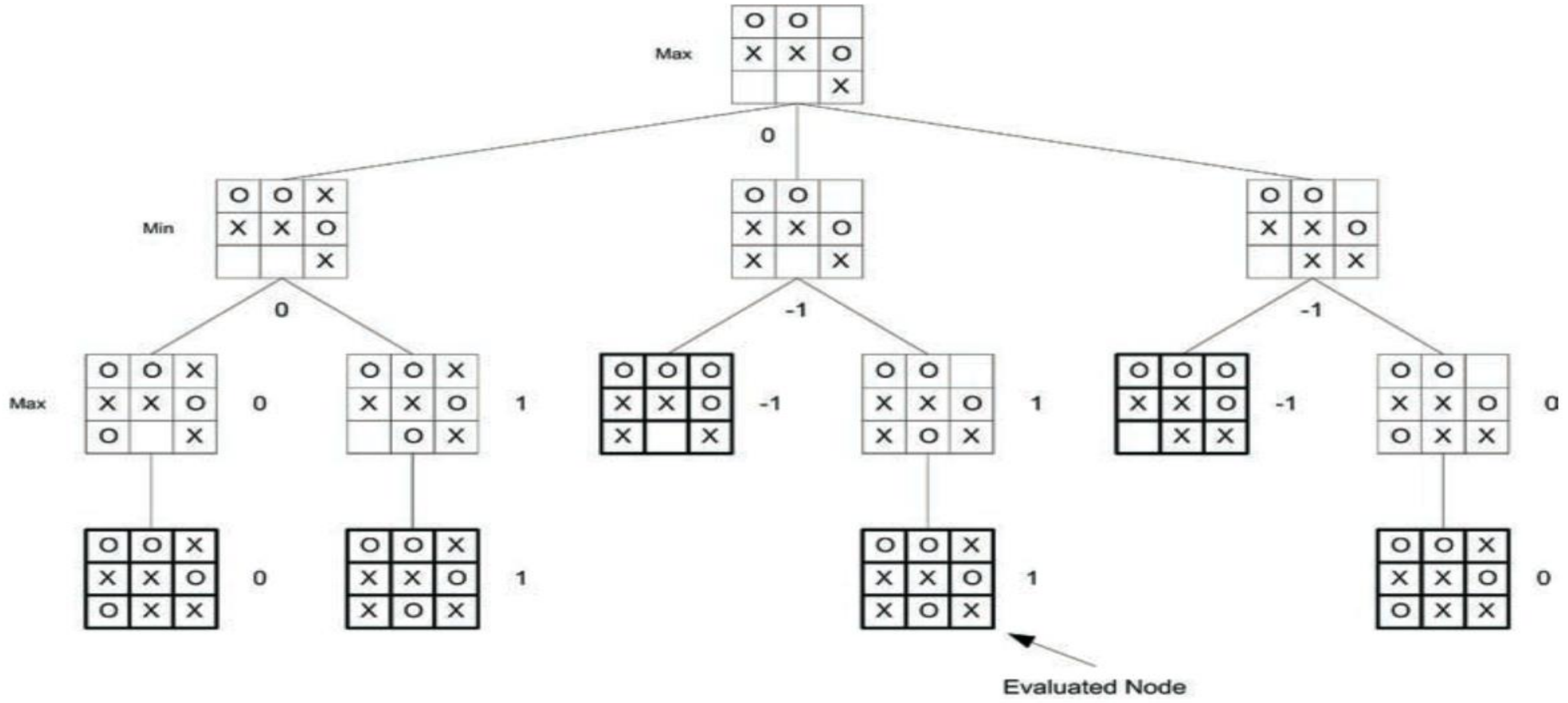


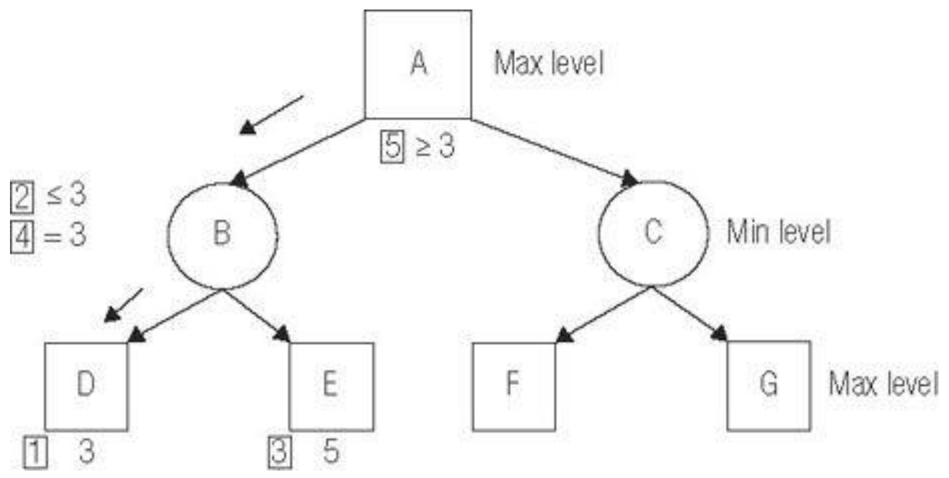
FIGURE 4.3: **End-game tree for a game of Tic-Tac-Toe.**

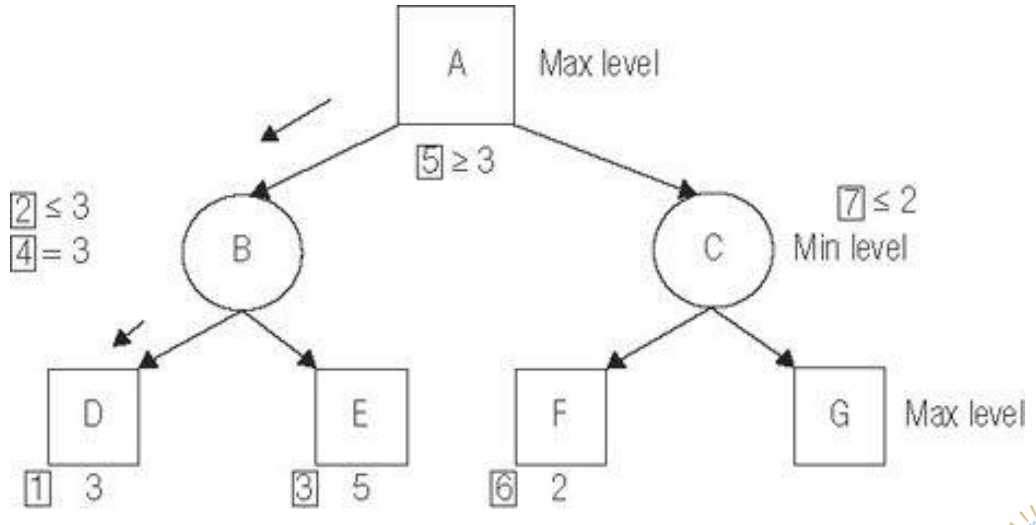
Alpha-Beta Pruning

- The problem with Minimax search is that the number of game states is exponential in the depth of the tree. Unfortunately, we couldn’t eliminate the exponent, but we can effectively cut it in half.
- We can borrow the idea of pruning to eliminate large parts of the tree from consideration. The particular technique we examine is called alpha–beta pruning.
- When applied to a standard Minimax tree, it returns the same move as Minimax would, but prunes away branches that cannot possibly influence the final decision.

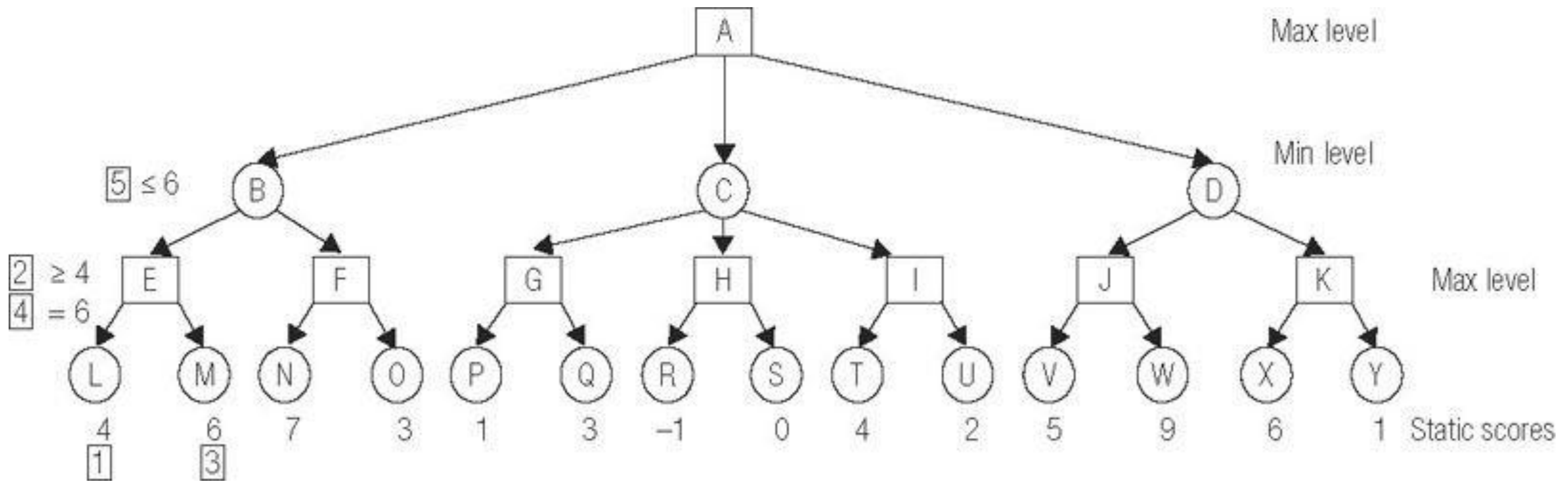
Alpha-Beta Pruning

1. Min discovers that game position D has a value of 3.
2. The value of position B is ≤ 3. This upper-bound value for B is referred to as a beta value for B.
3. The value of E is 5.
4. Consequently, at time 4, Min knows that the value of B equals 3.
5. A is worth at least 3
6. At time 6, Min observes that the value of a move to F is worth 2.
7. Therefore, at time 7, Min knows that C's worth is ≤ 2.
8. Max now knows that a move to C will return a value of 2 or less. Max will not be moving to C because a move to B ensures a return of 3. The value at G will not change this appraisal, so why look? Evaluation of this tree can now end.





Another example of alpha-beta pruning 1/2



1. We move left at every branch until leaf node L is encountered. Its static value is 4.

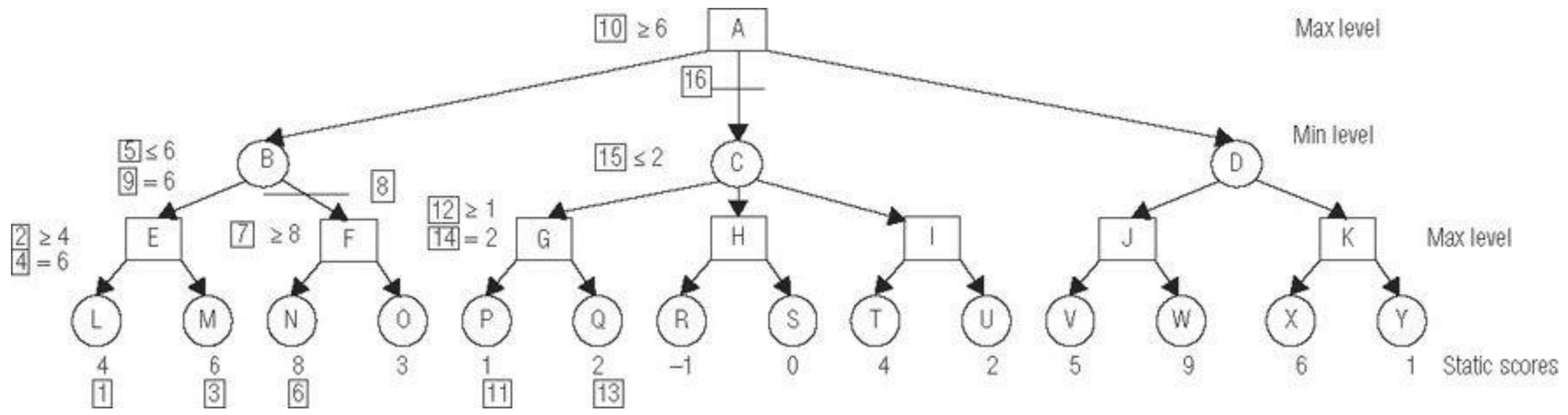
2. Max is guaranteed a score of at least 4. Again, this lower bound of 4 is referred to as an alpha value for E.

3. Max wants to ensure that no score higher than 4 exists in the subtree at E. At time 3, M’s value is equal to 6.

4. At time 4, node E has a value, which is now equal to 6.

5. At time 5, Min node B has a beta value of 6. Why? Try to answer this question before continuing with the next steps.

Another example of alpha-beta pruning 2/2



6. The entire subtree at F is pruned. The value at F is ≥ 8. Min at position B would never allow Max to get to position F.

7. The value at node B is now equal to 6.

8. The lower bound (alpha value) for Max is at the start position A.

9. Max wants to know if a value > 6 is available in the subtree at C. So the search next explores node P to derive its value, which is 1.

10. Now it is known that the value of G is ≥ 1.

11. Is G worth more than 1 to Max? To answer, the value at node Q must be obtained.

12. An exact value at node G is now known to equal 2.

13. Consequently, this value of G serves as an upper bound at position C (i.e., the beta value of C = 2).

14. Max observes that a move to node B is worth 6, but a move to C guarantees a return of at most 2. Max will not be moving to C and therefore this entire subtree can be pruned.

Alpha Cutoff vs. Beta cutoffs

- More generally, whenever a node (here node A) has an alpha value of x, the entire subtree whose root is the latter’s parent (here the subtree with root C) can be pruned. This is referred to as an alpha cutoff.
- Beta cutoffs (i.e., prunes caused by beta values for Min nodes), are defined analogously.

A third example of alpha-beta pruning

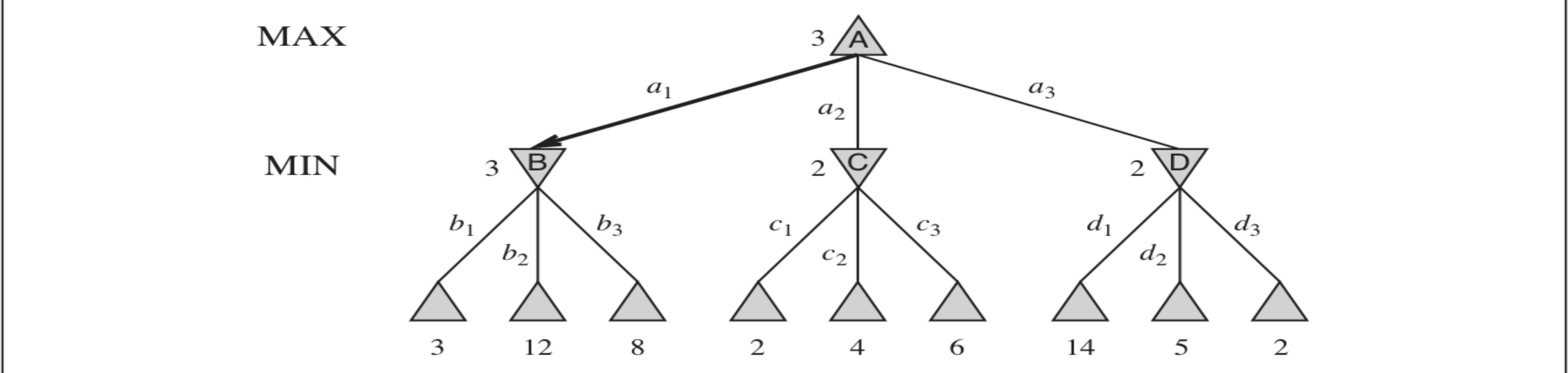


Figure 5.2 A two-ply game tree. The △ nodes are "MAX nodes," in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes." The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX's best move at the root is a_1, because it leads to the state with the highest minimax value, and MIN's best reply is b_1, because it leads to the state with the lowest minimax value.

A Simplification of the formula for MINIMAX

- The steps are explained in Figure next. The outcome is that we can identify the Minimax decision without ever evaluating two of the leaf nodes.

- Another way to look at this is as a simplification of the formula for MINIMAX. Let the two unevaluated successors of node C in *Figure 5.5* have values x and y. Then the value of the root node is given by

$$\begin{aligned}\text{MINIMAX}(root) &= \max(\min(3,12,8),\min(2,x,y),\min(14,5,2))\\ &= \max(3,\min(2,x,y),2)\\ &= \max(3,z,2) \text{ where } z = \min(2,x,y) \le 2\\ &= 3.\end{aligned}$$

- Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves.
    - α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
    - β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

Alpha-Beta Pruning

Alpha–beta search updates the values of α and β as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.

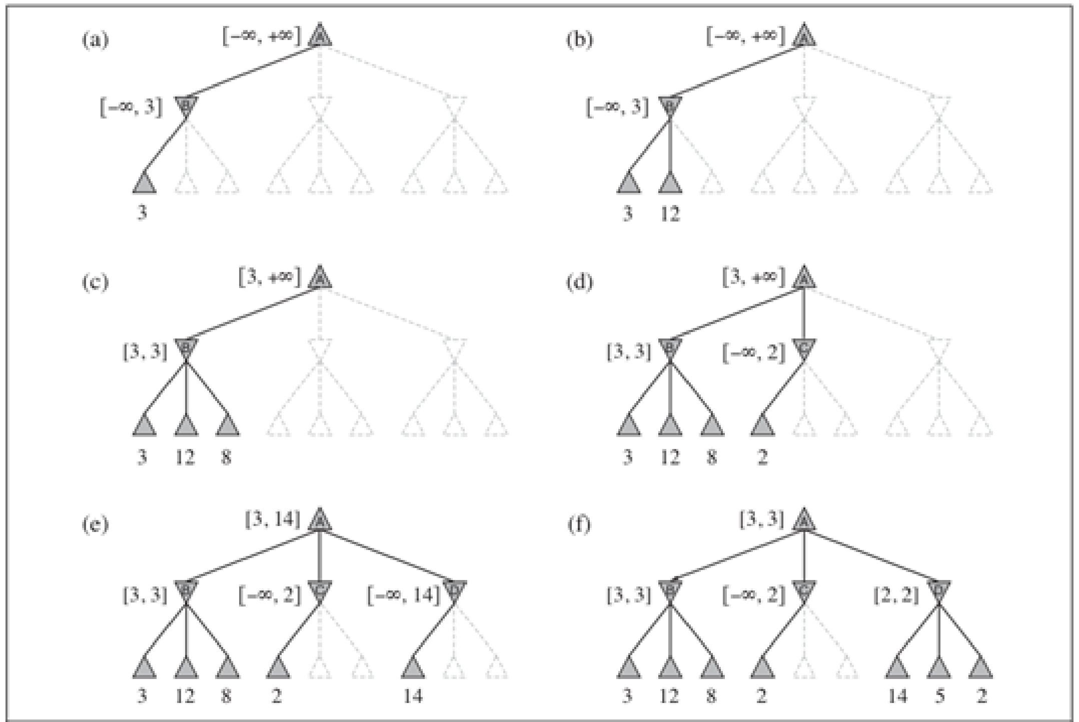
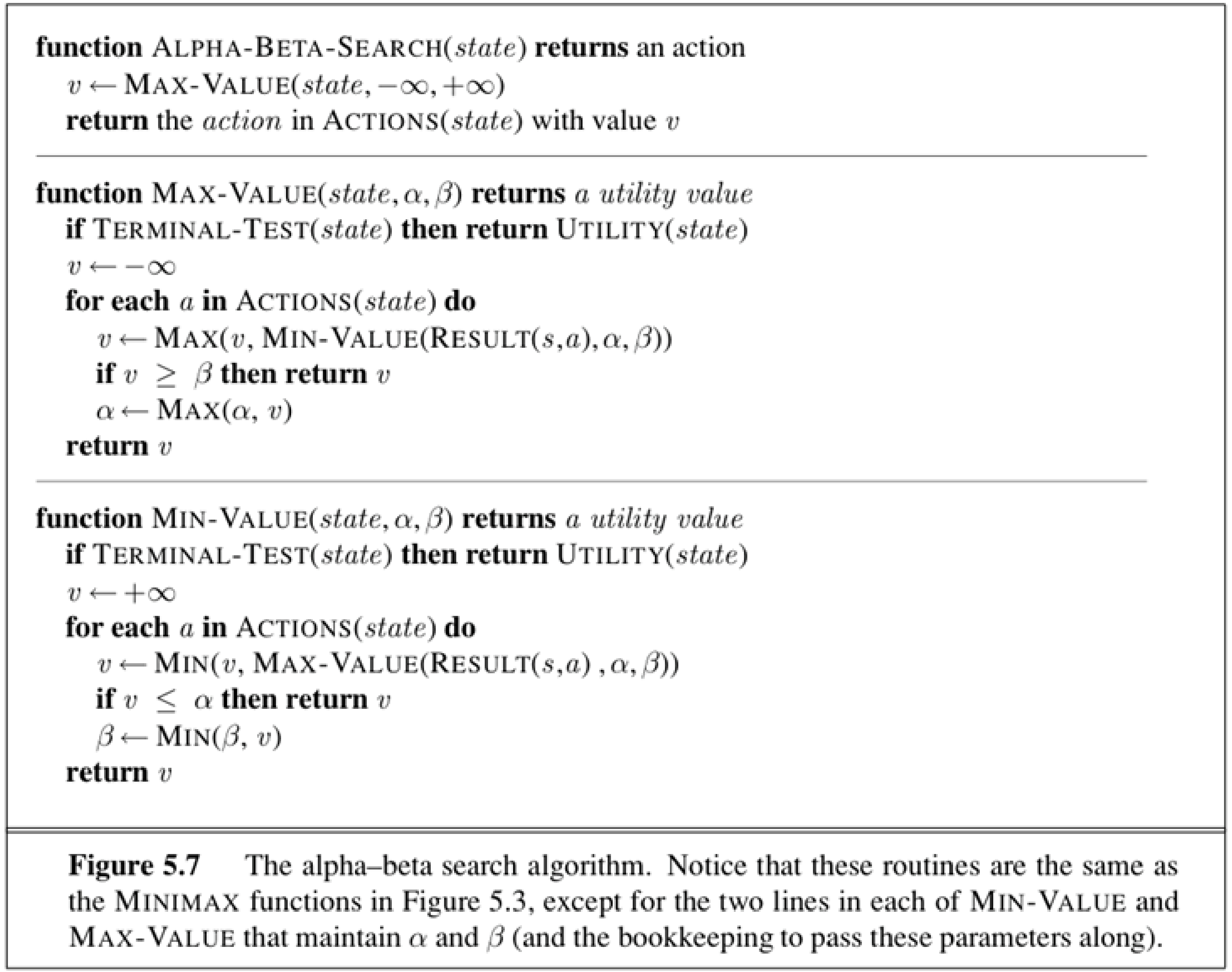


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Alpha-Beta Pruning- Algorithm

The is the complete algorithm. It will be good to trace its behavior when applied to the tree in Figure 5.5.



```
function ALPHA-BETA-SEARCH($state$) returns an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value $v$

function MAX-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha$, $\beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$

function MIN-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow +\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Contents for the next lecture

- Introduction to Machine Learning
- Introduction to Deep Learning (Artificial neural network)
- Natural Language Processing (NLP)

Any questions?