Introduction to Artificial Intelligence

Lecture 5

- Game Playing

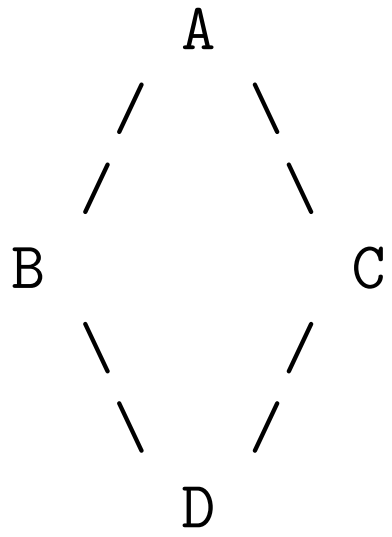
Reading: Graham

- Read Ch 1-3 and 6
- Skim Ch 4-5 and 7-9

Quiz

Given the following constraint satisfaction problem:

- Variables: A, B, C, D
- Domain: for each: r, g, y
- Constraints: Vars connected by an arc have different values:



Assign A = r and impose arc consistency on all arcs. What is the remaining domain for B, C, and D?

Search

- Formulation of search problems.
- Uninformed (blind) search algorithms.
- Informed (heuristic) search algorithms.
- Constraint Satisfaction Problems.
- * Game Playing Problems.

* we are here

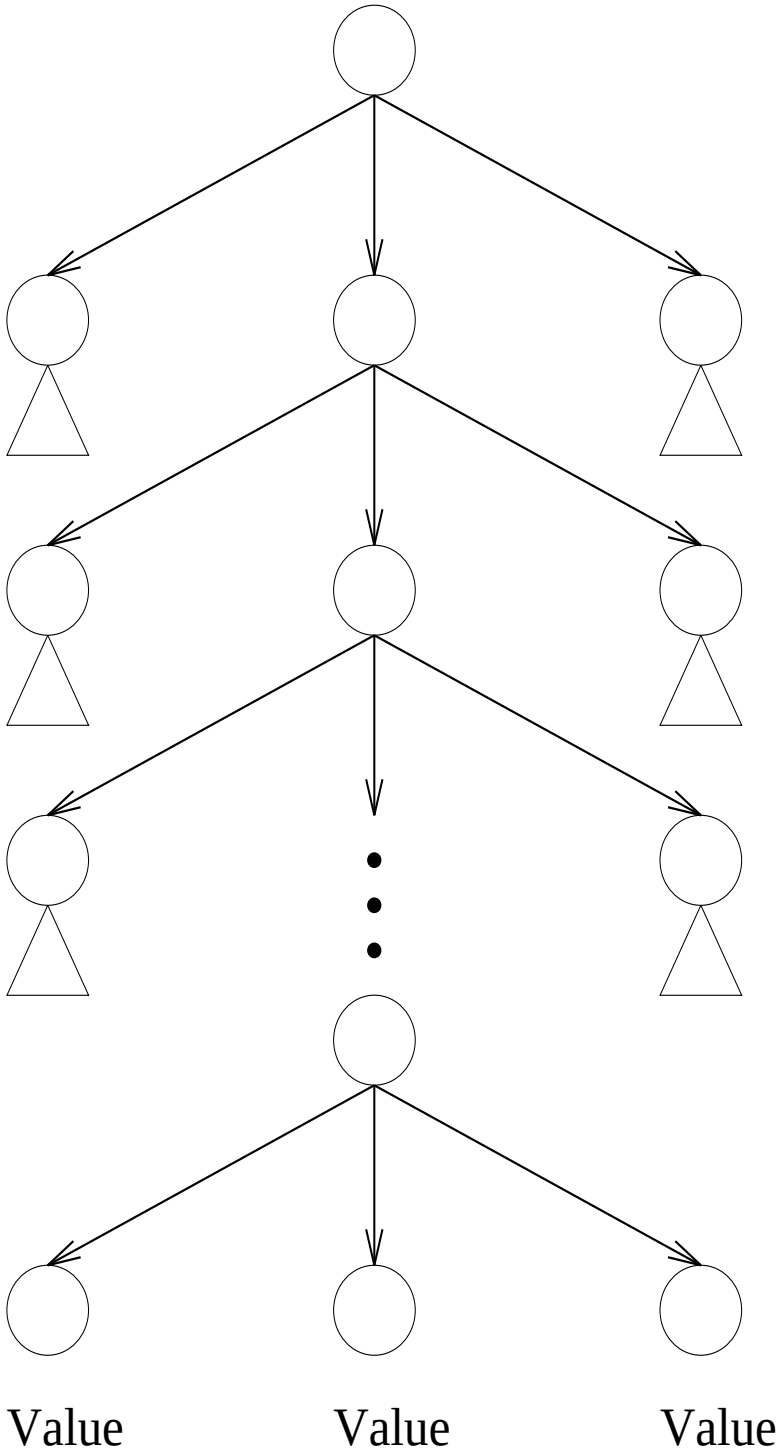
Game Playing

- having an opponent forces a game-playing program to deal with the **contingency problem**.
- uncertainty arises from the opponent and possibly from a dice or random variable that is part of the game.
- more popular games are too complex to solve, requiring the program to "take its best guess." For example, in chess, the search tree has 10^{40} nodes (with branching factor of 35).
- games are often played under strict time constraints (e.g., chess) and therefore must be very efficient.
- How do we make a "good" move when deciding on the "best" move is impossible?

Game Playing Space

- Initial state of the game.
- Operators defining legal moves.
- Terminal test defining end of game states.
- Utility or payoff function assigning numeric value to each end of game state.

Game Playing Tree



Values:

Max wins: 1

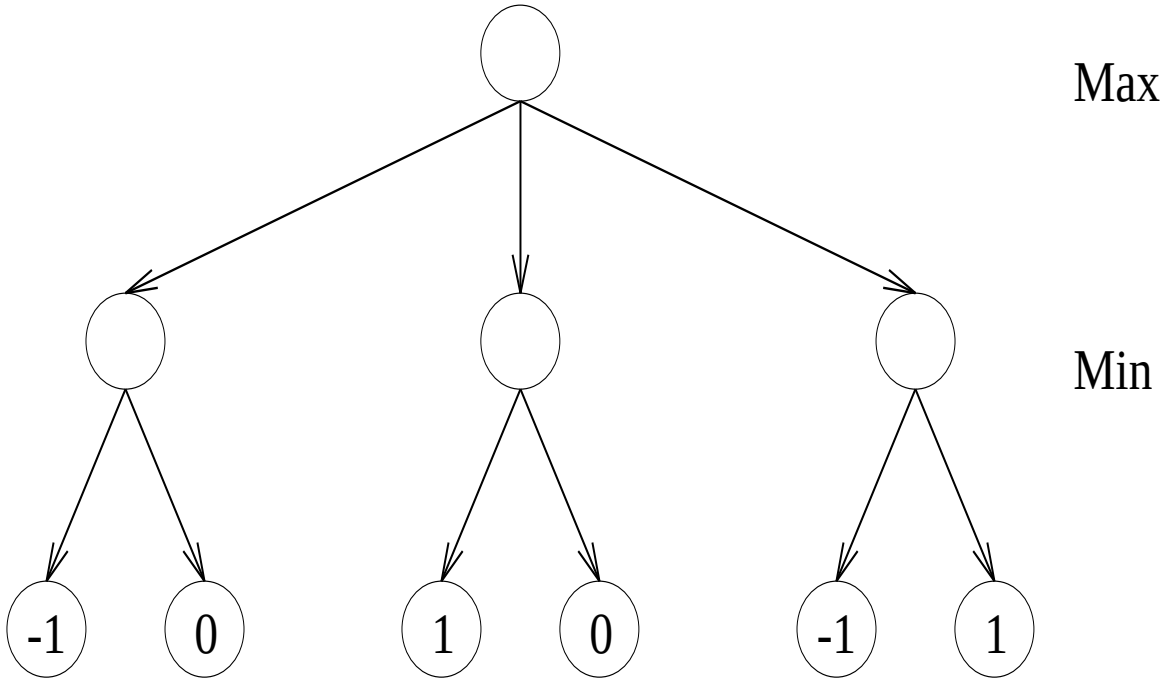
Min wins: -1

Draw: 0

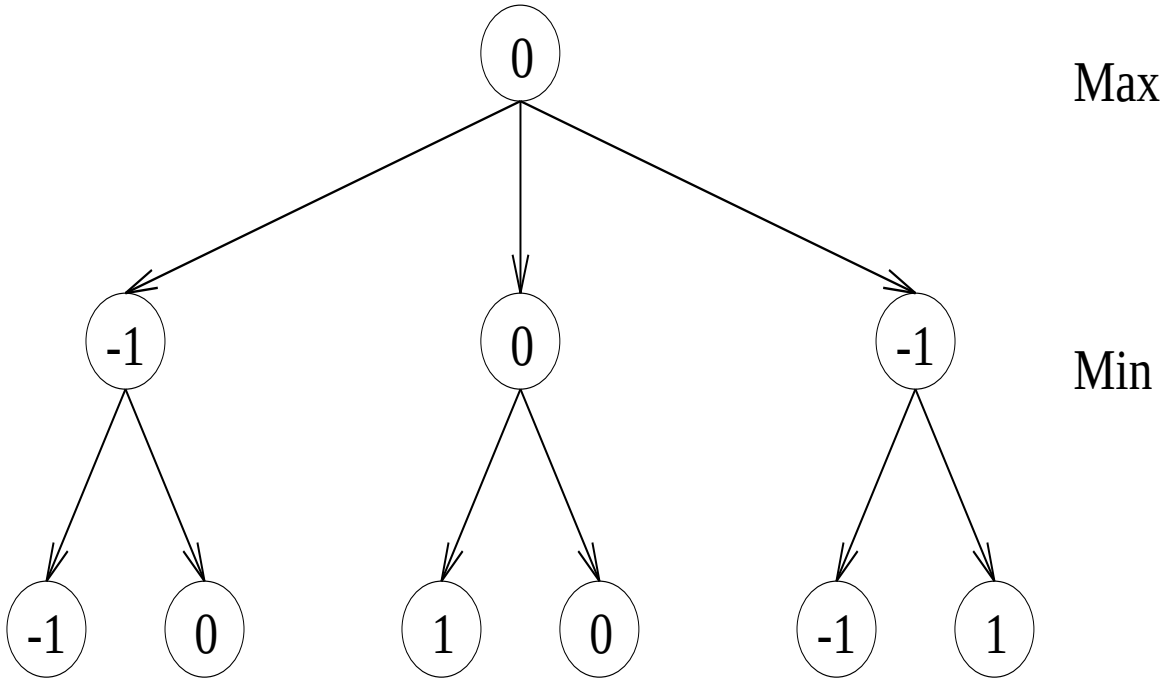
Minimax idea

When MAX moves, it must search for a sequence of moves that leads to a "winner" terminal state regardless of how MIN moves. MAX's **strategy** will find the best move for MAX for each possible move for MIN.

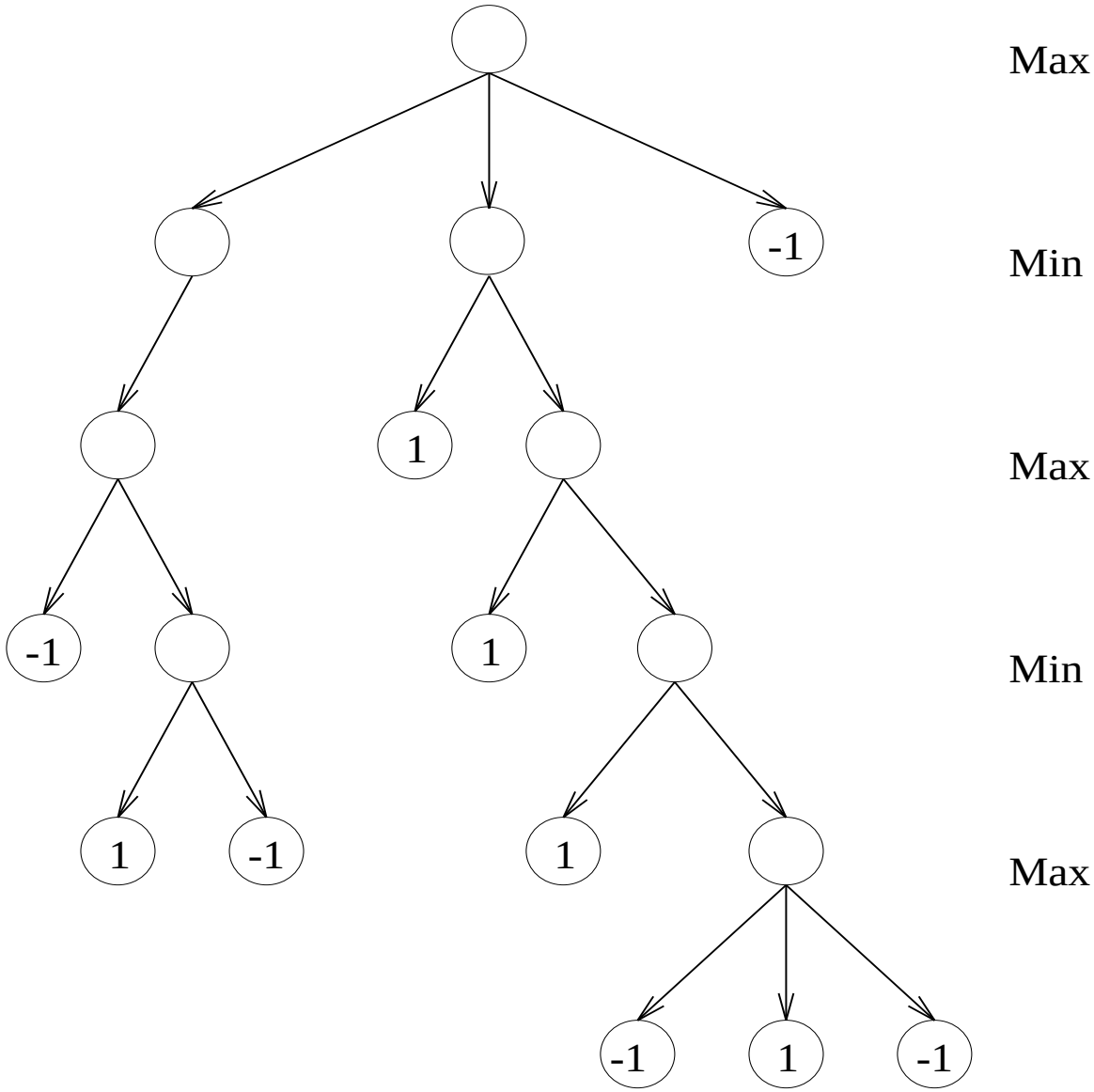
A Very Simple Game Tree



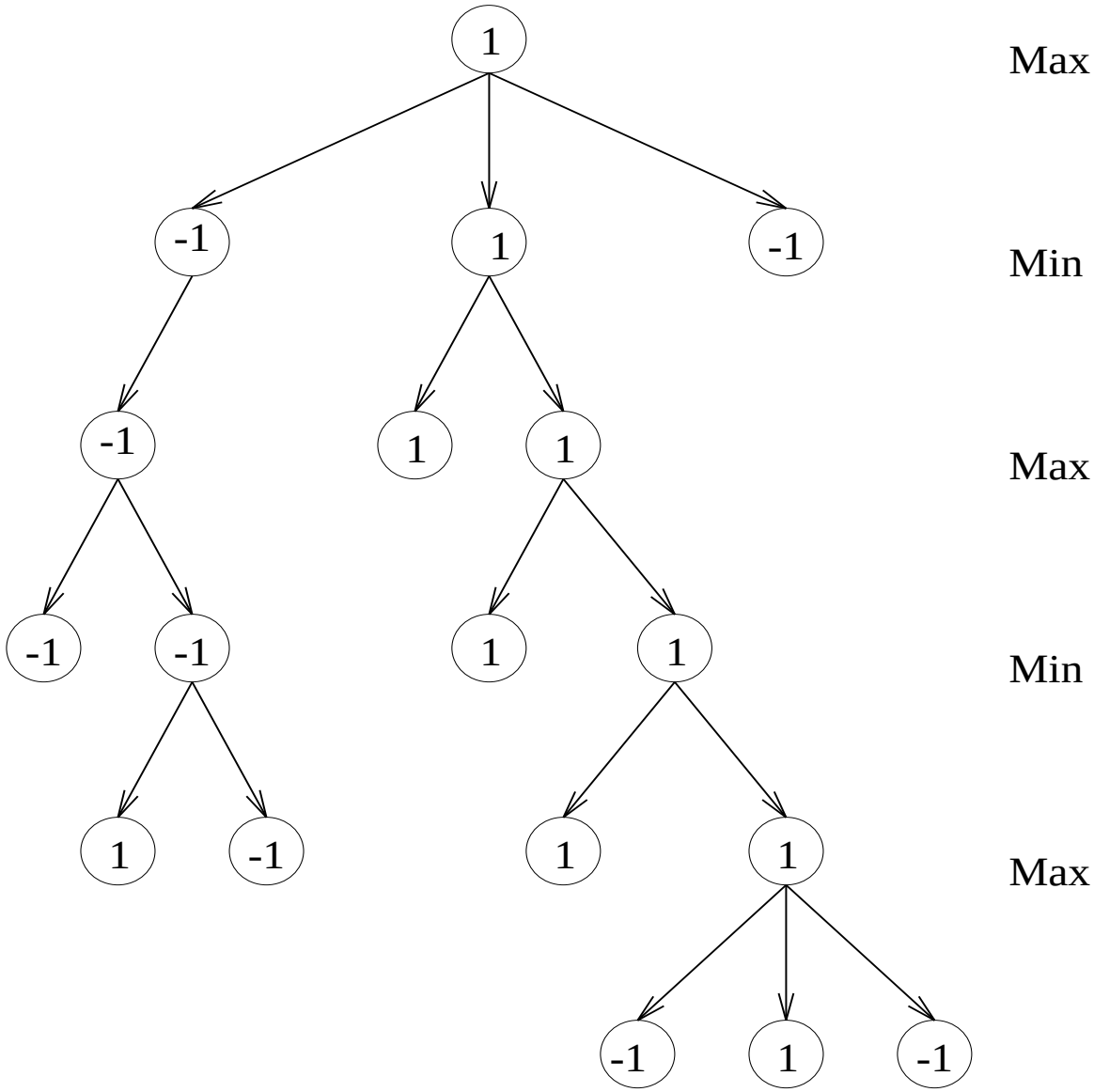
A Very Simple Game Tree



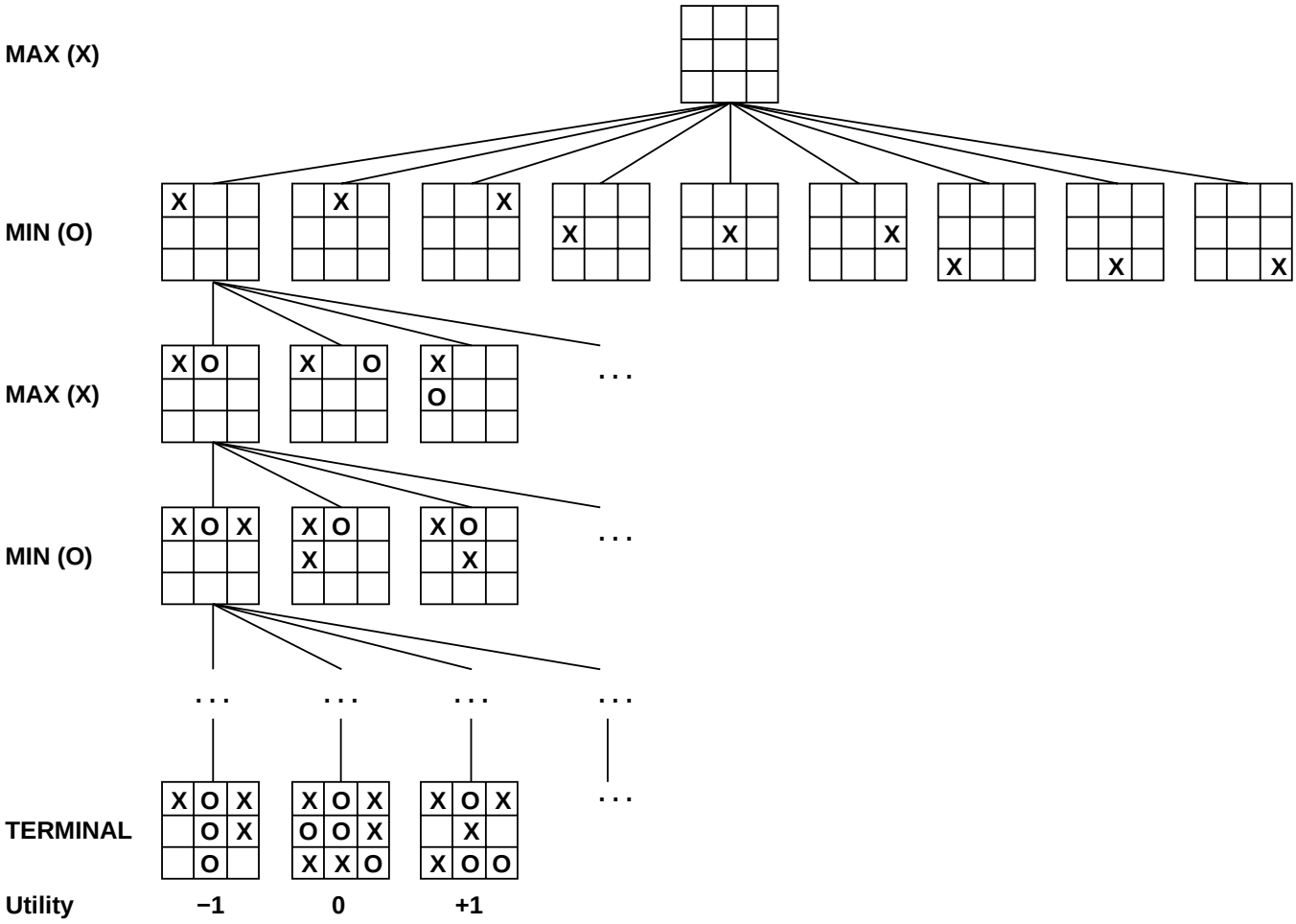
Another Game Tree



Another Game Tree



Another example: Tic-Tac-Toe



Partial search tree for Tic-Tac-Toe game.

Mini-Max Game Tree Search

```
Minimax(State,Sign):
If (State is the end of a game)
   Then Return(Value(State))
   Else 1. For each child Ci of State do Ei = Minimax(Ci,-Sign).
        2. If (Sign = 1)
              Then Return(Max of all Ei's)
              Else Return(Min of all Ei's)

Top Level Call Playing Max:

    Minimax(State,1)

Top Level Call Playing Min:

    Minimax(State,-1)
```

Complexity

With max depth m and branching factor b, time complexity is $O(b^m)$. Since algorithm is recursive, space complexity is linear in b and m.

Imperfect Decisions

For most games, it is impractical to search all the way to the terminal states. Therefore, we need to cut off the search and apply a **static evaluation function**.

Static Evaluation Functions

- Must trade-off accuracy of the evaluation function with the time required to compute it.
- In chess, material value is often a **weighted linear function** since each piece is treated independently.

Cutting Off Search

- limit search to fixed depth.
- iterative deepening within a fixed time limit.
- in a **quiescence search**, lookahead is extended until a stable position is found.
- in the **horizon problem**, the opponent is about to make an unavoidable, damaging move. The program, sensing this, tries to stall the opponent with useless moves that simply prolong the inevitable. If it "pushes" the inevitable over the search-depth "horizon", it thinks it can avoid the damage.

Static Evaluator for Tic Tac Toe

$$E(State) = 3X_2 + X_1 + -3O_2 - O_1$$

- Where X_i is the number of rows, columns and diagonals containing exactly i X's.
- And O_i is the number of rows, columns and diagonals containing exactly i O's.

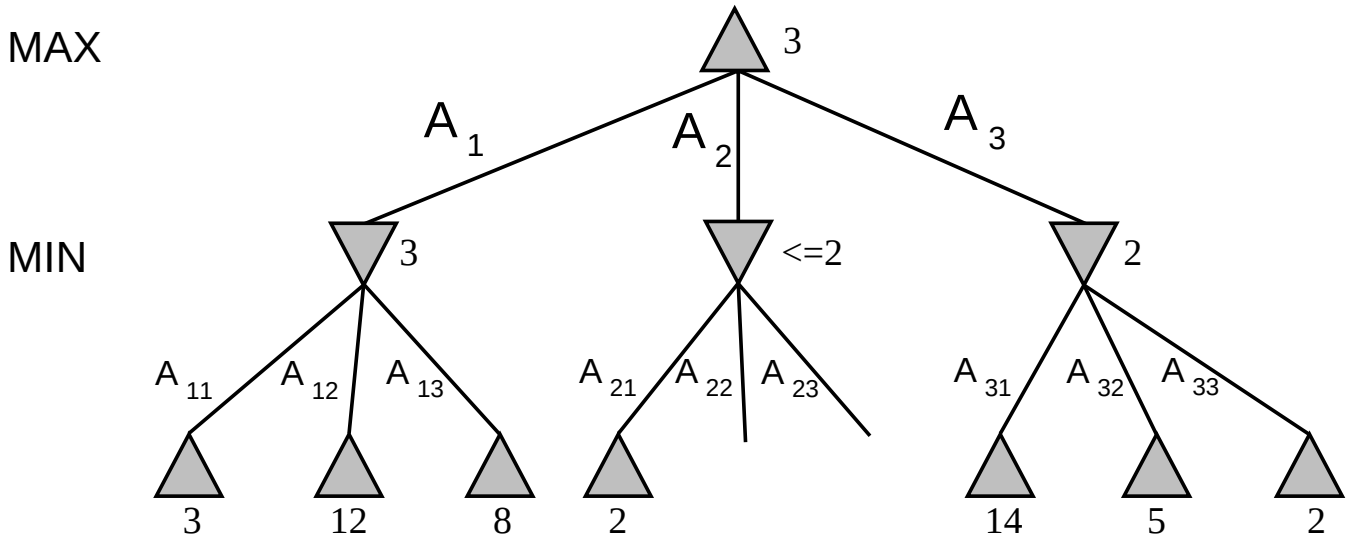
Mini-Max Game Tree Search with Static Evaluation

```
Minimax(State,Sign,Depth):
1. If (State is the end of a game) Then Return(Value(State))
2. If (Depth => Depth-Bound) Then Return(Static-Evaluation(State)).
3. For each child Ci of State do Ei = Minimax(Ci,-Sign,Depth+1).
4. If (Sign = 1)
      Then Return(Max of all Ei's)
      Else Return(Min of all Ei's)

Static-Evaluation(State):
;;Returns estimate of Minimax value of State.

Top Level Call Playing Max:

    Minimax(State,1,0)

Top Level Call Playing Min:

    Minimax(State,-1,0)
```

Alpha-Beta Pruning

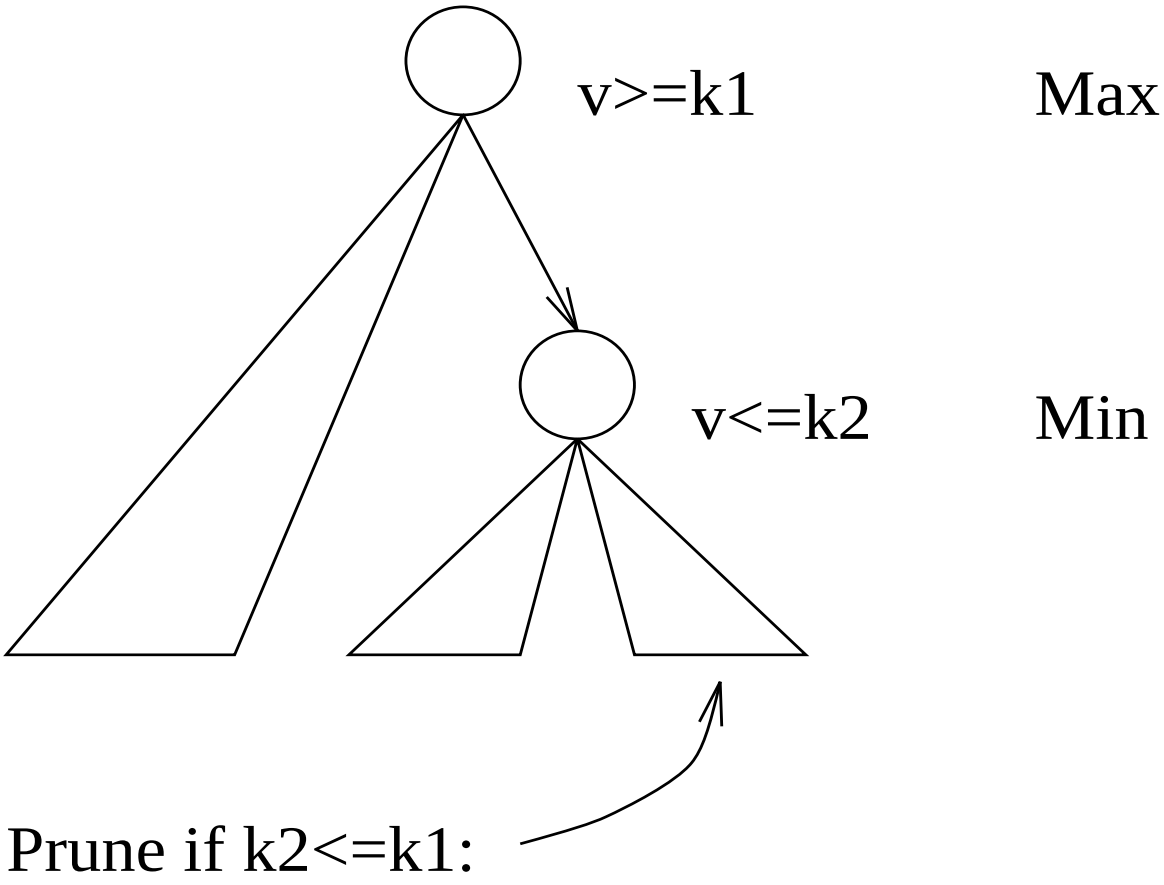
- enables a deeper search than can be accomplished by minimax by pruning fruitless branches of the search tree.
- if, at a node n in the tree, a player has a better choice at the node's parent (or ancestor), n will never be reached in play and we can therefore prune the subtree rooted at n.
- maintains α (the best choice for max) and β (the best choice for min) at each node.
- ordering the descendants (successors) from best to worst can improve the performance.

Alpha-beta pruning example

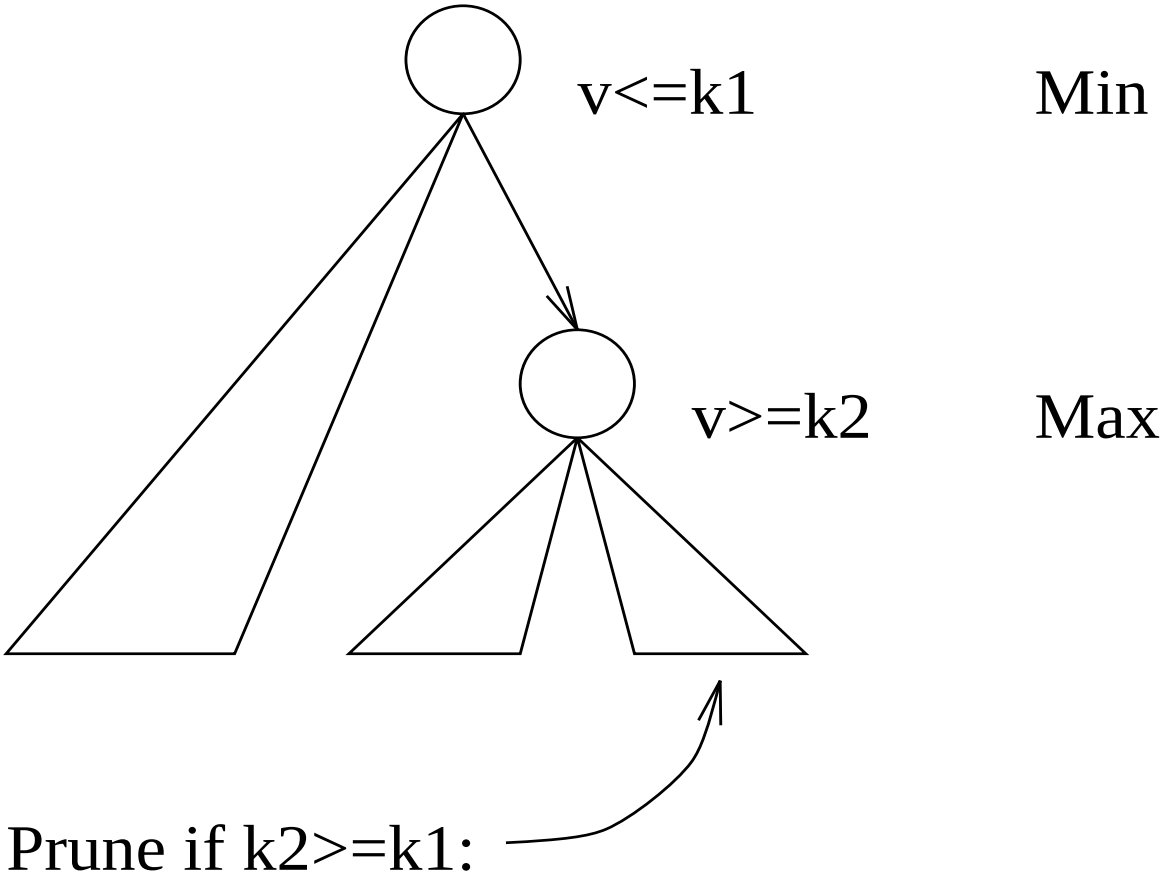


Two-ply game tree generated by alpha-beta.

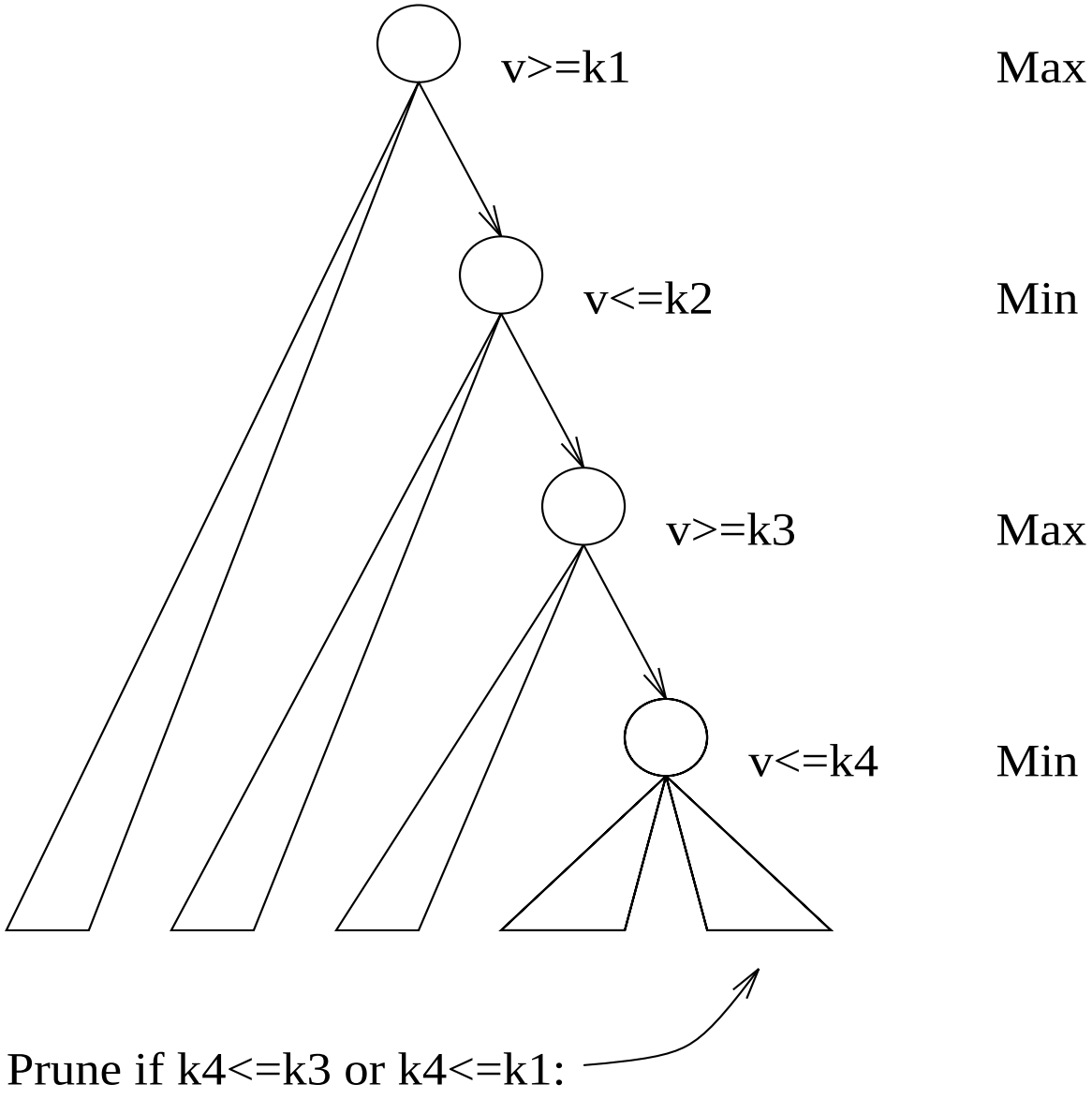
Alpha Prune



Beta Prune



Another Alpha Prune



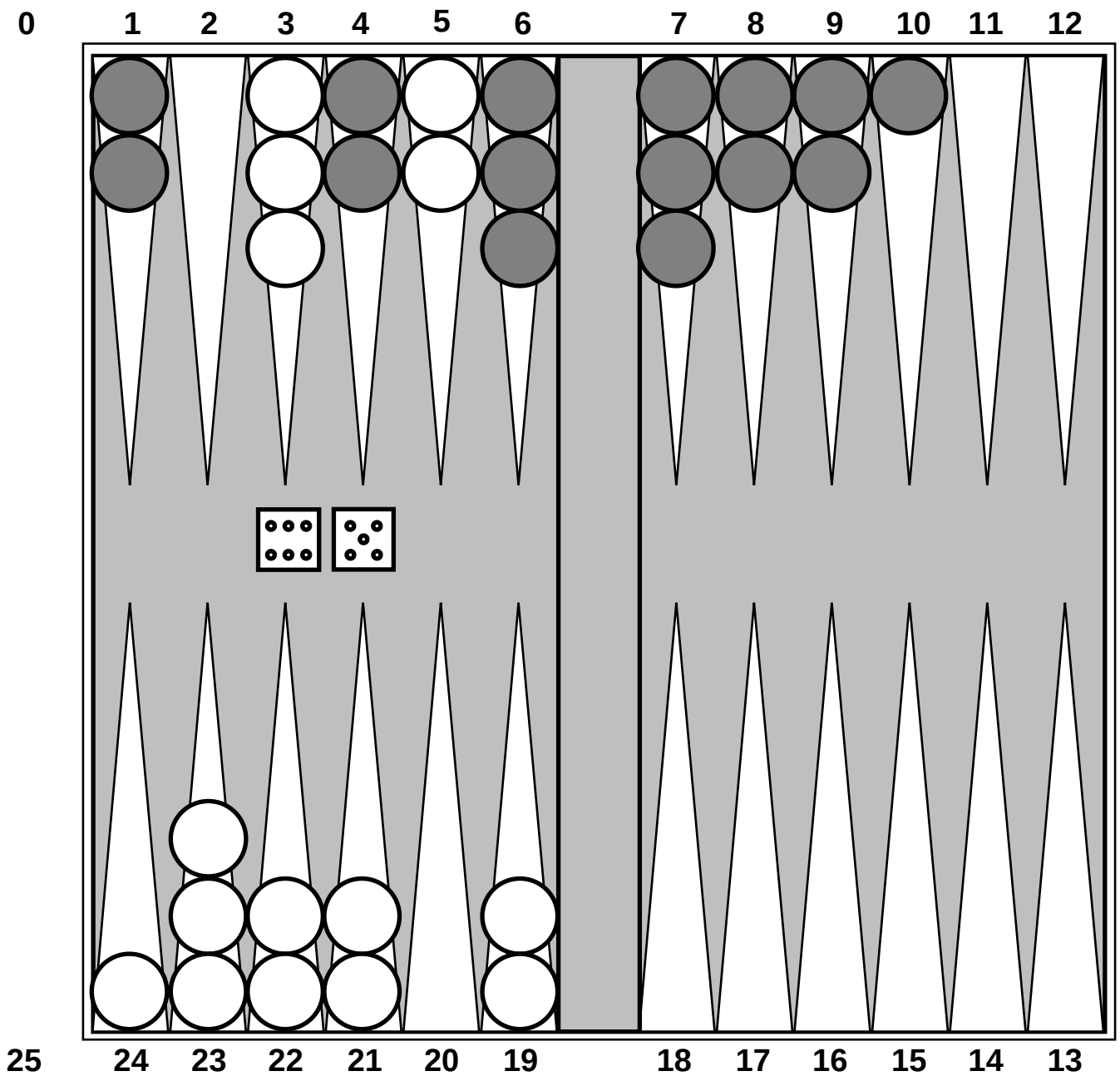
Mini-Max Game Tree Search with Static Evaluation and Alpha-Beta Pruning

```
Minimax(State,Sign,Depth,Alpha,Beta):
1. If (State is the end of a game) Then Return(Value(State))
2. If (Depth => Depth-Bound) Then Return(Static-Evaluation(State)).
3. Let M = Alpha.
4. For each child Ci of State do
    a. Ei = Minimax(Ci,-Sign,Depth+1,Beta,M).
    b. If (Sign = 1)
          Then a. M = Max(M,Ei)
               b. If M > Beta Then Return(M)
          Else a. M = Min(M,Ei)
               b. If M < Beta Then Return(M).
5. Return(M).

Static-Evaluation(State):
;;Returns estimate of Minimax value of State.

Top Level Call Playing Max:

  Minimax(State,1,0,-Infinity,+Infinity)

Top Level Call Playing Min:

  Minimax(State,-1,0,+Infinity,-Infinity)
```

Games with an Element of Chance

- Three types of nodes: max, min and chance.
- Compute expected values at chance nodes.
- Compute max at max nodes.
- Compute min at min nodes.
- complexity is $O(b^m n^m)$, where m is the number of outcomes of the random variable.

Backgammon position



White has just rolled 6-5 and has four legal moves: (5-10,5-11),(5-11,19-24), (5-10,10-16), and (5-11,11-16).

Game tree for backgammon position