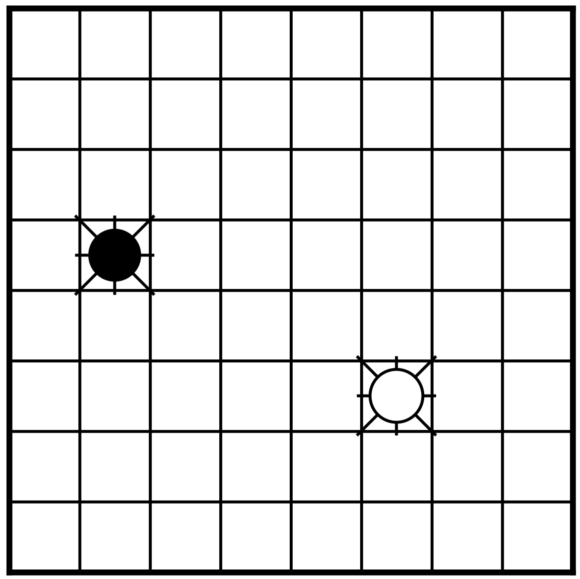
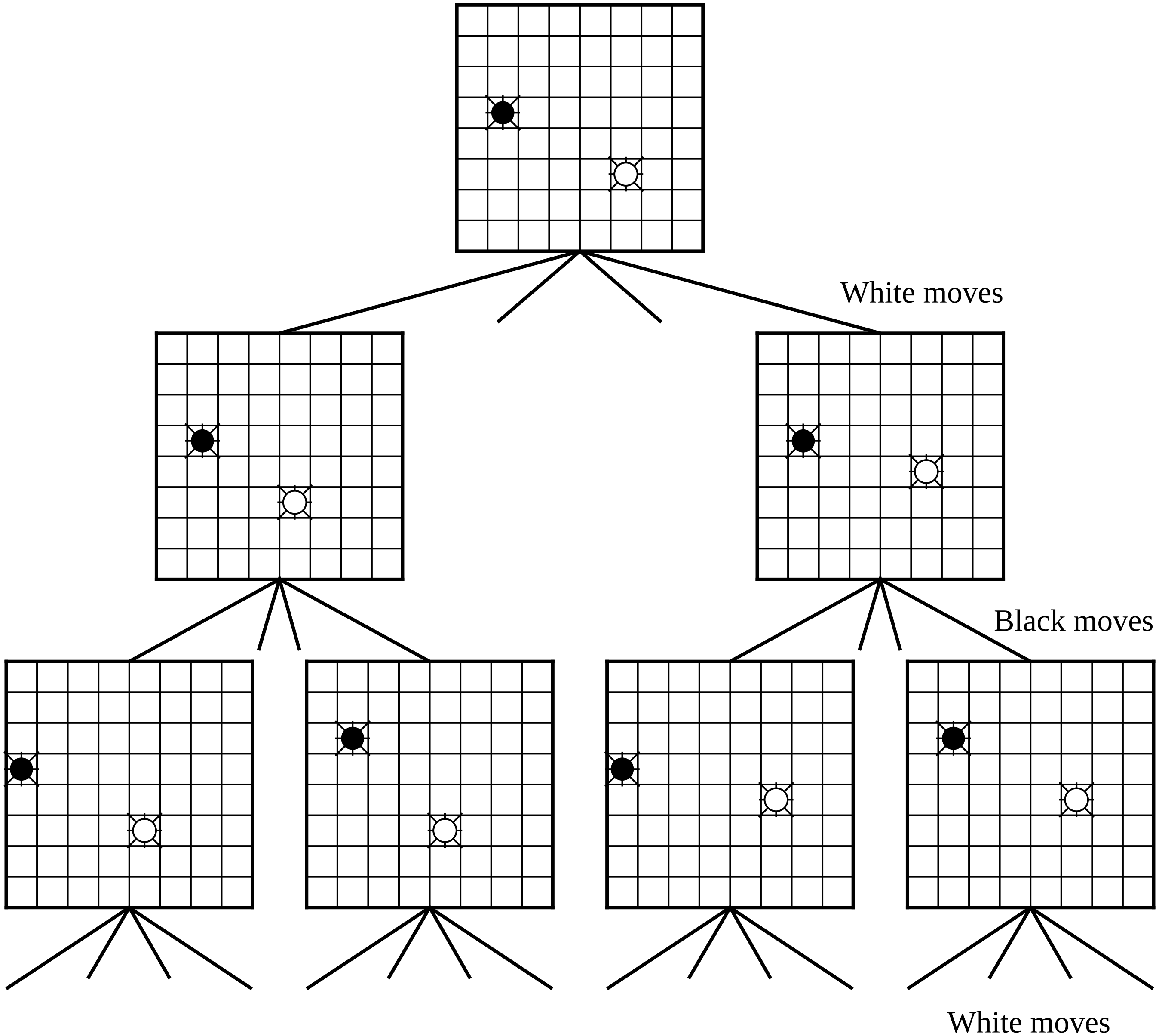
Adversarial Search



Two robots, "Black" and "White" can move in their adjacent cells. The goal of White is to be in the same cell with Black. The goal of Black is to prevent this from happening.


White moves
Black moves
White moves

Two-Agent Games

- Two-agent, perfect information, zero-sum games
- One player wins, the other player loses, or the results is a draw
- Many common games such as Chess, Checkers, and Go fall in this category
- Tic-Tac-Toe falls in the same category — simple, but we use it here
- Some games, such as Backgammon, involve an element of chance — more difficult to analyze
- State graphs can be used as before

The Minimax Procedure

- Two players: MIN and MAX, find a “best” move for MAX.
- MAX moves first, and then they alternate.
- Nodes at odd-numbered depths correspond to positions at which it is MIN’s move next; these are the MIN’s nodes (the root is at depth 0).
- The extent of searches is given in terms of lookahead measured in terms of pairs of alternating moves for MAX and MIN
- Use a static evaluation function to evaluate moves at leaf nodes.

The Minimax Procedure:

- The MIN and MAX have the same game tree.
- The MAX prefers a node having the largest evaluation, the MIN wants the opposite.
- We can assume that whenever the MIN chooses it well take the smallest number and the MAX will always take the largest.
- We can propagate the values upward based on these assumptions.

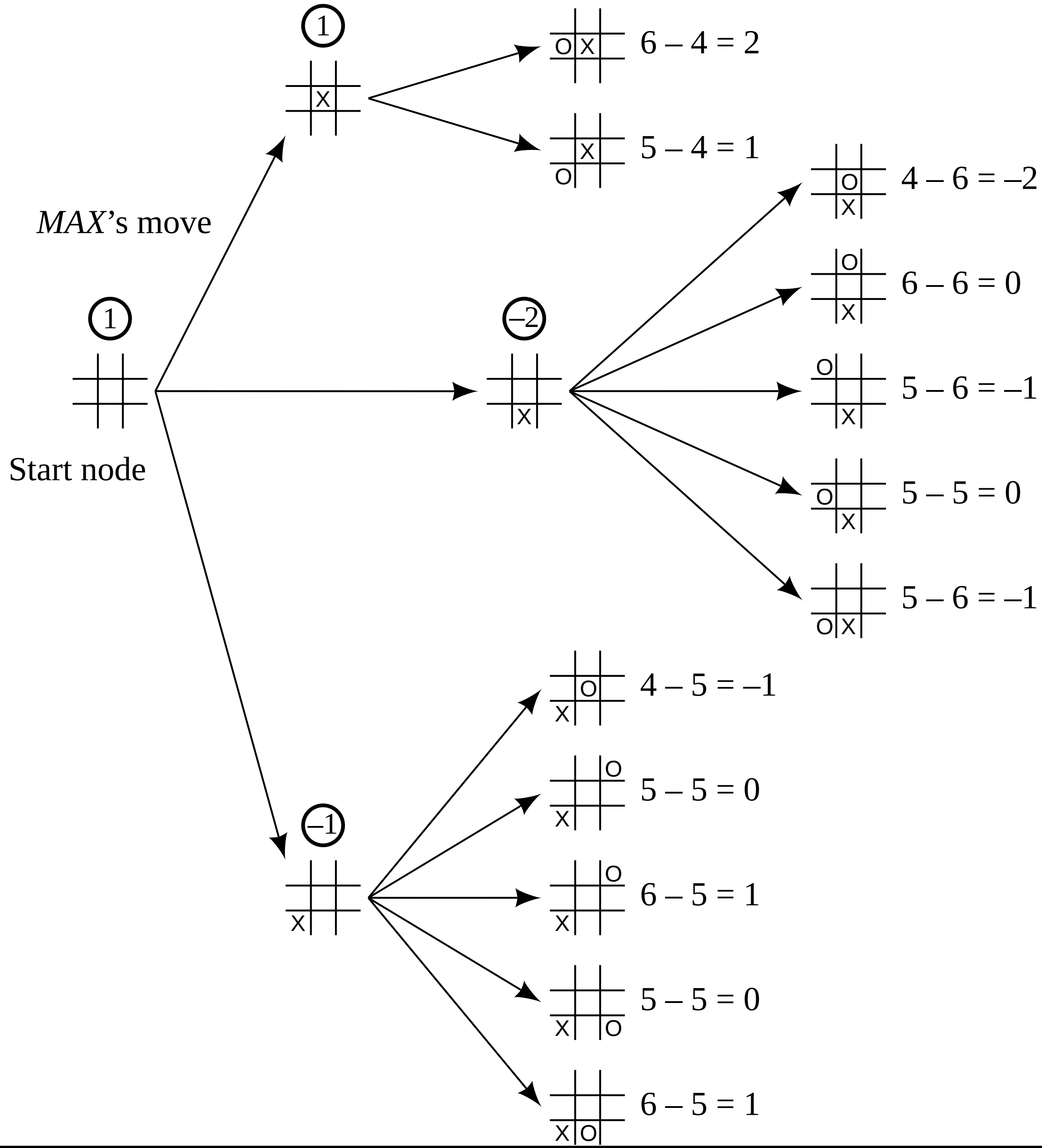
Tic-Tac-Toe

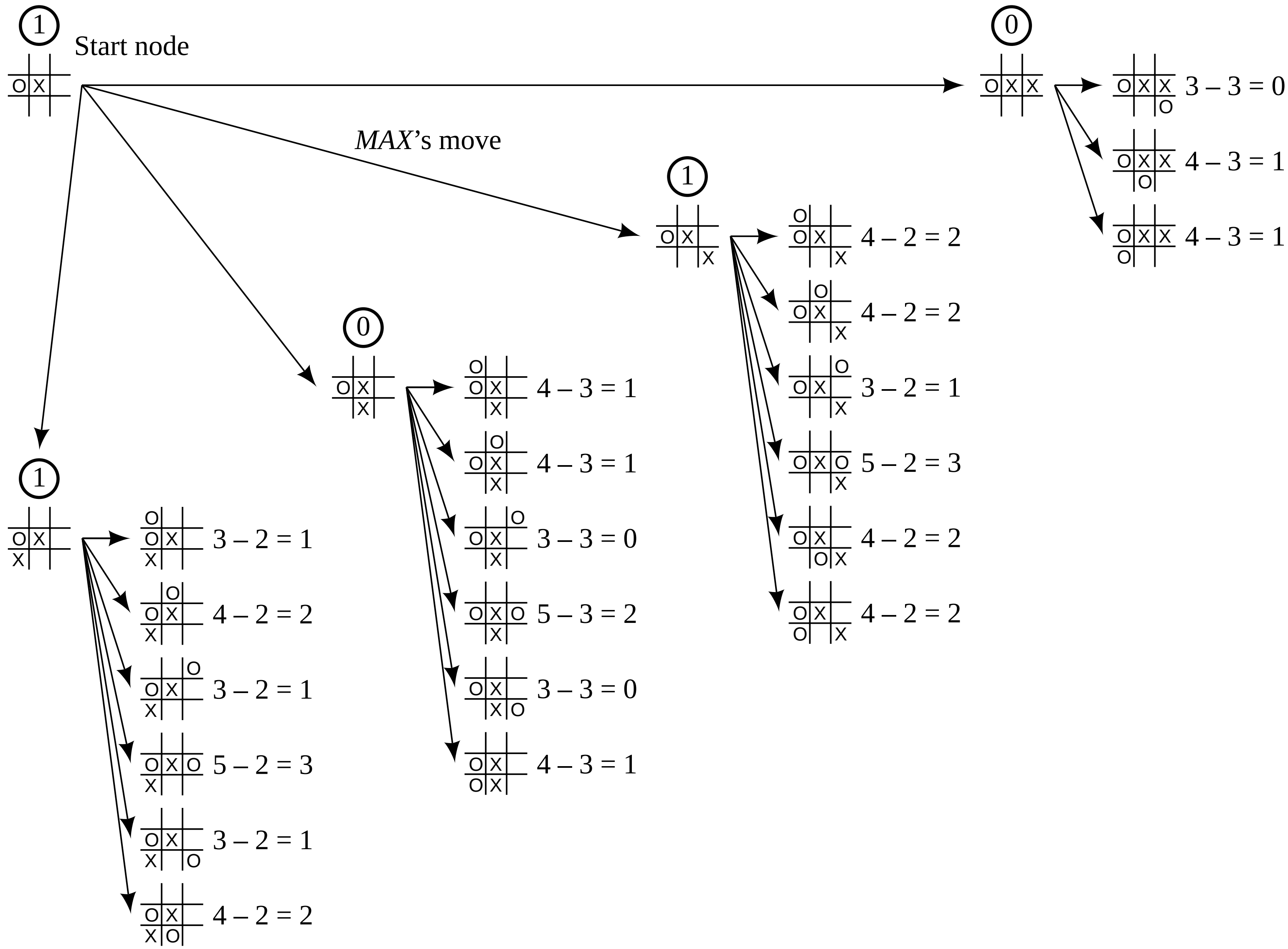
- Players alternate putting marks in a 3×3 array. The first player to have a complete row, column, or a diagonal filled in with his marks wins.
- Let our evaluation function, $e(p)$, of a position p be given by

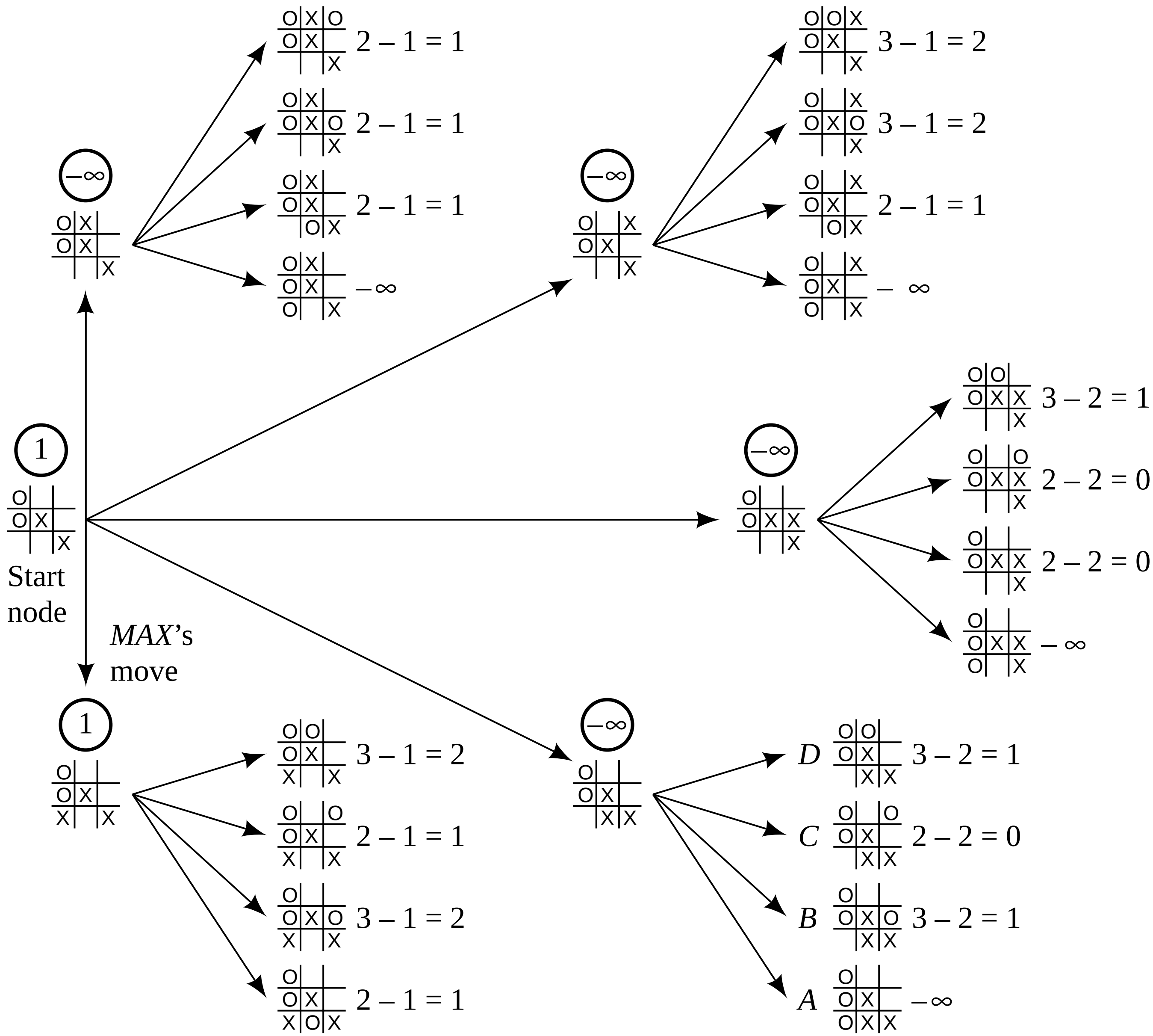
  If p is not a winning position for either player,
  $e(p) =$ (# of complete rows, columns, or diagonals that are still open for MAX) - (# of complete rows, columns, or diagonals that are still open for MIN).

  If p is a win for MAX,
  $e(p) = \infty$ (a very large positive number)

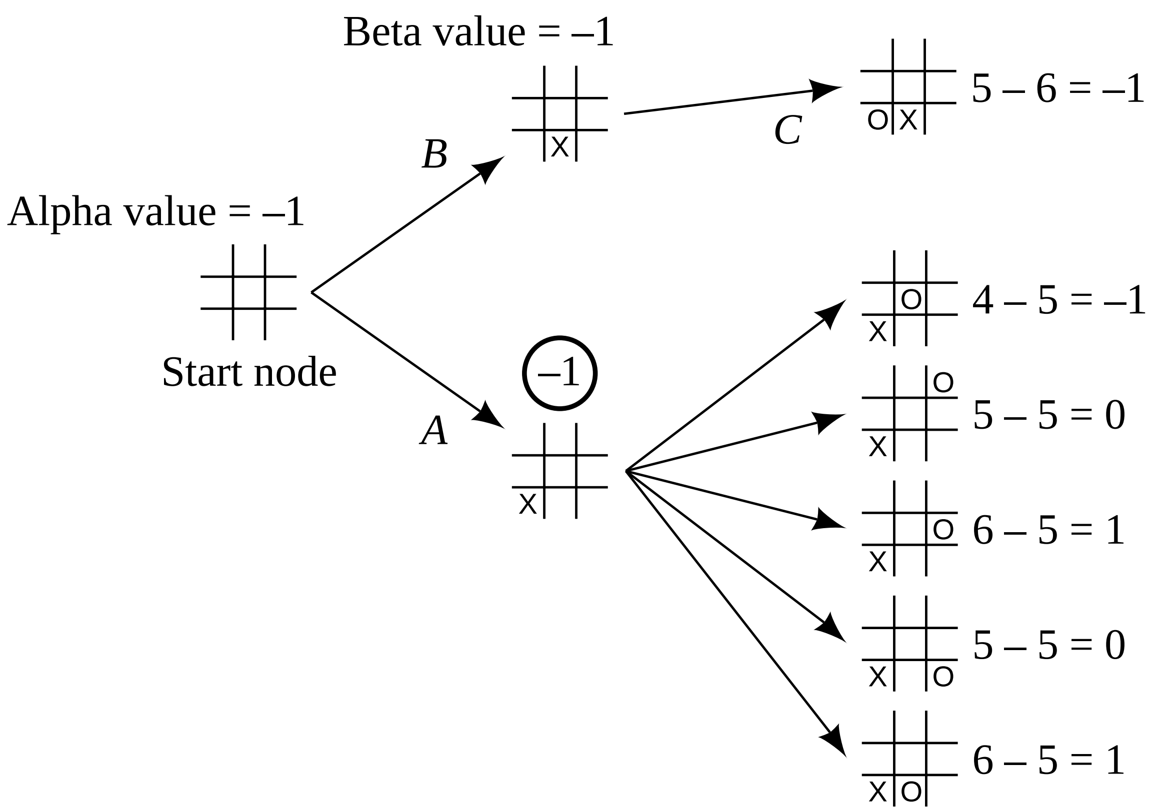
  If p is a win for MIN,
  $e(p) = -\infty$


1
6 – 4 = 2
5 – 4 = 1
MAX's move
4 – 6 = –2
6 – 6 = 0
1
–2
5 – 6 = –1
Start node
5 – 5 = 0
5 – 6 = –1
4 – 5 = –1
5 – 5 = 0
–1
6 – 5 = 1
5 – 5 = 0
6 – 5 = 1


Start node
MAX's move
1
0
1
0
1
3 – 3 = 0
4 – 3 = 1
4 – 3 = 1
4 – 2 = 2
4 – 2 = 2
3 – 2 = 1
5 – 2 = 3
4 – 2 = 2
4 – 2 = 2
4 – 3 = 1
4 – 3 = 1
3 – 3 = 0
5 – 3 = 2
3 – 3 = 0
4 – 3 = 1
3 – 2 = 1
4 – 2 = 2
3 – 2 = 1
5 – 2 = 3
3 – 2 = 1
4 – 2 = 2


2 – 1 = 1
2 – 1 = 1
2 – 1 = 1
–∞
3 – 1 = 2
3 – 1 = 2
2 – 1 = 1
– ∞
3 – 2 = 1
2 – 2 = 0
2 – 2 = 0
– ∞
1
Start node
MAX's move
1
3 – 1 = 2
2 – 1 = 1
3 – 1 = 2
2 – 1 = 1
D 3 – 2 = 1
C 2 – 2 = 0
B 3 – 2 = 1
A –∞

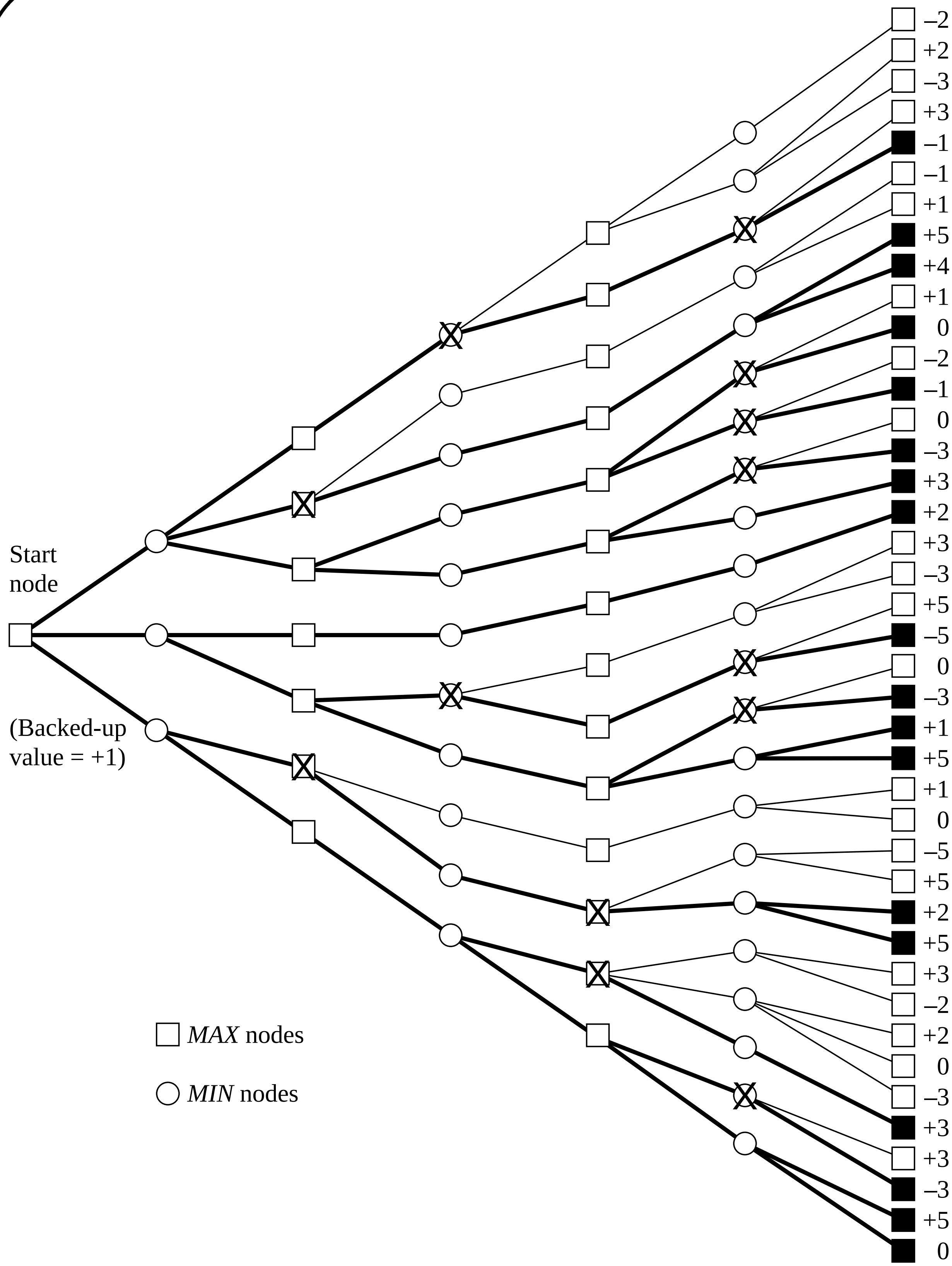
Part of the First Stage of Search in Tic-Tac-Toe



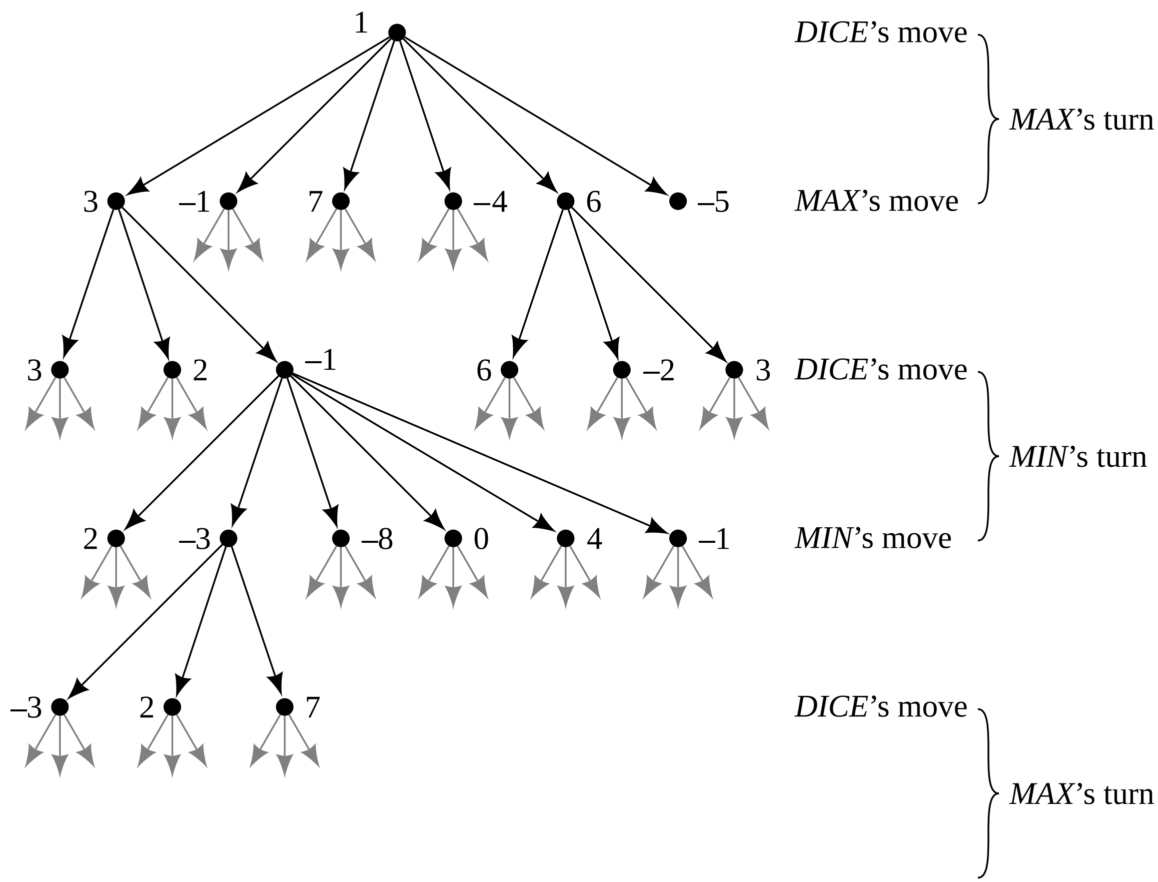
The MAX does not have to accept more than -1.

The Alpha-Beta Procedure

- Backed-up value for the MAX: a lower bound on *alpha value*.
- Backed-up value for the MIN: a lower bound on *beta value*.
- The alpha values of MAX nodes (including the start node) can never decrease.
- The beta values of MIN nodes can never icrease.
- The alpha value of a MAX node is set equal to the current largest final backed-up value of its successors.
- The beta value of a MIN node is set equal to the current smallest final backed-up value of its successors.


Start
node
(Backed-up
value = +1)
MAX nodes
MIN nodes
–2
+2
–3
+3
–1
–1
+1
+5
+4
+1
0
–2
–1
0
–3
+3
+2
+3
–3
+5
–5
0
–3
+1
+5
+1
0
–5
+5
+2
+5
+3
–2
+2
0
–3
+3
+3
–3
+5
0

Games of Chance



Use expected average values in evaluation.