



ARTIFICIAL INTELLIGENCE

2023/2024 Semester 2

Adversarial Search: Chapter 5

Section 1 – 4

Outline

- Games
- Optimal decisions in games
- α - β pruning
- Imperfect, real-time decisions

Games and adversarial search



Why study games?

- Games are a traditional hallmark of intelligence
- Games are easy to formalize
- Games can be a good model of real-world competitive activities
 - Military confrontations, negotiation, auctions, etc.

What kind of games?

- **Abstraction:** To describe a game we must capture every relevant aspect of the game. Such as:
 - Chess
 - Tic-tac-toe
 - ...
- **Accessible environments:** Such games are characterized by perfect information
- **Search:** game-playing then consists of a search through possible game positions
- **Unpredictable opponent:** introduces **uncertainty** thus game-playing must deal with **contingency problems**

Searching for the next move

- **Complexity:** many games have a huge search space
 - **Chess:** $b = 35, m = 100 \Rightarrow nodes = 35^{100}$
if each node takes about 1 ns to explore
then each move will take about 10^{50} millennia
to calculate.
- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate
 - 1. Pruning:** makes the search more efficient by discarding portions of the search tree that cannot improve quality result.
 - 2. Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search.

Two-player games

- A game formulated as a search problem:
 - Initial state: ?
 - Operators: ?
 - Terminal state: ?
 - Utility function: ?

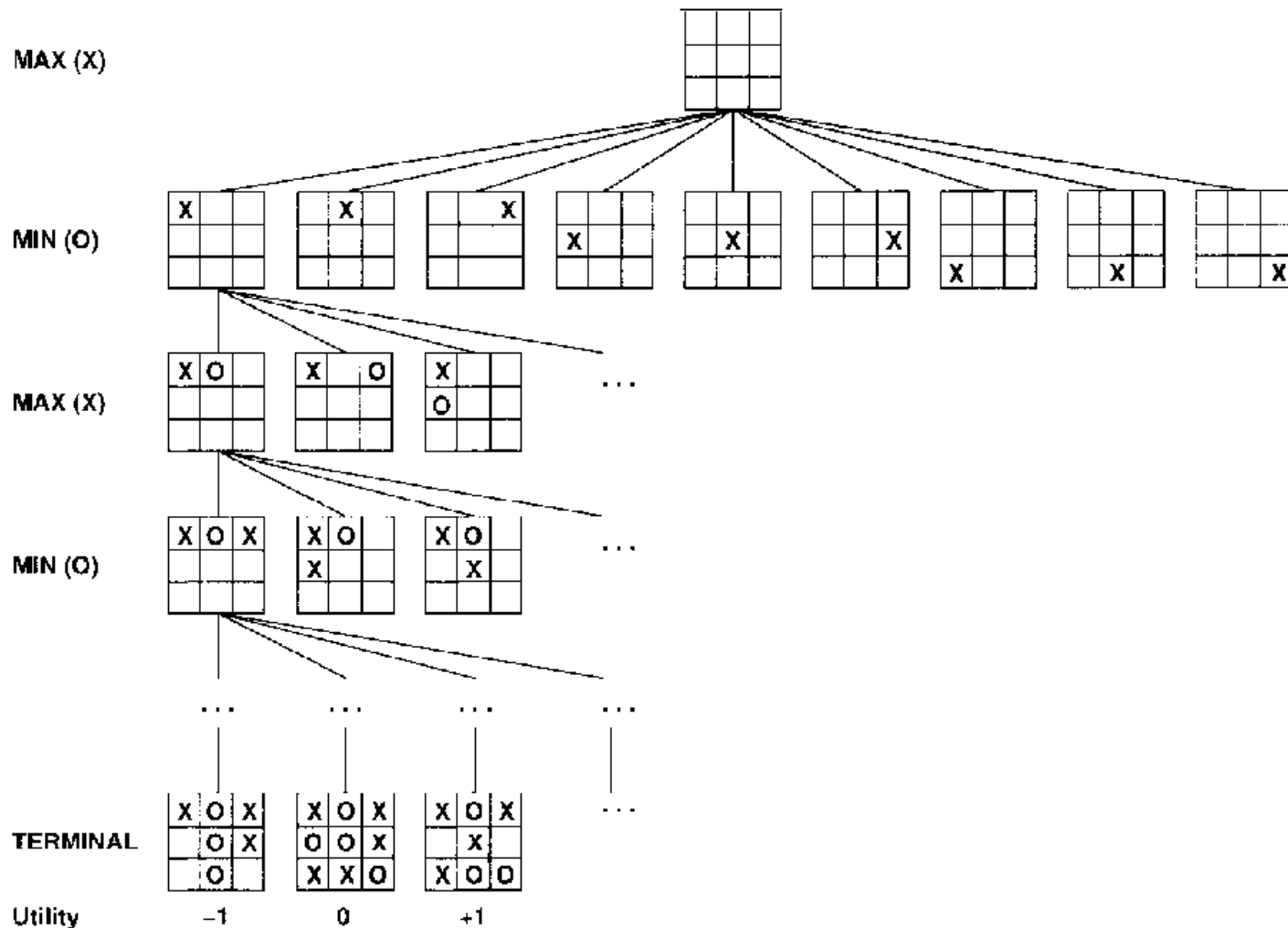
Two-player games

- A game formulated as a search problem:
 - Initial state: board position and turn
 - Operators: definition of legal moves
 - Terminal state: conditions for when game is over
 - Utility function: a numeric value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win. (AKA **payoff function**)

Games vs. search problems

- "Unpredictable" opponent \rightarrow specifying a move for every possible opponent reply
- For Chess, average branching 35; and search 50 moves by each player
 - Time limits (35^{100}) \rightarrow unlikely to find goal, must approximate

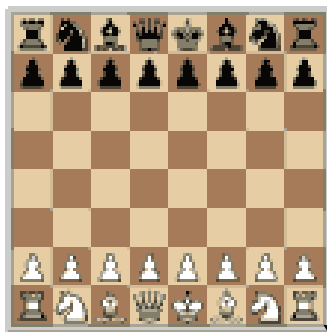
Example: Tic-Tac-Toe



Type of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

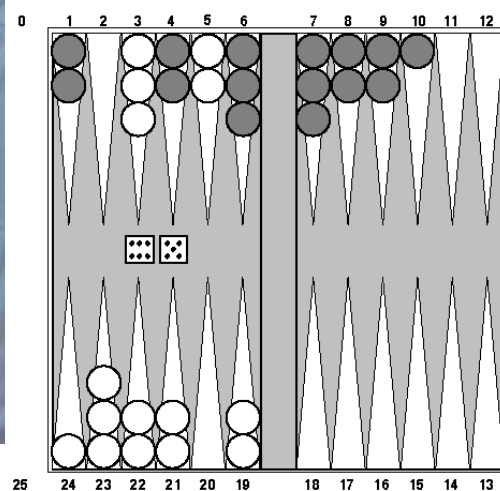
Type of games



The board set for play



Red to play



deterministic

chance

perfect information

chess, checkers,
go, othello

backgammon
monopoly

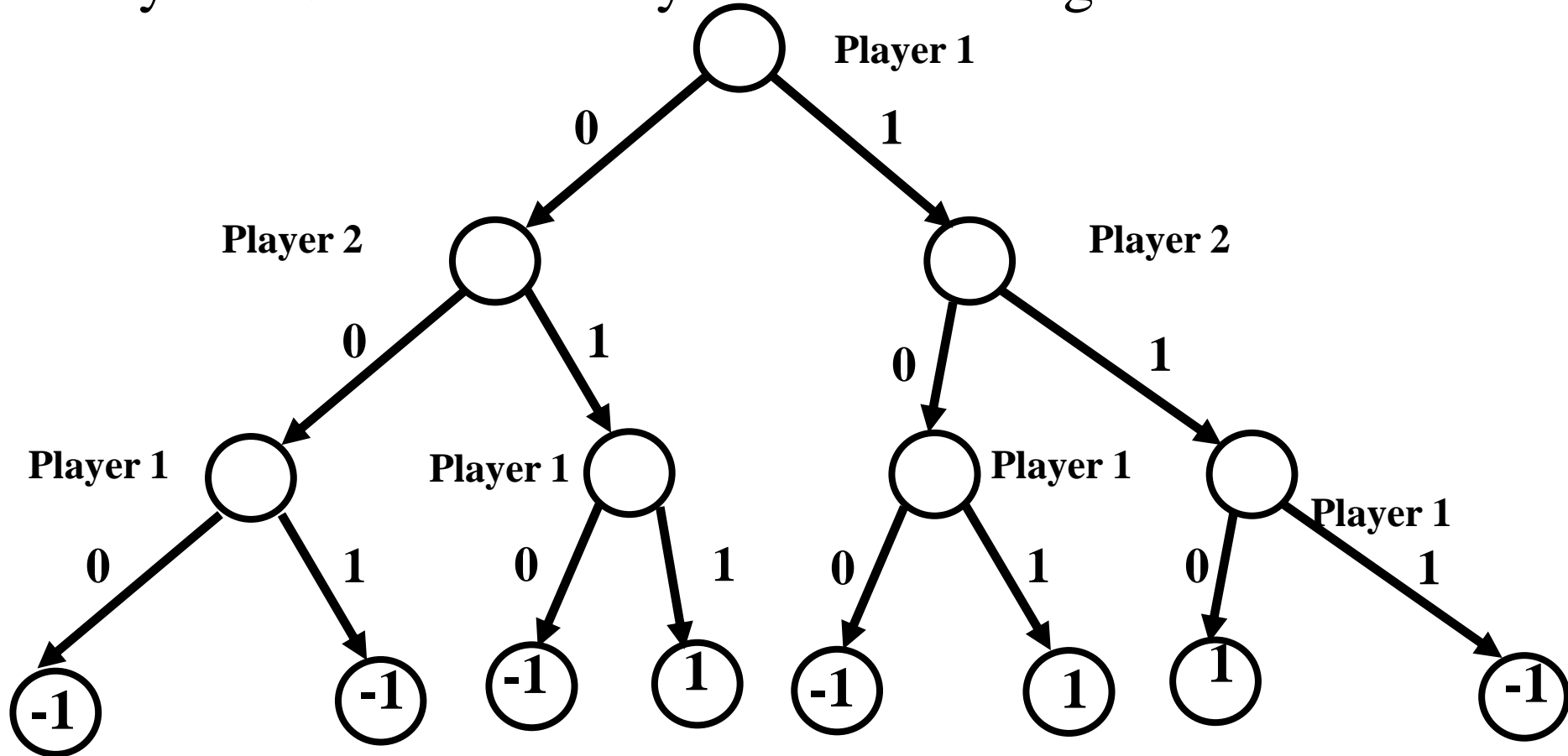
imperfect information

bridge, poker, scrabble
nuclear war



“Sum to 2” games

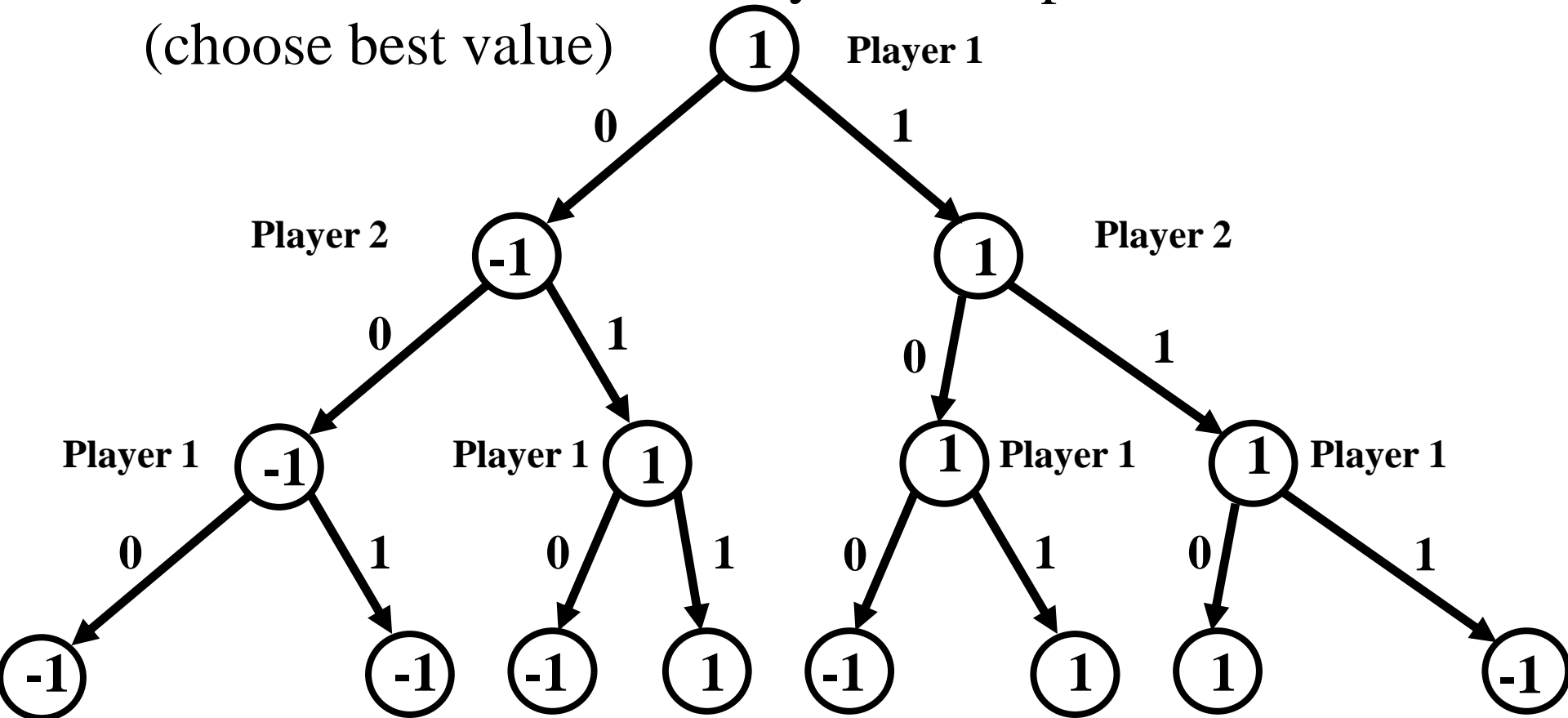
- Player 1 moves, then player 2, finally player 1 again
- Move = 0 or 1
- Player 1 wins if and only if all moves together sum to 2



Player 1's utility is in the leaves; player 2's utility is the negative of this

Backward induction (aka. minimax)

- From leaves upward, analyze best decision for player at node, give node a value
 - Once we know values, easy to find optimal action (choose best value)



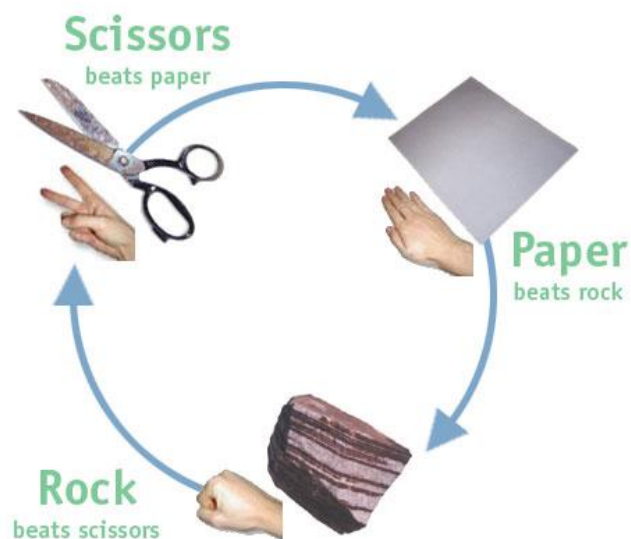
Two-Agent Games

- Two-agent, perfect information, zero-sum games
- Two agents move in turn until either one of them *wins* or the result is a draw.
- Each player has a complete model of the environment and of its own and the other's possible actions and their effects.



Simultaneous single-move games







- Players must choose their actions at the same time, without knowing what the others will do
 - Form of partial observability



Player 2

Normal form representation:

Player 1

			
	0,0	1,-1	-1,1
	-1,1	0,0	1,-1
	1,-1	-1,1	0,0

Payoff matrix

(row player's utility is listed first)

Is this a zero-sum game?

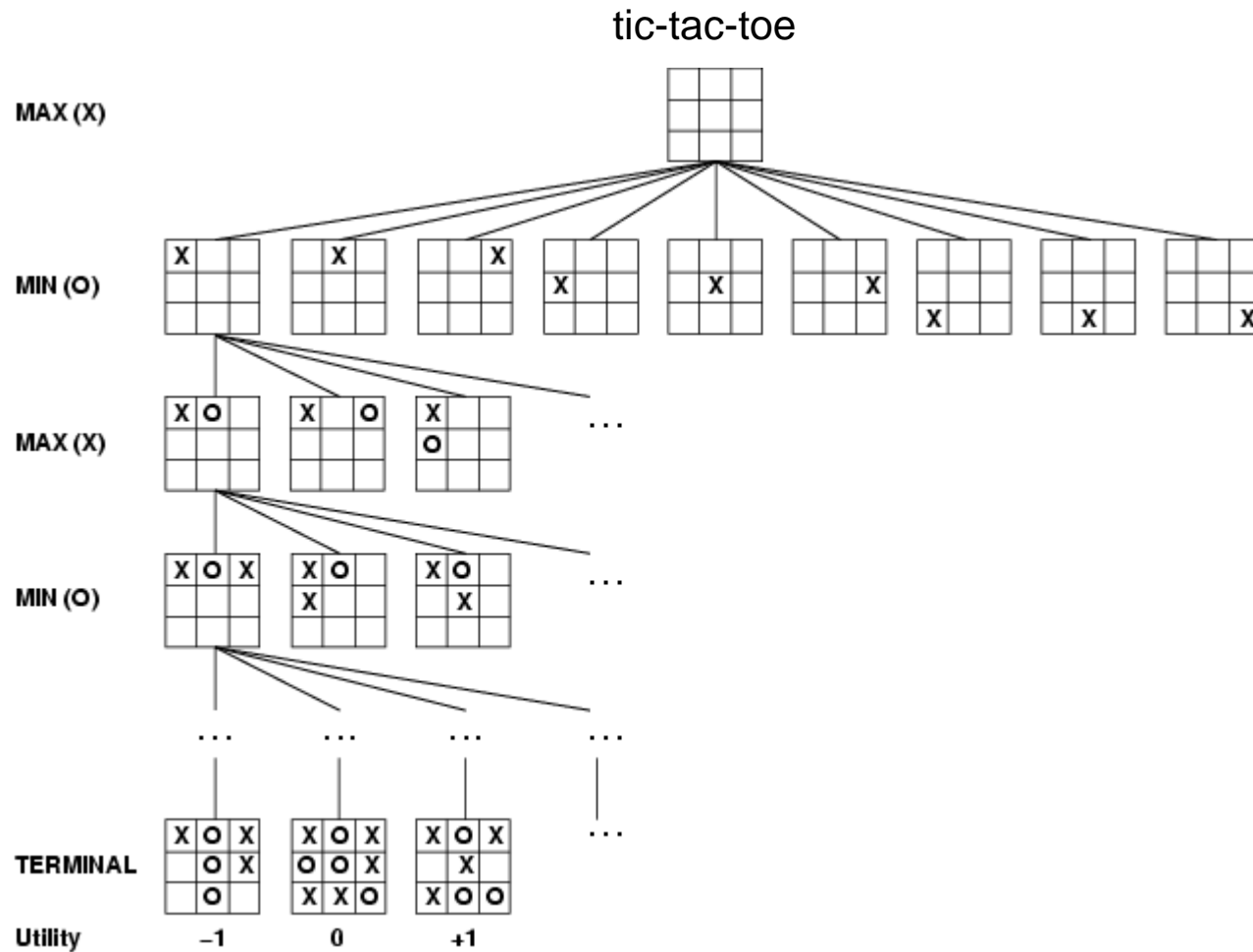
Minimax Procedure (1)

- Two player : *MAX* and *MIN*
- Task : find a “best” move for *MAX*
- Assume that *MAX* moves first, and that the two players move alternately.
- *MAX* node
 - nodes at even-numbered depths correspond to positions in which it is *MAX*’s move next
- *MIN* node
 - nodes at odd-numbered depths correspond to positions in which it is *MIN*’s move next

Minimax Procedure (2)

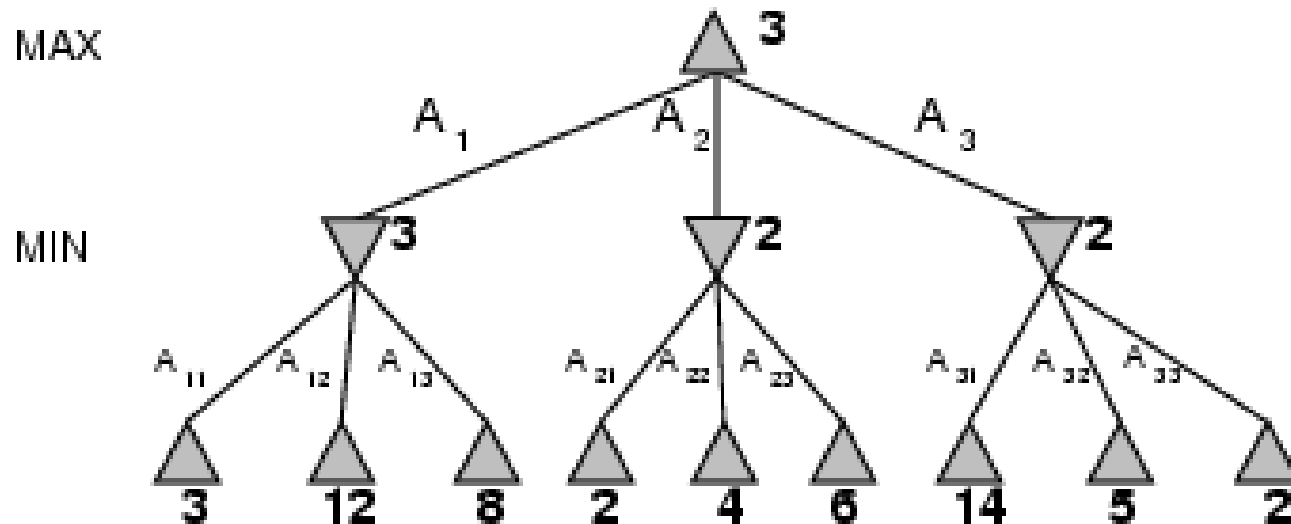
- Estimate of the best-first move
 - apply *a static evaluation function* to the leaf nodes
 - measure the “**worth**” of the leaf nodes.
 - The measurement is based on various features thought to influence this worth.
 - Usually, analyze game trees to adopt the convention
 - game positions favorable to *MAX* cause the evaluation function to have a **positive value**
 - positions favorable to *MIN* cause the evaluation function to have **negative value**
 - Values near zero correspond to game positions not particularly favorable to either *MAX* or *MIN*.

Game tree (2-player, deterministic, turns)

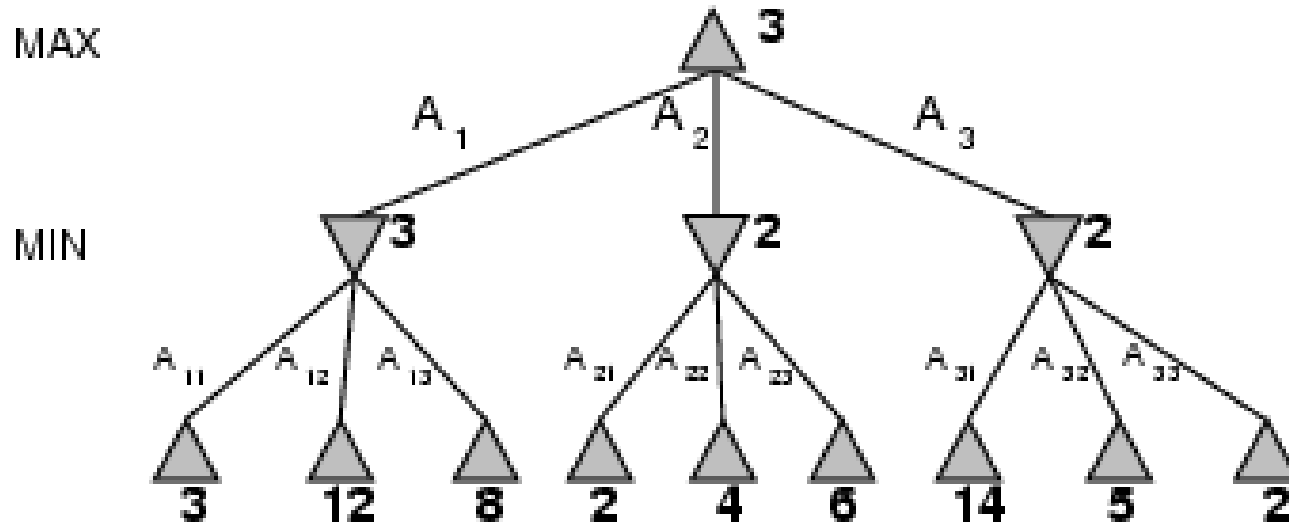


Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest **minimax value**
= best achievable payoff against best play
- E.g., 2-ply game:



Minimax



- **Minimax**($state$) =
 - Utility($state$) if $state$ is terminal
 - max **Minimax**(successors($state$)) if $player = \text{MAX}$
 - min **Minimax**(successors($state$)) if $player = \text{MIN}$

Minimax algorithm

function MINIMAX-DECISION(*state*) *returns an action*

$v \leftarrow \text{MAX-VALUE}(\textit{state})$

return the *action* in SUCCESSORS(*state*) with value *v*

function MAX-VALUE(*state*) *returns a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

for *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

return *v*

function MIN-VALUE(*state*) *returns a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow \infty$

for *a, s* in SUCCESSORS(*state*) **do**

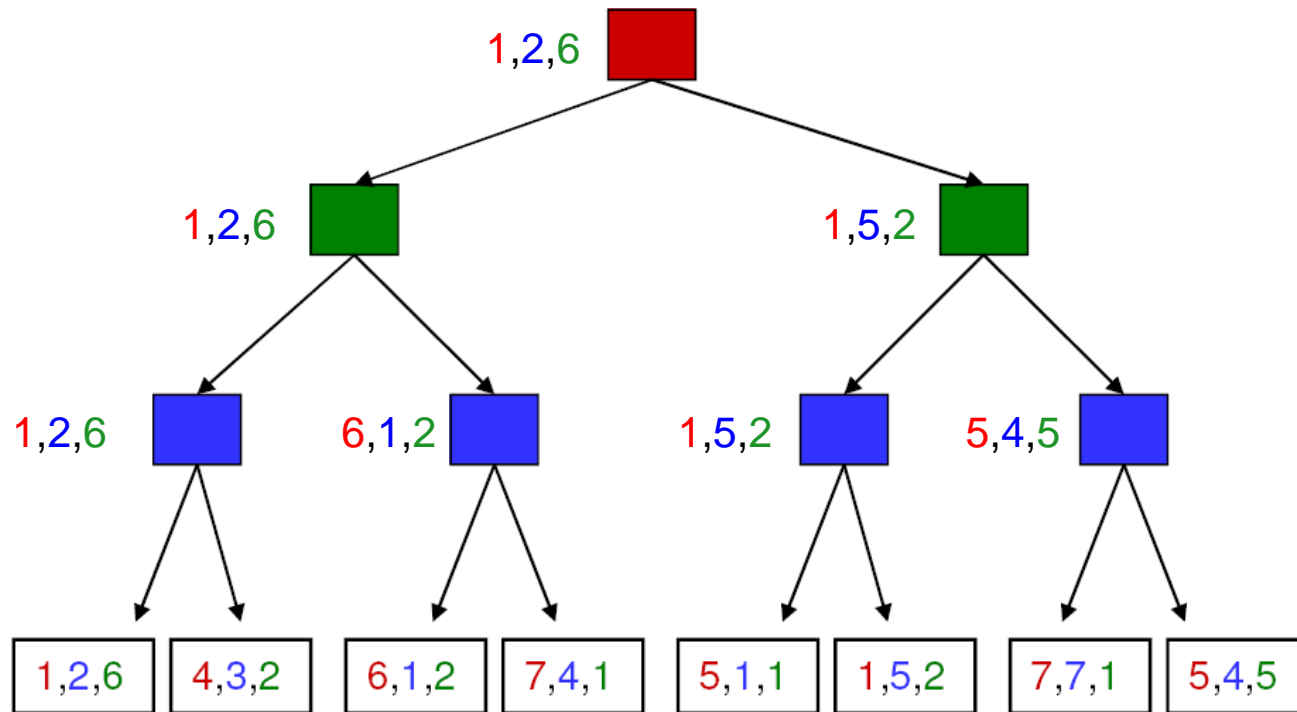
$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

return *v*

Properties of minimax

- Complete? Yes (if tree is finite)
 - Optimal? Yes (against an optimal opponent)
 - Time complexity? $O(b^m)$ (b -legal moves; m - max tree depth)
 - Space complexity? $O(bm)$ (depth-first exploration)
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
→ exact solution completely infeasible

More general games

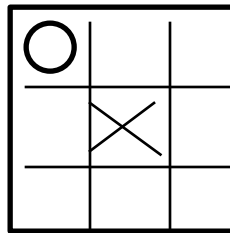
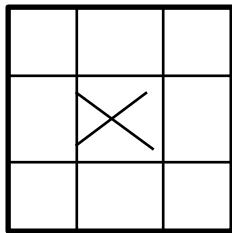


- More than two players, non-zero-sum
- Utilities are now tuples
- Each player maximizes their own utility at each node
- Utilities get propagated (*backed up*) from children to parents

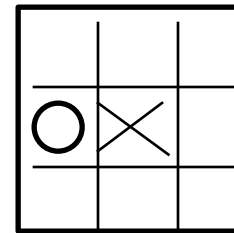
Example : Tic-Tac-Toe (1)

- *MAX* marks **crosses** *MIN* marks **circles**
- it is *MAX*'s turn to play first.
 - With a depth bound of 2, conduct a breadth-first search
 - evaluation function $e(p)$ of a position p
 - If p is not a winning for either player,
 $e(p) = (\text{no. of complete rows, columns, or diagonals that are still open for } MAX) - (\text{no. of complete rows, columns, or diagonals that are still open for } MIN)$
 - If p is a win of *MAX*, $e(p) = \infty$
 - If p is a win of *MIN*, $e(p) = -\infty$

- evaluation function $e(p)$ of a position p
 - If p is not a winning for either player,
 $e(p) = (\text{no. of complete rows, columns, or diagonals that are still open for } MAX) - (\text{no. of complete rows, columns, or diagonals that are still open for } MIN)$

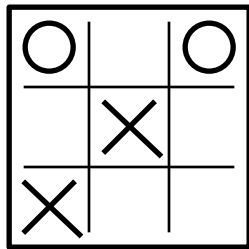


$$e(p) = 5 - 4 = 1$$

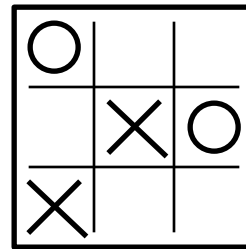


$$e(p) = 6 - 4 = 2$$

- evaluation function $e(p)$ of a position p
 - If p is not a winning for either player,
 $e(p) = (\text{no. of complete rows, columns, or diagonals that are still open for } MAX) - (\text{no. of complete rows, columns, or diagonals that are still open for } MIN)$

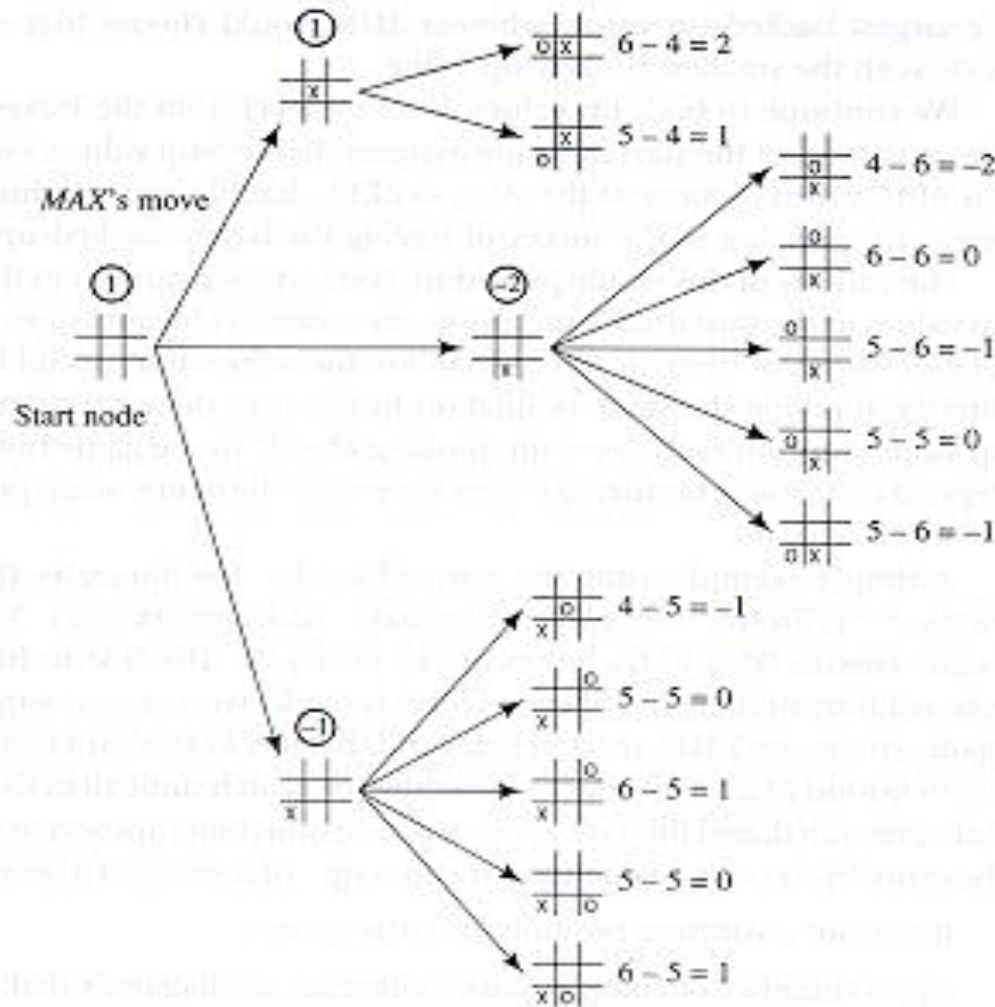


$e(p) =$

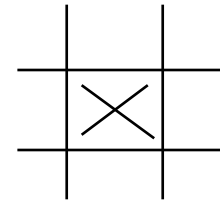


$e(p) =$

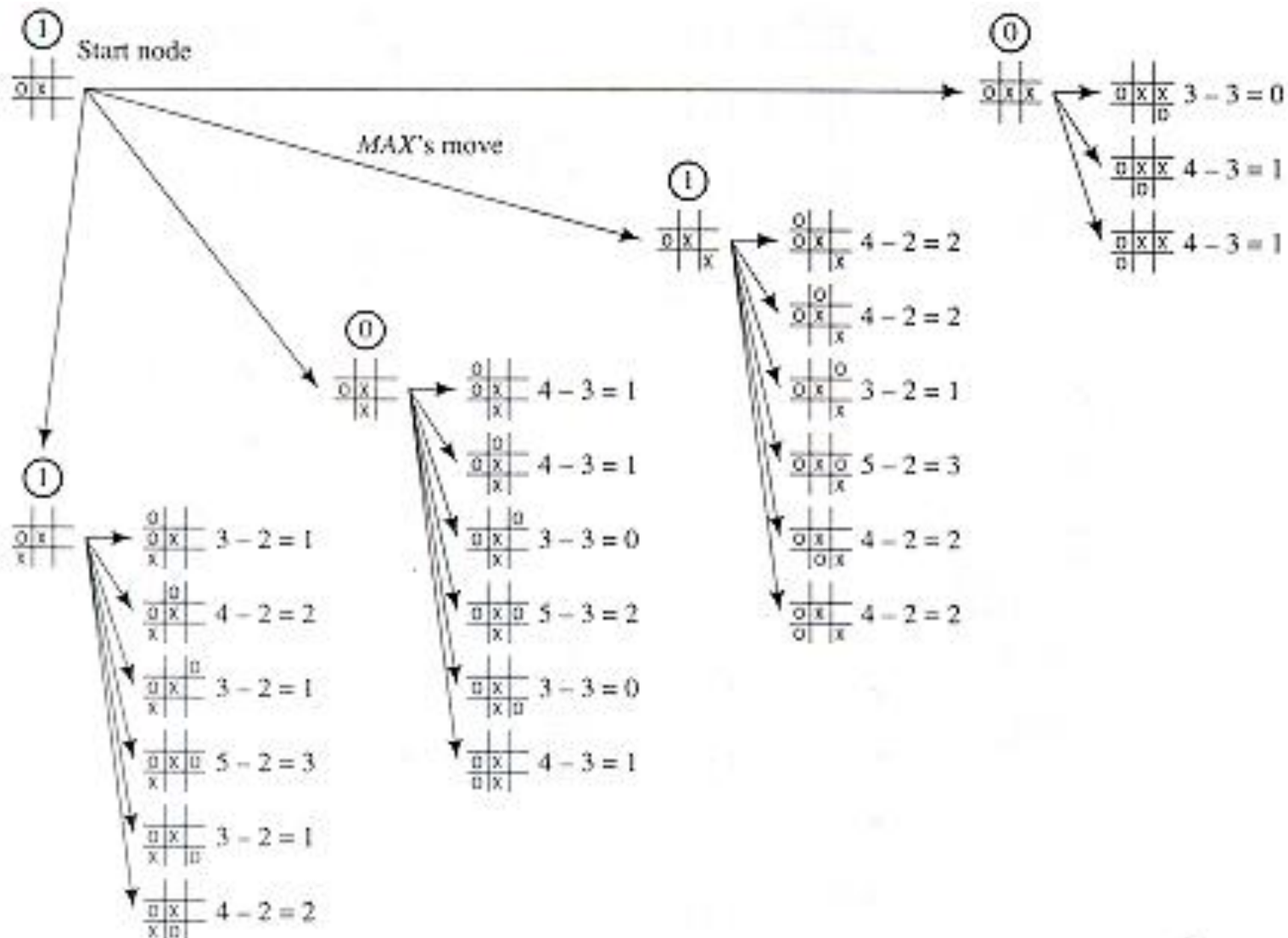
Example : Tic-Tac-Toe (2)



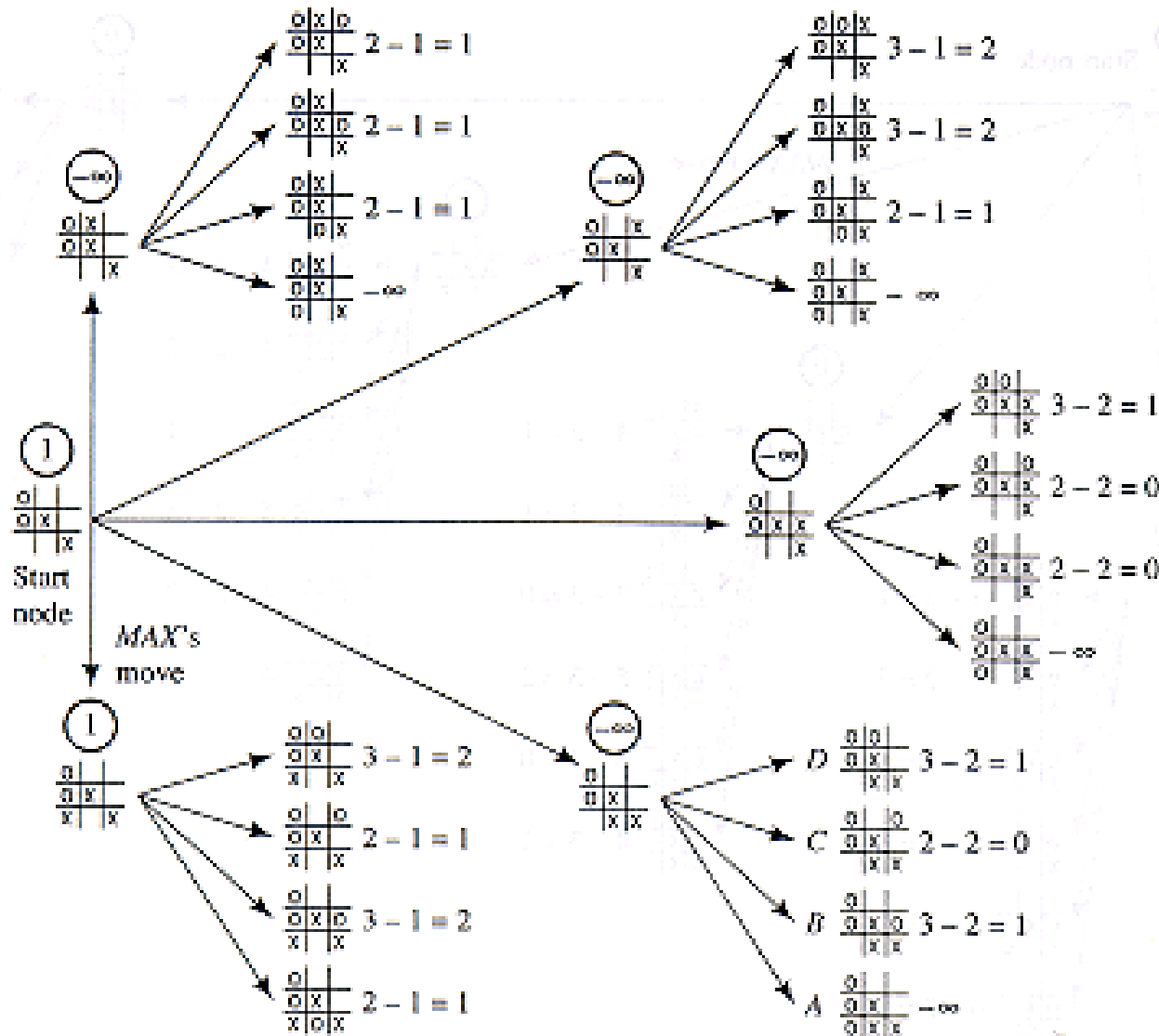
- First move



Example : Tic-Tac-Toe (3)

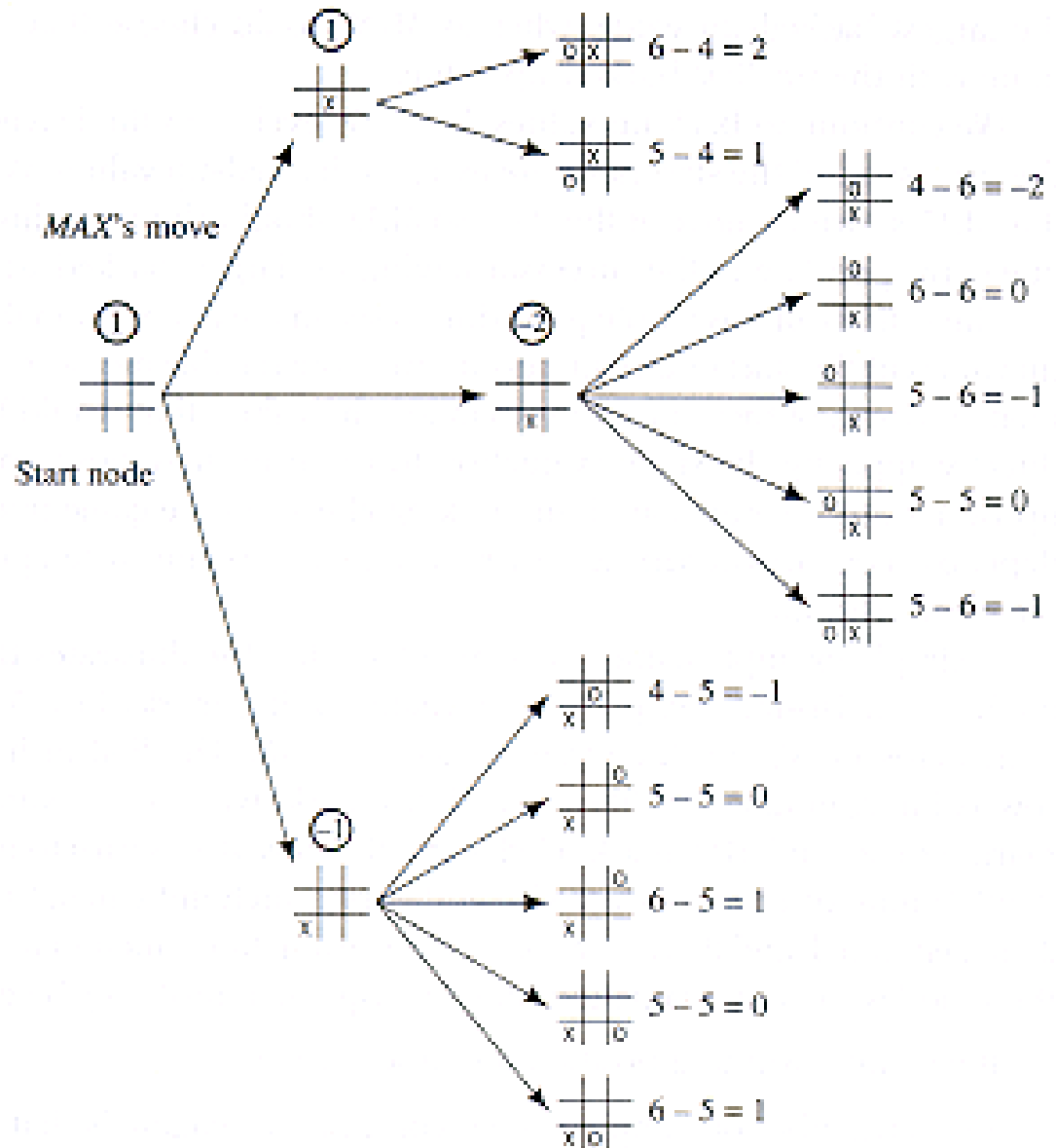


Example : Tic-Tac-Toe (4)

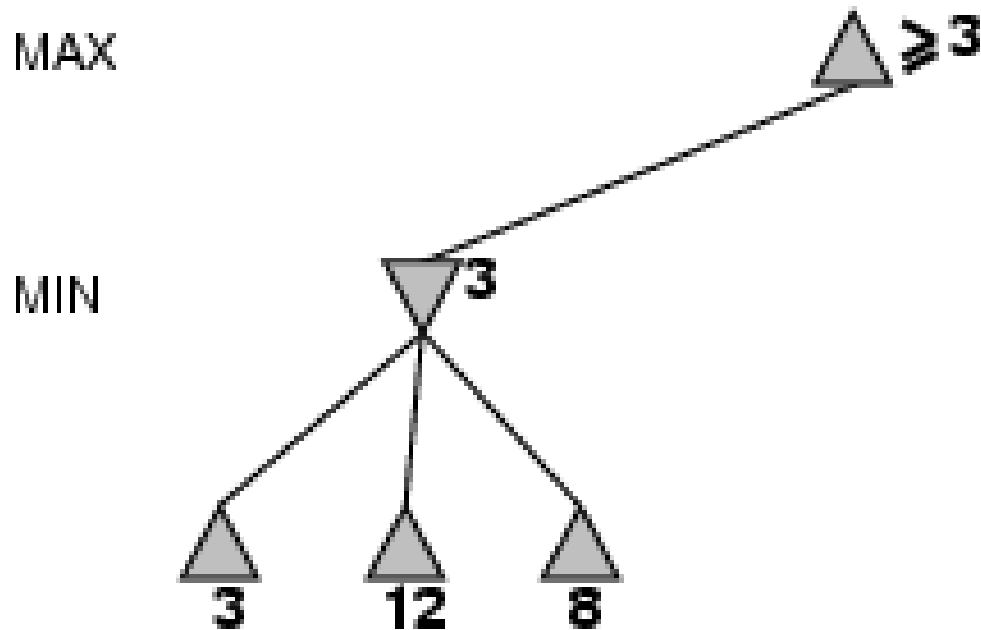


Question?

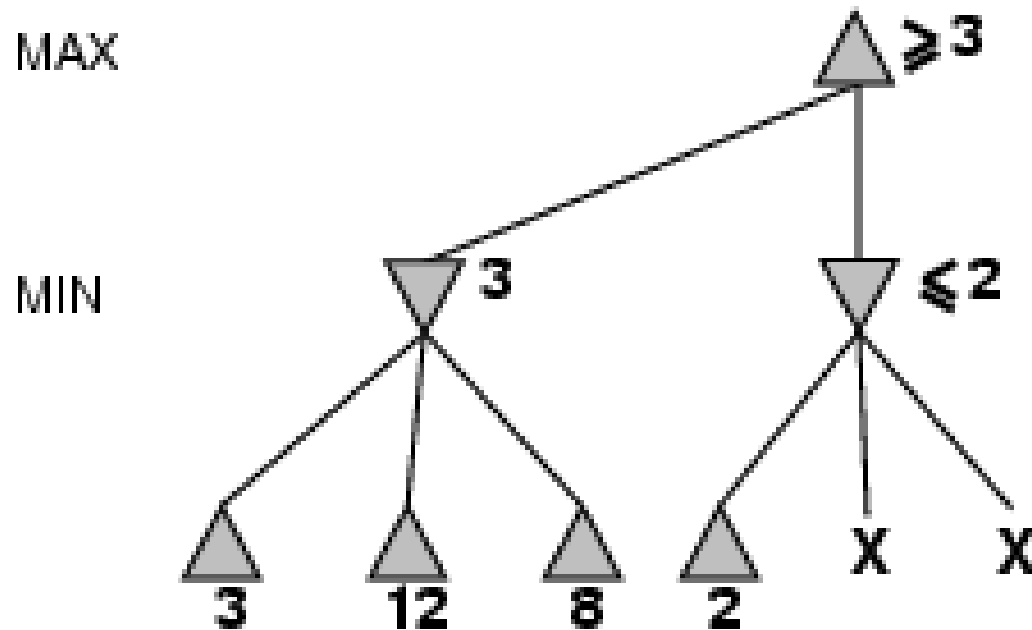
- How to improve search efficiency?
- It is possible to cut-off some unnecessary subtrees?



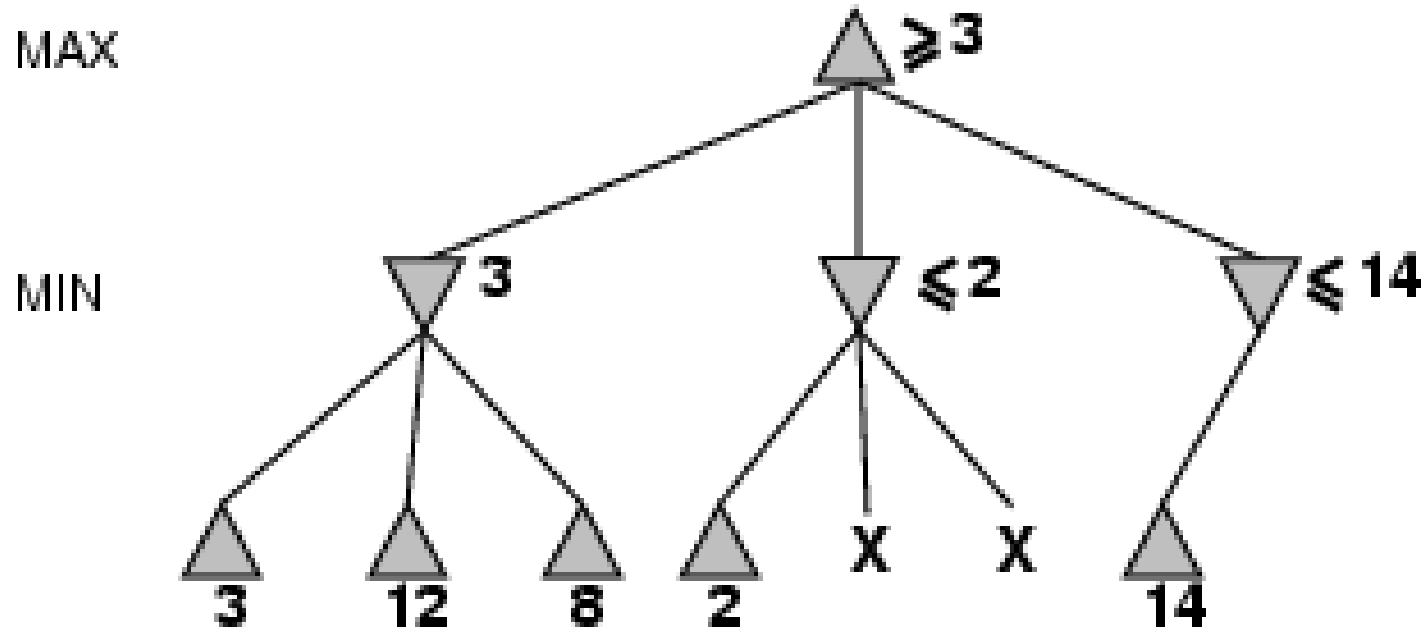
α - β Pruning Example



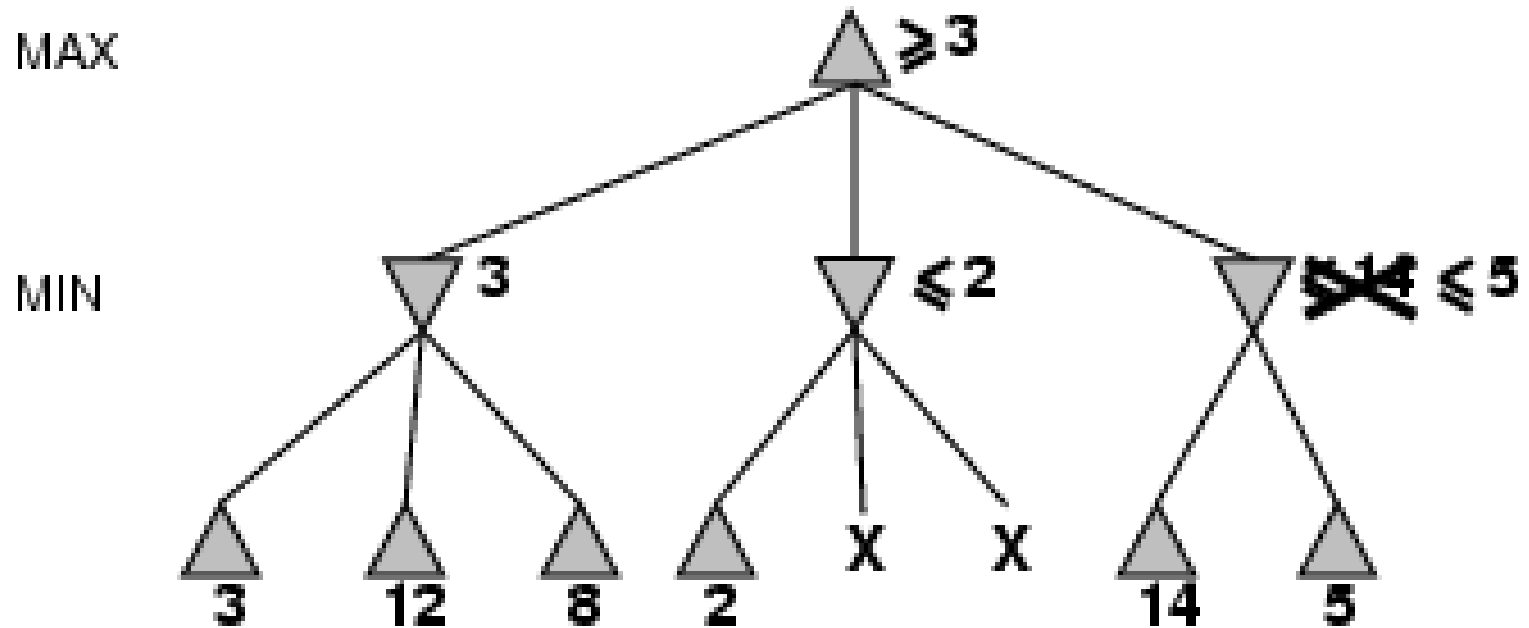
α - β pruning example



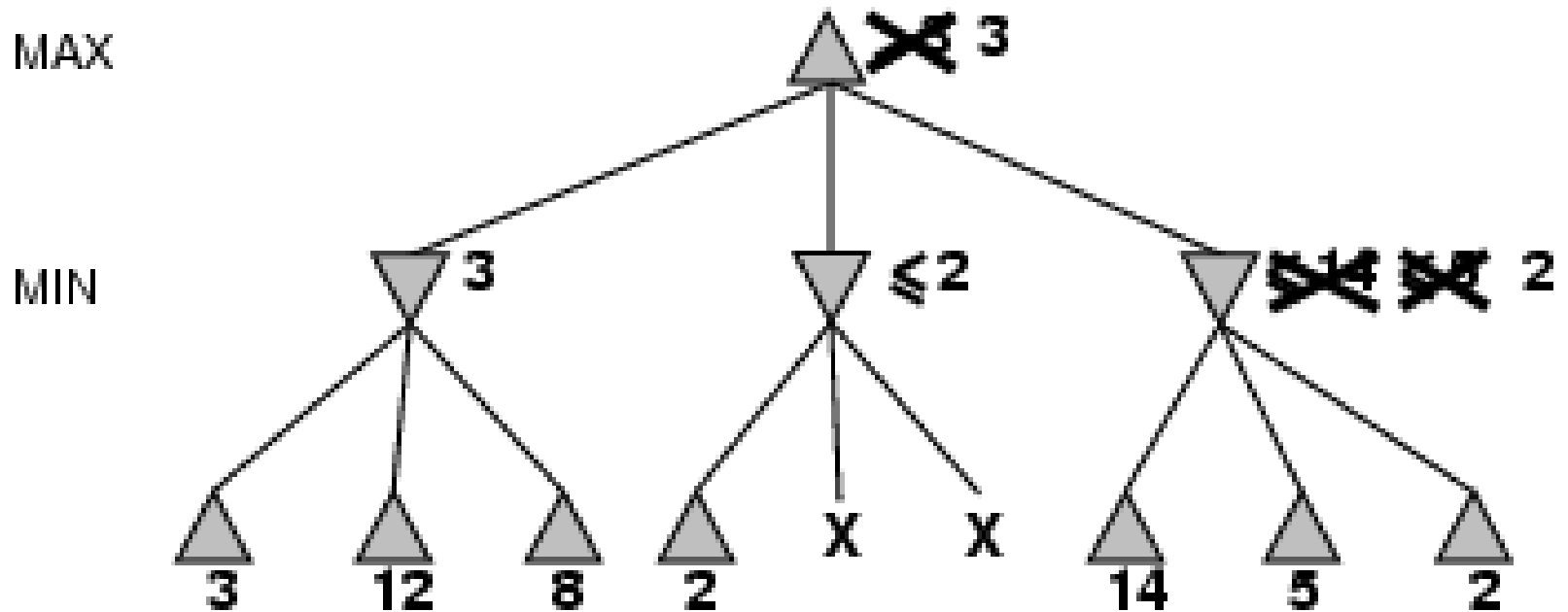
α - β pruning example



α - β pruning example



α - β pruning example



Properties of α - β

- ▶ Pruning **does not** affect final result
- ▶ Good move ordering improves effectiveness of pruning
- ▶ With "perfect ordering," time complexity = $O(b^{m/2})$
→ doubles depth of search

Why is it called α - β ?

- ▶ α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- ▶ If v is worse than α , *max* will avoid it
→ prune that branch
- ▶ Define β similarly for *min*

MAX

MIN

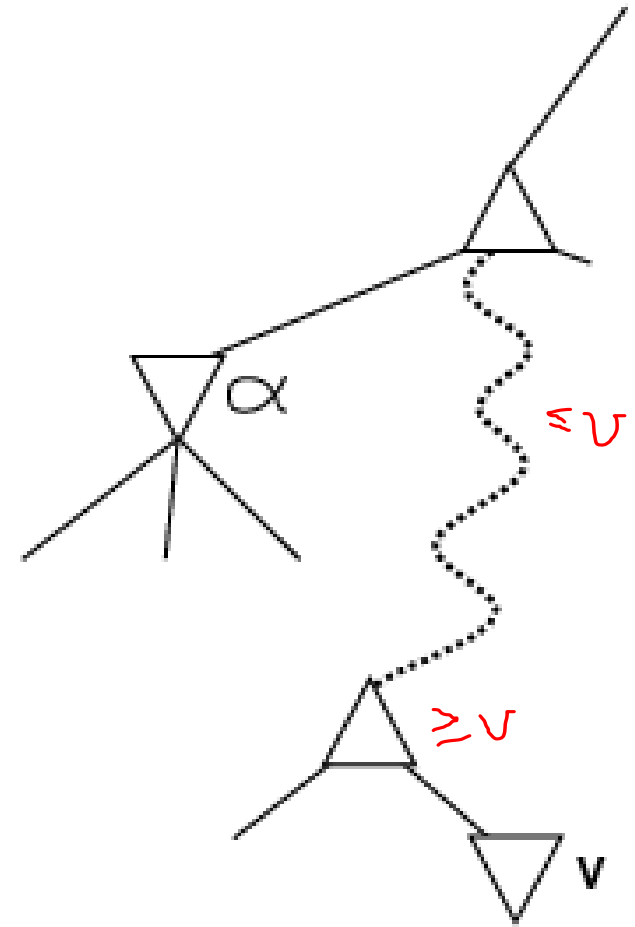
..

..

..

MAX

MIN



α - β pruning: search cutoff

- **Pruning**: eliminating a branch of the search tree from consideration without exhaustive examination of each node
- **α - β pruning**: the basic idea is to prune portions of the search tree that cannot improve the utility value of the max or min node, by just considering the values of nodes seen so far.
- **Does it work?** Yes, in roughly cuts the branching factor from b to \sqrt{b} resulting in double as far look-ahead than pure minimax

The α - β algorithm

function ALPHA-BETA-SEARCH(*state*) *returns an action*

inputs: *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

return the *action* in SUCCESSORS(*state*) with value *v*

function MAX-VALUE(*state*, α , β) *returns a utility value*

inputs: *state*, current state in game

α , the value of the best alternative for MAX along the path to *state*

β , the value of the best alternative for MIN along the path to *state*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

for *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

if $v \geq \beta$ **then return** *v*

$\alpha \leftarrow \text{MAX}(\alpha, v)$

return *v*

The α - β algorithm

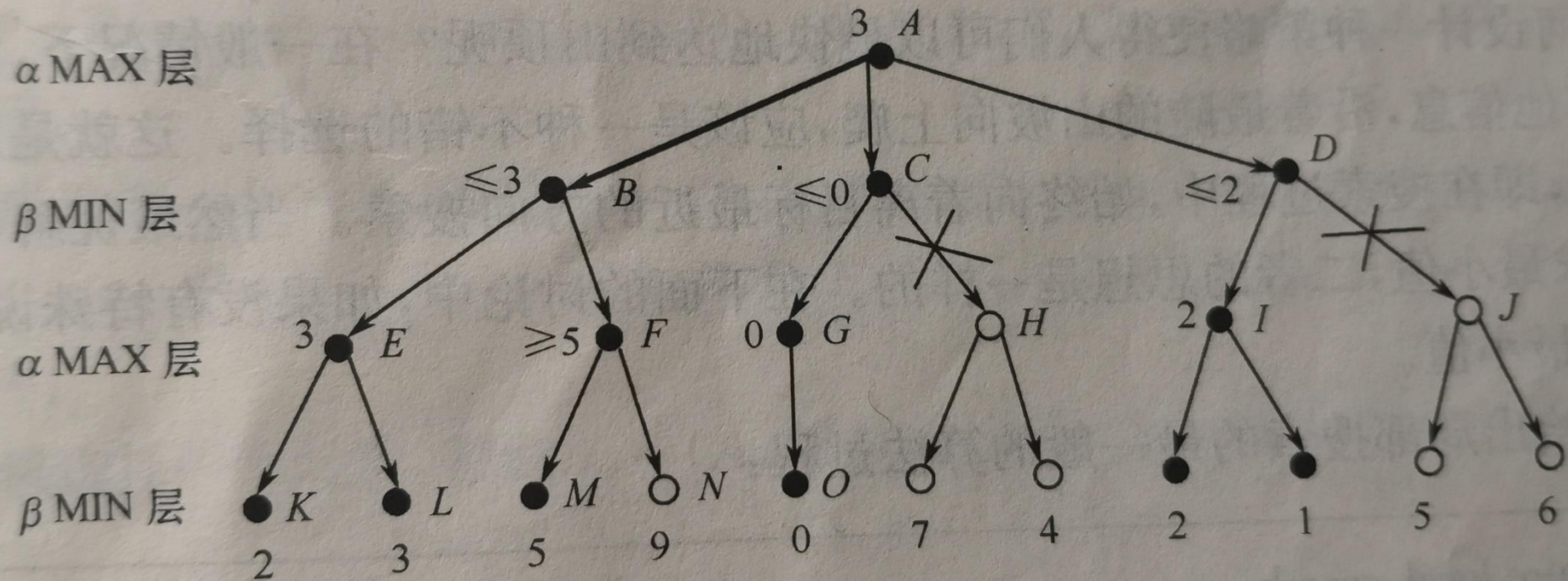
```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
            $\alpha$ , the value of the best alternative for MAX along the path to state
            $\beta$ , the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$ 
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return  $v$ 
```

More on the α - β algorithm

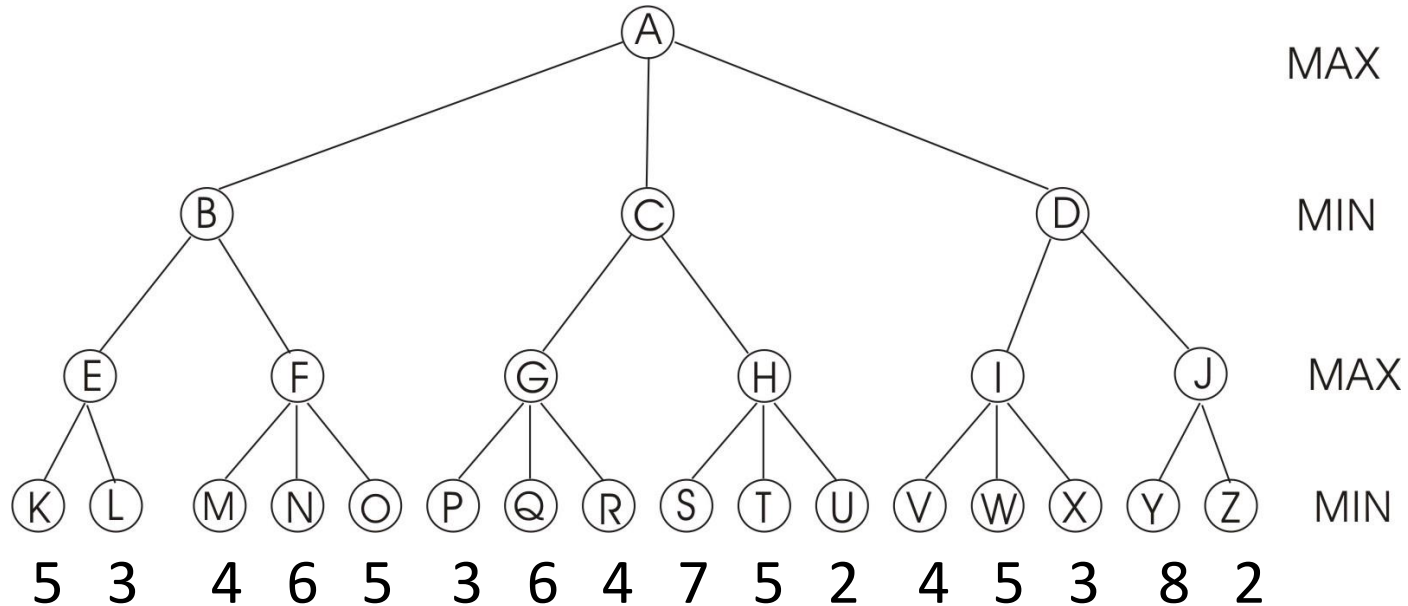
- Same basic idea as minimax, but keep track of α , β and prune (cut away) branches of the tree that we know will not contain the solution.
- Because minimax is depth-first, let's consider nodes along a given path in the tree. Then, as we go along this path, we keep track of:
 - α : Best choice so far for MAX
 - β : Best choice so far for MIN

An Example



An Exercise

- (a) Compute the backed-up values calculated by the minimax algorithm. Show your answer by writing values at the appropriate nodes in the above tree.
- (b) Which nodes will not be examined by the alpha-beta procedure?



Resource limits

Suppose we have 100 secs, explore 10^4 nodes/sec
→ 10^6 nodes per move

Standard approach:

- ▶ **cutoff test:**

 - e.g., depth limit

- ▶ **evaluation function**

 - = estimated desirability of position

Evaluation functions

- For chess, typically **linear** weighted sum of **features**

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

- e.g., $w_1 = 9$ with
- $f_1(s) = (\text{number of white queens}) - (\text{number of black queens})$, etc.
-
- First, the evaluation function should order the *terminal states in the same way as the* true utility function;
- Second, the **computation** must not take too long!
- Third, for nonterminal states, the evaluation function should be strongly correlated with the **actual chances of winning**.

Cutting off search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

**TERMINAL-TEST-->if CUTOFF-TEST(stated, depth) then
return EVAL(state)**

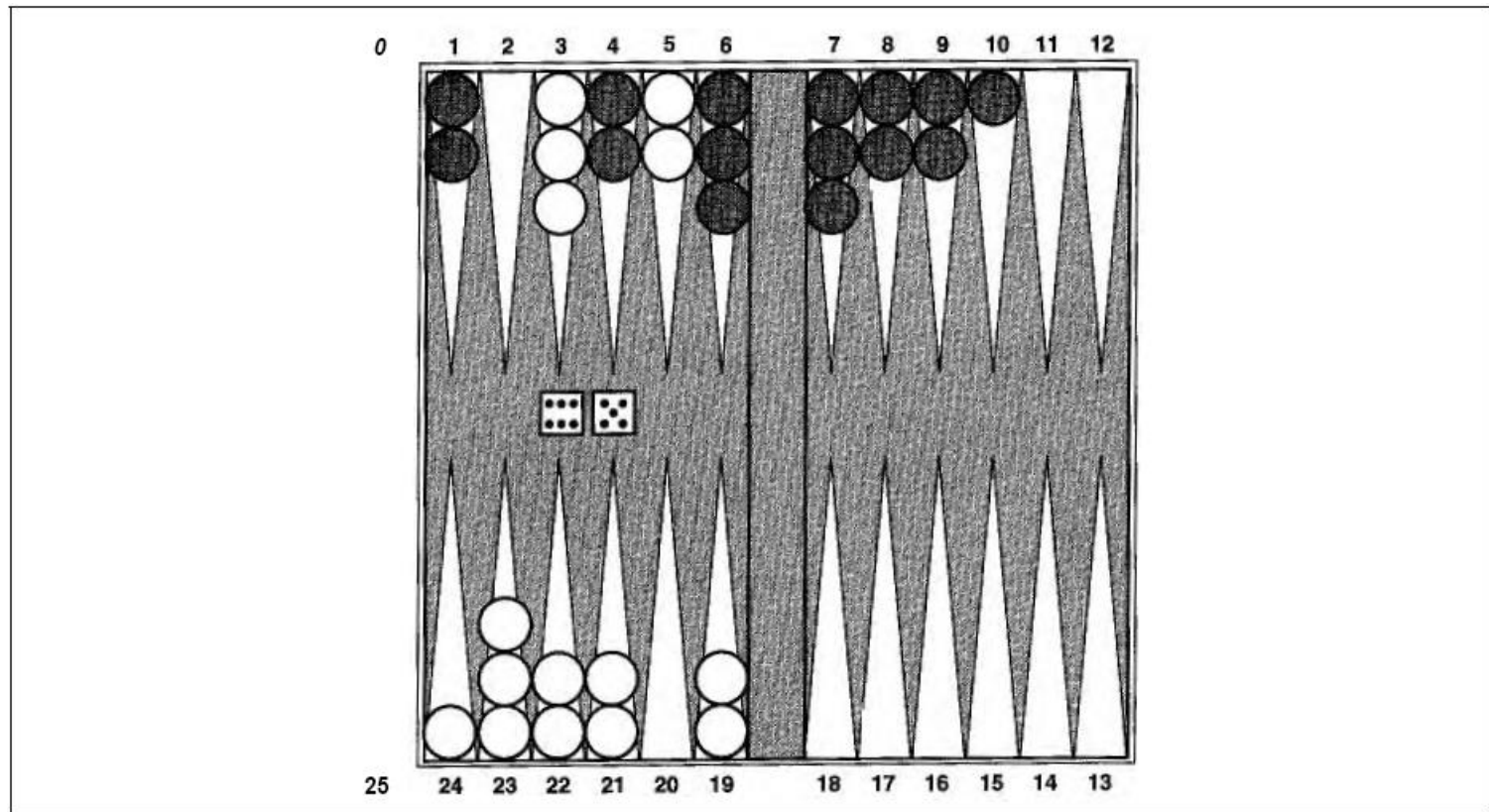
Does it work in practice?

$$b^m = 10^6, b=35 \rightarrow m=4$$

4-ply lookahead is a hopeless chess player!

- 4-ply \approx human novice
- 8-ply \approx typical PC, human master
- 12-ply \approx Deep Blue, Kasparov

Game Include an Element of Chance



Backgammon

Game Include an Element of Chance

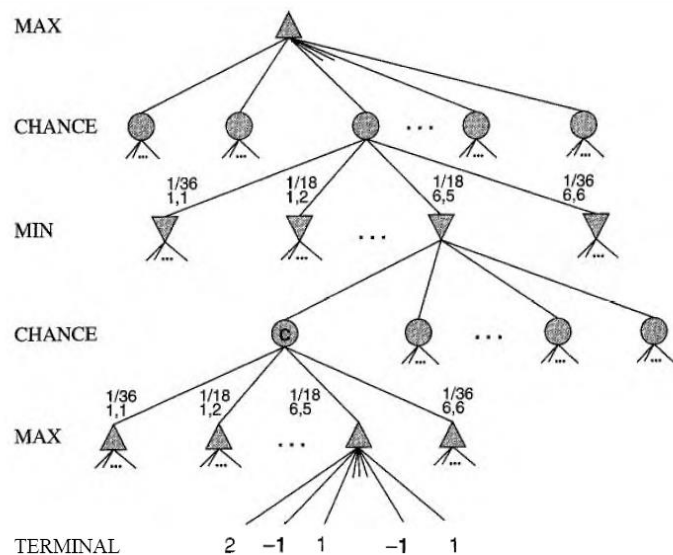


Figure 6.11 Schematic game tree for a backgammon position.

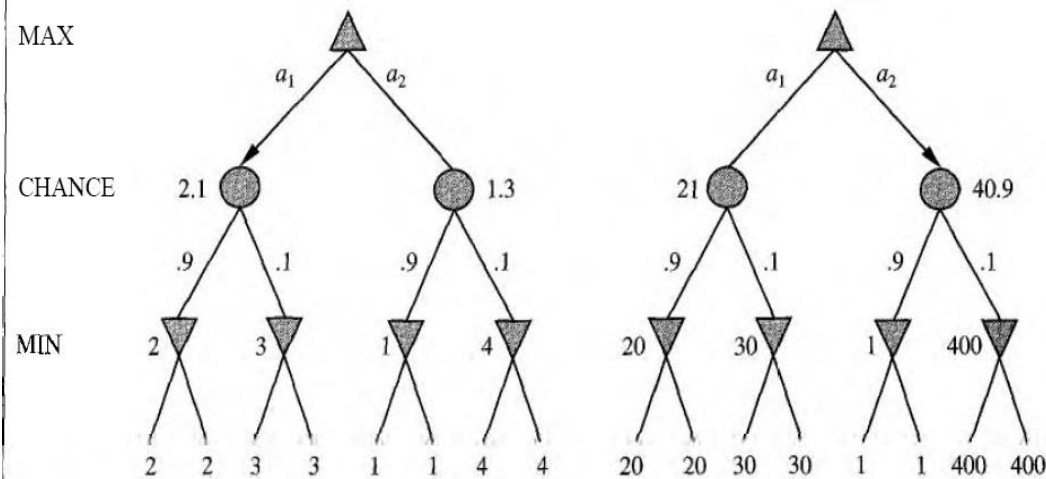


Figure 6.12 An order-preserving transformation on leaf values changes the best move.

$\text{EXPECTIMINIMAX}(n) =$

$$\begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in \text{Successors}(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

Deterministic games in practice

- Checkers: **Chinook** ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- Chess: **Deep Blue** defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- **Othello**: human champions refuse to compete against computers, who are too good.
- **Go**: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

The state of the art for some games

- Chess:
 - 1997: IBM Deep Blue defeats Kasparov
 - ... there is still debate about whether computers are really better
- Checkers:
 - Computer world champion since 1994
 - ... there was still debate about whether computers are really better...
 - until 2007: checkers solved optimally by computer
- Go:
 - Computers still not very good
 - Branching factor really high
 - Some recent progress: Alpha Go
- Poker:
 - Competitive with top humans in some 2-player games
 - 3+ player case much less well-understood

Is this of any value to society?

- Some of the techniques developed for games have found applications in other domains
 - Especially “adversarial” settings
- Real-world strategic situations are usually not two-player, perfect-information, zero-sum, ...
- But game theory does not need any of those
- Example application: security scheduling at airports

Summary

- Games are fun to work on!
- They illustrate several important points about AI
- Perfection is unattainable → must approximate
- Good idea to think about what to think about

作业

- 5.9