Outline

Optimal decisions

MiniMax Algorithm

α-β pruning

Imperfect, real-time decisions

Games vs. search problems

"Unpredictable" opponent → specifying a move for every possible opponent reply

For Chess, average branching 35; and search 50 moves by each player

- Time limits (35^100) → unlikely to find a goal, must approximate

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.

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

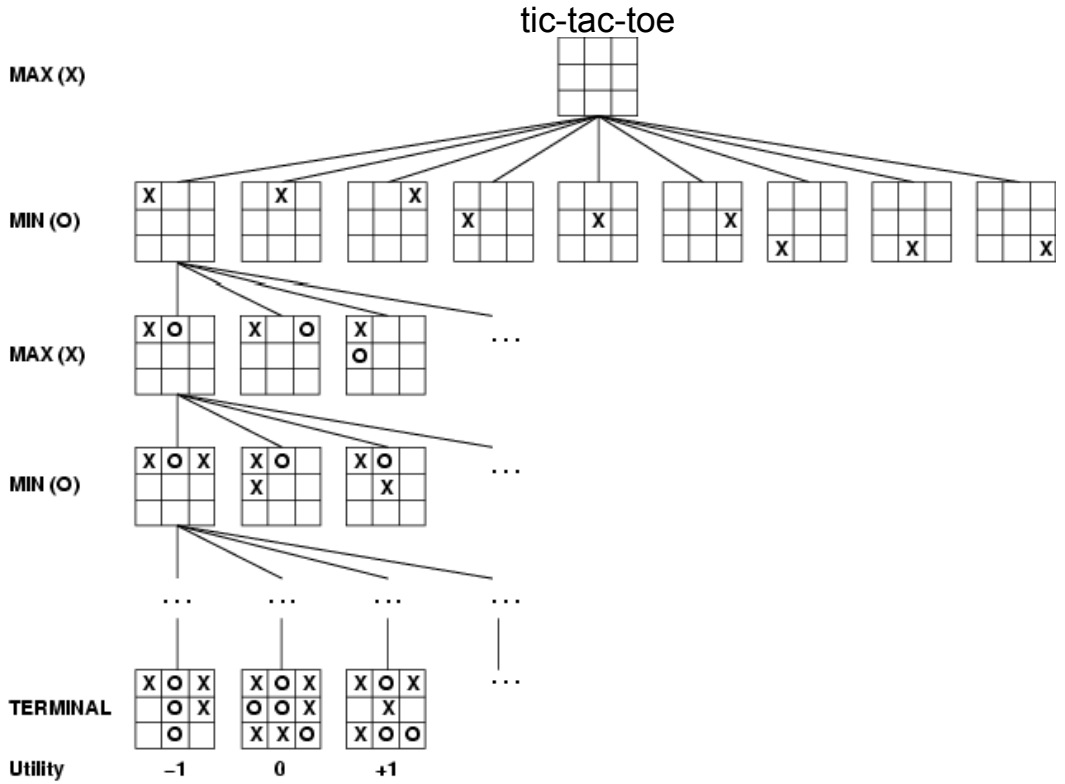
- nodes at odd-numbered depths correspond to positions in which it is *MIN*’s move next

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)

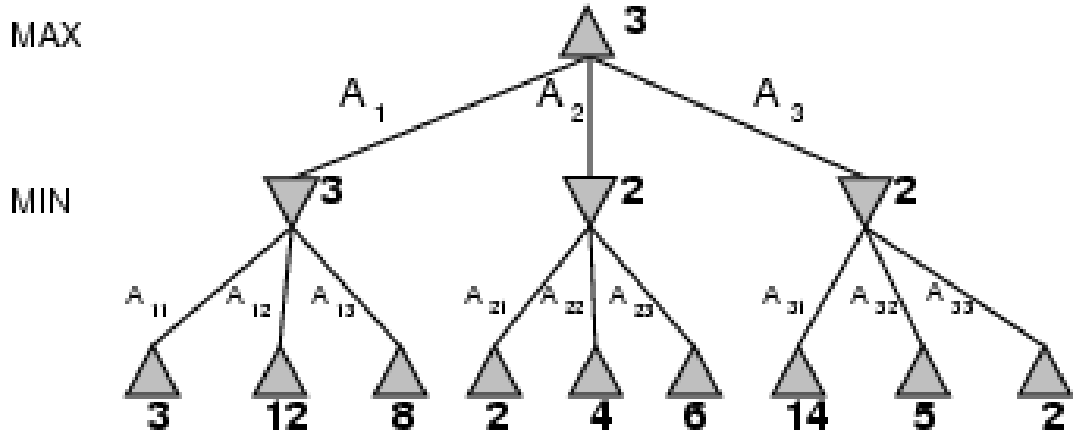


Minimax

The Perfect play for deterministic games

Idea: choose a move to a position with highest minimax value
= best achievable payoff against best play

E.g., 2-ply game:



Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(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 ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Properties of minimax

Complete? Yes (if the tree is finite)

Optimal? Yes (against an optimal opponent)

Time complexity? $O(b^m)$ (b-legal moves; m- max tree depth)

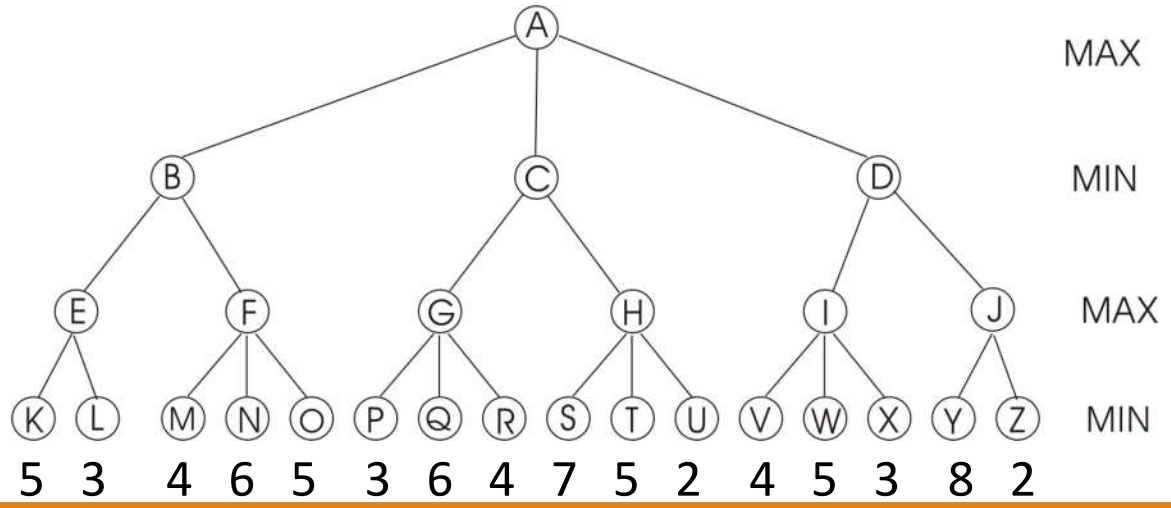
Space complexity? O(bm) (depth-first exploration)

- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

An Example

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?



Example : Tic-Tac-Toe (1)

MAX marks **crosses** *MIN* marks **circles**

it is *MAX*'s turn to play first.

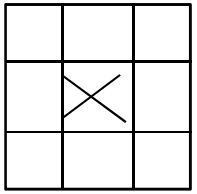
- With a depth bound of 2, conduct a breadth-first search
- evaluation function $e(p)$ of a position p
  - If p is not a winning for either player,

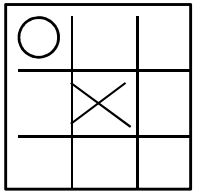
    $e(p)$ = (no. of complete rows, columns, or diagonals that are still open for *MAX*) - (no. of complete rows, columns, or diagonals that are still open for *MIN*)
  - If p is a win of *MAX*, $e(p) = \infty$
  - If p is a win of *MIN*, $e(p) = -\infty$

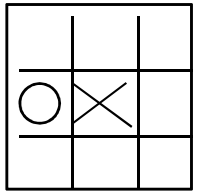
Example : Tic-Tac-Toe (2)

- evaluation function $e(p)$ of a position p
  - If p is not winning for either player,

    $e(p)$ = (no. of complete rows, columns, or diagonals that are still open for *MAX*) - (no. of complete rows, columns, or diagonals that are still open for *MIN*)





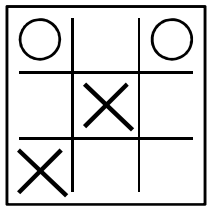


$e(p)=5-4=1$ $e(p)=6-4=2$

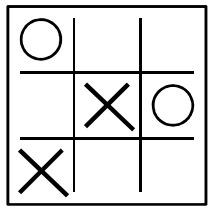
Example : Tic-Tac-Toe (3)

- evaluation function $e(p)$ of a position p
  - If p is not winning for either player,

    $e(p)$ = (no. of complete rows, columns, or diagonals that are still open for *MAX*) - (no. of complete rows, columns, or diagonals that are still open for *MIN*)

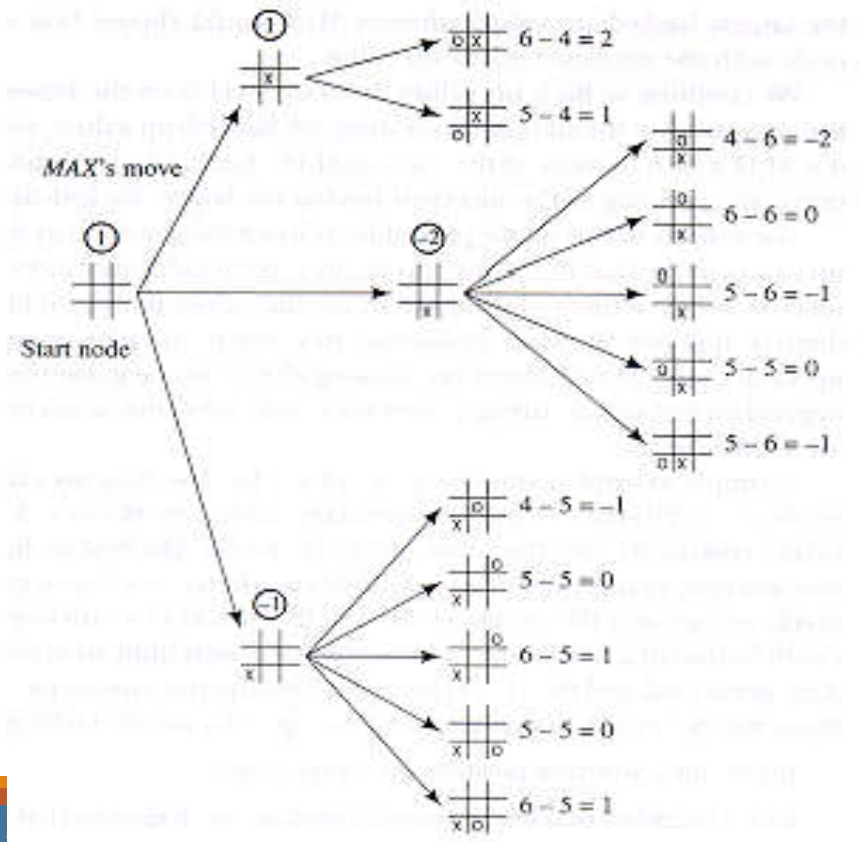


$e(p)=$

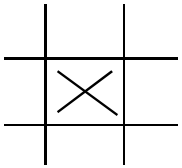


$e(p)=$

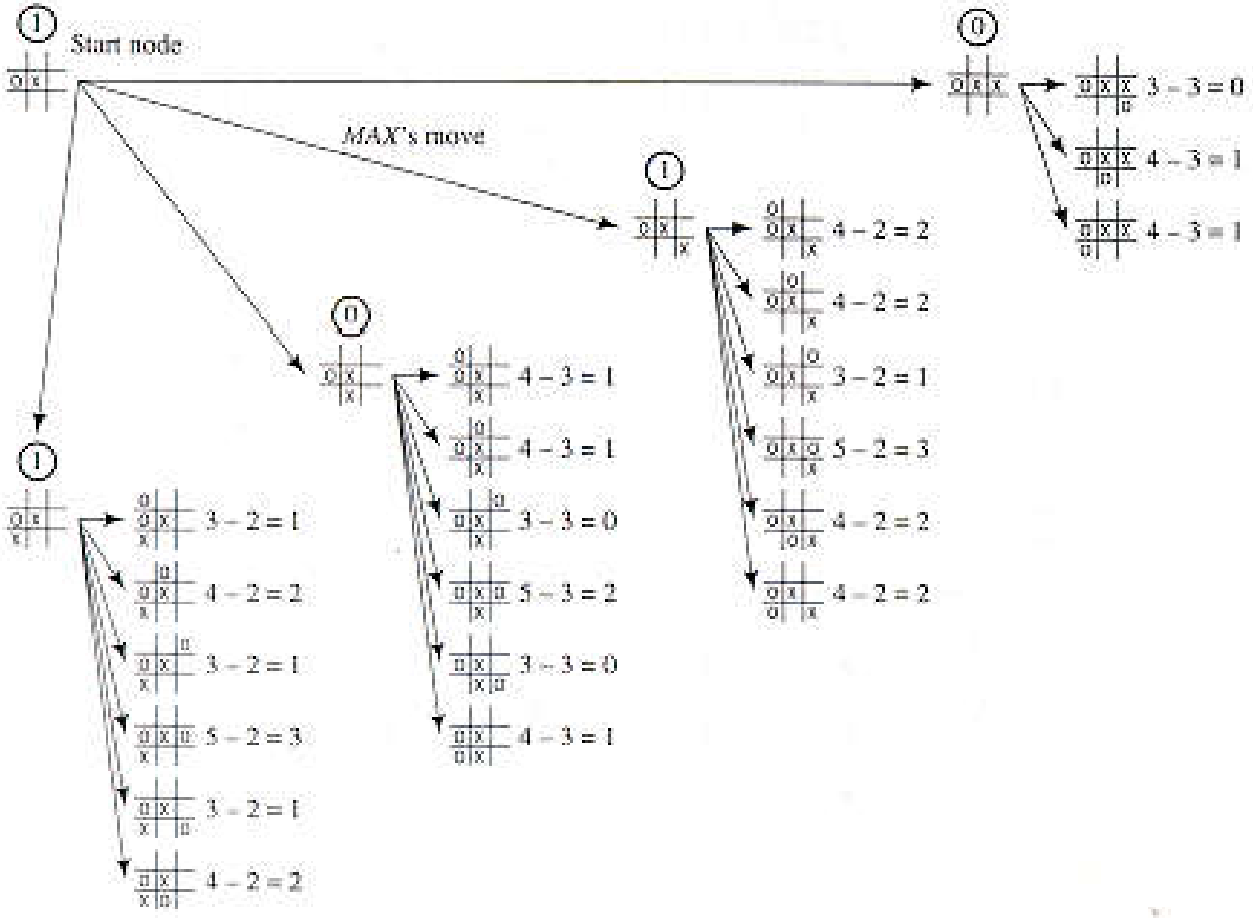
Example : Tic-Tac-Toe (4)



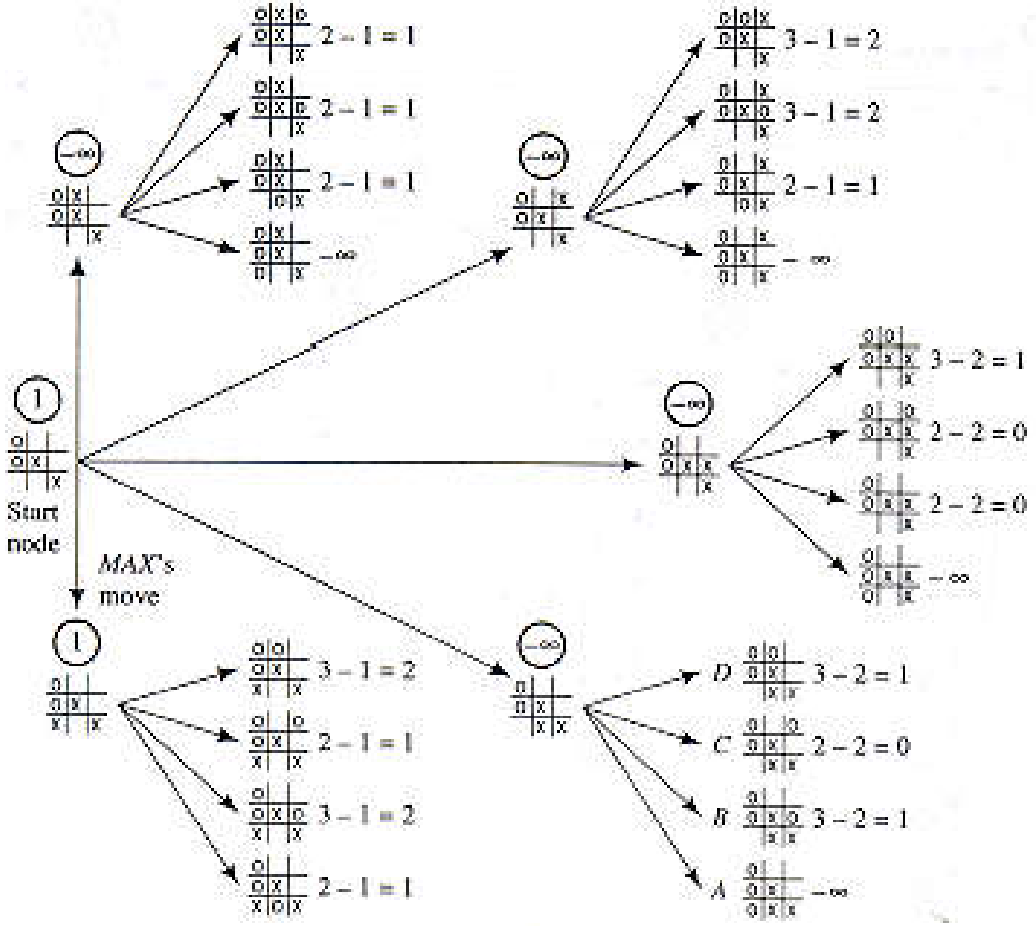
First move



Example : Tic-Tac-Toe (5)



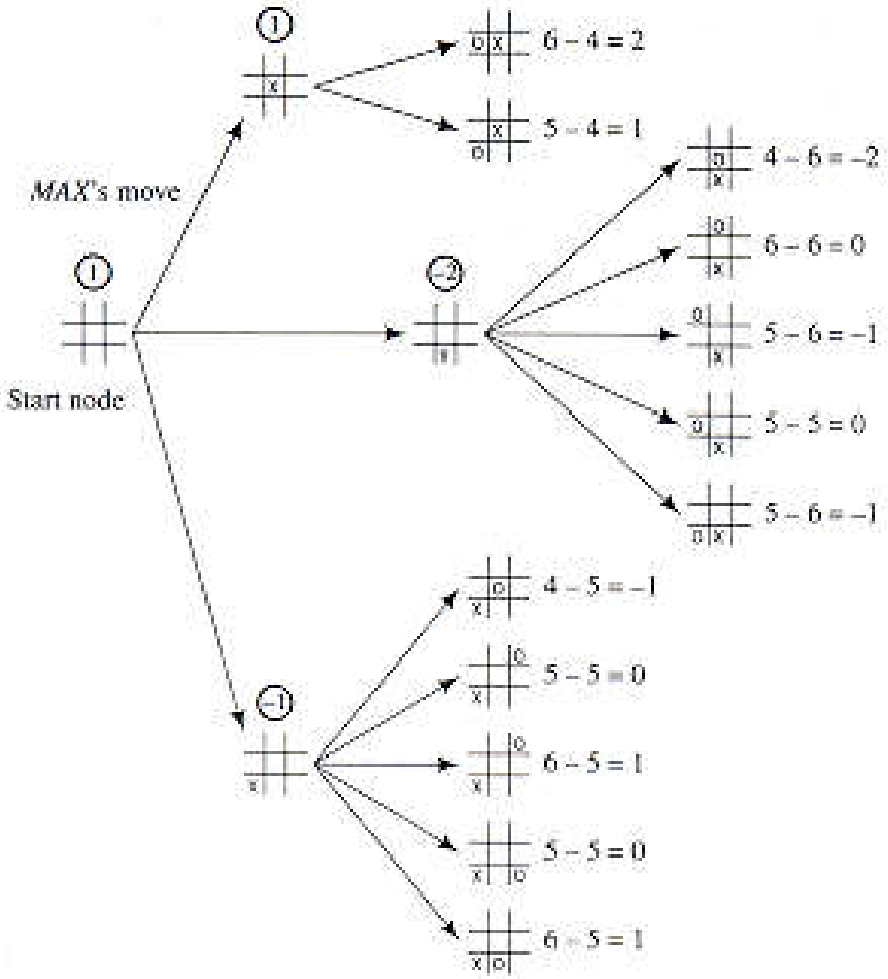
Example : Tic-Tac-Toe (6)



Question?

How to improve search efficiency?

It is possible to cut off some unnecessary subtrees?



Alpha-Beta Pruning

It is possible to compute the correct minimax decision without looking at every node in the game tree.

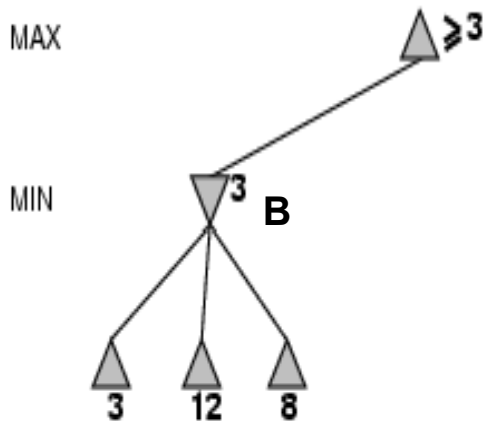
Alpha-beta pruning allows to eliminate large parts of the tree from consideration, without influencing the final decision.

Alpha-beta pruning gets its name from two parameters.

- They describe bounds on the values that appear anywhere along the path under consideration:
  - α = the value of the best (i.e., highest value) choice found so far along the path for MAX
  - β = the value of the best (i.e., lowest value) choice found so far along the path for MIN

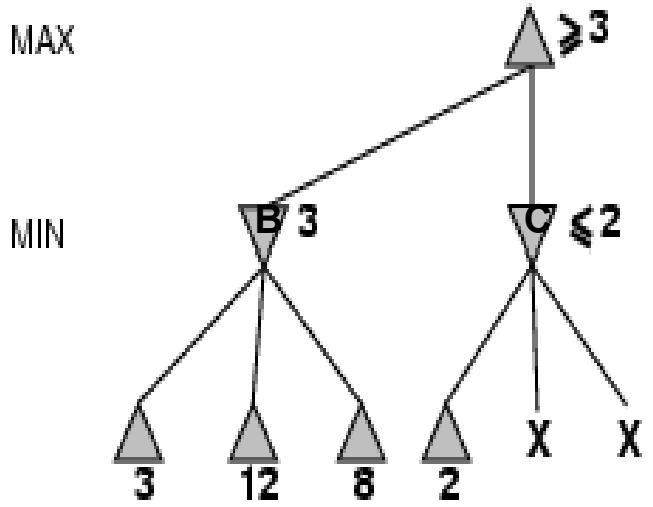
Alpha-Beta Pruning

- The leaves below *B* have the values 3, 12 and 8.
- The value of *B* is exactly 3.
- It can be inferred that the value at the root is *at least* 3, because MAX has a choice worth 3.

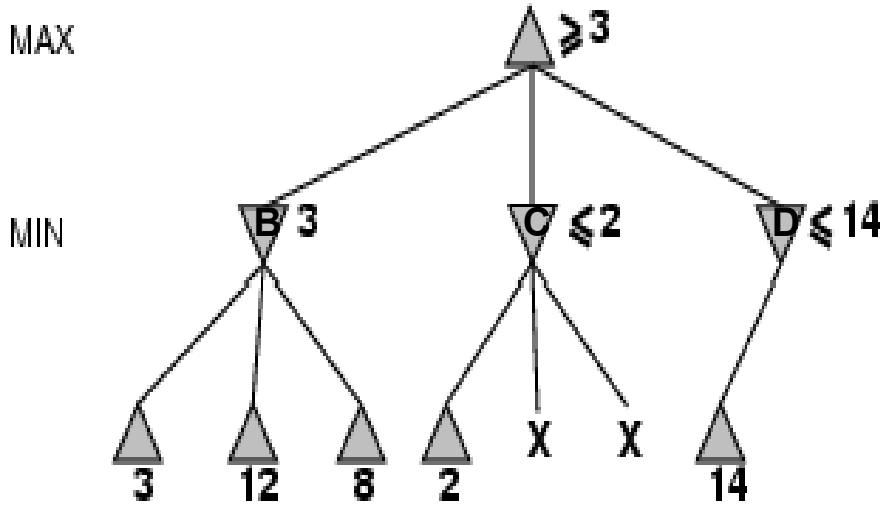


Alpha-Beta Pruning

- *C*, which is a MIN node, has a value of *at most* 2.
- But *B* is worth 3, so MAX would never choose *C*.
- Therefore, there is no point in looking at the other successors of *C*.



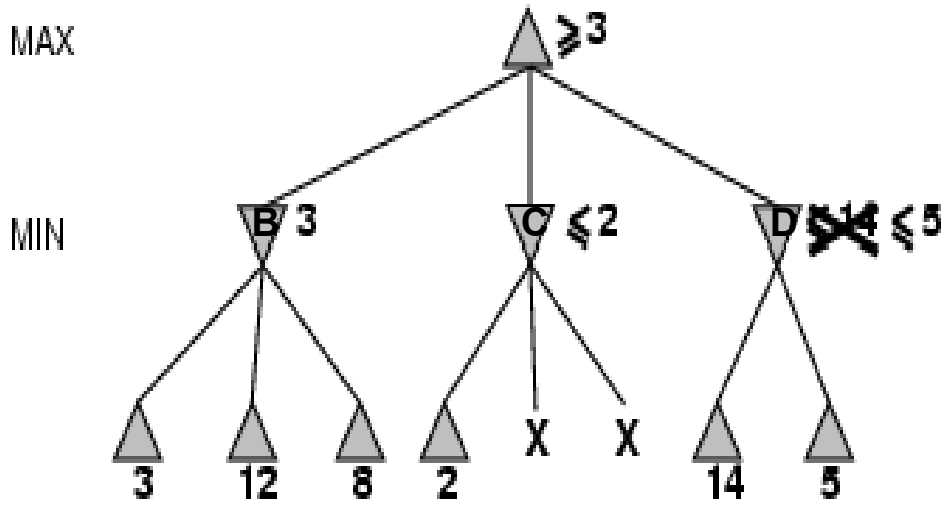
Alpha-Beta Pruning



***D*, which is a MIN node, is worth *at most* 14.**

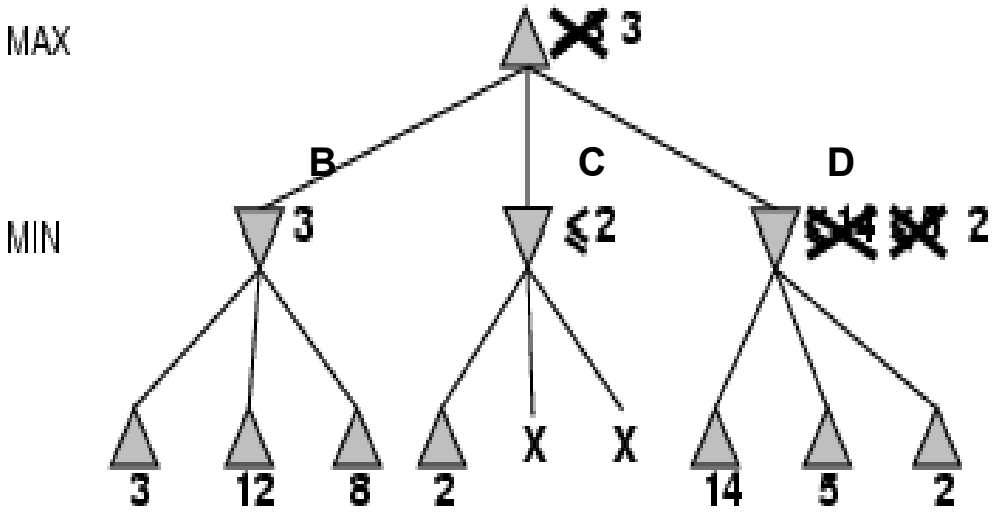
This is still higher than MAX's best alternative (i.e., 3), so *D*'s other successors are explored.

Alpha-Beta Pruning



The second successor of *D* is worth 5, so the exploration continues.

Alpha-Beta Pruning



- **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**

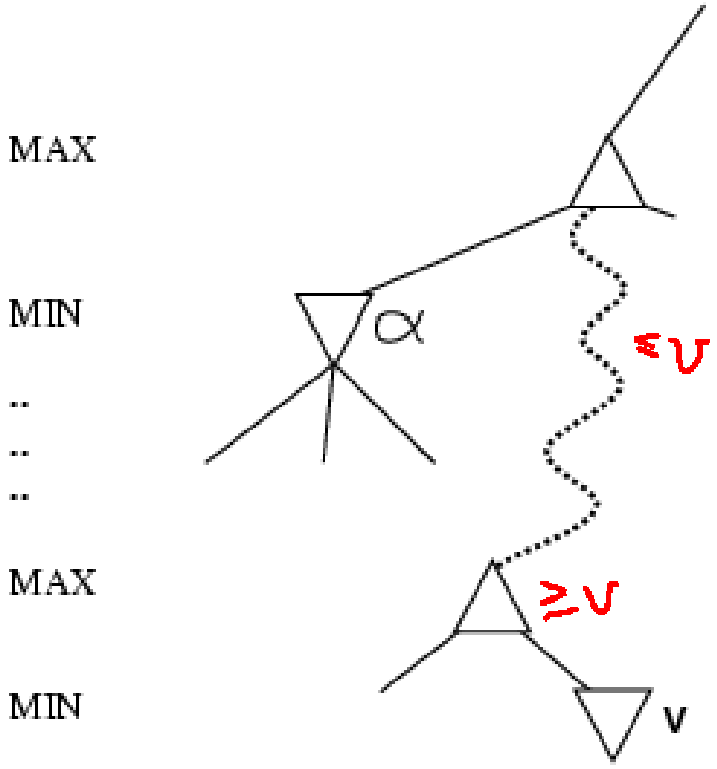
Properties of α-β

- 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
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

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*



The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state, −∞, +∞)
    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 ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

The α-β algorithm

```
function MIN-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 ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

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 (perhaps add quiescence search)

- evaluation function
  = estimated desirability of position

Evaluation functions

For chess, typically linear weighted sum of features

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

e.g., $w_1 = 9$ with

$f_1(s)$ = (number of white queens) – (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)

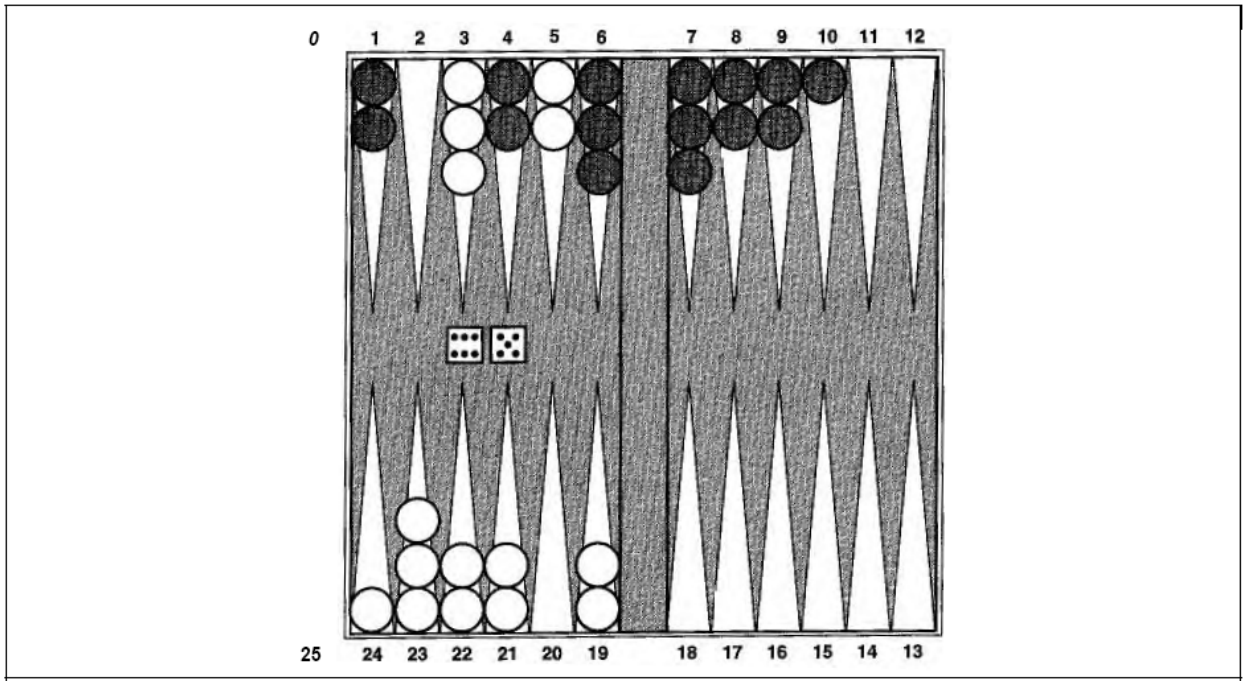
Does it work in practice?

$b^m = 10^6$, b=35 → m=4

4-ply lookahead is a hopeless chess player!

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

Game Include an Element of Chance



Backgammon

Game Include an Element of Chance

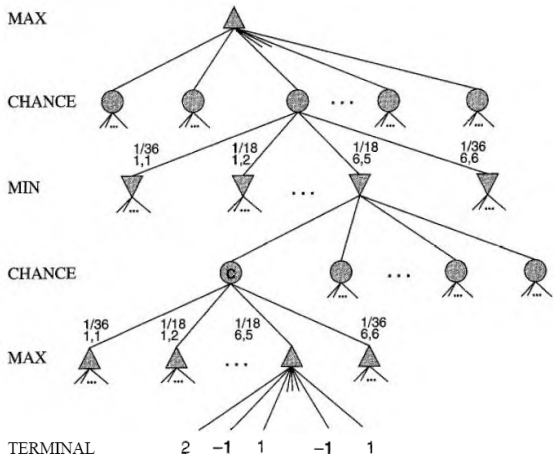


Figure 6.11 Schematic game tree for a backgammon position.

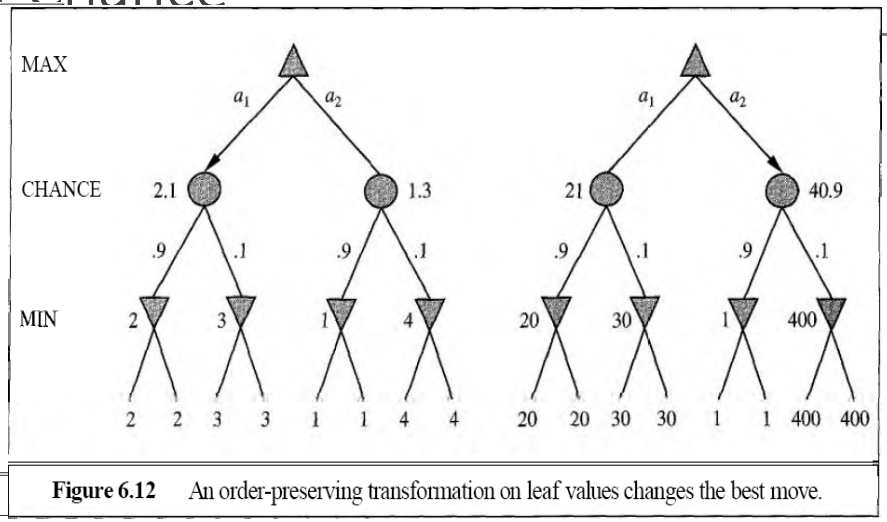
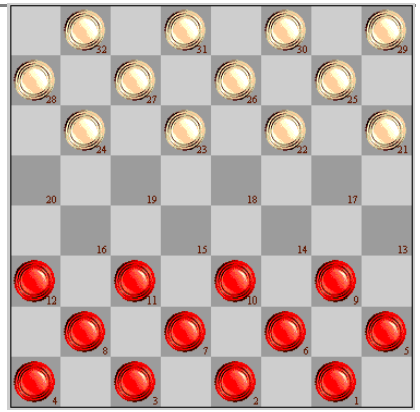


Figure 6.12 An order-preserving transformation on leaf values changes the best move.

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

Deterministic Games in Practice

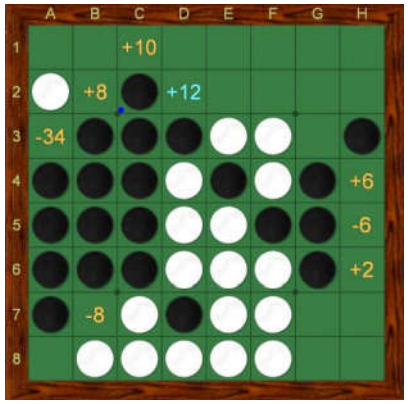
Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions. Checkers is now solved.



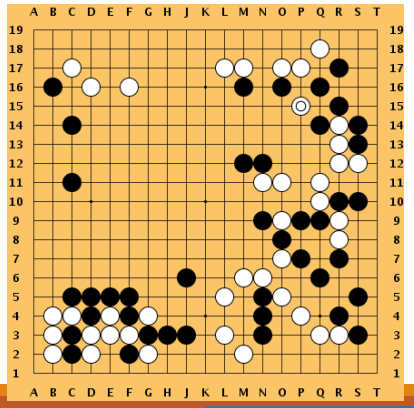
Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 plies. Current programs are even better, if less historic.

Deterministic Games in Practice

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, most programs use pattern knowledge bases to suggest plausible moves and aggressive pruning.



Summary

Have the games start with defining by the initial state (how the board is set up), the legal actions in each state, the result of each action, a terminal test (which says when the game is over), and a utility function that applies to terminal states.

In two-player zero-sum games which have perfect information, the minimax algorithm can select optimal moves by a depth-first enumeration of the game tree.

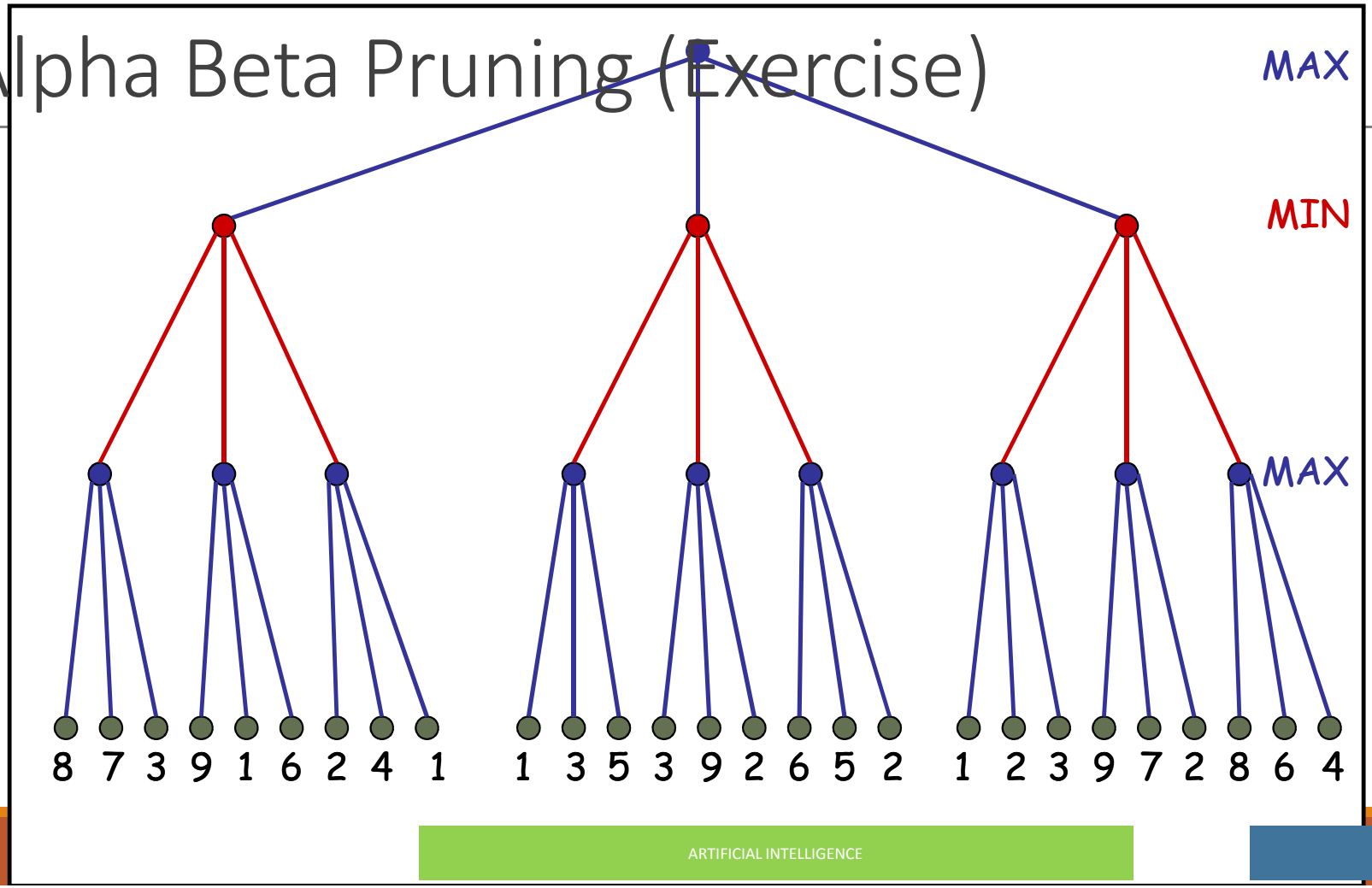
The alpha-beta search algorithm computes the same as the minimax but achieves much greater efficiency by eliminating subtrees that are probably irrelevant.

Summary

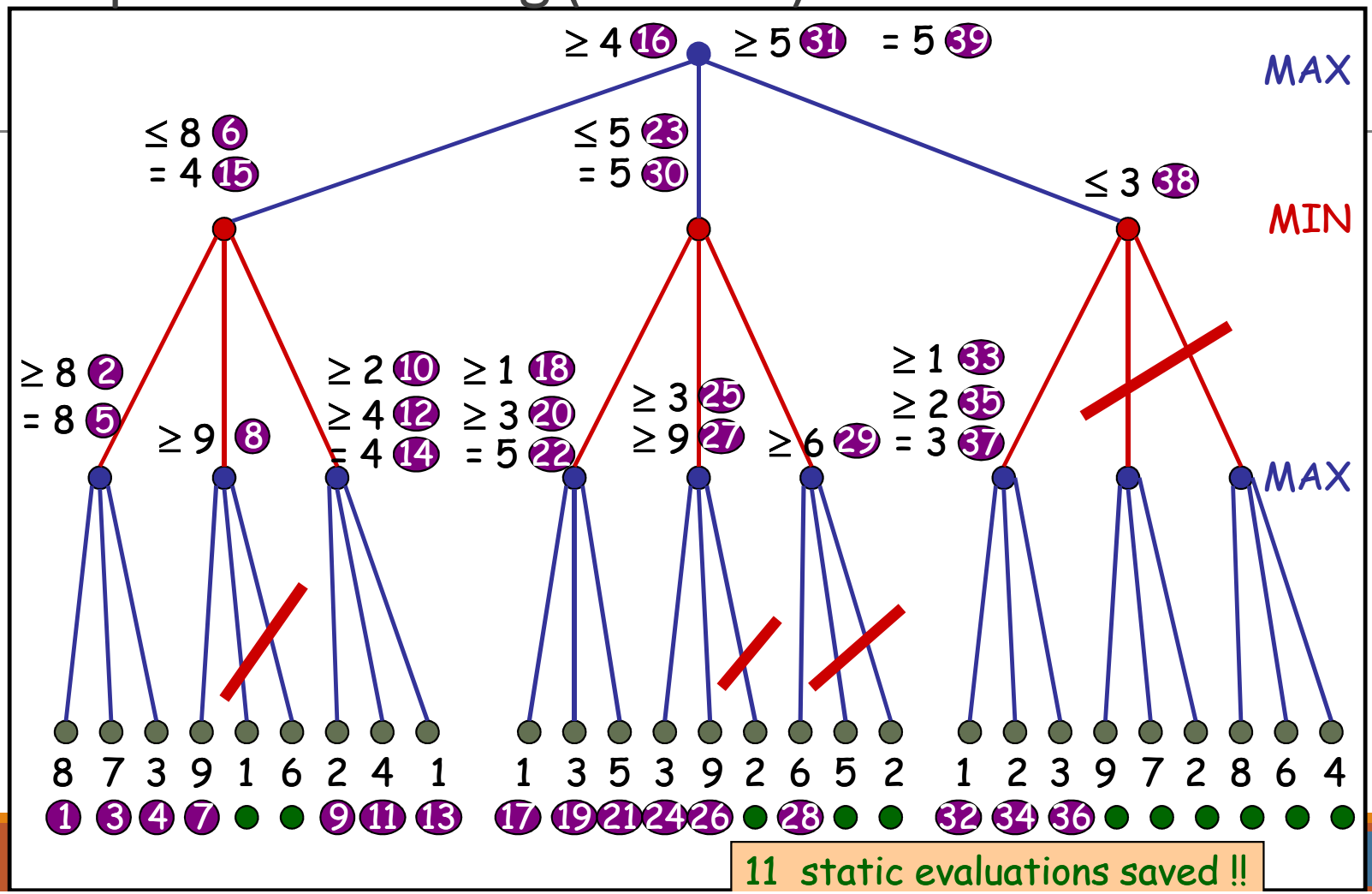
Alpha-betat is not feasible to consider the whole game tree (even with the alpha–beta) need to cut the search off at some point and apply a heuristic evaluation function that estimates the utility of a state.

Games of chance can be handled by an extension to the minimax algorithm that evaluates a chance node by taking the average utility of all its children, weighted by the probability of each child.

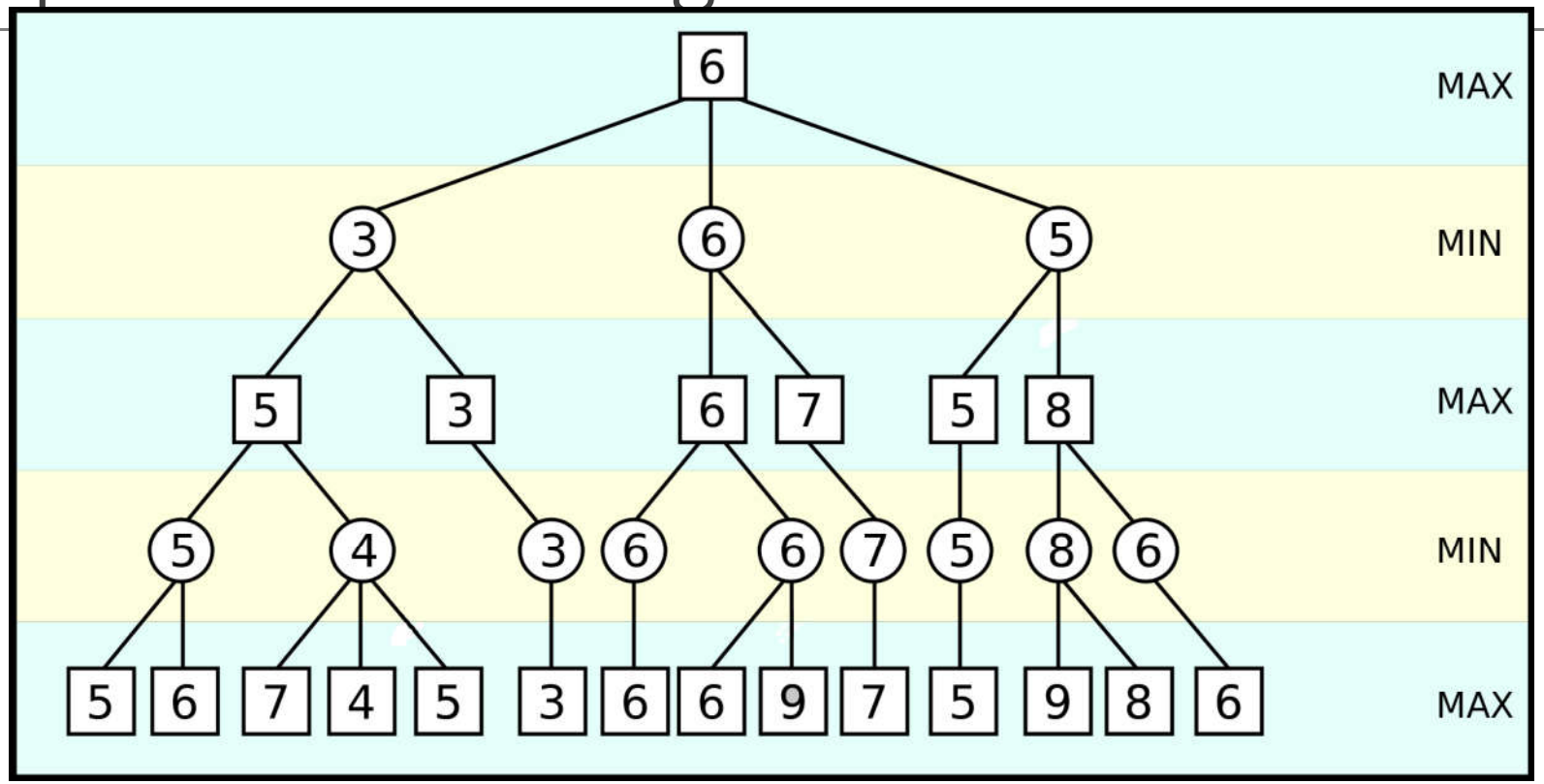
Alpha Beta Pruning (Exercise)



Alpha Beta Pruning (Exercise)



Alpha Beta Pruning



Alpha Beta Pruning (Solution)