BLG435E
Artificial Intelligence

Lecture 5: Adversarial Search

AI Games

- Agents' goals are in conflict
- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over

- At the end of the game
  - points are awarded to the winner
  - penalties are given to the loser

- Zero-sum games

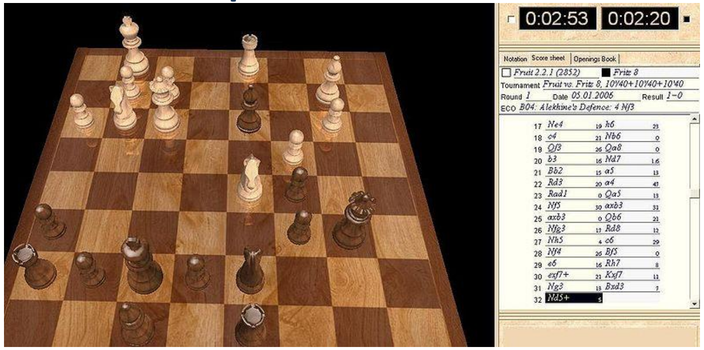
Types of games

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

What is this?

Deep Blue

- Against Garry Kasparov
  - 1996, in 1997 – won
  - Massively parallel, P2SC-based system with 30-nodes
    - each node containing a 120 MHz P2SC microprocessor
    - Written in C and ran under the AIX OP.
    - Capable of evaluating 200 million positions per second
    - search to a depth of 14 moves, to a maximum of twenty or even more moves in some situations
- Komodo is the last champion
  - International Computer Chess Tournament

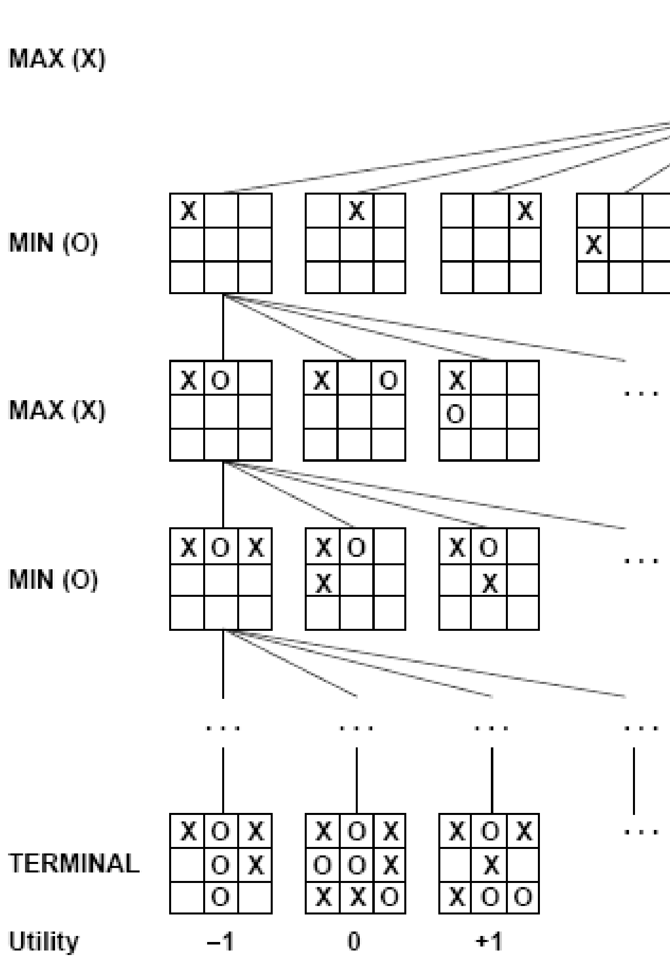


Game formulation

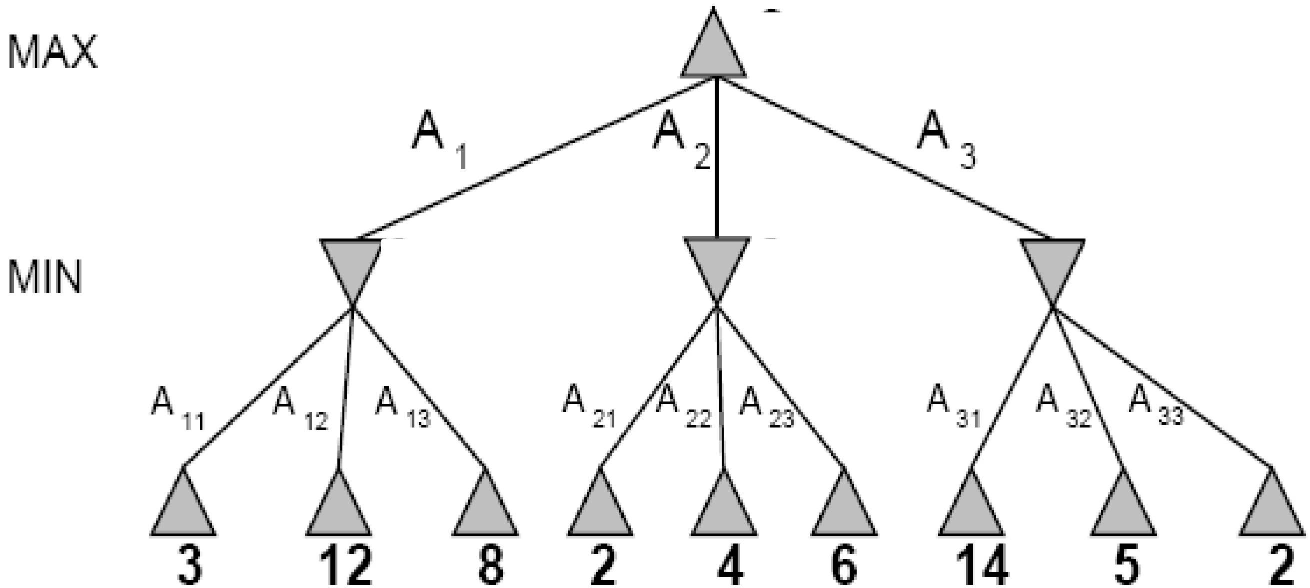
- A game is formally defined
  - initial state
  - successor function
  - terminal test (terminal state)
  - utility function (objective, payoff)

- Game tree: the initial state and the legal moves
- ply: the depth of the search tree (ply of lookahead)

Tic-Tac-Toe



Optimal Strategies

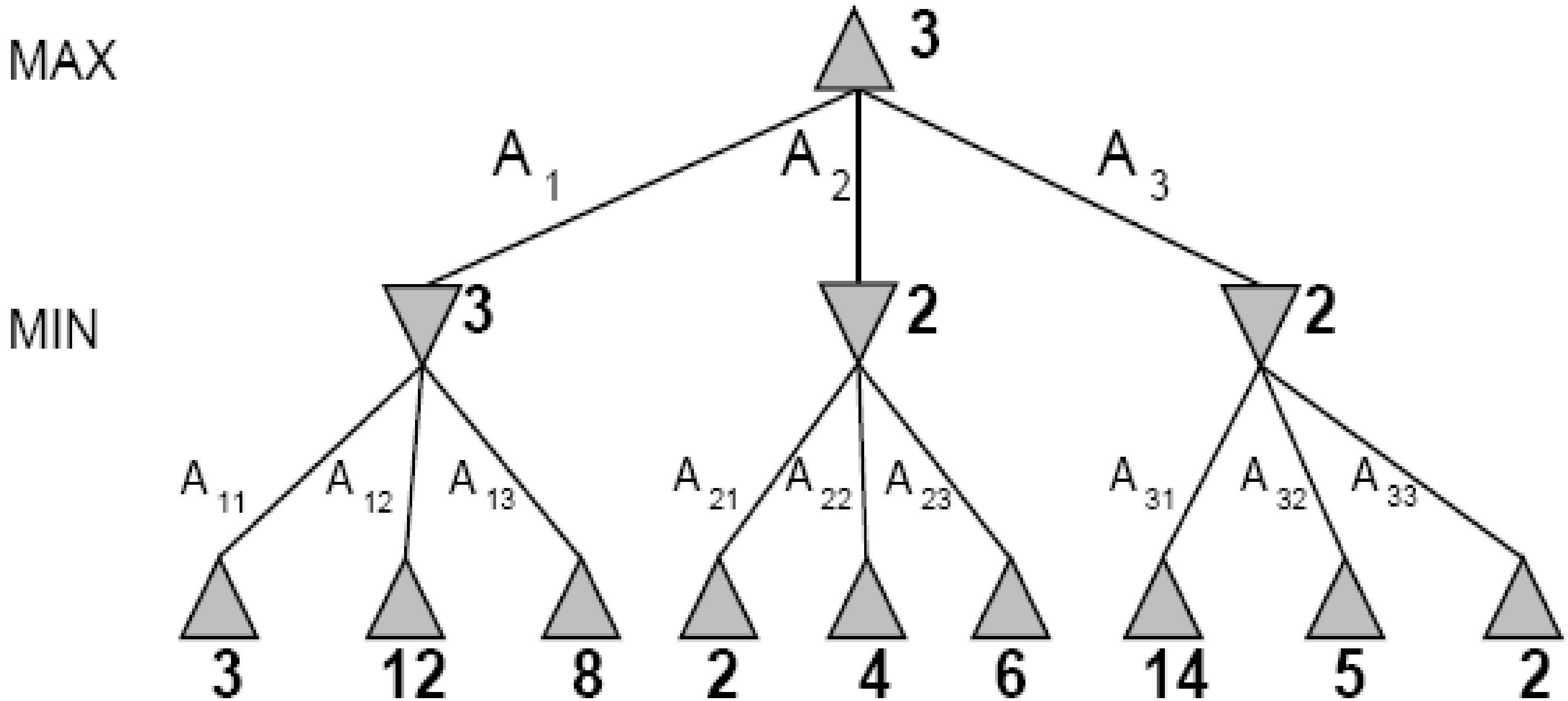


Minimax

- Optimal strategy -> minimax value of each node
  - MINIMAX-VALUE(n), utility (for MAX) of being in the corresponding state
  - Assuming both players play optimally to the end
  - Best achievable payoff against best play
  - MAX will prefer to move to a state of maximum value

$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n), & \text{if } n \text{ is a terminal state} \\ \max_s (\text{MINIMAX-VALUE}(s)), & \text{if } n \text{ is a MAX node} \\ \min_s (\text{MINIMAX-VALUE}(s)), & \text{if } n \text{ is a MIN node} \end{cases}$$

Minimax



MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

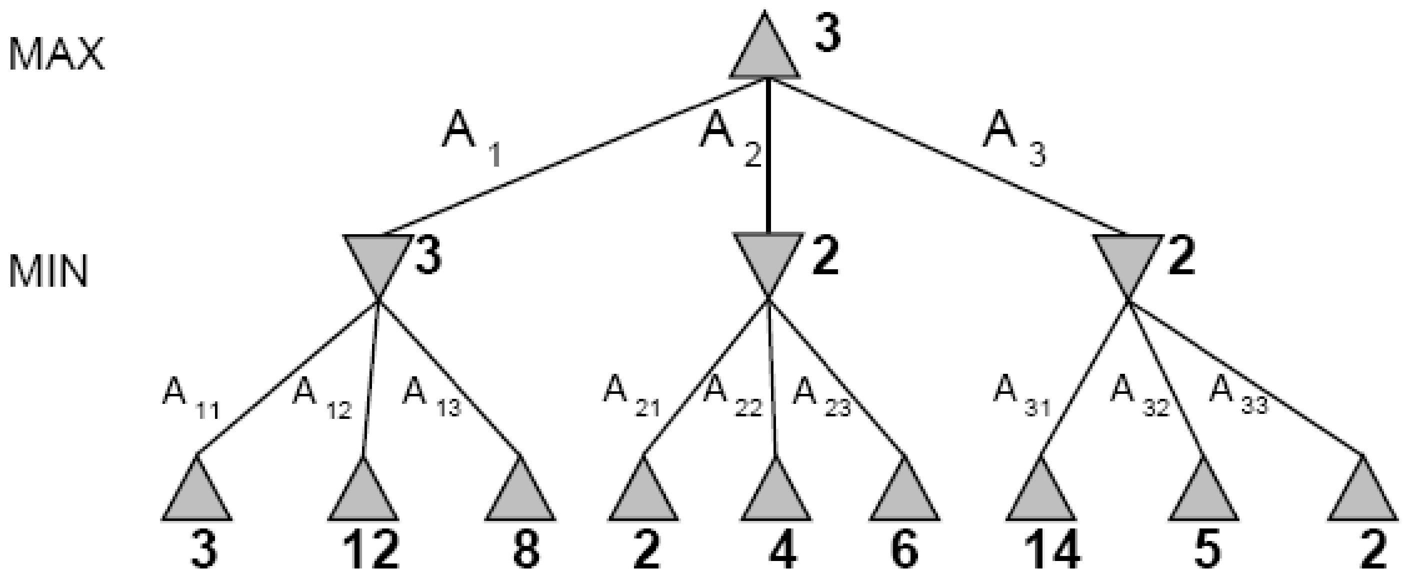
Minimax Algorithm

```
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
```

Minimax Algorithm



- What is the optimal move for MAX?
- What if MIN does not play optimally?

Properties of Minimax

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

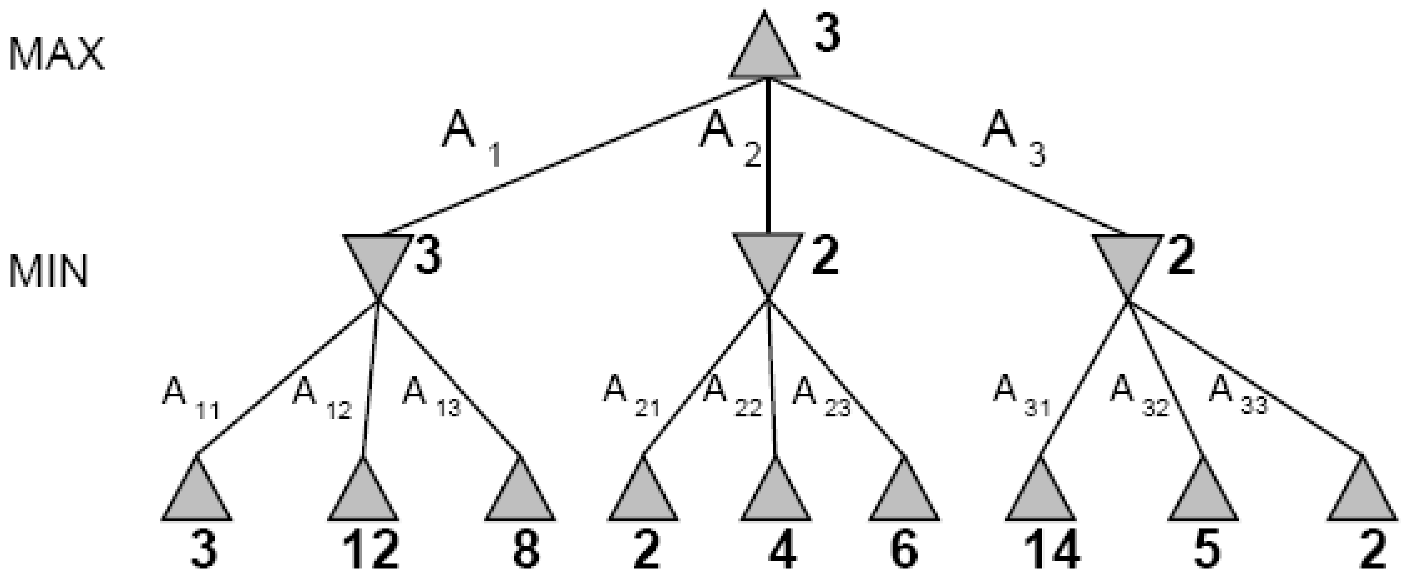
Properties of Minimax

- Complete? Yes, if tree is finite (chess has specific rules for this)
- Optimal? Yes, against an optimal opponent. Otherwise??
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  - exact solution completely infeasible

- Do we need to explore every path?

Do we need to explore every path?



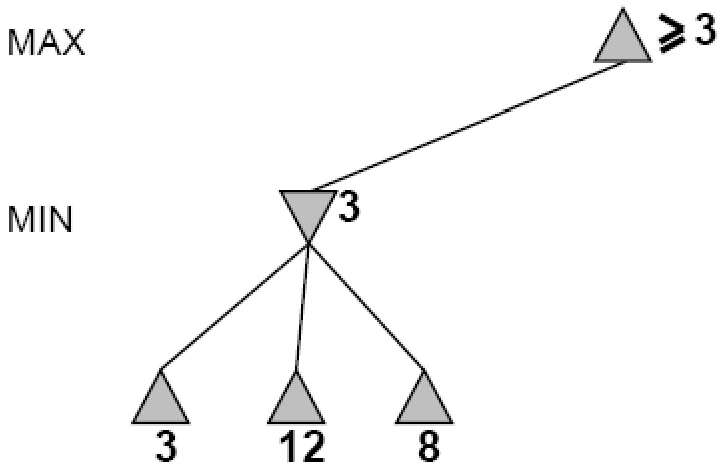
α - β Pruning

- Prunes the game tree
  - Branches that are irrelevant

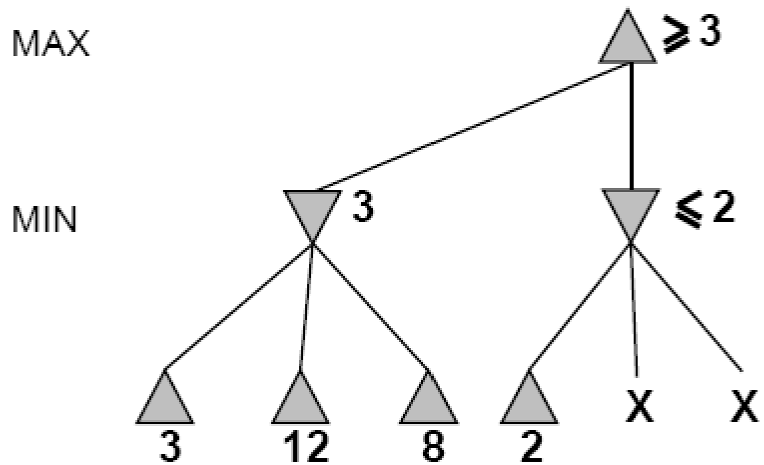
- The same move as minimax would is selected

- Identify the minimax decision without evaluating all of the leaf nodes

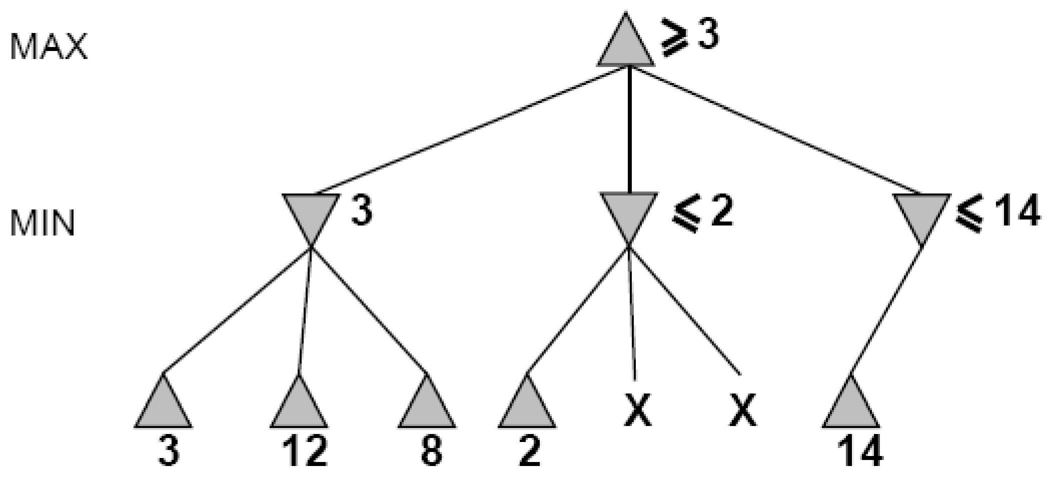
α - β Pruning Example



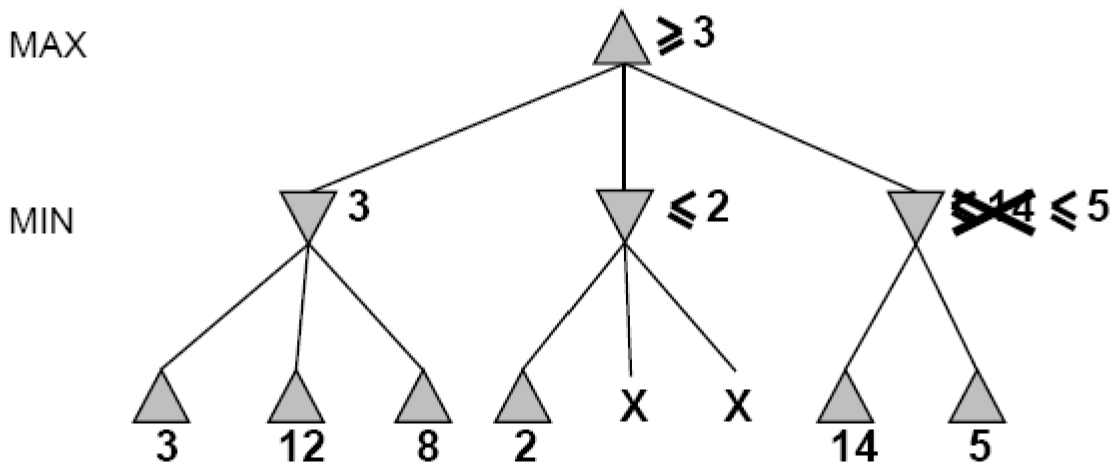
α - β Pruning Example



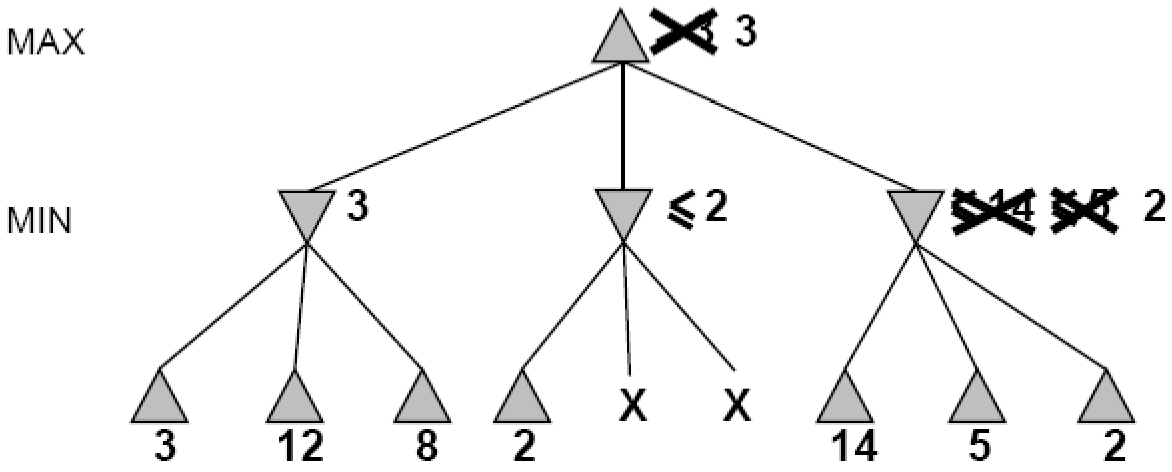
α - β Pruning Example



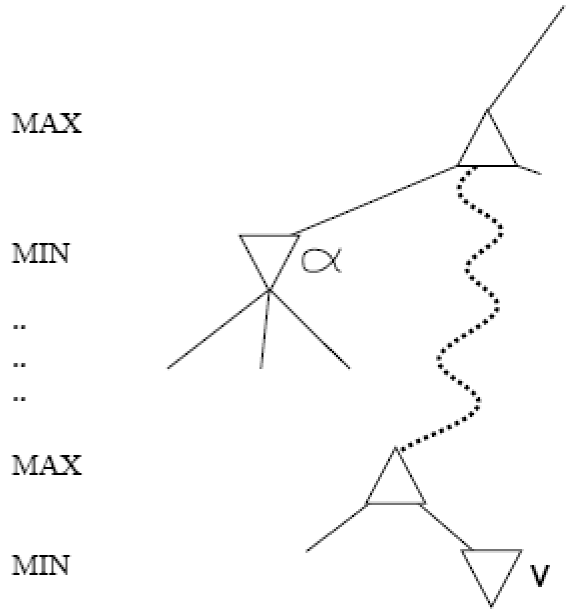
α - β Pruning Example



α - β Pruning Example

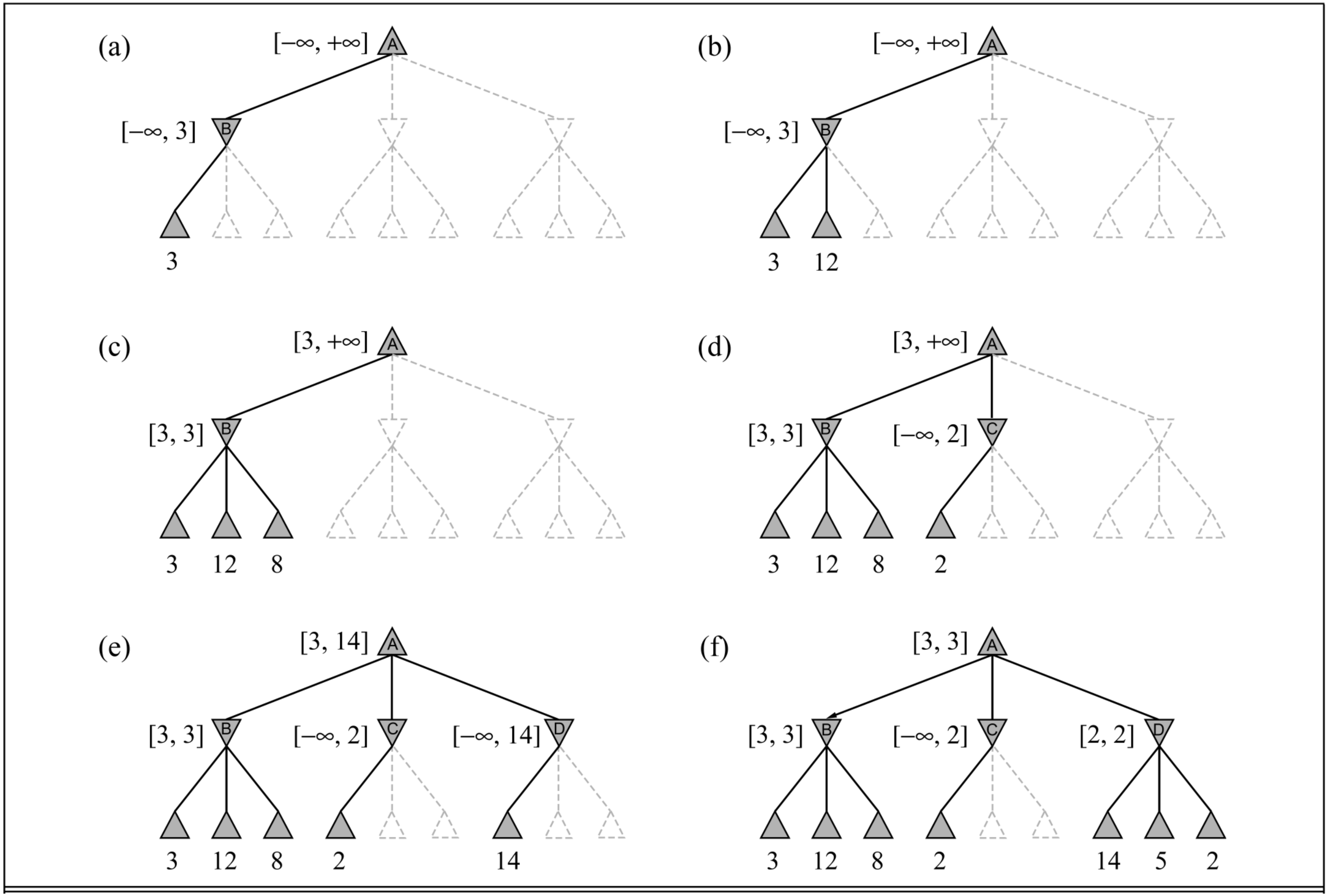


Why it is called α - β Pruning?



- α is the best value (to MAX) found so far off the current path
- If v is worse than α, MAX will avoid it → prune that branch
- Define β similarly for MIN

α - β Pruning Illustration



α - β Pruning Algorithm

```
function ALPHA-BETA-DECISION(state) returns an action
   return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

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

function MIN-VALUE(state, α, β) returns a utility value
   same as MAX-VALUE but with roles of α, β reversed
```

α - β Pruning 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
```

Properties of α - β Algorithm

- Pruning does not affect the final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering", time complexity : $O(b^{m/2})$
  - doubles solvable depth
- Unfortunately, 35^{50} is still impossible!

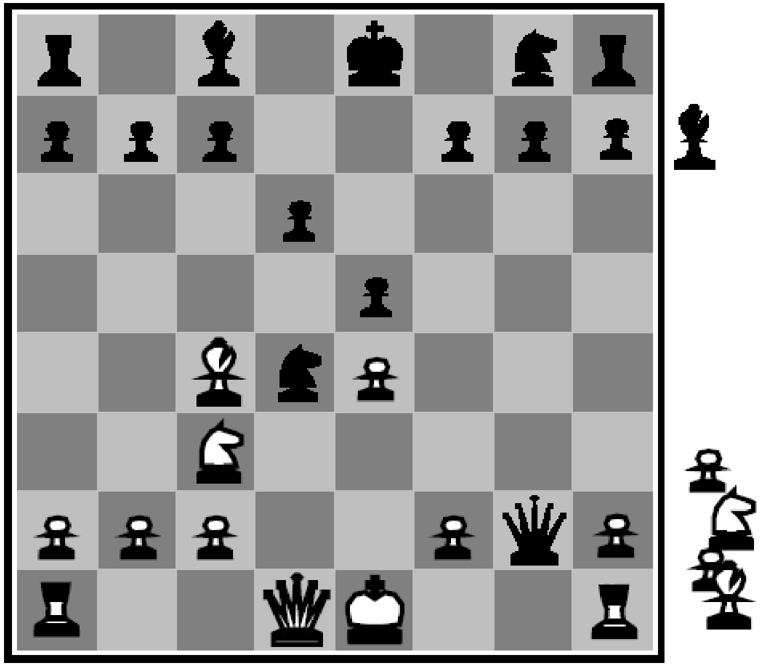
Resource Limits

- Shannon's 1950 paper: Programming a computer for playing chess
  - Use CUTOFF-TEST instead of TERMINAL-TEST
    - depth limit
  - Use EVAL instead of UTILITY
    - evaluation function that estimates desirability of position

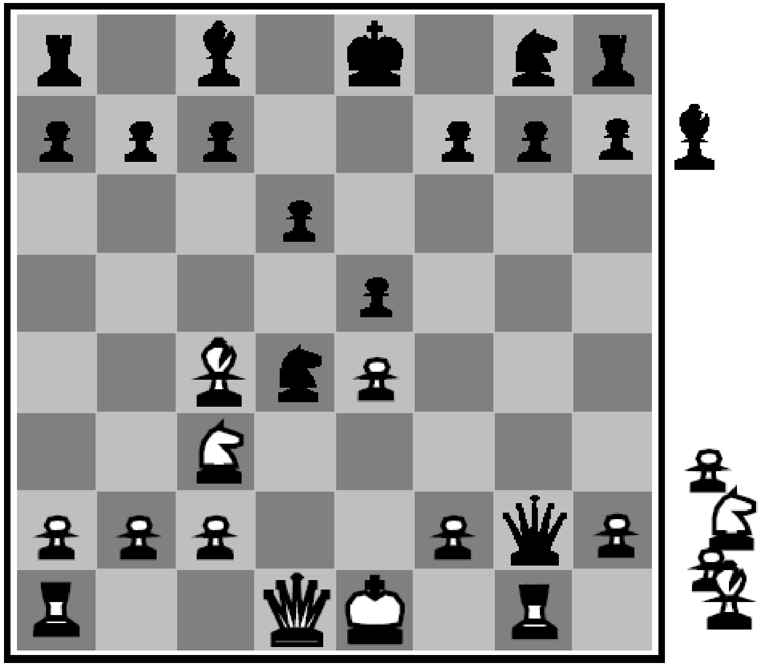
Evaluation Functions

- Estimate of the expected utility of the game
- The performance is dependent on the quality of the evaluation function
- The evaluation function
  - Should give higher scores to better positions
  - Should order the terminal states as the utility function
  - Computation must not take too long
  - For non-terminal states, the evaluation function should be correlated with the actual chances of winning

Evaluation Functions



(a) White to move

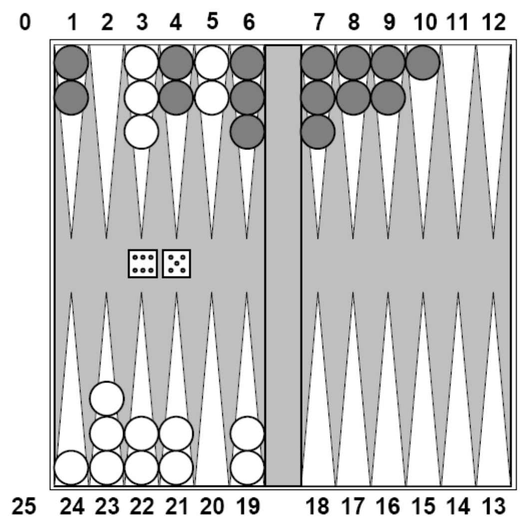


(b) White to move

- $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - $w_1 = 9$ with $f_1(s)$ = (number of white queens) - (number of black queens), etc.

Nondeterministic Games

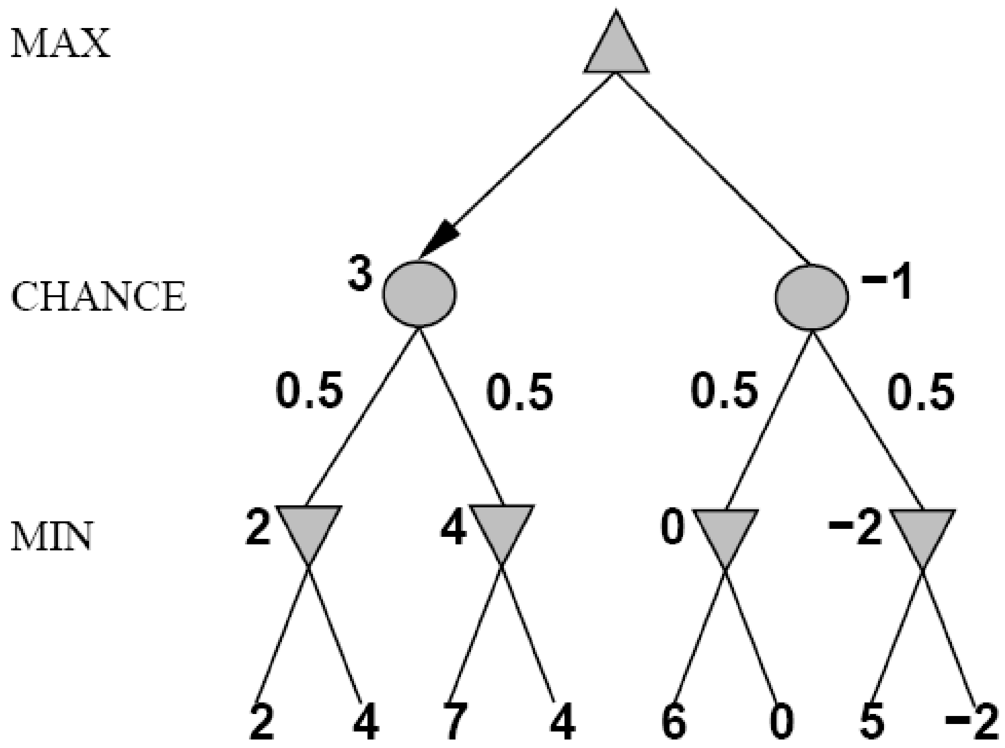
- In nondeterministic games, chance introduced by dice, card-shuffling



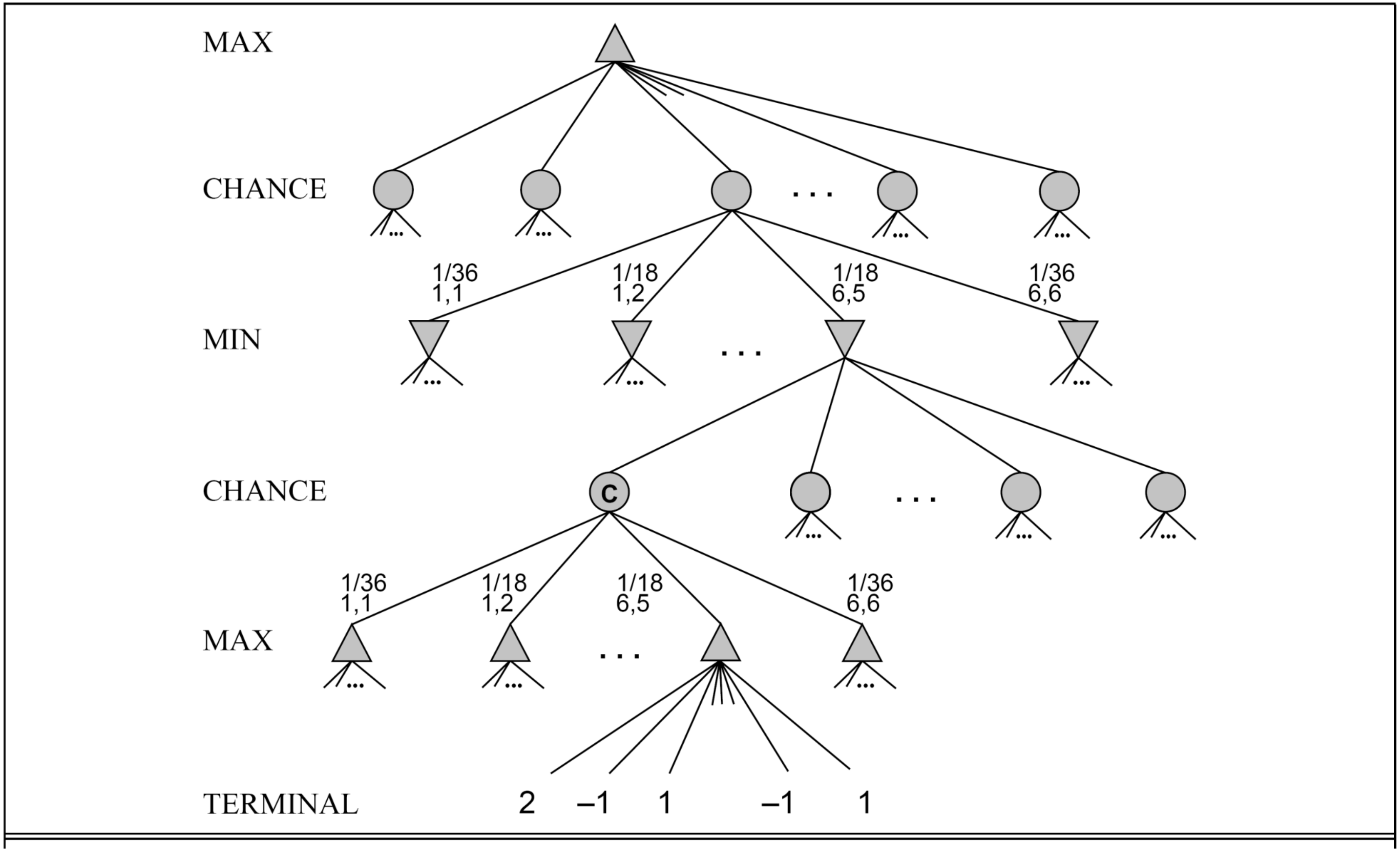
- White does not know what black is going to roll
  - Cannot construct a standard game tree
  - Chance nodes

A simplified example

- With coin flipping:



Backgammon Game Tree



Algorithm for nondeterministic games

- EXPECTIMINIMAX gives perfect play
  - Just like MINIMAX, except we must also handle chance nodes:

- if state is a MAX node then
  - return the highest EXPECTIMINIMAX-Value of SUCCESSORS(state)
- if state is a MIN node then
  - return the lowest EXPECTIMINIMAX-Value of SUCCESSORS(state)
- if state is a chance node then
  - return average of EXPECTIMINIMAX-Value of SUCCESSORS(state)

Practice