Adversarial Search
Lecture 4

Juhan Ernits
Department of Computer Science
Tallinn University of Technology
Juhan.ernits@ttu.ee
2015

Outline

- Optimal decisions
- α-β pruning
- Imperfect, real-time decisions
- Stochastic games
- Partially observable games

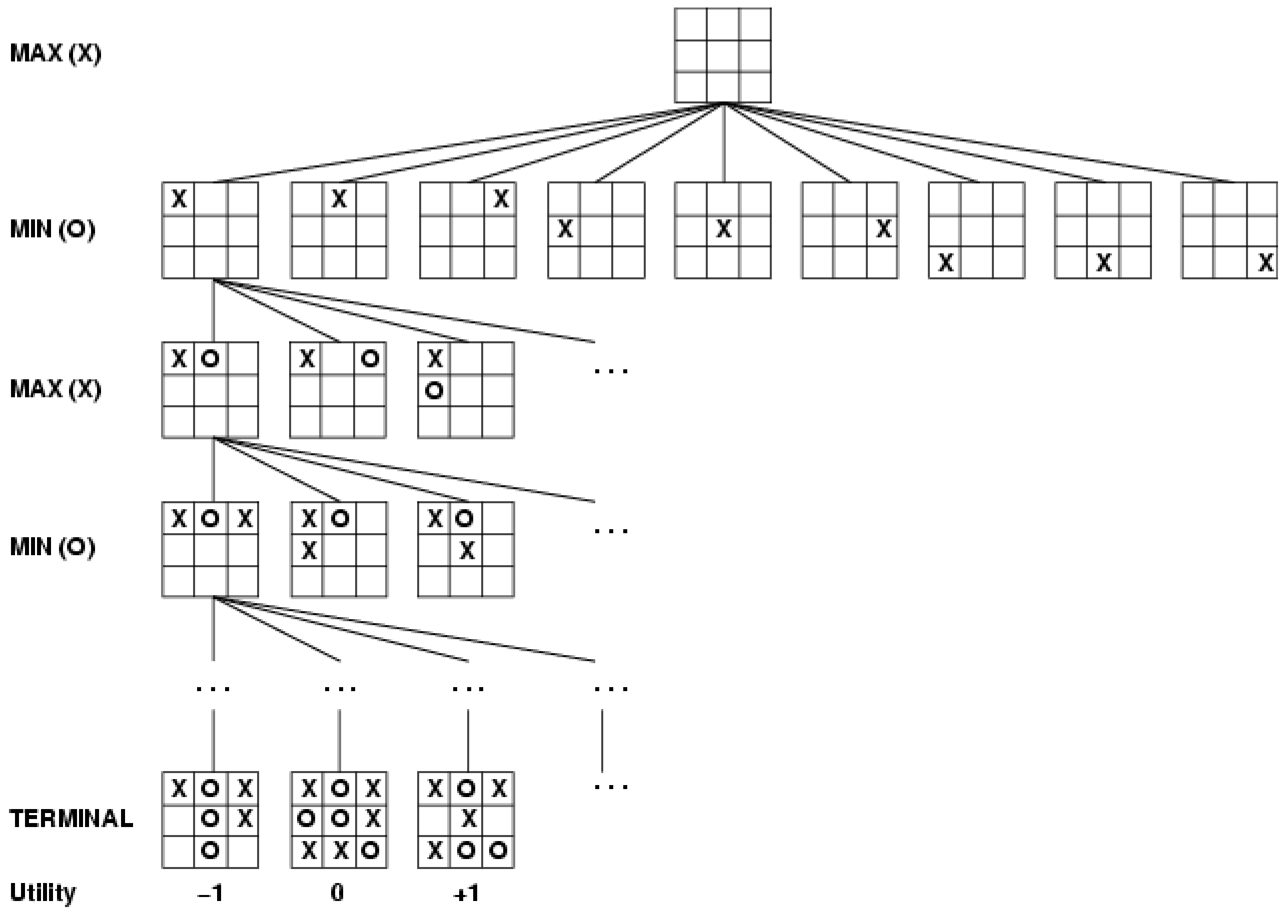
Games

- Mathematical **game theory**, a branch of economics, views any multiagent environment as a game
- In AI most common games are zero-sum games of perfect information (think chess)
  - Deterministic, fully observable environments, two players act alternately, one wins, the other loses
- Some games have elements of **imperfect information**, e.g. Bridge

Games vs. search problems

- "Unpredictable" opponent
  - specifying a move for every possible opponent reply
- Time limits
  - unlikely to find goal, must approximate

Game tree (2-player, deterministic, turns)

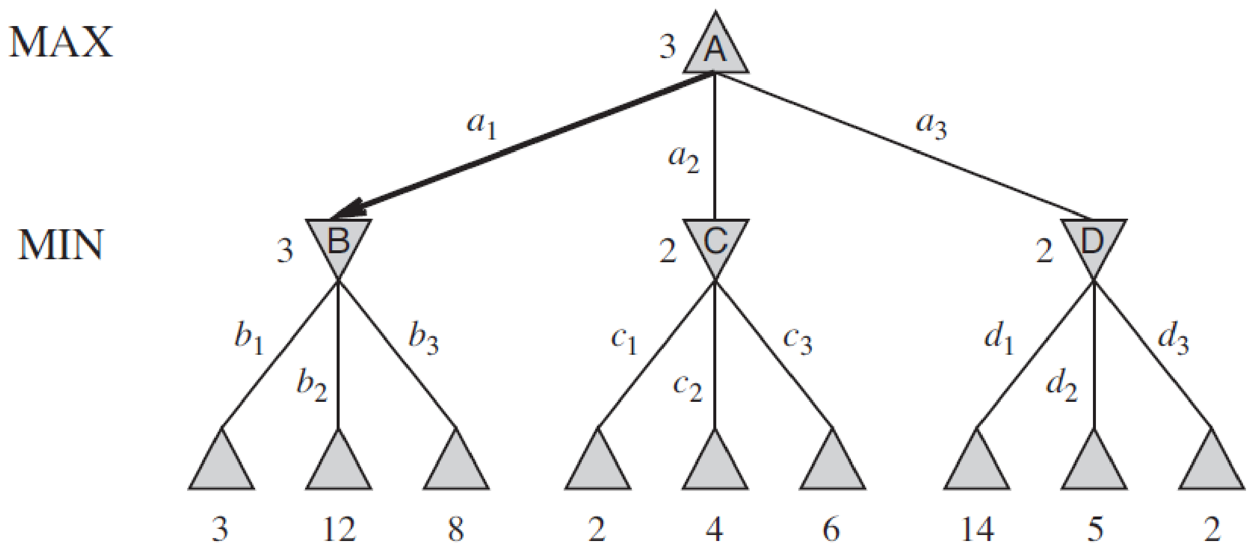


Game as a search problem

- S_0 : initial state
- *Player(s)*: defines which player can move
- *Actions(s)*: set of legal moves in s
- *Result(s,a)*: transition model, defining the result of a move
- *Terminal-Test(s)*: check whether the game is over
- *Utility(s,p)*: utility aka objective aka payoff function defines the final numeric value of a game ending in a terminal state.

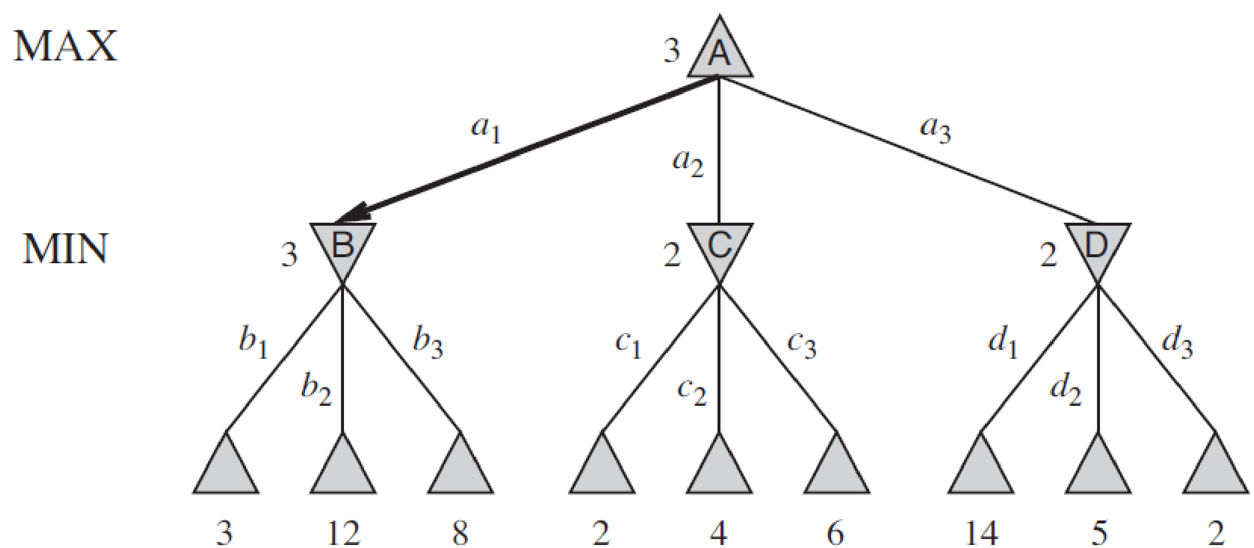
Minimax

- 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

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$



Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a\,\in\,\text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v

Minimax algorithm

```
def minimax_decision(state, game):
    """Given a state in a game, calculate the best move by searching
    forward all the way to the terminal states. [Fig. 5.3]"""

    player = game.to_move(state)

    def max_value(state):
        if game.terminal_test(state):
            return game.utility(state, player)
        v = -infinity
        for a in game.actions(state):
            v = max(v, min_value(game.result(state, a)))
        return v

    def min_value(state):
        if game.terminal_test(state):
            return game.utility(state, player)
        v = infinity
        for a in game.actions(state):
            v = min(v, max_value(game.result(state, a)))
        return v

    # Body of minimax_decision:
    return argmax(game.actions(state),
                  lambda a: min_value(game.result(state, a)))
```

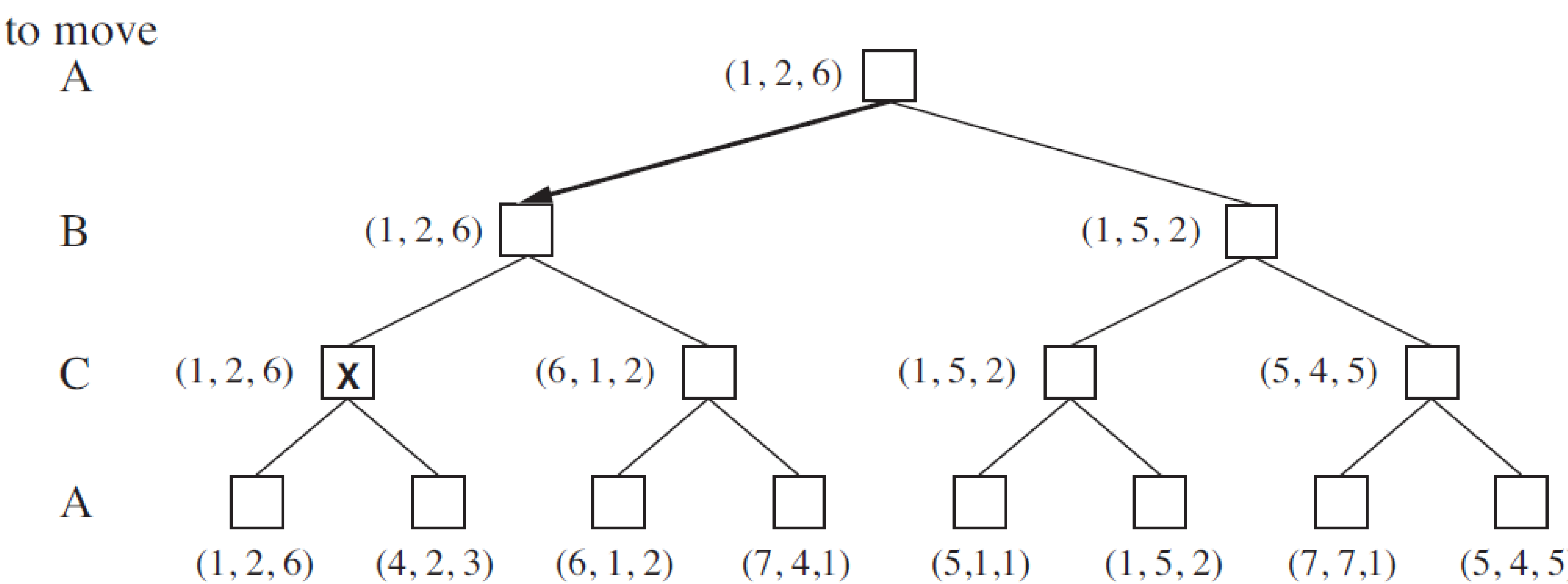
Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  → exact solution is completely infeasible, $35^{100} \sim 10^{154}$ nodes in the search tree or “only” 10^{40} states in search graph

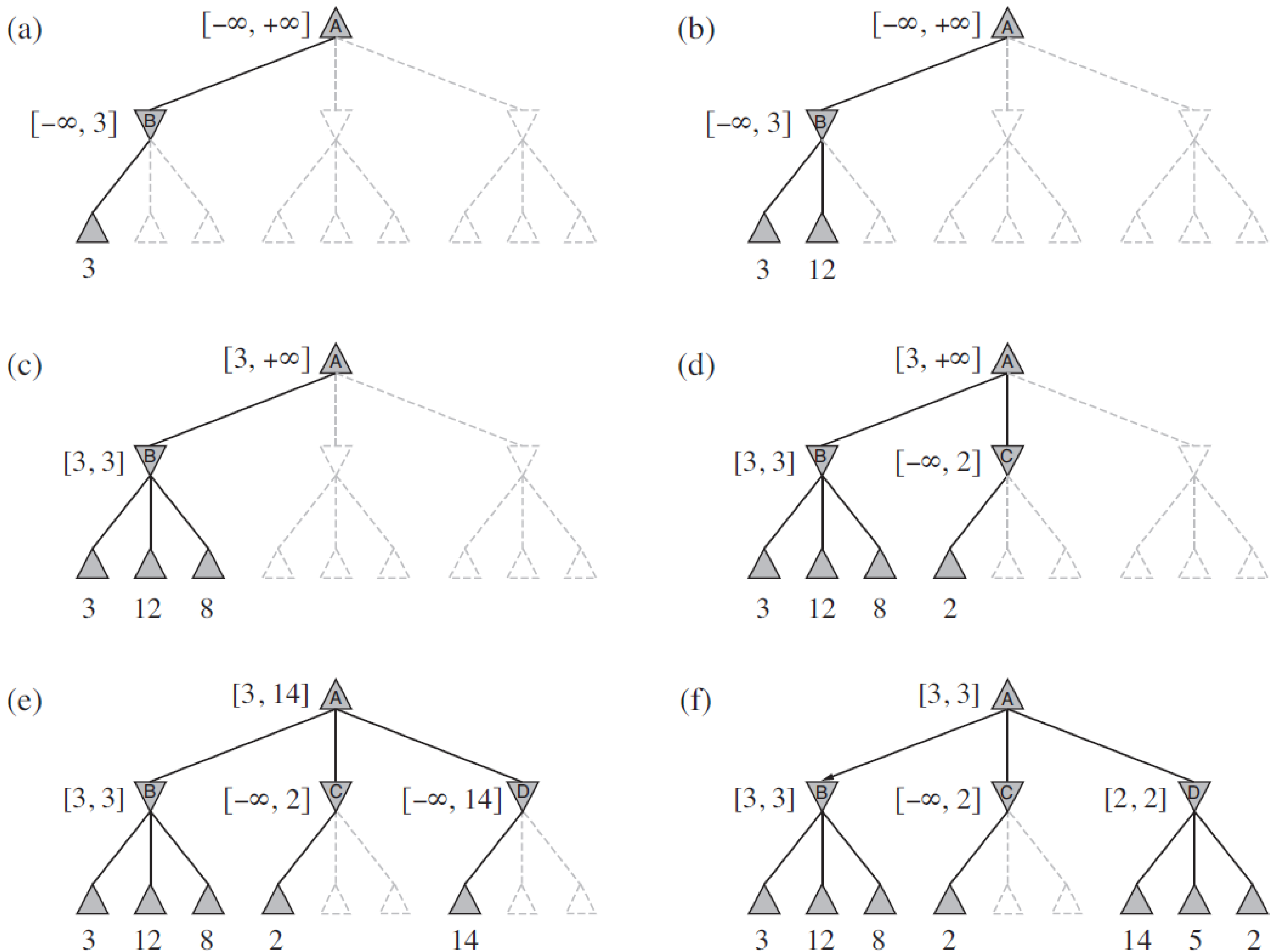
More than 2 player games

- It is possible to store vectors of utility values at nodes.
- Multiplayer games typically involve **alliances**



Can minimax be optimized?

α-β pruning example

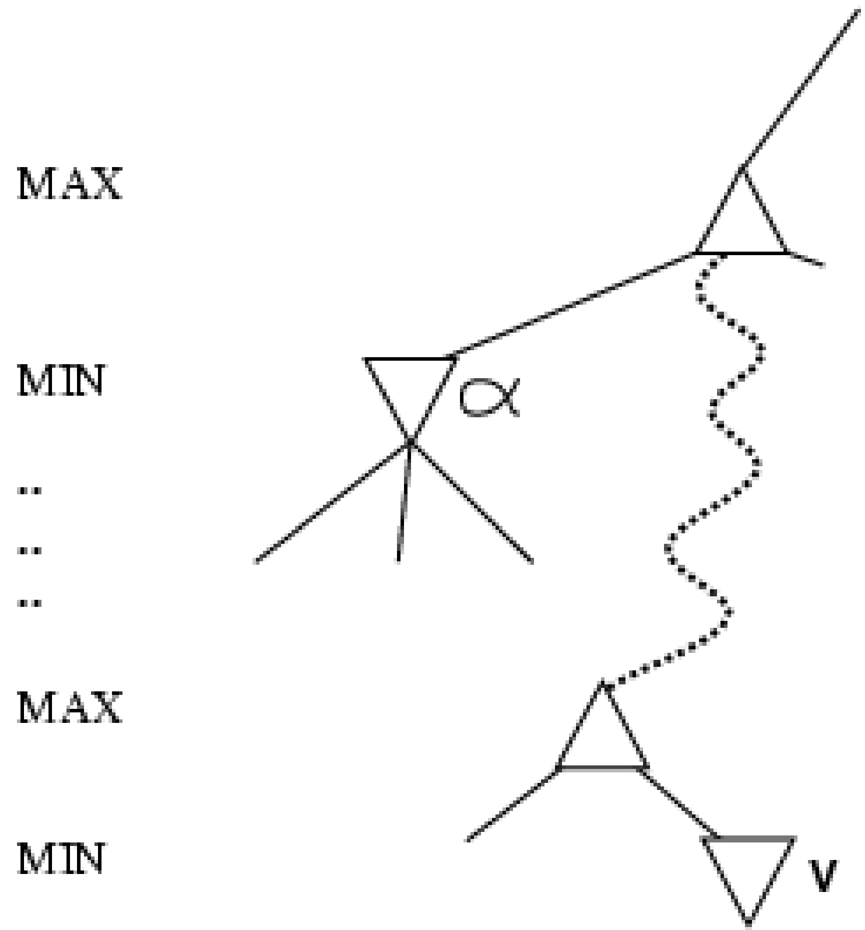


Properties of α-β

- 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
- 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
    v ← MAX-VALUE(state, −∞, +∞ )
    return the action in ACTIONS(state) with value v
```

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

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

The α-β algorithm

```
def alphabeta_full_search(state, game):
    """Search game to determine best action; use alpha-beta pruning.
    As in [Fig. 5.7], this version searches all the way to the leaves."""

    player = game.to_move(state)

    def max_value(state, alpha, beta):
        if game.terminal_test(state):
            return game.utility(state, player)
        v = -infinity
        for a in game.actions(state):
            v = max(v, min_value(game.result(state, a), alpha, beta))
            if v >= beta:
                return v
            alpha = max(alpha, v)
        return v

    def min_value(state, alpha, beta):
        if game.terminal_test(state):
            return game.utility(state, player)
        v = infinity
        for a in game.actions(state):
            v = min(v, max_value(game.result(state, a), alpha, beta))
            if v <= alpha:
                return v
            beta = min(beta, v)
        return v

    # Body of alphabeta_search:
    return argmax(game.actions(state),
                  lambda a: min_value(game.result(state, a),
                                      -infinity, infinity))
```

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

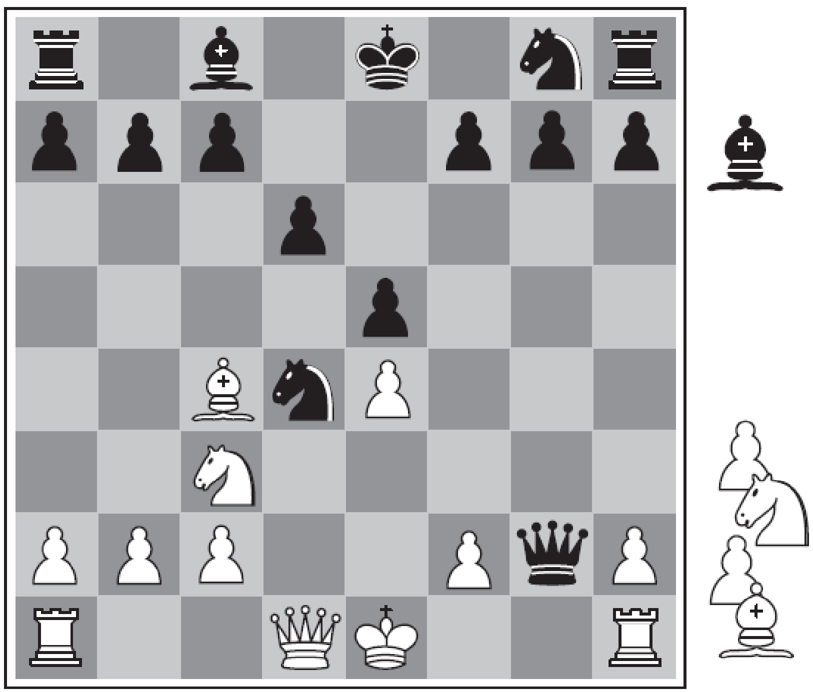
- For chess, typically linear weighted sum of features

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

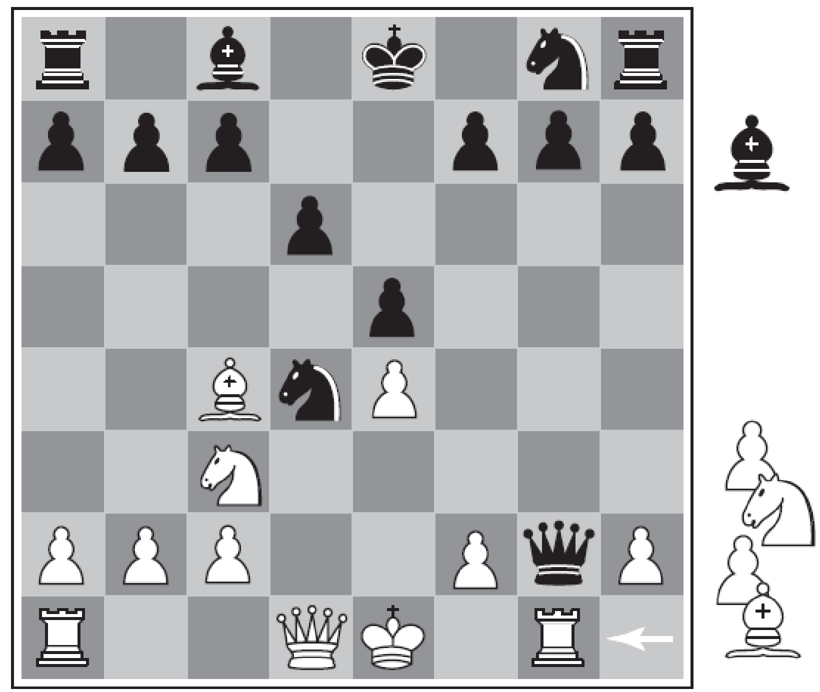
- e.g., $w_1 = 9$ with

  $f_1(s)$ = (number of white queens) – (number of black queens), etc.

Evaluation in Chess



(a) White to move



(b) White to move

Tricks

- Using databases of previous games
- Using knowledge from books (e.g. openings)
- Quiescence search
- Singular extensions
  - Try “killer moves” after the horizon has been reached
- Retrogade minimax – do unmoves from desired outcome
  - Used for fully exploring endgames

Cutting off search

MinimaxCutoff is identical to *MinimaxValue* except

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

Does it work in practice?

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

4-ply lookahead is a hopeless chess player!

– 4-ply ≈ human novice
– 8-ply ≈ typical PC, human master
– 14-ply ≈ Deep Blue, Kasparov
– 18-ply ≈ Hydra

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.
  - July 19, 2007 - The journal *Science* publishes Schaeffer's team's article "Checkers Is Solved", presenting their proof that the best a player playing against Chinook can achieve is a draw.

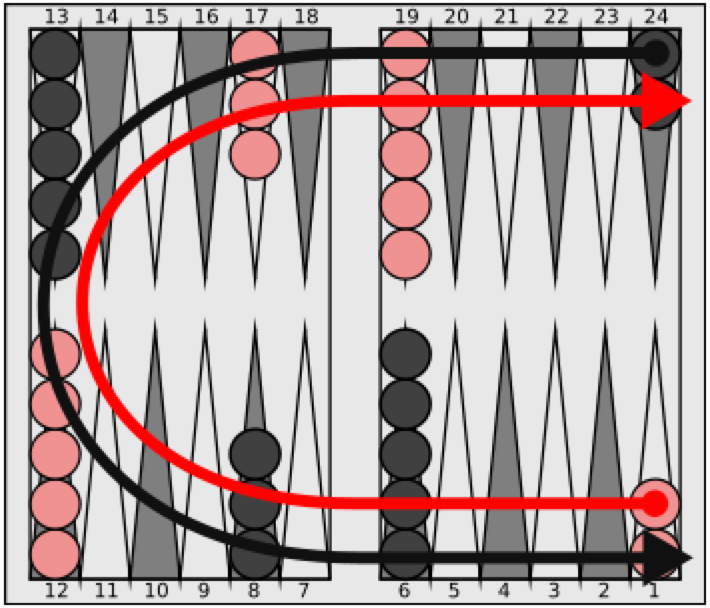
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

- Othello (Reversi): human champions refuse to compete against computers, who are too good.

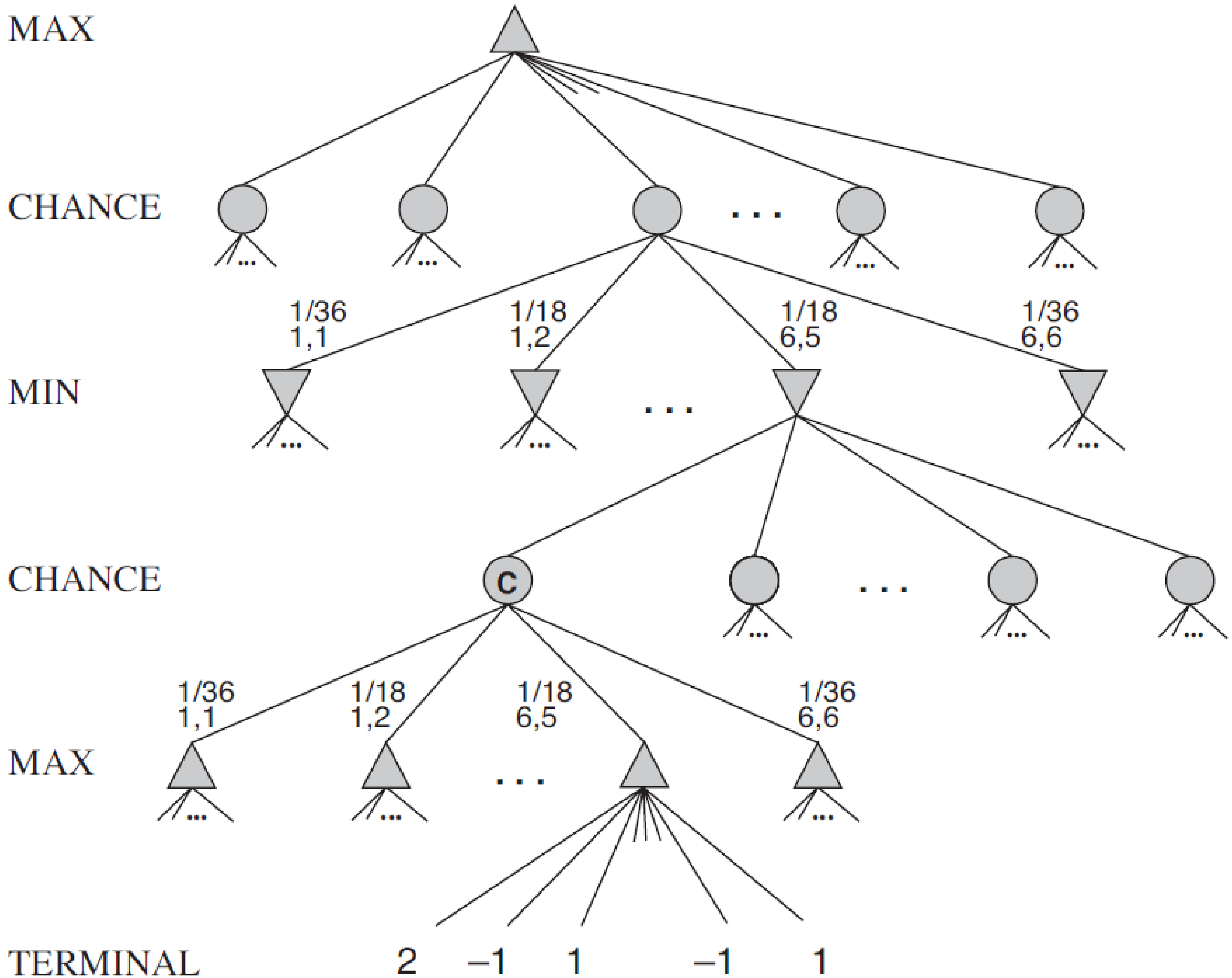
- Go: human champions refused to compete against computers, who were too bad for a long time. But since 2011 Go playing programs have successfully competed against human masters (Zen, Crazy Stone)

Stochastic games

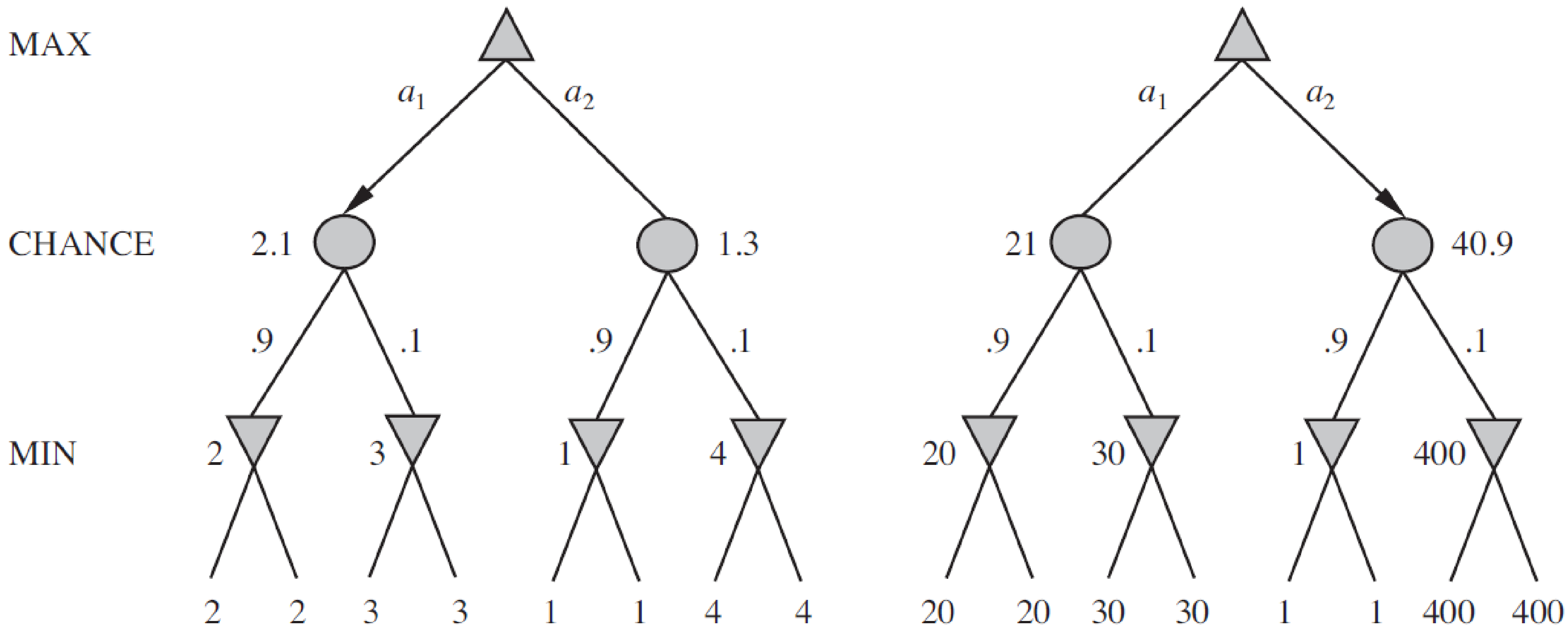
- Backgammon



Game tree of Backgammon



Effect of evaluation functions



ExpectiMinimax

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

Partially observable games

- Krigspiel – only part of the chess board is observable
- Bridge:
  - Deal s occurs with probability P(s)
  - We want
    - argmax a sum(s) P(s) Minimax(Result(s,a))
    - Monte Carlo sampling

Summary

- Games illustrate several important points about AI
- Perfection is unattainable → must approximate
- Good idea to think about what to think about
- Partial observability and belief states will be dealt in more detail later in the course

Problems

- Consider the problem of solving two 8-puzzles.
  - Give a complete problem formulation (according to the style introduced in classical search).
  - How large is the reachable state space? Give exact numerical expression.
  - Suppose we make the problem adversarial as follows: th two players take turns moving; a coin is flipped to determine the puzzle on which to make a move in that turn; and th winner is the first to solve one puzzle. Which algorithm can be used to choose a move?
  - Will someone always win eventually if both play perfectly?