Artificial Intelligence

Adversarial Search I.

December 1st, 2009

Tomáš Horváth

[Stuart Russell, Peter Norvig: Artificial Intelligence – A Modern Approach, Prentice Hall, 2003]

What should be ***discussed*** today

- deterministic games
  - environment of competitive agents
  - as search problem
- minimax algorithm
  - properties
  - $\alpha-\beta$ prunning
  - some heuristics
- elements of chance

Games in AI

- mathematical game theory
  - a branch of economics
  - multi-agent environment as a game where agents have significant influence on each other
    - competitive or cooperative

- Why are games interesting for AI?
  - hard problems to solve

zero-sum games

- deterministic, fully observable environments
  - two competitive agents (i.e. two players)
  - alternate actions
  - the utility values are sum to zero at the end
    - winner (+1), loser (-1), equal (0)
    - if one agent wins the other necessarily loses
      - <u>adversarial</u> situation

- find a strategy specifying the move for every possible opponent reply

problem formulation

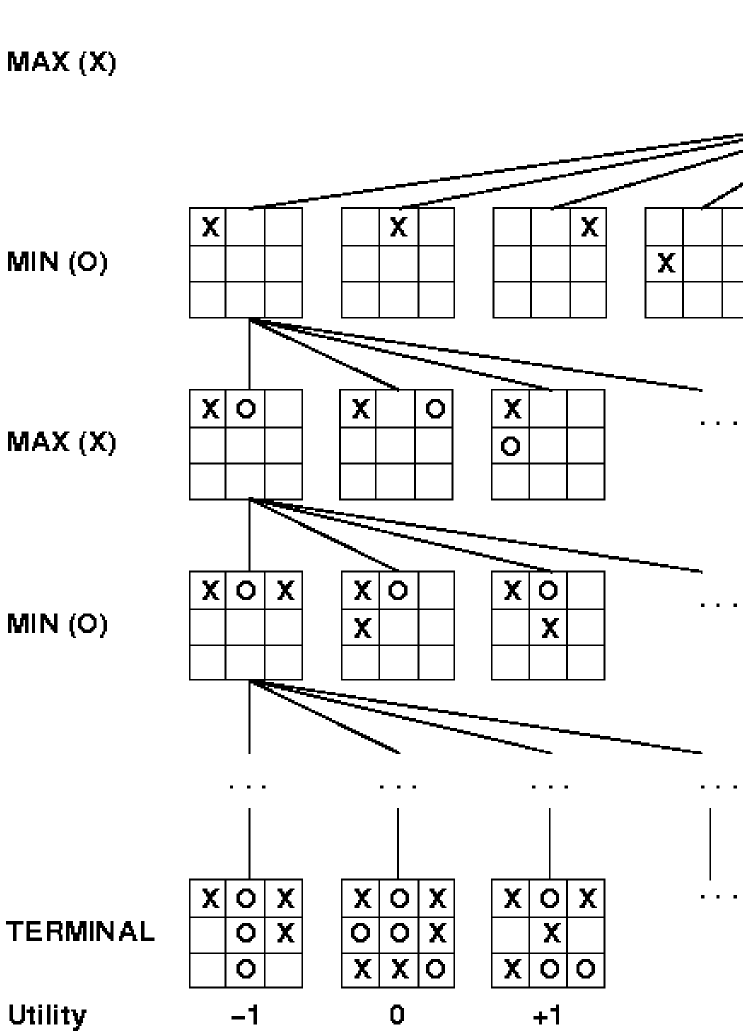
- initial state
  - board position and player to move
- successor function
  - returns a list of (move, state) pairs which indicate a legal move and the resulting action
- terminal test
  - determines when the game is over, i.e. the game reached one of a so-called <u>terminal states</u>
- utility function
  - gives numeric value for terminal states (-1, 0, +1)

problem formulation

- two players called “MIN” and “MAX”
    - names “Stan” & “Pan” were already booked by Hollywood :-)
  - MAX is playing a strategy for maximizing its utility
    - MAX moves first
  - MIN is trying to minimize MAX's utility

- How can we represent this problem?
  - e.g. for the game TIC-TAC-TOE

game tree



- represents the happening in the game
- each level in the tree
    - belongs to one player to move
    - half turn = ply

strategy

- in a normal search problem
  - optimal solution is a sequence of moves to a terminal state with utility value = +1
- but in a game
  - MIN has impact on the moves of MAX
- an optimal strategy is determined by examining a “value” of each node
  - we call this value <u>minimax value</u>

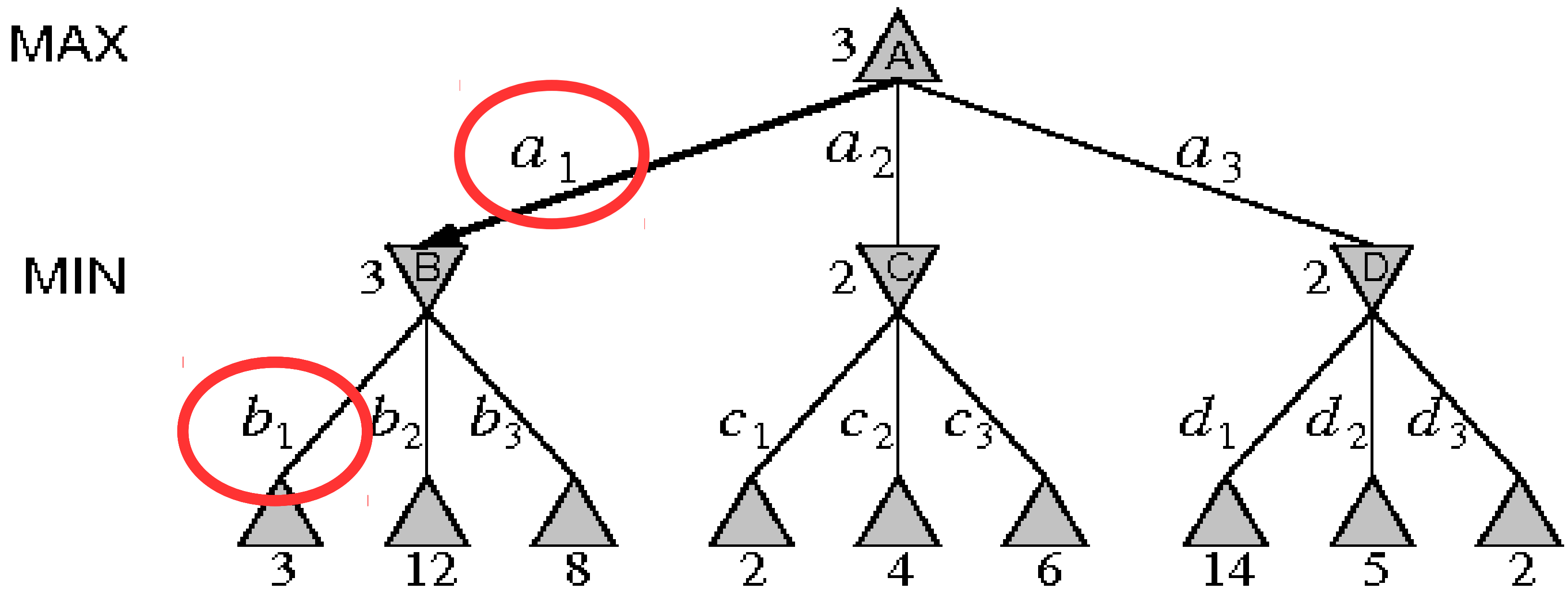
minimax value

- computed for every node in the game tree

- MINIMAX-VALUE(n) =
  - UTILITY(n)
    - if n is a terminal state
  - $\max_{s \in \text{Successors}(n)}$ MINIMAX-VALUE(s)
    - if n is a MAX node
  - $\min_{s \in \text{Successors}(n)}$ MINIMAX-VALUE(s)
    - if n is a MIN node

optimal decisions

- MAX moves to states with highest minimal values
- MIN moves to states with lowest maximal values



minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
   inputs: state, current state in game

   return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
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

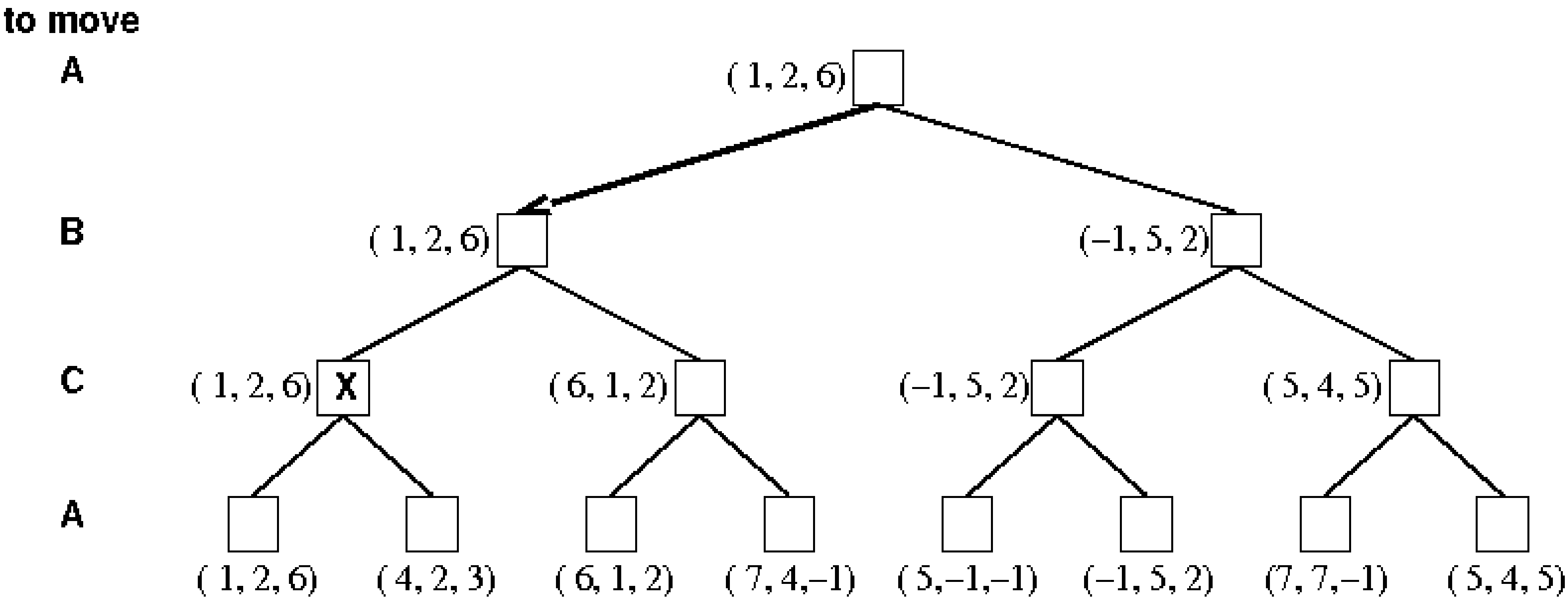
- properties
  - performs a complete depth-first exploration of a game tree
  - time complexity O(b^m)
    - m = maximal depth
    - b = legal moves at each point
  - space complexity
    - O(b*m)
      - if generates all successors at once
    - O(m)
      - if generates successors one at a time

more players

- <u>vector</u> of values in the nodes
  - instead of single values
  - gives utility of the state for each player

- which state a given player chooses?

more players

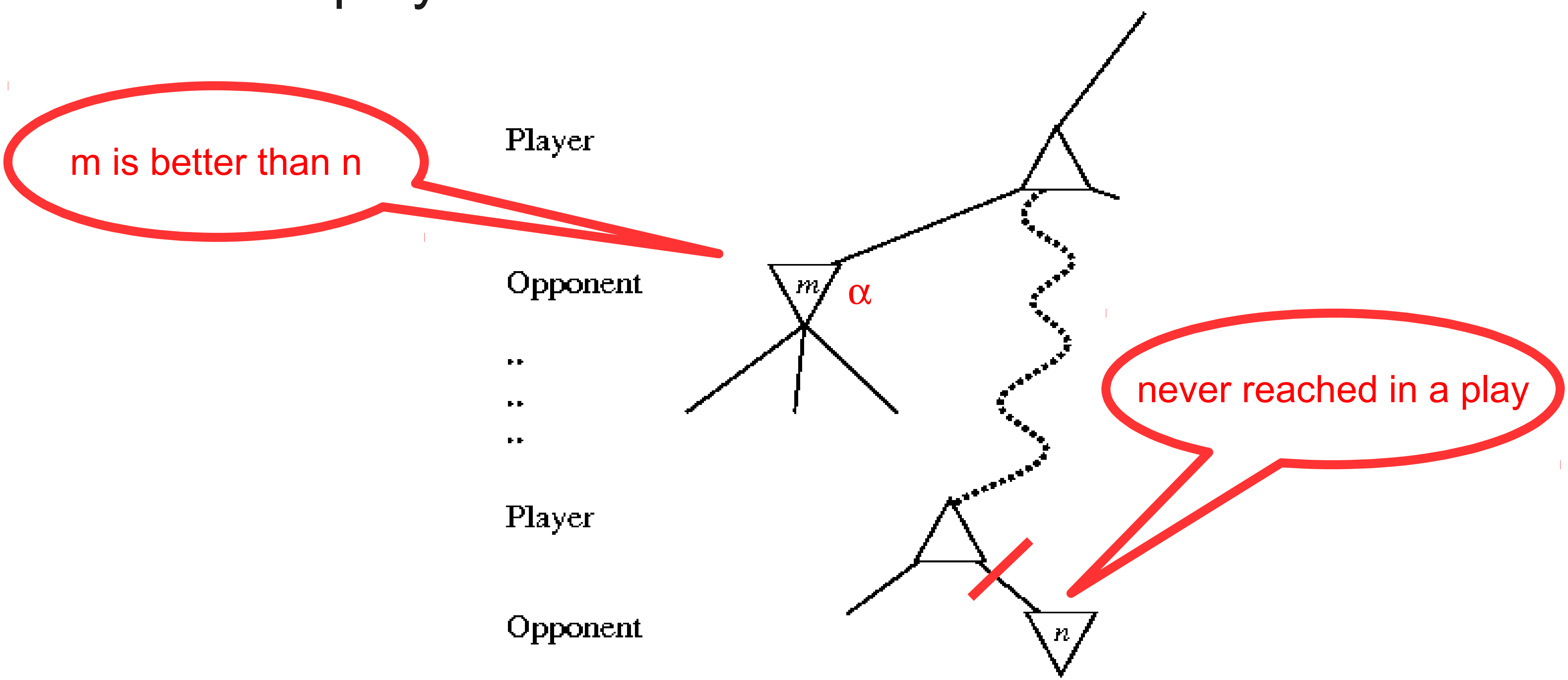


alliances

- when the players in weak positions attack the player(s) in strong positions.
  - is it a natural consequence of optimal strategies?
  - in case of two players
    - consider a terminal state (1000,1000) with 1000 as the highest possible utility value for each player
      - the optimal strategy for both players is to reach this state, i.e. they will automatically cooperate

alpha-beta prunning

- basic idea
  - eliminate nodes which will be never reached in the actual play



alpha-beta prunning

- MINIMAX-VALUE(root) =

  = max(min(3,12,8), min(2,x,y), min(14,5,2))

  = max(3, min(2,x,y), 2)

  = max(3, z, 2)   where $z \leq 2$

  = 3

alpha-beta prunning

- two parameters (α,β)
  - bounds on the backed-up values
  - α = the value of the best choice we have found so far at any choice point along the path for MAX
    - best choice = the highest value
  - β = the value of the best choice we have found so far at any choice point along the path for MIN
    - best choice = the lowest value

alpha-beta algorithm

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

---

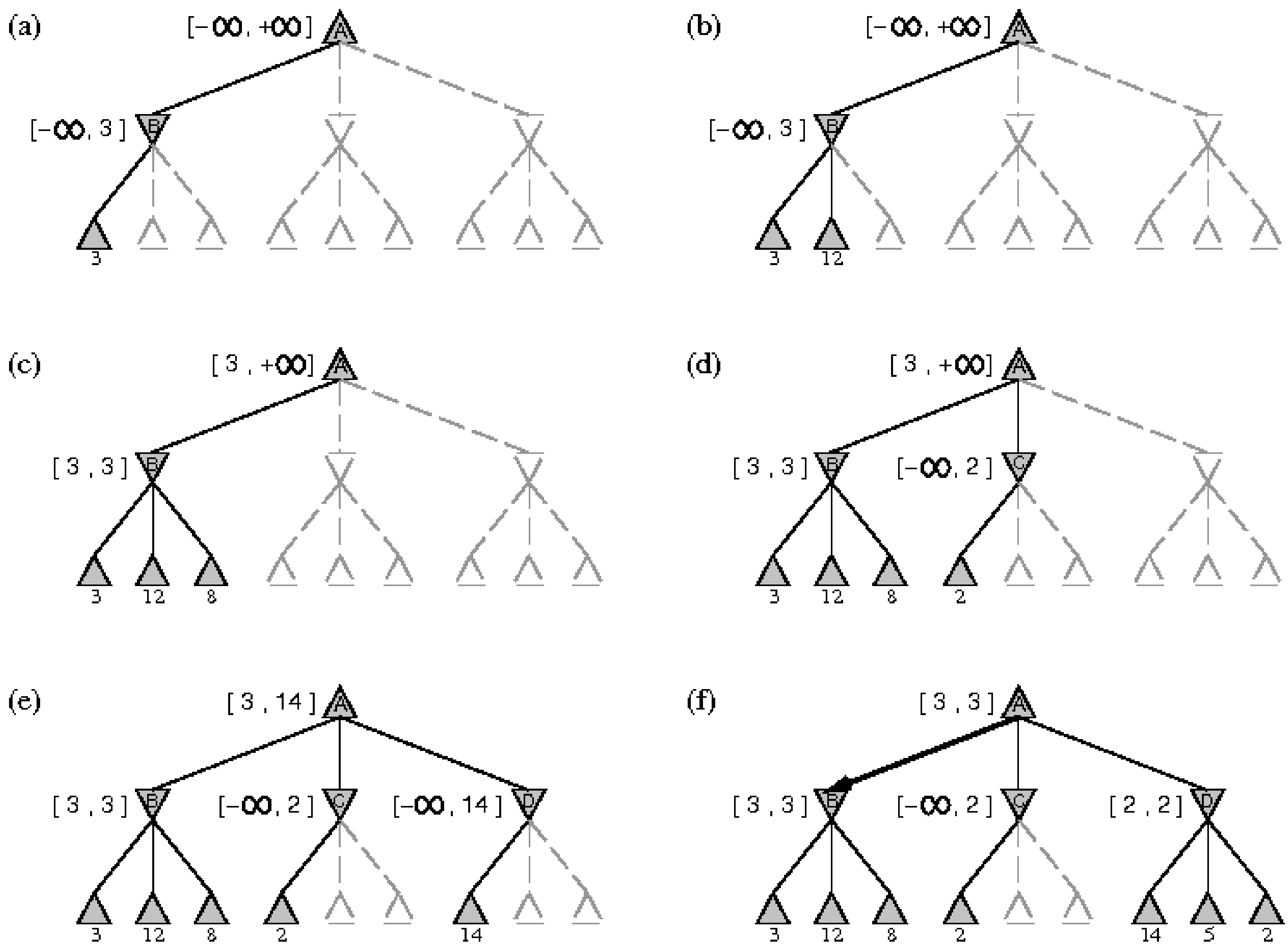
function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

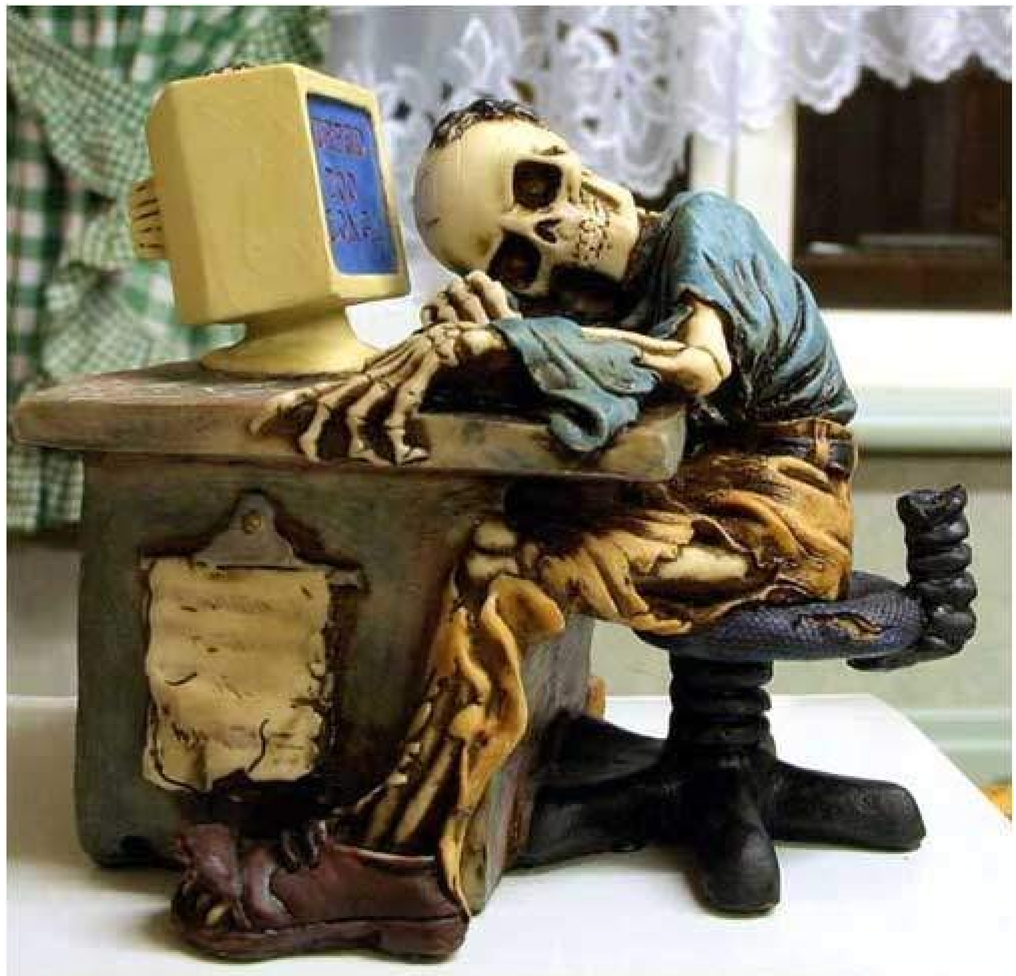
alpha-beta algorithm



alpha-beta algorithm

- properties
  - finds the same strategy as the minimax algorithm
    - the effectiveness is dependent on the order in which the successors are examined
  - time complexity
    - „ideal“ ordering of child-nodes: $O(b^{(m/2)})$
    - random ordering: $O(b^{(3m/4)})$

real-time decisions



computer on the move...

transpositions

- different permutations of the move sequence that end up in the same position
  - eliminating the transpositions
  - transpositions table
    - a hash table of previously seen positions
    - is it practical if evaluating many nodes to keep all of them in a transposition table?

evaluation function

- estimate of the expected utility of the game from a given position
  - UTILITY function $\Rightarrow$ heuristic EVALuation function
  - terminal test $\Rightarrow$ cutoff test
- how to design EVAL
  - EVAL should order terminal states in the same way as the UTILITY function
  - computation of EVAL must be effective
  - EVAL should be strongly correlated with the actual chances of winning

evaluation function

- features of the state
  - define various categories of states
    - each category contain states that leads to win, to draws and to losses
- expected value
  - weighted average of the outcomes of the states in the category
    - (0.72 * (+1)) + (0.20 * (-1)) + (0.08 * 0) = 0.52

evaluation function

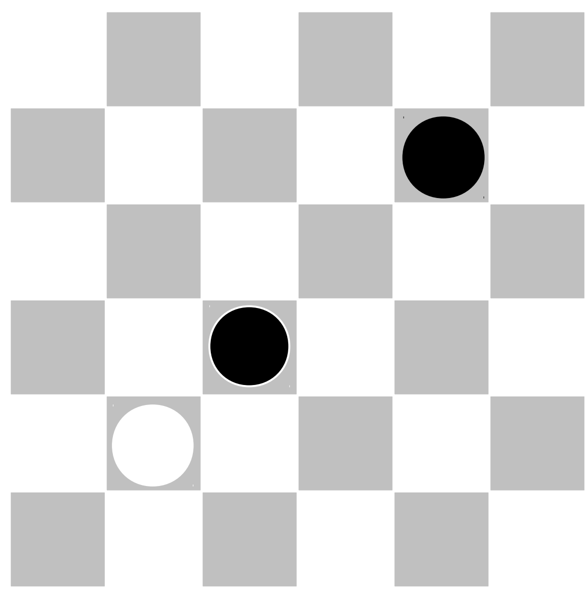
- material value
  - numerical contributions from each feature
    - chess: pawn = 1; knight, bishop = 3; rook = 5; queen = 9
- weighted linear function
  - $EVAL(s) = w_1 * f_1(s) + ... + w_n * f_n(s)$
    - w_i ... weight
    - f_i ... feature
- non-linear combination can be also used

cutting off the search

- cutoff test
  - determines when to use EVAL
  - if CUTOFF-TEST(state, depth) then return EVAL(s)
  - problem
    - may be applied when it is unfavorable

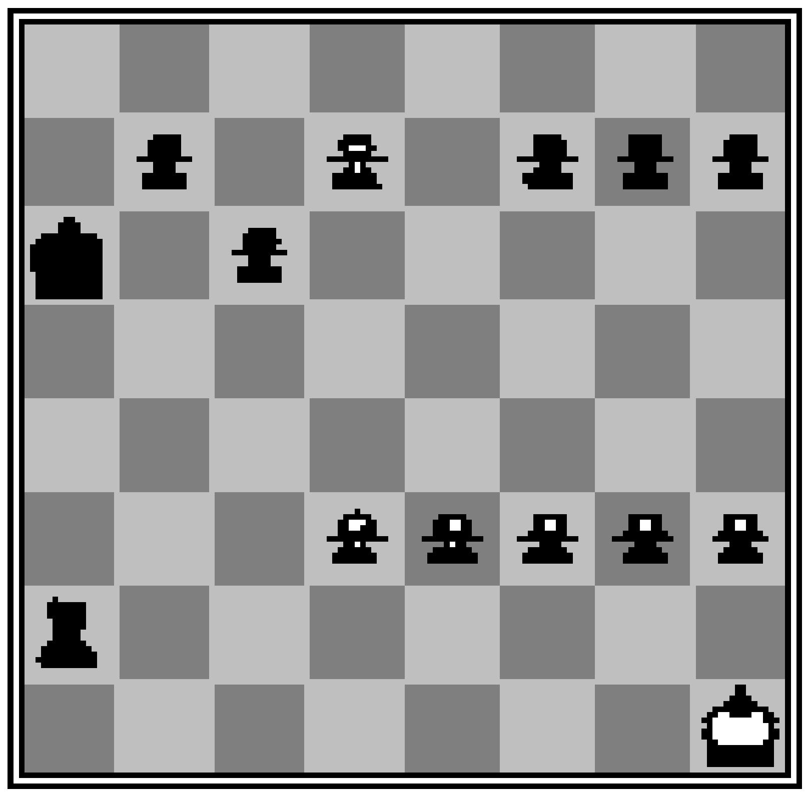
quiescence search

- when material values are used
  - quiescent position
    - where is unlikely to exhibit wild swings in value in the near future
  - only apply EVAL in quiescent positions



horizon effect

- arises when the program is facing a move by the opponent that causes serious damage and is ultimately unavoidable



Black to move

other considerations

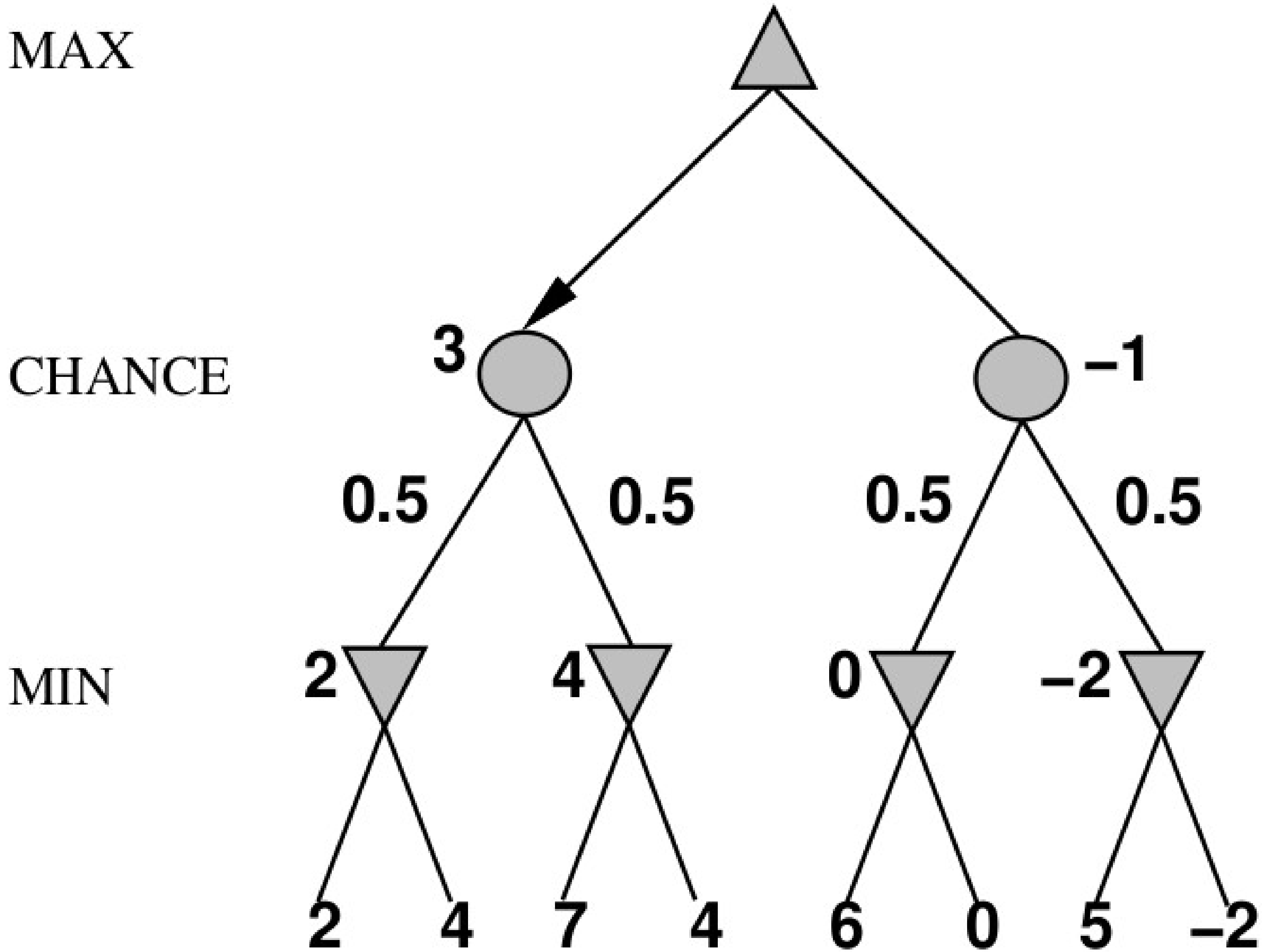
- singular extension
  - move that is clearly better than all other moves in a given position
  - idea: expand just the “better” moves
- forward pruning
  - some moves at a given node are pruned immediately without further consideration
    - e.g. symmetric moves

games with elements of chance

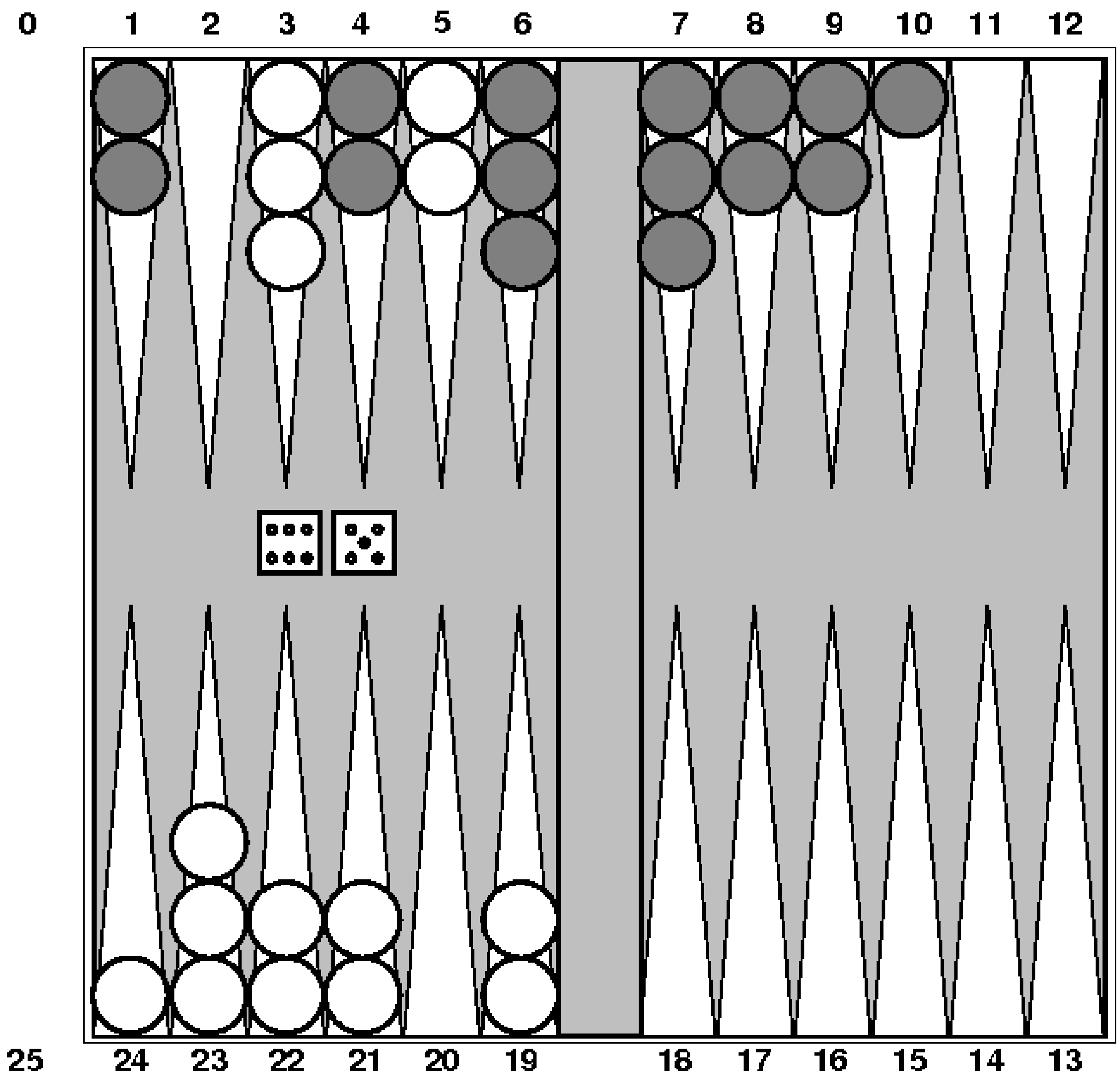
- random element included in a game
  - throwing the dice
  - backgammon

- we can't construct the standard game tree
  - a tree for such a game includes <u>chance nodes</u>
    - labeled with
      - the roll
      - the chance the roll occurs

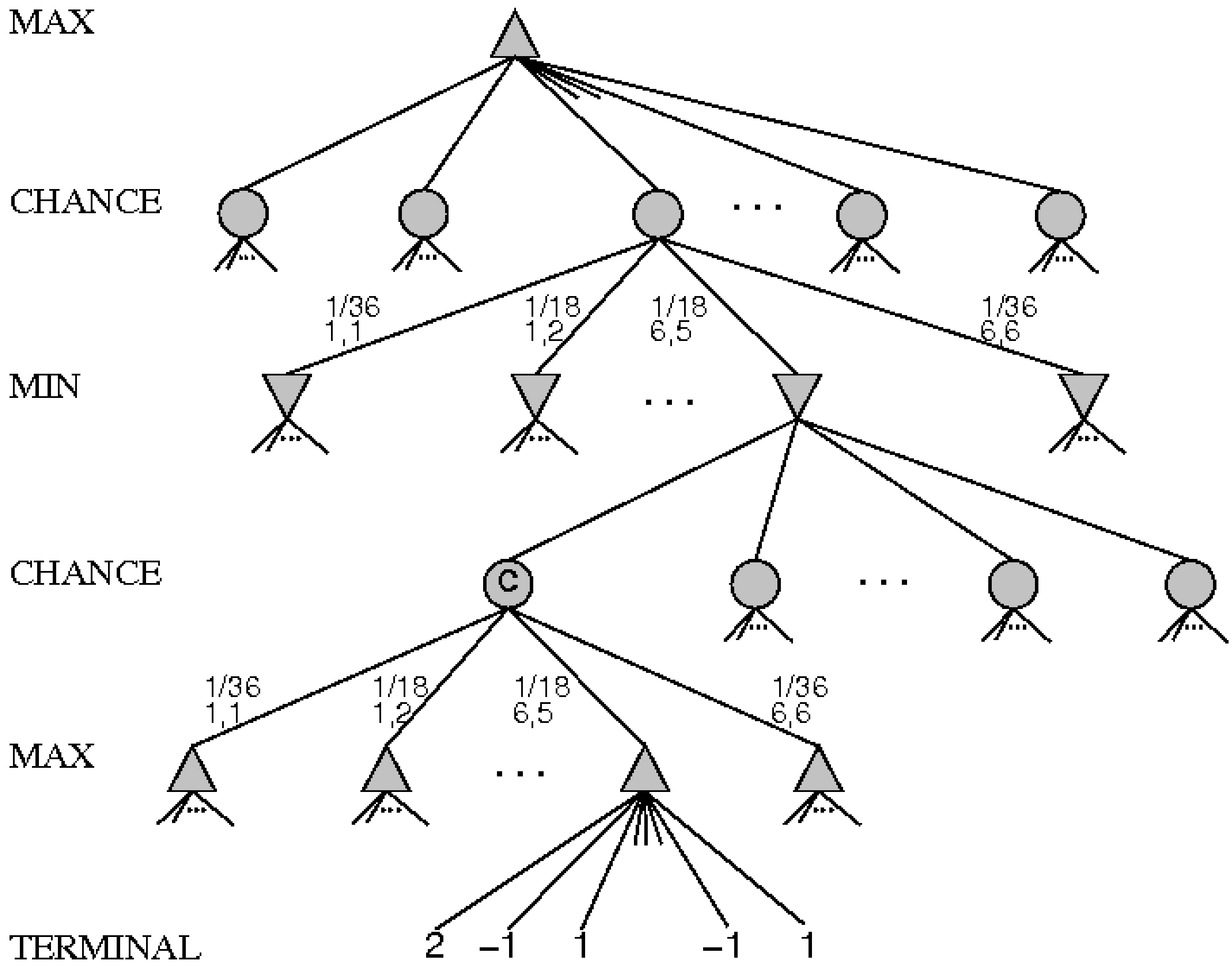
chance nodes



backgammon



backgammon tree



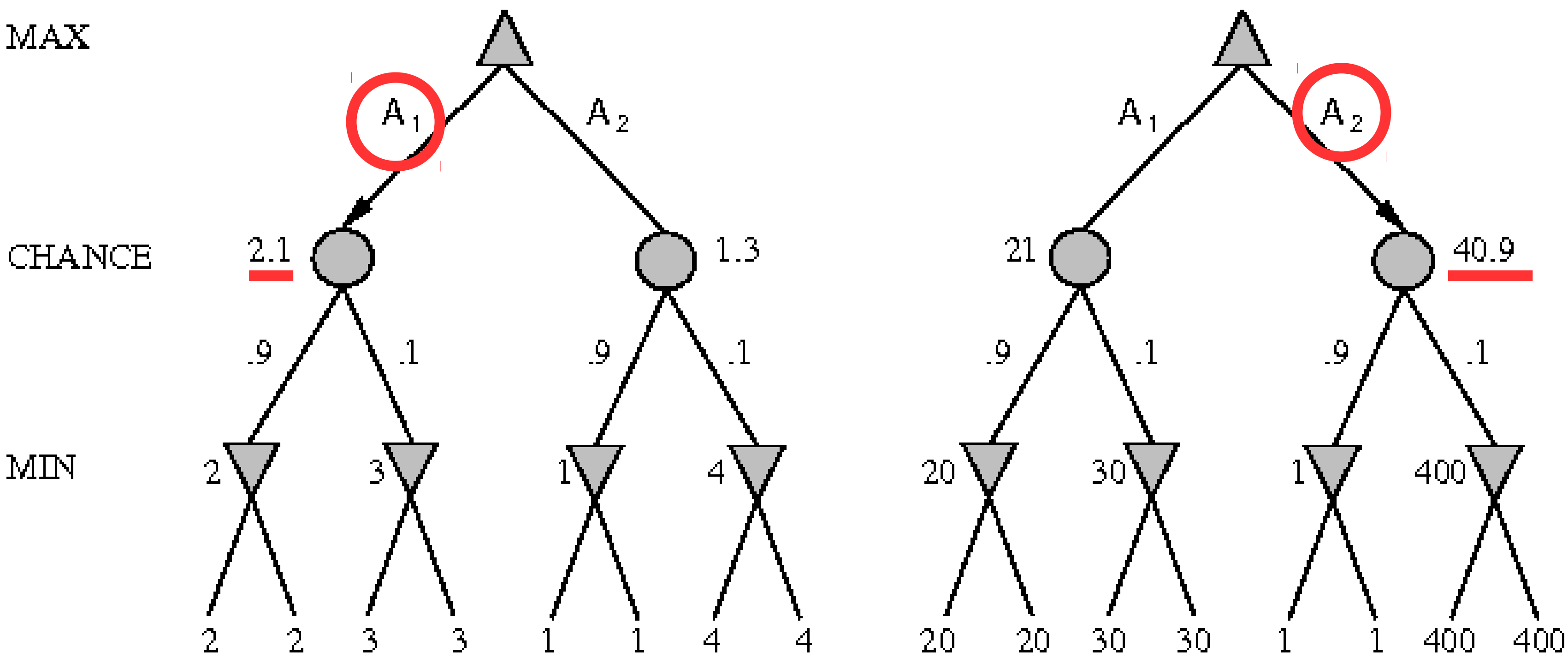
expectiminimax value

- expected values instead of definite minimax values

- EXPECTIMINIMAX(n) =
  - UTILITY(n)
    - if n is a terminal state
  - $\max_{s \in \text{Successors}(n)}$ EXPECTIMINIMAX(s)
    - if n is a MAX node
  - $\min_{s \in \text{Successors}(n)}$ EXPECTIMINIMAX(s)
    - if n is a MIN node
  - $\sum_{s \in \text{Successors}(n)}$ P(s) * EXPECTIMINIMAX(s)
    - if n is a chance node

digression

- exact values do matter in case of chance nodes
  - EVAL could be a positive linear transformation of the expected utility of the position



games with imperfect information

- belief states
  - Day 1: Road A leads to a heap of gold pieces; Road B leads to fork. Take the left fork and you'll find a mound of jewels, but take the right fork and you'll be run over by a bus.
  - Day 2: Road A leads to a heap of gold pieces; Road B leads to fork. Take the right fork and you'll find a mound of jewels, but take the left fork and you'll be run over by a bus.
  - Day 3: Road A leads to a heap of gold pieces; Road B leads to fork. Guess correctly and you'll find a mound of jewels, but guess incorrectly and you'll be run over by a bus.

  - road B is optimal on day 1 and on day 2
    - is road B therefore optimal on day 3?
      - averaging over clairvoyance suggests the road B...

Summary

- games as search problems
- minimax
  - assumes that opponent plays optimally
  - utility function
  - pruning
- real-time decisions
  - cutoff
  - EVAL functions as search heuristics
- elements of chance
  - expected values of chance
- games with imperfect information
  - optimal decisions depend on information state, not real state