Adversarial Search

Lecturer: Eibe Frank

Based on "Artificial Intelligence"
by S. Russell and P. Norvig
Sections 6.1-6.6

- Games
- Optimal decisions in games
- Alpha-beta pruning
- Imperfect, real-time decisions
- Games that include an element of chance
- State-of-the-art game programs

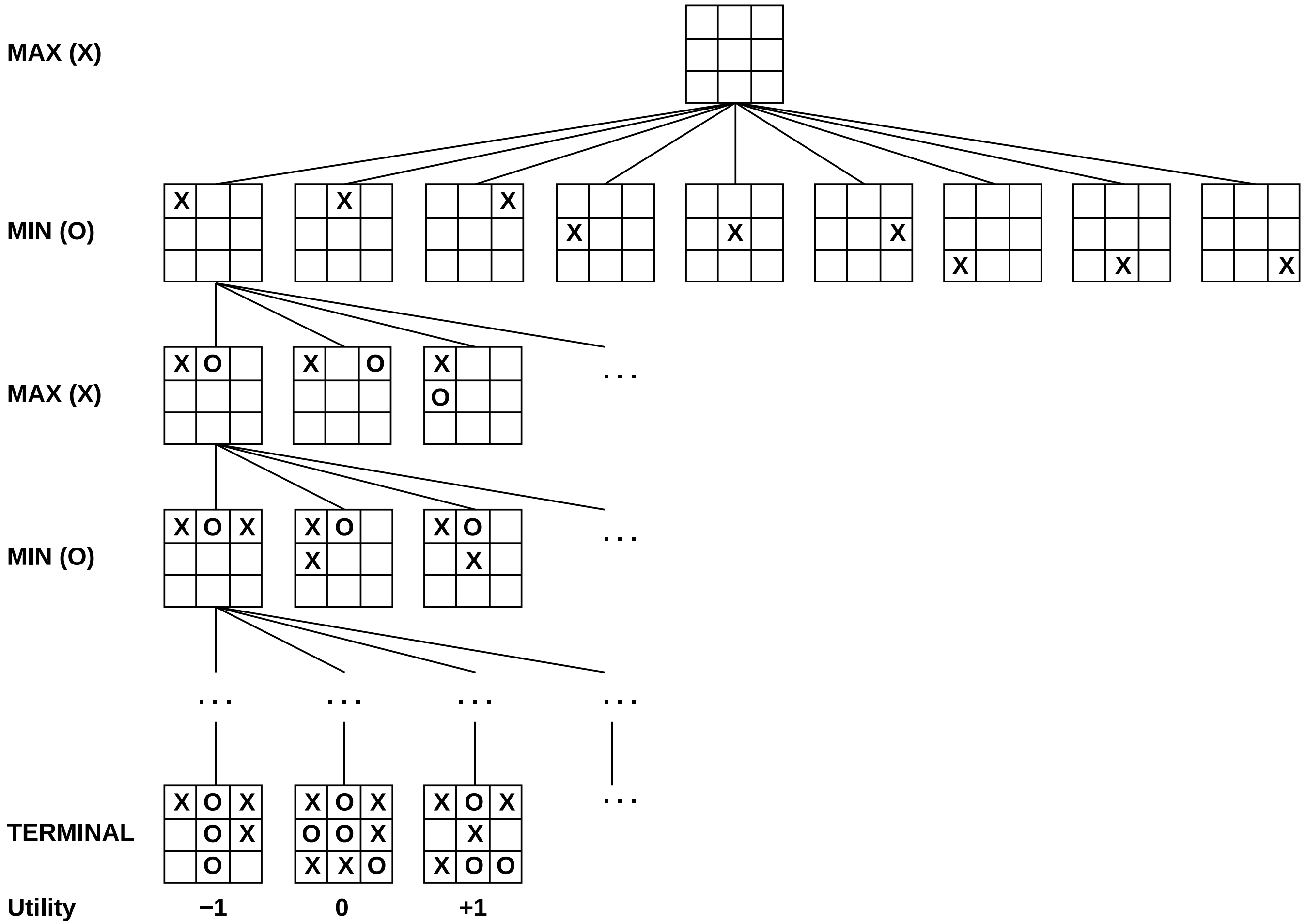
Games

- **Competitive** multi-agent environments give rise to **adversarial search problems** (also known as **games**)
- Unpredictability of other agent introduces **contingencies**
- Mathematical **game theory** views any multiagent environment as a game
- **Zero-sum games**: utility values at end of game are always equal and opposite
- Games of **perfect information**: fully-observable environments
- Abstract nature makes games useful for AI research
  - States and actions are easy to represent

A game as a search problem

- Has the following components:
  - **Initial state**: includes the board position and identifies the player to move
  - **Successor function**: returns list of (*move, state*) pairs, each indicating a legal move and the resulting state
  - **Terminal test**: determines when the game is over (i.e., when we are in a **terminal state**)
  - **Utility function**: gives a numeric value in terminal states (i.e., -1, 0, +1 in chess)
- We will call the first player MAX and the second player MIN (and state utility values from MAX's perspective)
- Initial state and legal moves define **game tree**

A partial game tree for tic-tac-toe



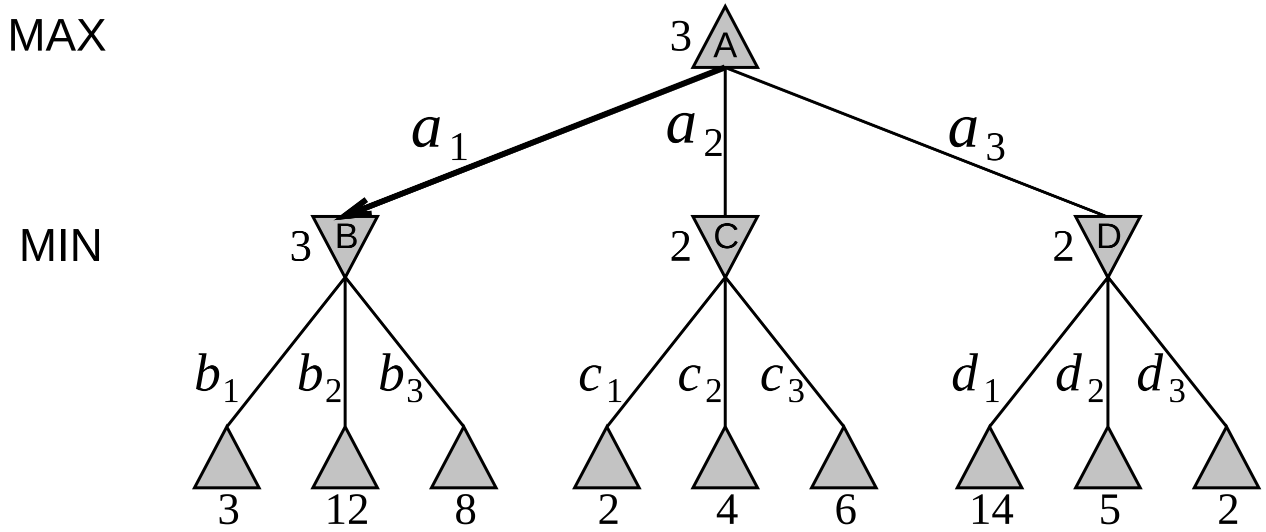
Optimal strategies

- MIN has something to say about outcome of game: need contingent **strategy**
- Optimal strategy: leads to outcome at least as good as any other strategy when playing infallible opponent
- Optimal strategy can be determined using **minimax value** of each node:
  - Utility (for MAX) of being in state, *assuming both players play optimally* from there to the end of the game

$$MINIMAX-VALUE(n) =$$

$$\begin{cases} Utility(n) & \text{if } n \text{ is a terminal state} \\ max_{s \in Successors(n)} MINIMAX-VALUE(s) & \text{if } n \text{ is a MAX node} \\ min_{s \in Successors(n)} MINIMAX-VALUE(s) & \text{if } n \text{ is a MIN node} \end{cases}$$

A two-ply game tree



The 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
```

Properties of the minimax algorithm

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

Properties of the minimax algorithm

- Complete?
  - Yes, if tree is finite
- Time complexity?
- Space complexity?
- Optimal?

Properties of the minimax algorithm

- Complete?
  - Yes, if tree is finite
- Time complexity?
  - $O(b^m)$ (depth-first exploration)
- Space complexity?
- Optimal?

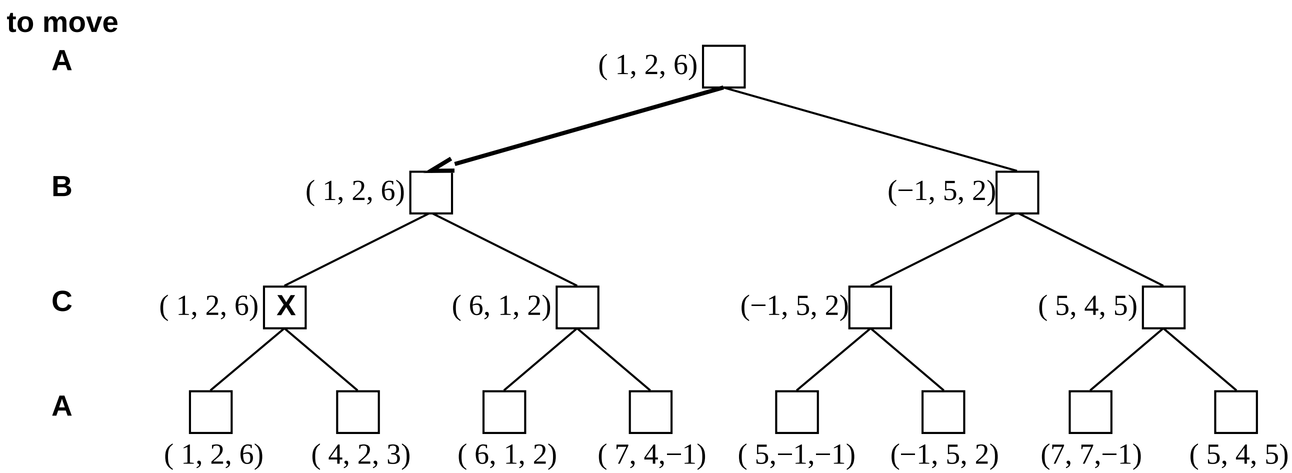
Properties of the minimax algorithm

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

Properties of the minimax algorithm

- Complete?
  - Yes, if tree is finite
- Time complexity?
  - $O(b^m)$ (depth-first exploration)
- Space complexity?
  - $O(bm)$
- Optimal?
  - Yes, against an optimal opponent. Otherwise? No.

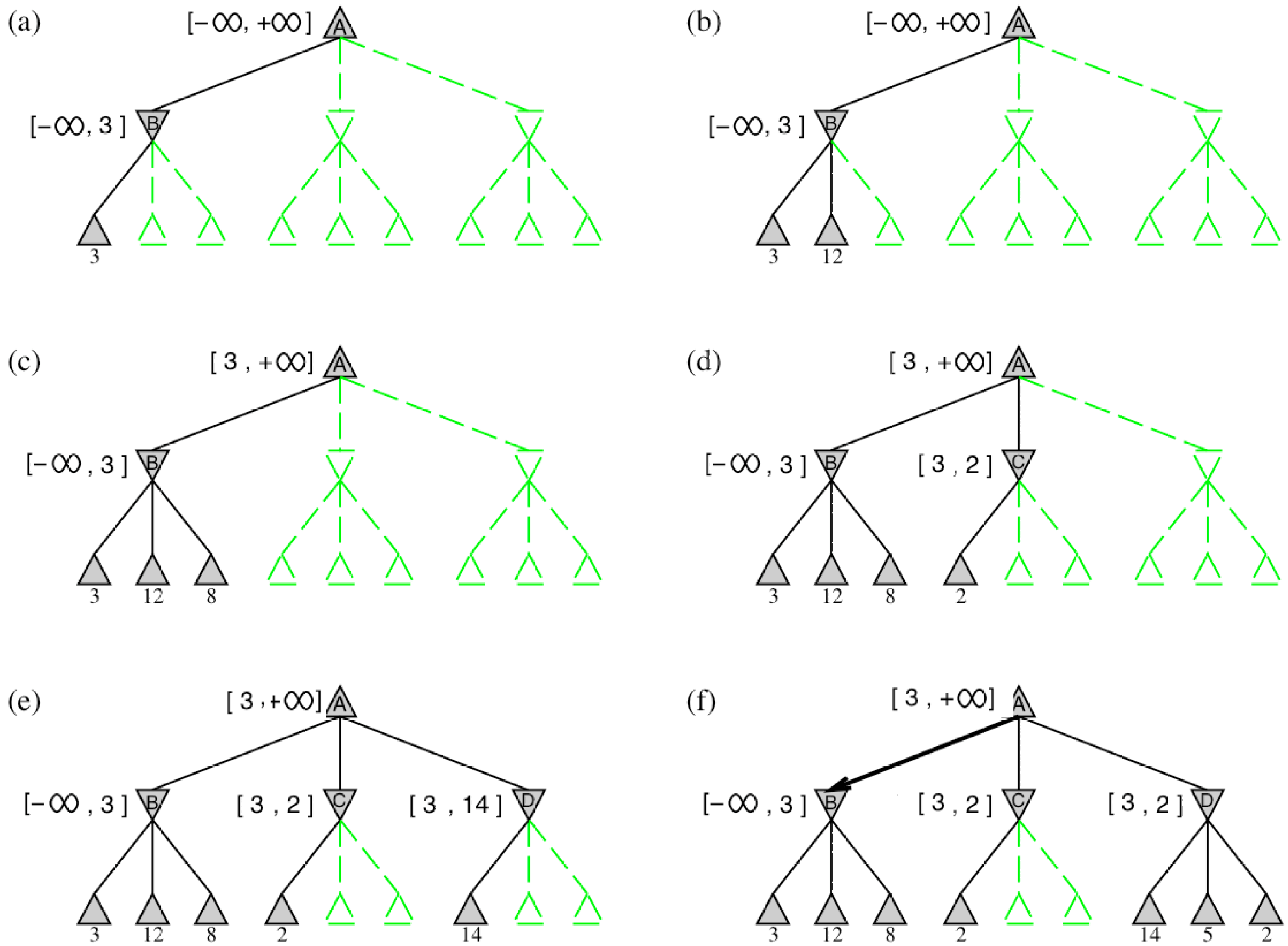
Tree for game with three players



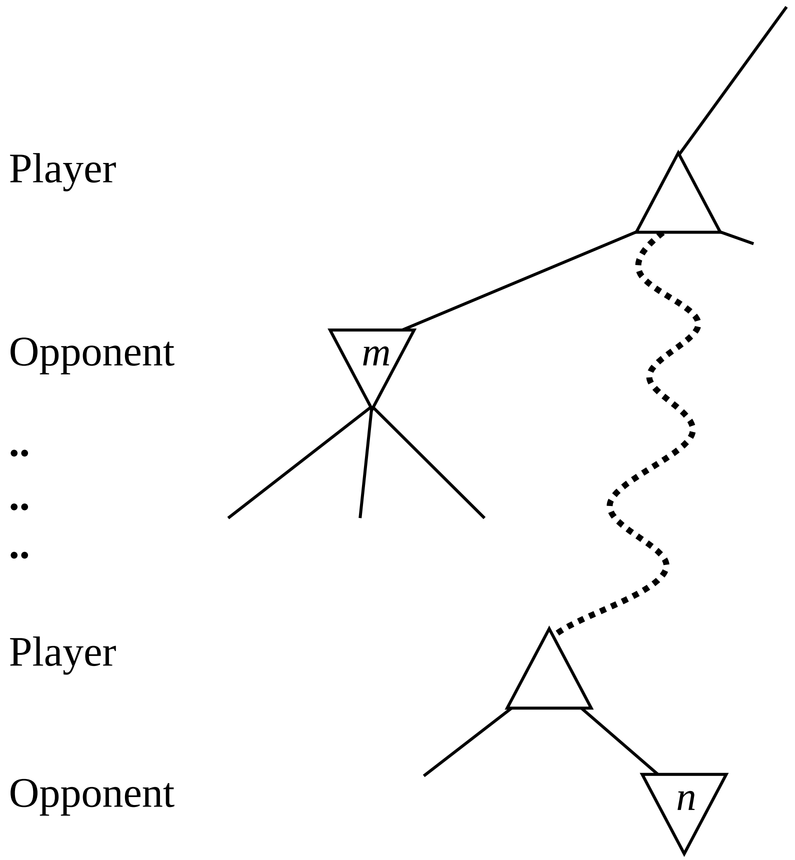
Alpha-beta pruning

- Minimax search: number of game states to be examined is exponential in number of moves
- **Alpha-beta pruning** can effectively cut exponent in half
- It turns out that we can compute the correct minimax decision without looking at every node in the game tree
- Idea: **prune** branches that cannot possibly influence the final decision
- Maintains two parameters:
  - α = the value of the best choice found so far at any choice point along the path for MAX
  - β = the value of the best choice found so far at any choice point along the path for MIN

Alpha-beta pruning example



Alpha-beta pruning: the general case



Alpha-beta pruning: the 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/β, the value fo the best alternative for MAX/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))
    α ← MAX(α, v)
    if v ≥ β then return v
  return v
```

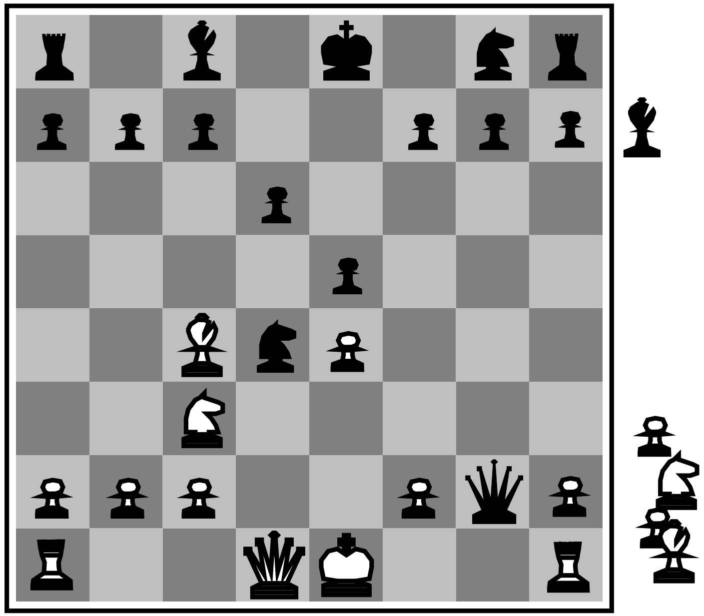
```
function MIN-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α/β, the value fo the best alternative for MAX/MIN along the path to state

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

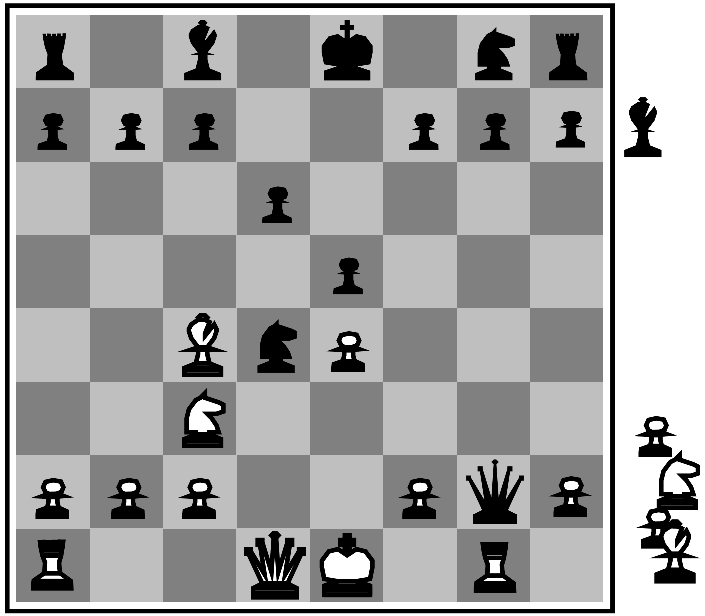
Alpha-beta pruning: properties

- Pruning *does not* affect the final result
- Good move odering improves effectiveness of pruning
- With "perfect" ordering, time complexity = $O(b^{m/2})$
  - *Doubles* depth of search
  - Can easily reach depth 8 and play good chess
- A "simple" example of the value of reasoning about which computations are relevant (a form of *metareasoning*)

Evaluation functions



(a) White to move



(b) White to move

- Often linear combination:
  $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$
- E.g.: $w_1 = 9$ with $f_1(s)$=(number of white queens) - (number of black queens), etc.

Cutting off search

- Next step: replace terminal test in alpha-beta pruning by

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

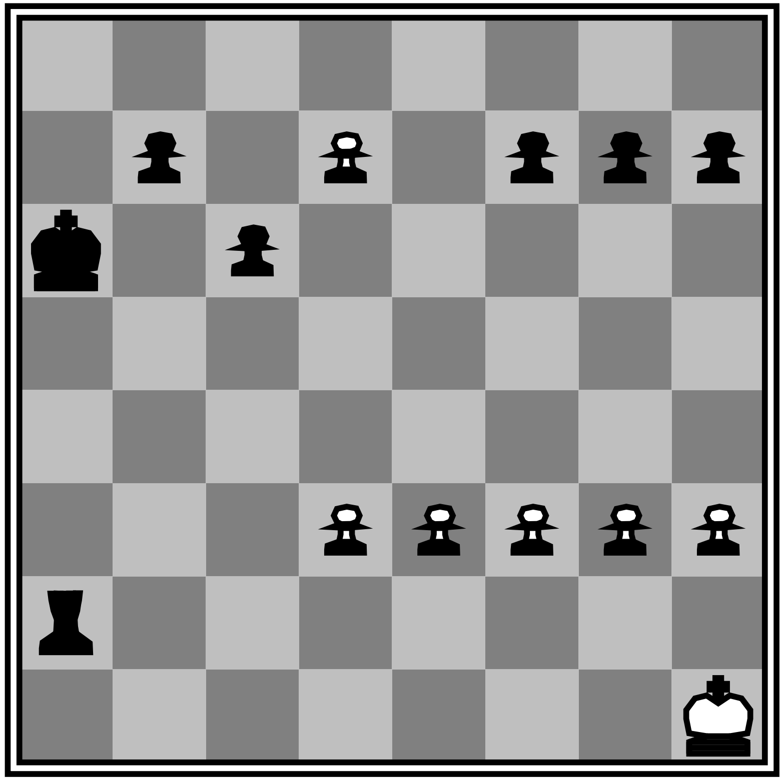
  – E.g., use estimate provided by evaluation function once certain depth has been reached (instead of searching further)

- Problem: abrupt swings in evaluation function
  – Trick: **quiescence search** expands states further that may lead to large changes in the evaluation function

- Problem: **horizon effect**
  – Trick: **singular extensions** (explore moves further that are "clearly" better)

- Further speedup by **forward pruning**: some moves are pruned immediately without further consideration (dangerous)

The horizon effect



Black to move

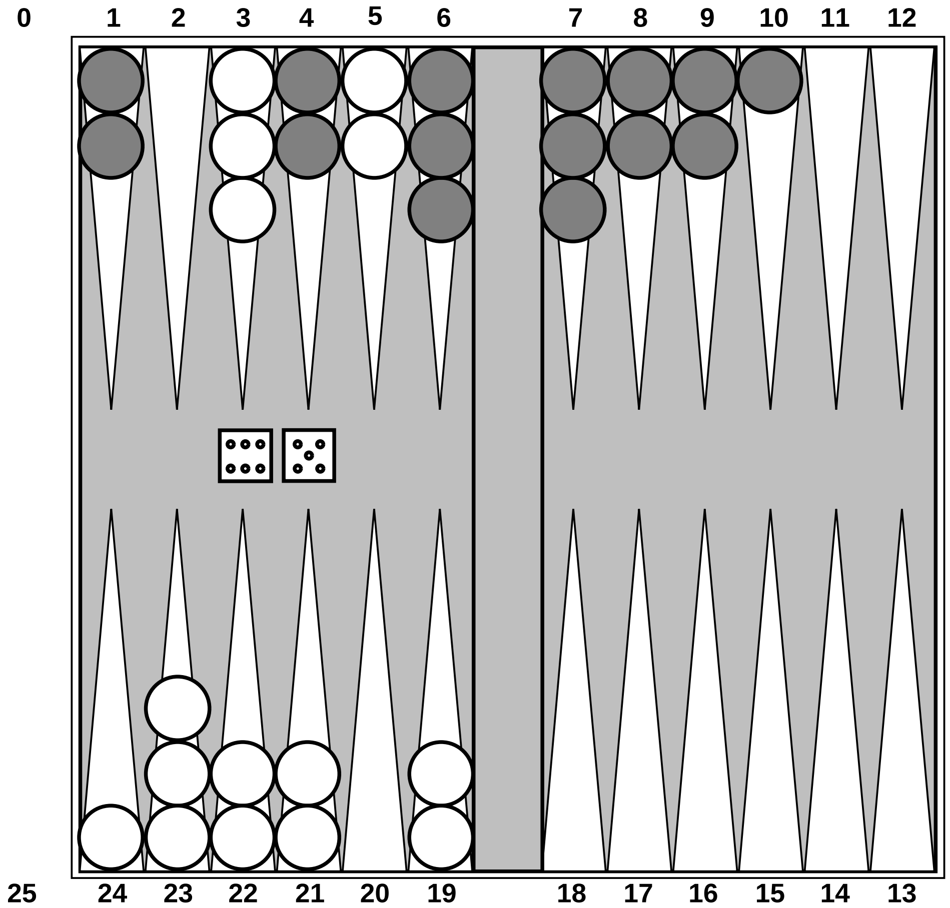
Games including element of chance

- Example: backgammon
- Game tree includes **chance nodes**
- Need to replace **minimax value** by **expectiminimax value**
- Gives perfect play

$$EXPECTIMINIMAX(n) =$$

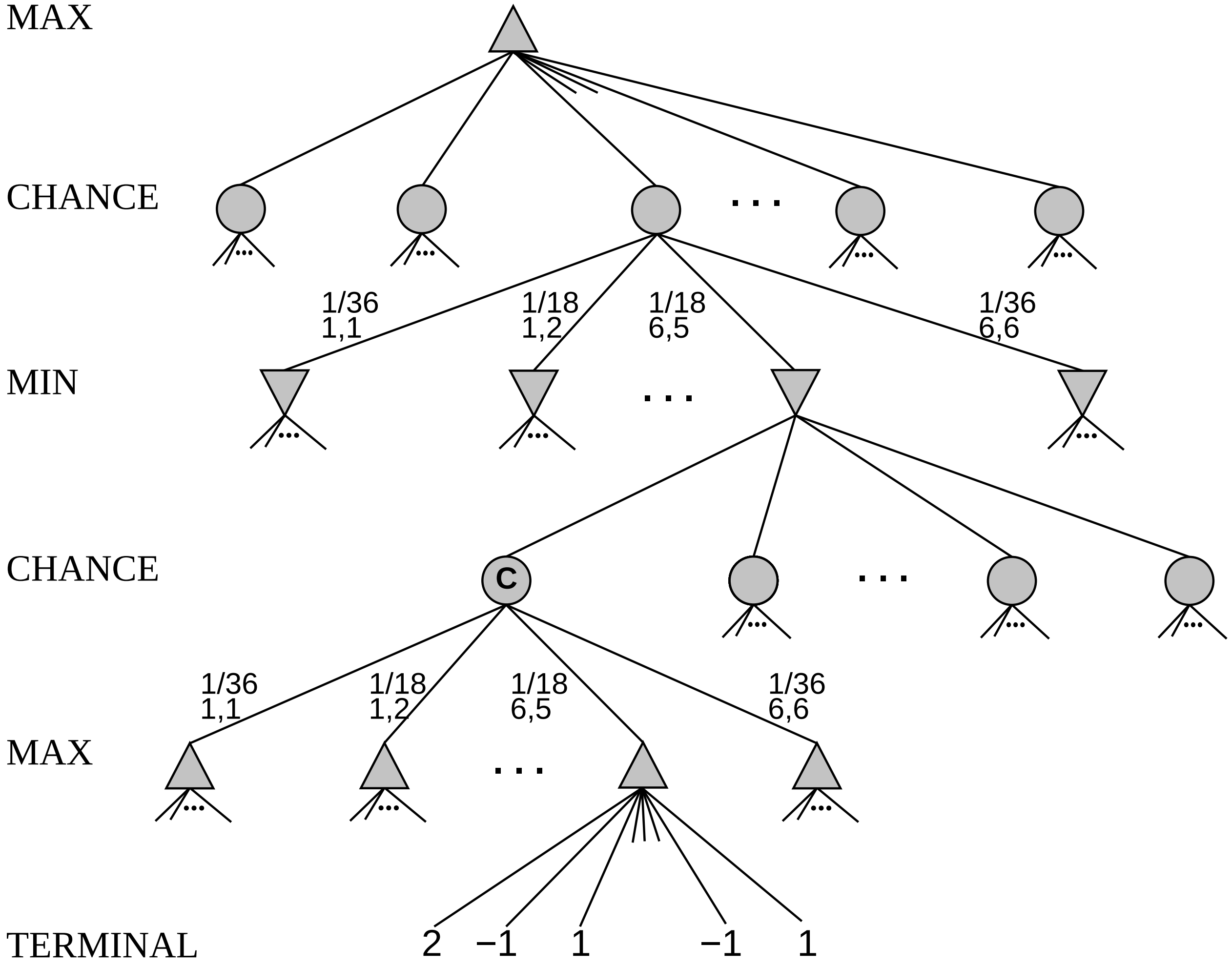
$$\begin{cases} Utility(n) & \text{if } n \text{ is a terminal state} \\ max_{s \in Successors(n)} EXPECTIMINIMAX(s) & \text{if } n \text{ is a MAX node} \\ min_{s \in Successors(n)} EXPECTIMINIMAX(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in Successors(n)} P(s) \times EXPECTIMINIMAX(s) & \text{if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

A backgammon position

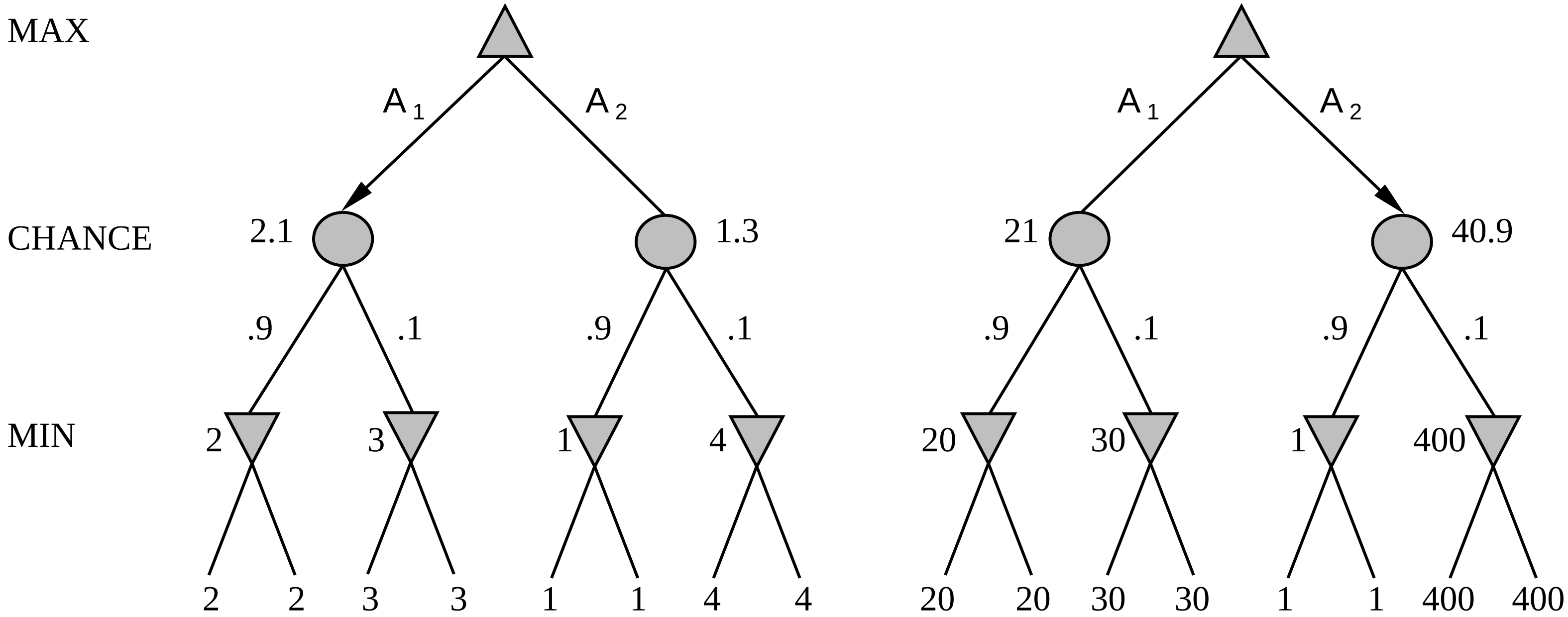


- Legal moves: (5-10, 5-11), (5-11, 19-24), (5-10, 10-16), and (5-11, 11-16)

Schematic game tree for a backgammon position



Position evaluation in games with chance nodes



Complexity of EXPECTIMINIMAX

- Has to consider all possible dice rolls: time complexity $O(b^m n^m)$, where n is the number of distinct rolls
- I.e. chance factor introduces huge extra cost
- It is possible to adapt alpha-beta pruning to game trees with chance nodes
  - Treatment of MAX and MIN nodes remains the same
  - Pruning decision for chance node can be made by computing upper bound on expected value
  - Requirement: bounds on possible values of utility function (e.g. all values are between +3 and -3)
  - Why? Otherwise we would need to explore all successors because average could be anything

State-of-the-art game programs (I)

- Chess: *Deep Blue*
  - Parallel computer with 30 standard processors, and 480 custom chess processors
  - Often reaches depth 14 of the search tree
  - Uses iterative-deepening alpha-beta search, evaluation function with 8,000 features, opening book, endgame database
- Checkers: *Chinook*
  - Uses alpha-beta search, pre-computed database of 444 billion positions with eight or fewer pieces on board
  - Developer Schaeffer believes checkers could be solved completely by enlarging endgame database

State-of-the-art game programs (II)

- Othello: *Logistello*
  - Smaller search space than chess (usually 5 to 15 legal moves)
  - Defeated human world champion 6 games to none
- Backgammon: *TD-GAMMON*
  - Learns evaluation function using reinforcement learning with neural network techniques
  - Search depth 2 or 3
  - Ranked among top 3 players in world after playing a million training games against itself

State-of-the-art game programs (III)

- Go: *Goemate* and *Go4++*
  - Ranked as weak amateurs
  - Branching factor starts at 361 (board is 19×19)!
  - Programs use pattern recognition with limited search
- Bridge: *GIB*
  - Bridge is multiplayer game of imperfect information
  - GIB averages over **belief states**, taking a random sample of 100 arrangements (there can be 10 million)
  - Uses **explanation-based generalization** to compute and cache rules for optimum play
  - Came 12th in one contest at the 1998 human world championship