Foundations of Artificial Intelligence

G2. Board Games: Minimax Search and Evaluation Functions

Malte Helmert

University of Basel

May 14, 2025

Foundations of Artificial Intelligence

May 14, 2025 — G2. Board Games: Minimax Search and Evaluation Functions

G2.1 Minimax Search

G2.2 Evaluation Functions

G2.3 Summary

Board Games: Overview

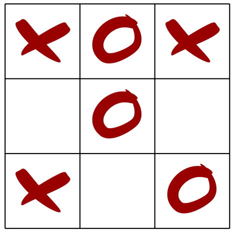
chapter overview:

- G1. Introduction and State of the Art
- G2. Minimax Search and Evaluation Functions
- G3. Alpha-Beta Search
- G4. Stochastic Games
- G5. Monte-Carlo Tree Search Framework
- G6. Monte-Carlo Tree Search Variants

G2.1 Minimax Search

Example: Tic-Tac-Toe

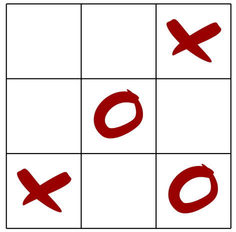
consider it's the turn of player ✗:



If the utility for win/draw/lose for player ✗ is +1/0/-1,
what is an appropriate utility value for the depicted position?

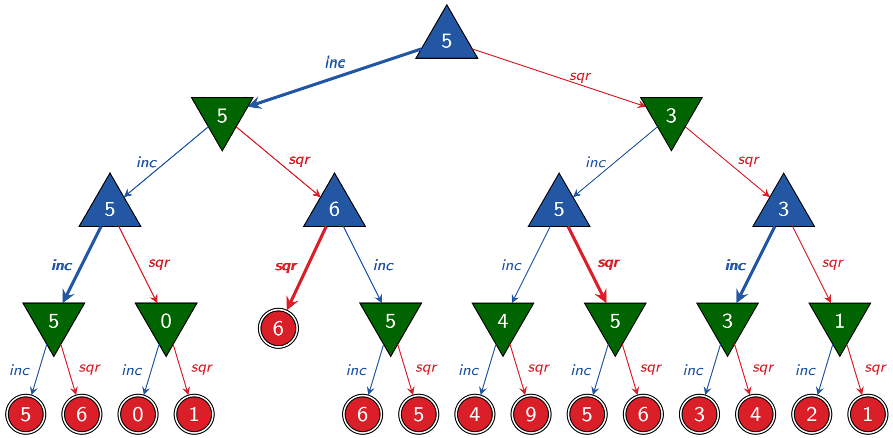
Example: Tic-Tac-Toe

consider it's the turn of player ✗:



And what about this one?

Idea and Example



- depth-first search in game tree
- determine utility value of terminal position with utility function
- strategy: action that maximizes utility value (minimax decision)
- compute utility value of inner nodes from below to above through the tree:
  - MIN's turn: utility value is minimum of utility values of children
  - MAX's turn: utility value is maximum of utility values of children

Minimax: Pseudo-Code

```
function minimax(p)
if p is terminal position:
    return ⟨utility(p), none⟩
best_move := none
if player(p) = MAX:
    v := −∞
else:
    v := ∞
for each ⟨move, p′⟩ ∈ succ(p):
    ⟨v′, best_move′⟩ := minimax(p′)
    if (player(p) = MAX and v′ > v) or
       (player(p) = MIN and v′ < v):
        v := v′
        best_move := move
return ⟨v, best_move⟩
```

Discussion

- minimax is the simplest (decent) search algorithm for games
- yields optimal strategy (in the game-theoretic sense, i.e., under the assumption that the opponent plays perfectly)
- MAX obtains at least the utility value computed for the root, no matter how MIN plays
- if MIN plays perfectly, MAX obtains exactly the computed value

Limitations of Minimax

What if the size of the game tree is too big for minimax?

$\rightsquigarrow$ heuristic alpha-beta search

- heuristics (evaluation functions): rest of this chapter
- alpha-beta search: next chapter

G2.2 Evaluation Functions

Evaluation Functions

Definition (evaluation function)

Let $\mathcal{S}$ be a game with set of positions S.
An evaluation function for $\mathcal{S}$ is a function

$$h : S \to \mathbb{R}$$

which assigns a real-valued number to each position $s \in S$.

Looks familiar? Commonalities? Differences?

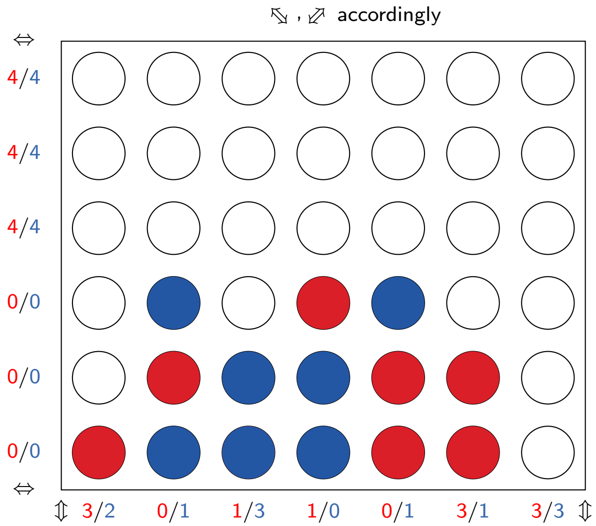
Intuition

- problem: game tree too big
- idea: search only up to predefined depth
- depth reached: estimate the utility value according to heuristic criteria (as if terminal position had been reached)

accuracy of evaluation function is crucial

- high values should relate to high "winning chances"
- at the same time, the evaluation should be efficiently computable in order to be able to search deeply

Example: Connect Four



evalution function: difference of number of possible lines of four

General Method: Linear Evaluation Functions

expert knowledge often represented with weighted linear functions:

$$h(s) = w_0 + w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s),$$

where w_i are weights and f_i are features.

- assumes that feature contributions are mutually independent (usually wrong but acceptable assumption)
- features are (usually) provided by human experts
- weights provided by human experts or learned automatically

General Method: Linear Evaluation Functions

expert knowledge often represented with weighted linear functions:

$$h(s) = w_0 + w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s),$$

where w_i are weights and f_i are features.

example: evaluation function in chess (cf. Lolli 1763)

feature	f_p^{player}	f_k^{player}	f_b^{player}	f_r^{player}	f_q^{player}
no. of pieces	pawn	knight	bishop	rook	queen
weight for MAX	1	3	3	5	9
weight for MIN	−1	−3	−3	−5	−9

often additional features based on pawn structure, mobility, ...

$$\rightsquigarrow h(s) = f_p^{\mathrm{MAX}}(s) + 3f_k^{\mathrm{MAX}}(s) + 3f_b^{\mathrm{MAX}}(s) + 5f_r^{\mathrm{MAX}}(s) + 9f_q^{\mathrm{MAX}}(s)$$
$$- f_p^{\mathrm{MIN}}(s) - 3f_k^{\mathrm{MIN}}(s) - 3f_b^{\mathrm{MIN}}(s) - 5f_r^{\mathrm{MIN}}(s) - 9f_q^{\mathrm{MIN}}(s)$$

General Method: State Value Networks

alternative: evaluation functions based on neural networks

- value network takes position features as input (usually provided by human experts)
- and outputs utility value prediction
- weights of network learned automatically

example: value network of AlphaGo

- start with policy network trained on human expert games
- train sequence of policy networks by self-play against earlier version
- final step: convert to utility value network (slightly worse informed but much faster)

⇝ Mastering the game of Go with deep neural networks and tree search (Silver et al., 2016)

How Deep Shall We Search?

- objective: search as deeply as possible within a given time
- problem: search time difficult to predict
- solution: iterative deepening
  - sequence of searches of increasing depth
  - time expires: return result of previously finished search
  - overhead acceptable (⇝ Chapter B8)
- refinement: search deeper in "turbulent" states (i.e., with strong fluctuations of the evaluation function) ⇝ quiescence search
  - example chess: deepen the search after capturing moves

G2.3 Summary

Summary

- Minimax is a tree search algorithm that plays perfectly (in the game-theoretic sense), but its complexity is $O(b^d)$ (branching factor b, search depth d).
- In practice, the search depth must be bounded ⇝ apply evaluation functions.