

# Grundlagen der Künstlichen Intelligenz

## 41. Brettspiele: Einführung und Minimax-Suche

Malte Helmert

Universität Basel

22. Mai 2015

# Einordnung

Einordnung:

Brettspiele

Umgebung:

- **statisch** vs. **dynamisch**
- **deterministisch** vs. **nicht-deterministisch** vs. **stochastisch**
- **vollständig** vs. **partiell** vs. **nicht beobachtbar**
- **diskret** vs. **stetig**
- **ein Agent** vs. **mehrere Agenten (Gegenspieler)**

Lösungsansatz:

- **problemspezifisch** vs. **allgemein** vs. **lernend**

# Brettspiele: Überblick

## Kapitelüberblick:

- 41. Einführung und Minimax-Suche
- 42. Alpha-Beta-Suche und Ausblick

Einführung  
●ooooooooo

Minimax-Suche  
oooooooo

Bewertungsfunktionen  
oooo

Zusammenfassung  
oo

# Einführung

# Warum Brettspiele?

Brettspiele sind eines der ältesten Gebiete der KI  
(Shannon, Turing 1950).

- sehr abstrakte Form von Problem, leicht zu formalisieren
- benötigen offensichtlich „Intelligenz“ ([oder?](#))
- Traum von einer intelligenten Maschine, die Schach spielt, ist älter als der elektronische Computer
- vgl. von Kempelens „Schachtürke“ (1769),  
Torres y Quevedos „El Ajedrecista“ (1912)

# Eingrenzung

Wir betrachten Brettspiele mit folgenden Eigenschaften:

- aktuelle Situation durch **endliche Menge** von **Positionen**  
(= Zuständen) repräsentierbar
- Situationsänderungen durch **endliche Menge** von **Zügen**  
(= Aktionen) repräsentierbar
- es gibt **zwei Spieler**, von denen in jeder Position
  - einer **am Zug** ist
  - oder es ist eine **Endposition**
- Endposition haben **Nutzenbewertung**
- Nutzen von Spieler 2 immer Gegenteil von Nutzen von Spieler 1 (**Nullsummenspiel**)
- „endlose“ Spielverläufe gelten als Remis (Nutzen 0)
- kein Zufall, keine geheimen Informationen

# Beispiel: Schach

## Beispiel (Schach)

- Positionen beschrieben durch:
  - Stellung der Figuren
  - Wer ist am Zug?
  - en-passant- und Rochade-Rechte
- Züge gegeben durch Spielregeln
- Endpositionen: Matt- und Patt-Stellungen der beiden Spieler
- Nutzen der Endpositionen aus Sicht des ersten Spielers (Weiss) zum Beispiel:
  - +1 wenn Schwarz matt
  - 0 bei Patt
  - -1 wenn Weiss matt

# Abgrenzungen

Wichtige Klassen von Spielen, die wir **nicht** berücksichtigen:

- mit Zufall (z. B. Backgammon)
- mit mehr als zwei Spielern (z. B. Halma)
- mit verdeckter Information (z. B. Bridge)
- mit gleichzeitigen Zügen (z. B. Diplomacy)
- ohne Nullsummeneigenschaft („Spiele“ aus der Spieltheorie  
~~ Auktionen, Wahlverfahren, Wirtschaft, Politik, . . . )
- . . . und viele weitere Generalisierungen

Viele dieser Spieltypen können mit ähnlichen/erweiterten Algorithmen behandelt werden.

# Formalisierung

Brettspiele gegeben durch **Zustandsräume**

$\mathcal{S} = \langle S, A, cost, T, s_0, S_* \rangle$  mit zwei Erweiterungen

- **Spielerfunktion *player***:  $S \setminus S_* \rightarrow \{1, 2\}$  gibt an, welcher der beiden Spieler am Zug ist
- **Nutzenfunktion *u***:  $S_* \rightarrow \mathbb{R}$  gibt Nutzen (aus Sicht von Spieler 1) in Endpositionen an.

sonstige Änderungen:

- Aktionskosten *cost* werden nicht benötigt

(Wir haben ähnliche Definitionen inzwischen oft gesehen und gehen daher nicht weiter ins Detail.)

# Terminologie

Im Kontext von Brettspielen oft abweichende Begriffe für Dinge, die wir bereits kennen:

- Zustand, Zielzustand, etc.  $\rightsquigarrow$  **Position**, **Endposition** etc.
- Aktion  $\rightsquigarrow$  **Zug**
- Suchbaum  $\rightsquigarrow$  **Spielbaum**

# Spezielle vs. allgemeine Algorithmen

- Wir betrachten hier Verfahren, die für gute Performance auf spezielle Brettspiele **zugeschnitten** werden müssen, z. B. durch Implementierung einer geeigneten **Bewertungsfunktion**.
  - ~ vgl. Kapitel zu informierten Suchverfahren
- analog zur Verallgemeinerung von Suchverfahren auf deklarativ beschriebene Probleme (**Handlungsplanung**) können auch Brettspiele in einem allgemeinen Rahmen betrachtet werden, wo **Spielregeln** (Zustandsräume) **Teil der Eingabe** sind
  - ~ **general game playing**, jährliche Wettbewerbe seit 2005

# Warum sind Brettspiele schwierig?

Ebenso wie klassische Suchprobleme haben (interessante) Brettspiele **astronomisch grosse Zustandsräume**:

- **Schach**: ca.  $10^{40}$  erreichbare Zustände;  
Partie mit 50 Zügen/Spieler und Verzweigungsgrad 35:  
Baumgrösse ca.  $35^{100} \approx 10^{154}$
- **Go**: mehr als  $10^{100}$  Zustände;  
Partie mit ca. 300 Zügen, Verzweigungsgrad ca. 200:  
Baumgrösse ca.  $200^{300} \approx 10^{690}$

# Warum sind Brettspiele schwierig?

Ebenso wie klassische Suchprobleme haben (interessante) Brettspiele **astronomisch grosse Zustandsräume**:

- **Schach**: ca.  $10^{40}$  erreichbare Zustände;  
Partie mit 50 Zügen/Spieler und Verzweigungsgrad 35:  
Baumgrösse ca.  $35^{100} \approx 10^{154}$
- **Go**: mehr als  $10^{100}$  Zustände;  
Partie mit ca. 300 Zügen, Verzweigungsgrad ca. 200:  
Baumgrösse ca.  $200^{300} \approx 10^{690}$

Dazu kommt, dass es nicht mehr reicht,  
einen Lösungspfad zu finden:

- benötigt wird eine **Strategie**, die auf alle möglichen Verhaltensweisen des Gegners reagiert
- üblicherweise implementiert als Algorithmus,  
der „on demand“ den nächsten Zug liefert

# Algorithmen für Brettspiele

Gute Algorithmen für Brettspiele:

- sehen **möglichst weit voraus** (tiefe Suche)
- betrachten nur **interessante Teile** des Spielbaums  
(selektive Suche, analog zu heuristischen Suchverfahren)
- nehmen **möglichst genaue Bewertung** von Positionen vor  
(Evaluationsfunktionen, analog zu Heuristiken)

Einführung  
oooooooooo

Minimax-Suche  
●oooooooo

Bewertungsfunktionen  
oooo

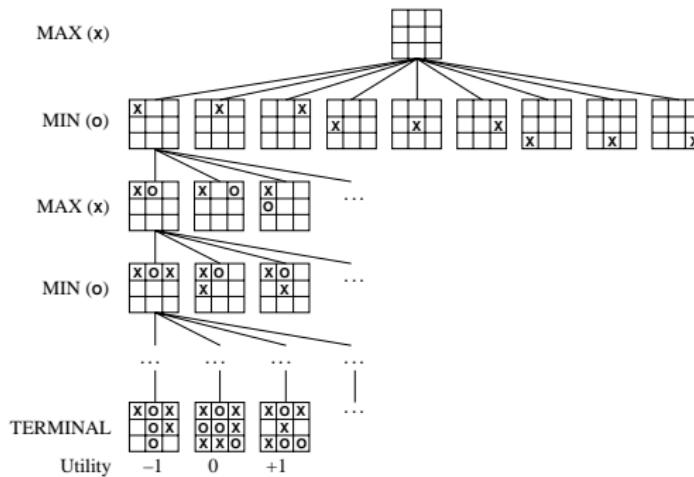
Zusammenfassung  
oo

# Minimax-Suche

# Terminologie für Zwei-Personen-Spiele

- **Spieler** werden traditionell **MAX** und **MIN** genannt.
- Wir wollen Züge für MAX berechnen  
(MIN ist der Gegner).
- MAX versucht seinen Nutzen in der erreichten Endposition  
(gegeben durch die Funktion  $u$ ) zu **maximieren**.
- MIN versucht  $u$  zu **minimieren** (was MINs Nutzen maximiert)

## Beispiel: Tic-Tac-Toe

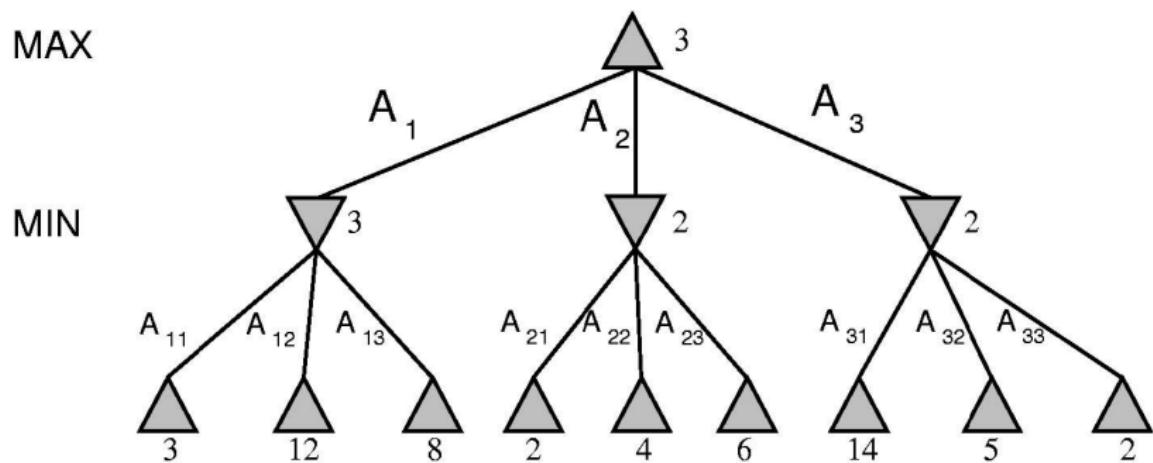


- **Spielbaum** mit Spieler am Zug (MAX/MIN) links markiert
- in letzter Reihe **Endpositionen** mit ihrem **Nutzen**
- Grösse des Spielbaums?

# Minimax: Berechnung

1. **Tiefensuche** durch den Spielbaum
2. Wende Nutzenfunktion auf Endpositionen an.
3. Von unten nach oben durch den Baum berechne Nutzen von inneren Knoten wie folgt:
  - MIN ist am Zug:  
Nutzen ist **Minimum** der Nutzenwerte der Kinder
  - MAX ist am Zug:  
Nutzen ist **Maximum** der Nutzenwerte der Kinder
4. Zugauswahl für MAX in der Wurzel:  
wähle einen Zug, der den berechneten Nutzenwert maximiert  
**(Minimax-Entscheidung)**

## Minimax: Beispiel



# Minimax: Diskussion

- **Minimax** ist der einfachste (brauchbare) Spielsuchalgorithmus
- Führt zu optimaler Strategie\* (im Sinne der Spieltheorie, d. h. unter Annahme perfekter Gegenwehr), ist aber für komplexe Spiele zu zeitaufwändig.
- Egal, wie der Gegner spielt, wird **mindestens** der für die Wurzel berechnete Nutzenwert erreicht.
- Spielt der Gegner perfekt, wird **genau** dieser Wert erreicht.

(\*) bei Spielen, die nicht in Zyklen geraten können;  
ansonsten wird es komplizierter (da der Baum unendlich wird)

# Minimax: Pseudo-Code

(geht von alternierender Spielerreihenfolge aus)

```
function MINIMAX-DECISION(state) returns an action
  return arg maxa ∈ ACTIONS(s) MIN-VALUE(RESULT(s, a))

function MAX-VALUE(s) returns a utility value
  if TERMINAL-TEST(s) then return UTILITY(s)
  v ← −∞
  for each a in ACTIONS(s) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(s) returns a utility value
  if TERMINAL-TEST(s) then return UTILITY(s)
  v ← ∞
  for each a in ACTIONS(s) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Was, wenn der Spielbaum zu gross für Minimax ist?  
~~ approximieren durch **Bewertungsfunktionen**

Einführung  
oooooooooo

Minimax-Suche  
ooooooo

Bewertungsfunktionen  
●ooo

Zusammenfassung  
oo

# Bewertungsfunktionen

# Bewertungsfunktionen

- **Problem:** Spielbaum zu gross
- **Idee:** suche nur bis zu einer bestimmten Tiefe
- wenn diese Tiefe erreicht ist, **schätze** den Nutzen anhand **heuristischer Kriterien** (als wäre eine Endposition erreicht)

## Beispiel (Bewertungsfunktion in Schach)

- **Material:** Bauer 1, Springer 3, Läufer 3, Turm 5, Dame 9  
positives Vorzeichen für Figuren von MAX, negatives bei MIN
- **Bauernstruktur, Mobilität, ...**

Daumenregel: 3-Punkte-Vorteil  $\rightsquigarrow$  sicherer Sieg

**Gute Bewertungsfunktionen sind entscheidend!**

- Hohe Werte sollten hohen „Gewinnchancen“ entsprechen, damit Verfahren gut funktioniert.
- Gleichzeitig sollte Bewertung schnell berechnet werden, um tief suchen zu können.

# Lineare Bewertungsfunktionen

Am häufigsten werden **gewichtete lineare Funktionen** verwendet:

$$w_1 f_1 + w_2 f_2 + \cdots + w_n f_n$$

wobei die  $w_i$  **Gewichte** und die  $f_i$  **Features** sind.

- enthält Annahme, dass Beiträge der Features **unabhängig** sind (normalerweise falsch, aber vertretbar)
- erlaubt effiziente **inkrementelle Berechnung**, wenn Features sich nicht in jedem Zug ändern
- Gewichte können automatisch gelernt werden
- Features stammen (in der Regel) von menschlichen Experten

# Wie tief suchen?

- **Ziel:** In gegebener Bedenkzeit möglichst tief suchen
- **Problem:** Suchzeit schwer vorherzusehen
- **Lösung: iteratives Vertiefen**
  - Abfolge von Suchen, die immer tiefer gehen
  - Zeit läuft ab: liefere Ergebnis letzter abgeschlossener Suche

# Wie tief suchen?

- **Ziel:** In gegebener Bedenkzeit möglichst tief suchen
- **Problem:** Suchzeit schwer vorherzusehen
- **Lösung: iteratives Vertiefen**
  - Abfolge von Suchen, die immer tiefer gehen
  - Zeit läuft ab: liefere Ergebnis letzter abgeschlossener Suche
- **Verfeinerung:** Suchtiefe nicht uniform, sondern tiefer in „unruhigen“ Positionen (mit grossen Schwankungen der Bewertungsfunktion) ↵ **quiescence search**
  - **Beispiel Schach:** Suche vertiefen, wenn Figurentausch begonnen, aber nicht abgeschlossen wurde

Einführung  
oooooooooo

Minimax-Suche  
ooooooo

Bewertungsfunktionen  
oooo

Zusammenfassung  
●○

# Zusammenfassung

# Zusammenfassung

- **Brettspiele** können verstanden werden als Erweiterung von klassischen Suchproblemen um einen **Gegenspieler**.
- Beide Spieler versuchen eine Endposition mit (für sie) **maximalem Nutzen** zu erreichen.
- **Minimax** ist ein Baumsuchalgorithmus, der perfekt spielt (im Sinne der Spieltheorie), aber Aufwand  $O(b^d)$  hat (Verzweigungsgrad  $b$ , Suchtiefe  $d$ )
- in der Praxis muss Suchtiefe oft begrenzt werden; dann Anwendung von **Bewertungsfunktionen** (meist Linearkombinationen von Features)