Grundlagen der Künstlichen Intelligenz

18. Brettspiele

Malte Helmert

Universität Basel

24. Mai 2013

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

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:
  - $+100$ wenn Schwarz matt
  - 0 bei Patt
  - -100 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 $\rightsquigarrow$ Auktionen, Wahlverfahren, Wirtschaft, Politik, . . .)
- . . . und viele weitere Generalisierungen

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

Formalisierung

Brettspiele gegeben durch Zustandsräume
$\mathcal{S} = \langle S, A, \mathit{cost}, T, s_0, S_\star \rangle$ mit zwei Erweiterungen

- Spielerfunktion $\mathit{player} : S \setminus S_\star \to \{1, 2\}$ gibt an, welcher der beiden Spieler am Zug ist
- Nutzenfunktion $u : S_\star \to \mathbb{R}$ 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:

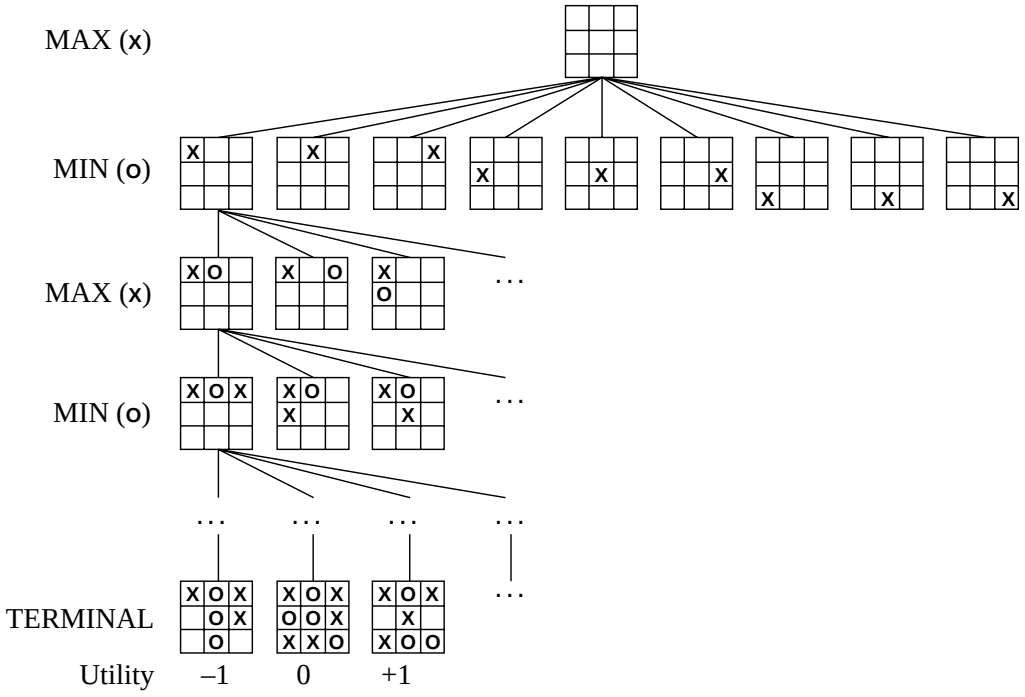
- sehen möglichst weit voraus (tiefe Suche)
- betrachten nur interessante Teile des Spielbaums (selektive Suche, analog zu heuristischen Suchverfahren)
- nehmen möglichst genaue Bewertung von Positionen vor (Evaluationsfunktionen, analog zu Heuristiken)

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

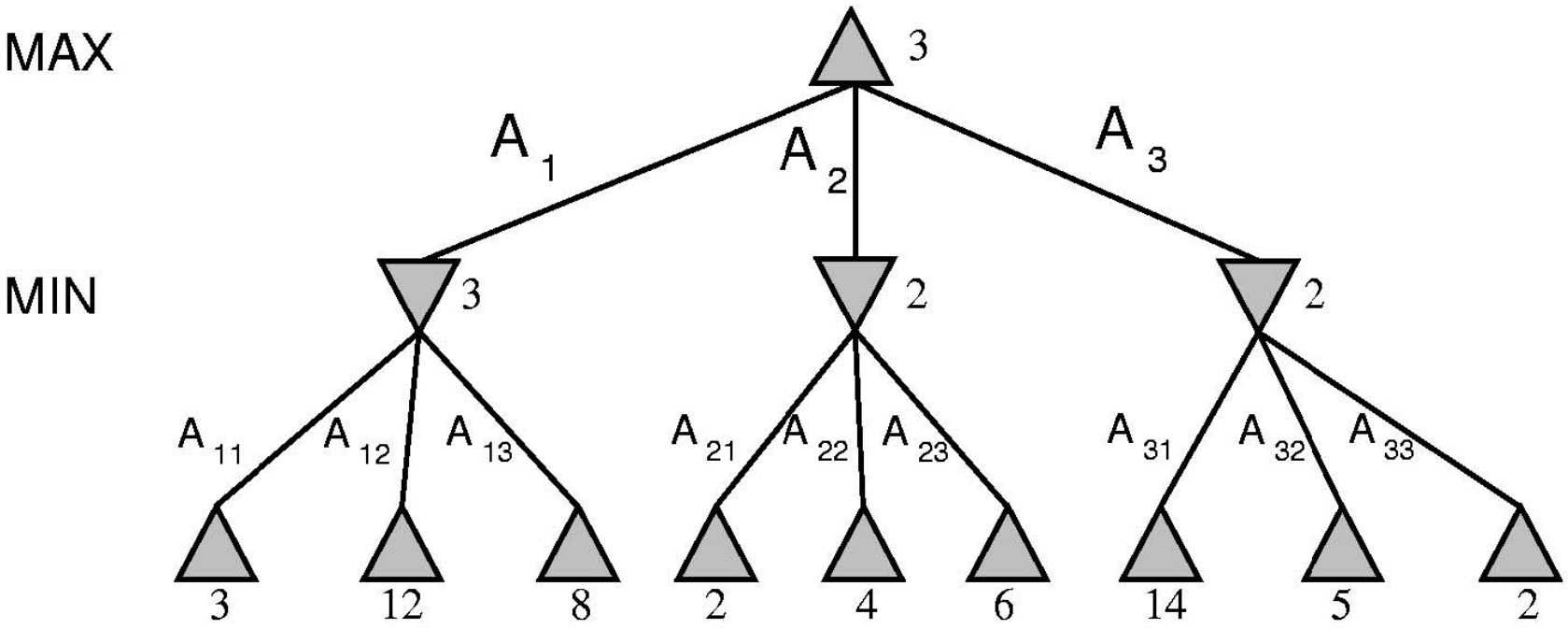


- Spielbaum mit Spieler am Zug (MAX/MIN) links markiert
- in letzter Reihe Endpositionen mit ihrem Nutzen
- Grösse des Spielbaums?

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 $\arg\max_{a \in \text{ACTIONS}(s)}$ 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*

Was, wenn der Spielbaum zu gross für Minimax ist?
$\rightsquigarrow$ approximieren durch Bewertungsfunktionen

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 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?

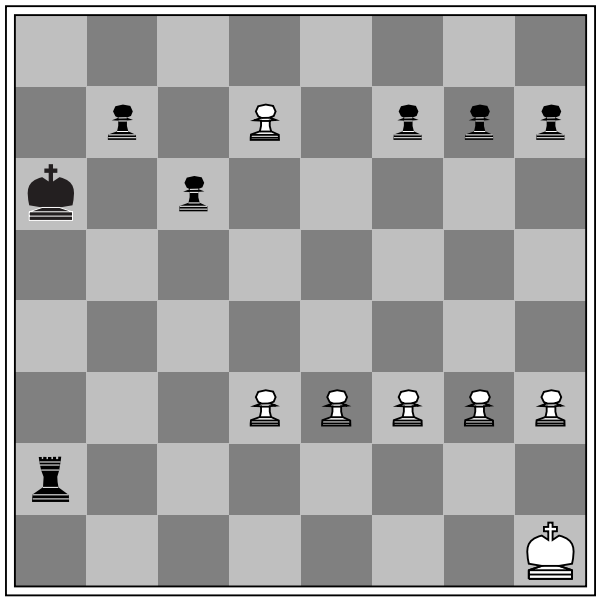
- Ziel: In gegebener Bedenkzeit möglichst tief suchen
- Problem: Suchzeit schwer vorherzusehen
- Lösung: iteratives Vertiefen
  - Abfolge von Suchen, die immer tiefer gehen
  - Zeit läuft ab: liefere Ergebnis letzter abgeschlossener Suche

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) $\rightsquigarrow$ quiescence search
  - Beispiel Schach: Suche vertiefen, wenn Figurentausch begonnen, aber nicht abgeschlossen wurde

Problem bei begrenzter Suchtiefe: Horizont-Problem

Problem: tiefenbeschränkte Suche kann in manchen Fällen kritische Aspekte „hinter den Suchhorizont schieben“

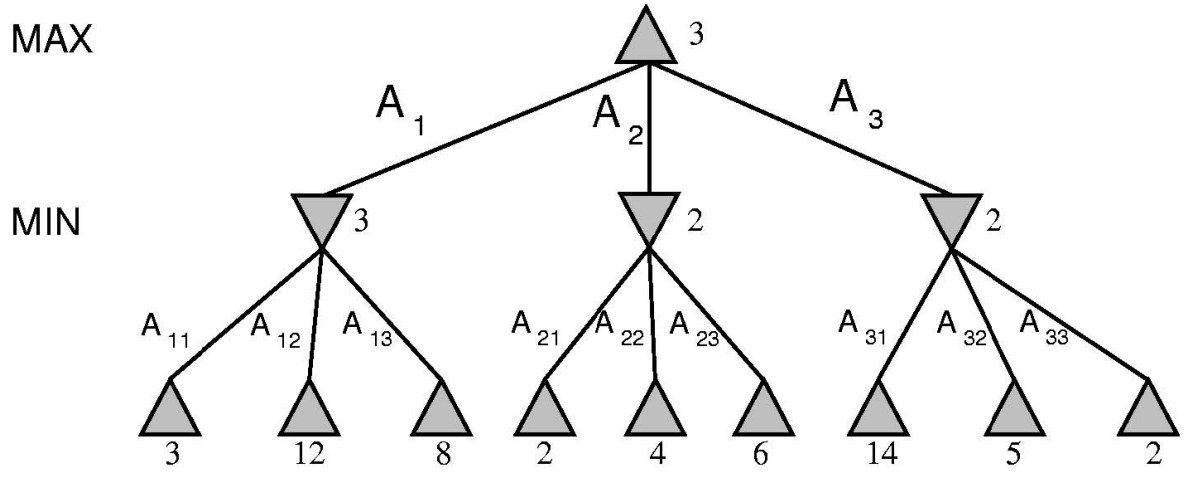


Black to move

- Schwarz hat Materialvorteil
- . . . wird aber verlieren (weisser Bauer wird umgewandelt)
- Suche, die nicht extrem tief ist, übersieht dies, da Schwarz vorher oft Schach bieten kann

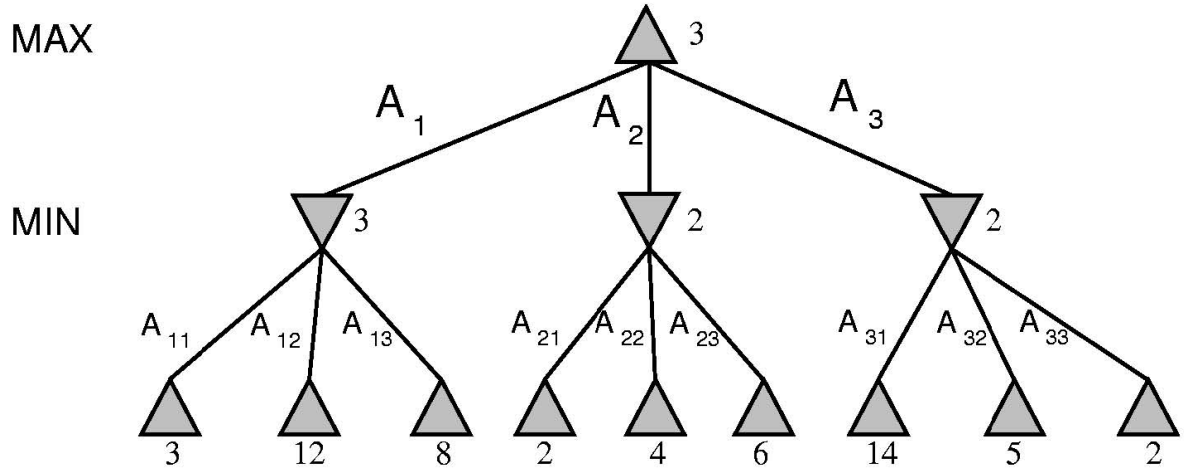
Alpha-Beta-Suche

Alpha-Beta-Suche



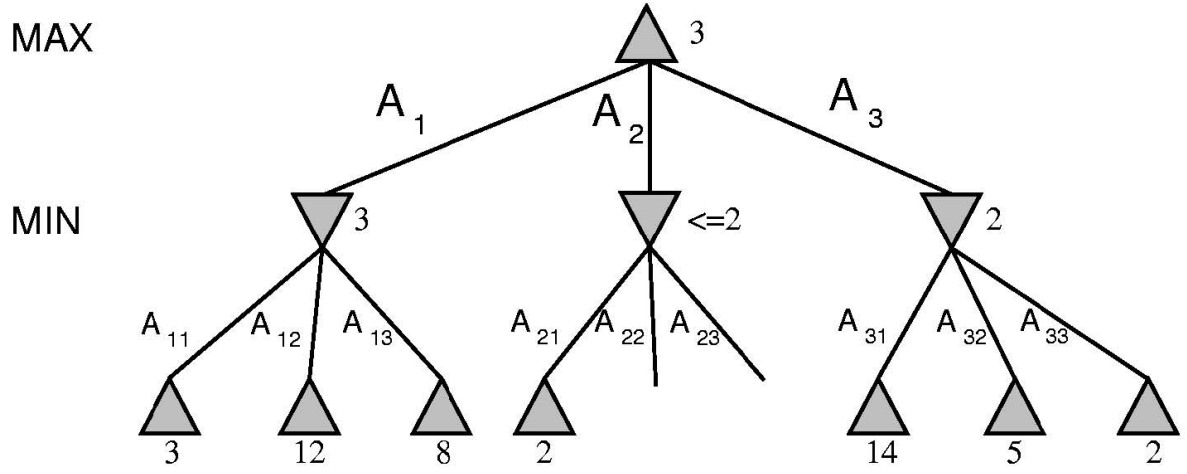
Kann man Sucharbeit sparen?

Alpha-Beta-Suche

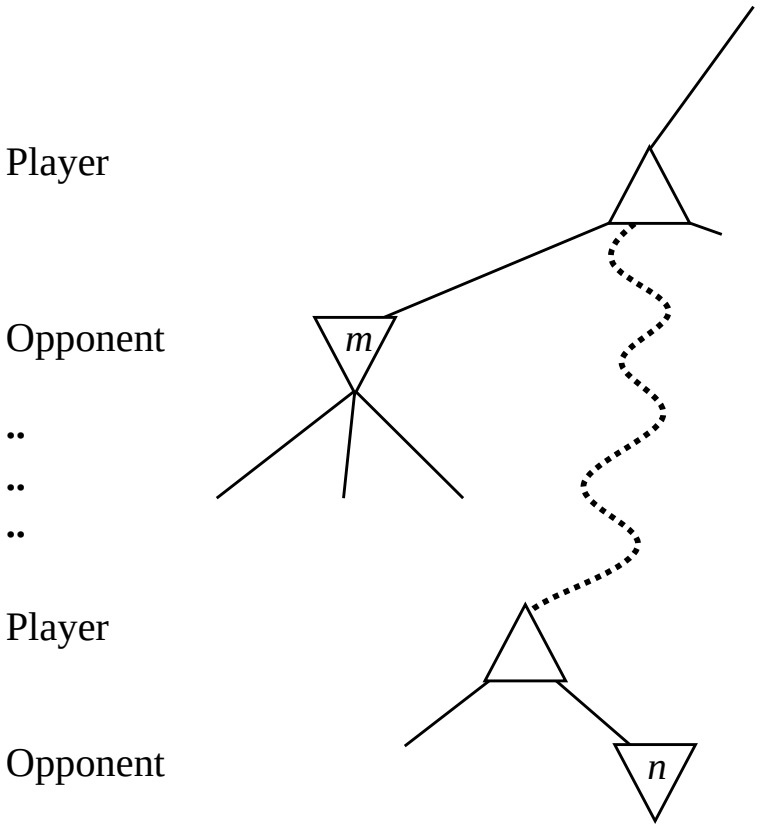


Kann man Sucharbeit sparen?

Nicht alle Knoten müssen betrachtet werden!



Alpha-Beta-Suche: allgemein



Falls $m > n$ gilt, wird Knoten mit Nutzen in n bei perfektem Spiel nie erreicht!

Alpha-Beta-Suche: Idee

Idee: Führe in Minimax-Tiefensuche zwei Zahlenwerte α und β mit, so dass für jeden rekursiven Aufruf gilt:

Alpha-Beta-Suche: Idee

Idee: Führe in Minimax-Tiefensuche zwei Zahlenwerte α und β mit, so dass für jeden rekursiven Aufruf gilt:

- Wenn der Nutzen im aktuellen Teilbaum $\leq \alpha$ ist, interessiert er uns nicht, weil MAX dann bei perfektem Spiel diesen Teilbaum nicht betreten muss
- Wenn der Nutzen im aktuellen Teilbaum $\geq \beta$ ist, interessiert er uns nicht, weil MIN dann bei perfektem Spiel diesen Teilbaum nicht betreten muss

Alpha-Beta-Suche: Idee

Idee: Führe in Minimax-Tiefensuche zwei Zahlenwerte α und β mit, so dass für jeden rekursiven Aufruf gilt:

- Wenn der Nutzen im aktuellen Teilbaum $\leq \alpha$ ist, interessiert er uns nicht, weil MAX dann bei perfektem Spiel diesen Teilbaum nicht betreten muss
- Wenn der Nutzen im aktuellen Teilbaum $\geq \beta$ ist, interessiert er uns nicht, weil MIN dann bei perfektem Spiel diesen Teilbaum nicht betreten muss

Wenn $\alpha \geq \beta$ im Teilbaum gilt, ist der Teilbaum uninteressant und muss nicht weiter durchsucht werden (α-β-Pruning).

Alpha-Beta-Suche: Idee

Idee: Führe in Minimax-Tiefensuche zwei Zahlenwerte α und β mit, so dass für jeden rekursiven Aufruf gilt:

- Wenn der Nutzen im aktuellen Teilbaum $\leq \alpha$ ist, interessiert er uns nicht, weil MAX dann bei perfektem Spiel diesen Teilbaum nicht betreten muss
- Wenn der Nutzen im aktuellen Teilbaum $\geq \beta$ ist, interessiert er uns nicht, weil MIN dann bei perfektem Spiel diesen Teilbaum nicht betreten muss

Wenn $\alpha \geq \beta$ im Teilbaum gilt, ist der Teilbaum uninteressant und muss nicht weiter durchsucht werden (α-β-Pruning).

Wird Alpha-Beta-Suche mit $\alpha = -\infty$ und $\beta = +\infty$ gestartet, ist Ergebnis identisch zu Minimax bei weniger Suchaufwand.

Alpha-Beta-Suche: Pseudo-Code

function ALPHA-BETA-SEARCH($state$) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

---

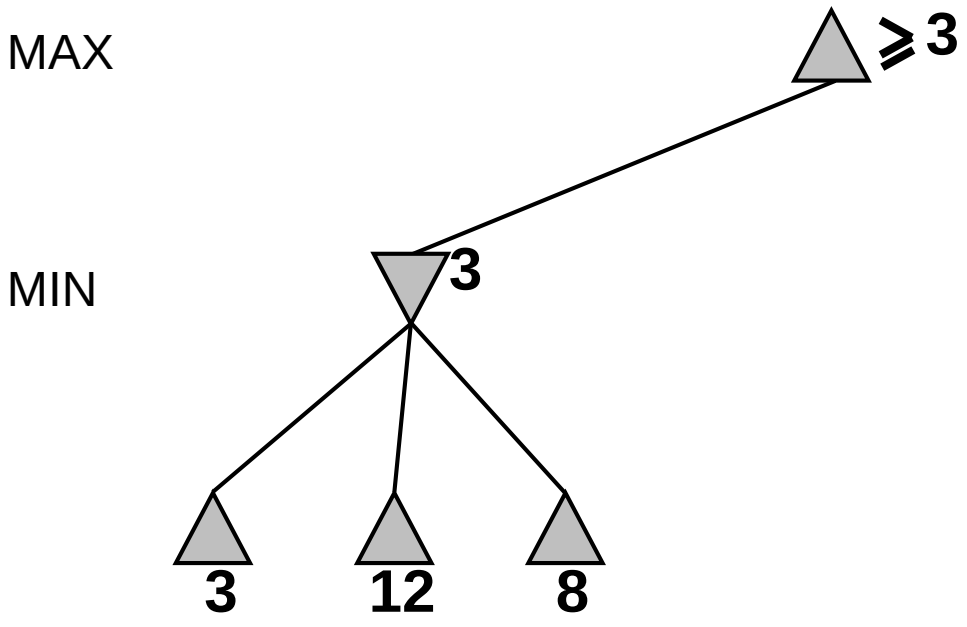
function MAX-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v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

---

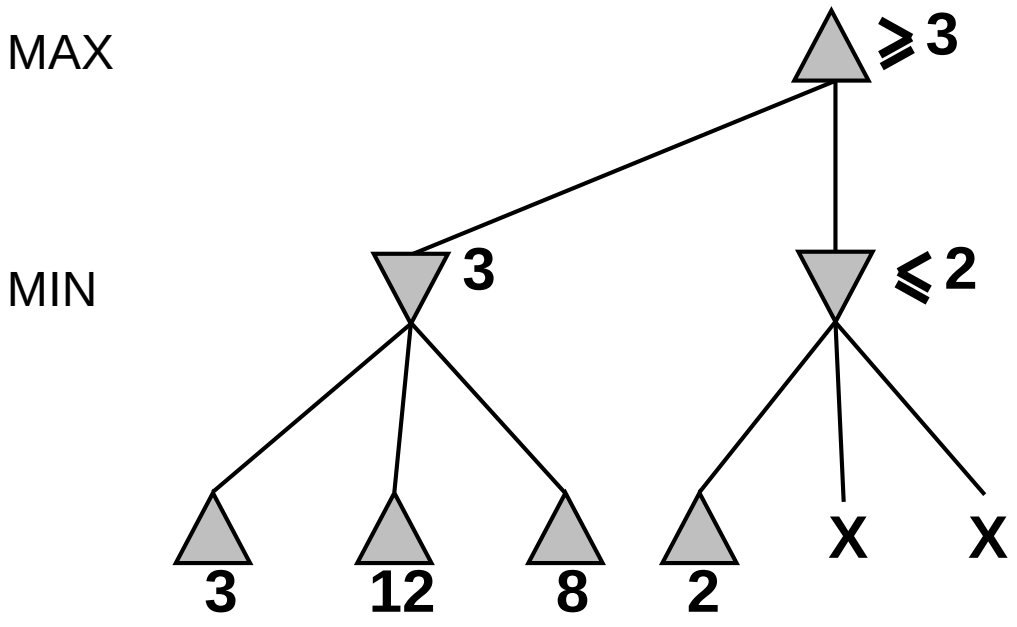
function MIN-VALUE($state, \alpha, \beta$) **returns** a $utility$ $value$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return**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

Anmerkung: bei mehreren identisch bewerteten Zügen muss ALPHA-BETA-SEARCH den ersten Maximierer wählen.

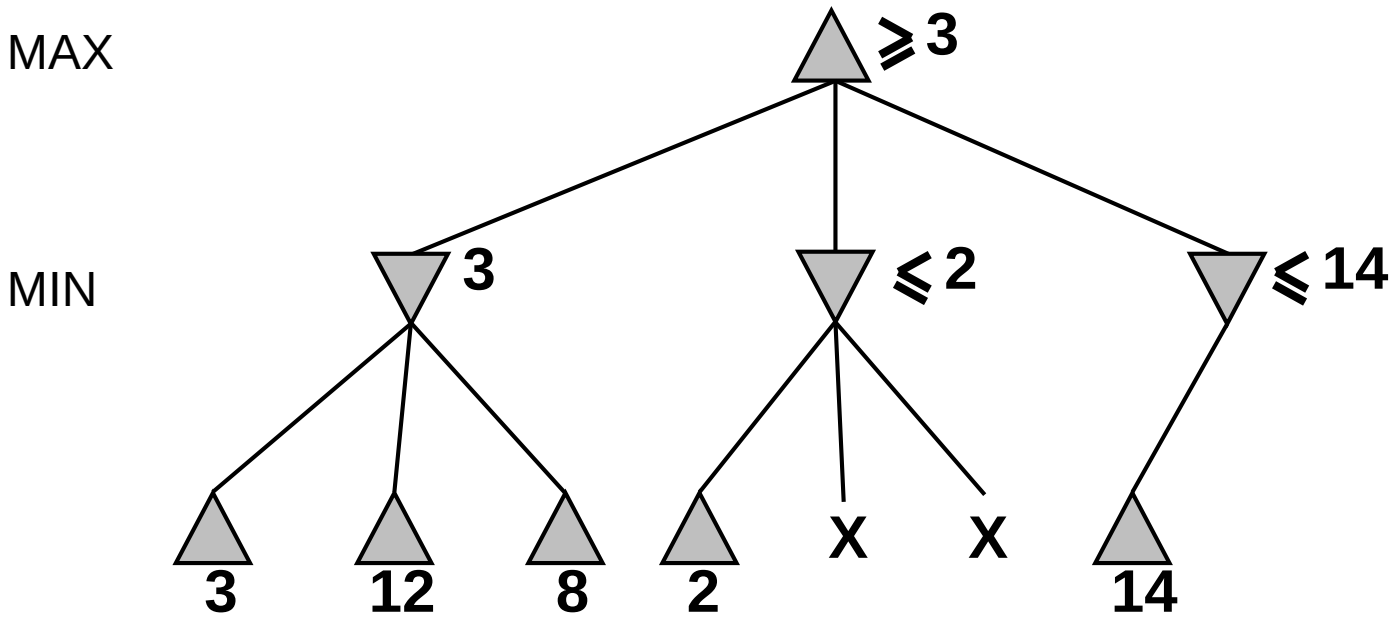
Alpha-Beta-Suche: Beispiel



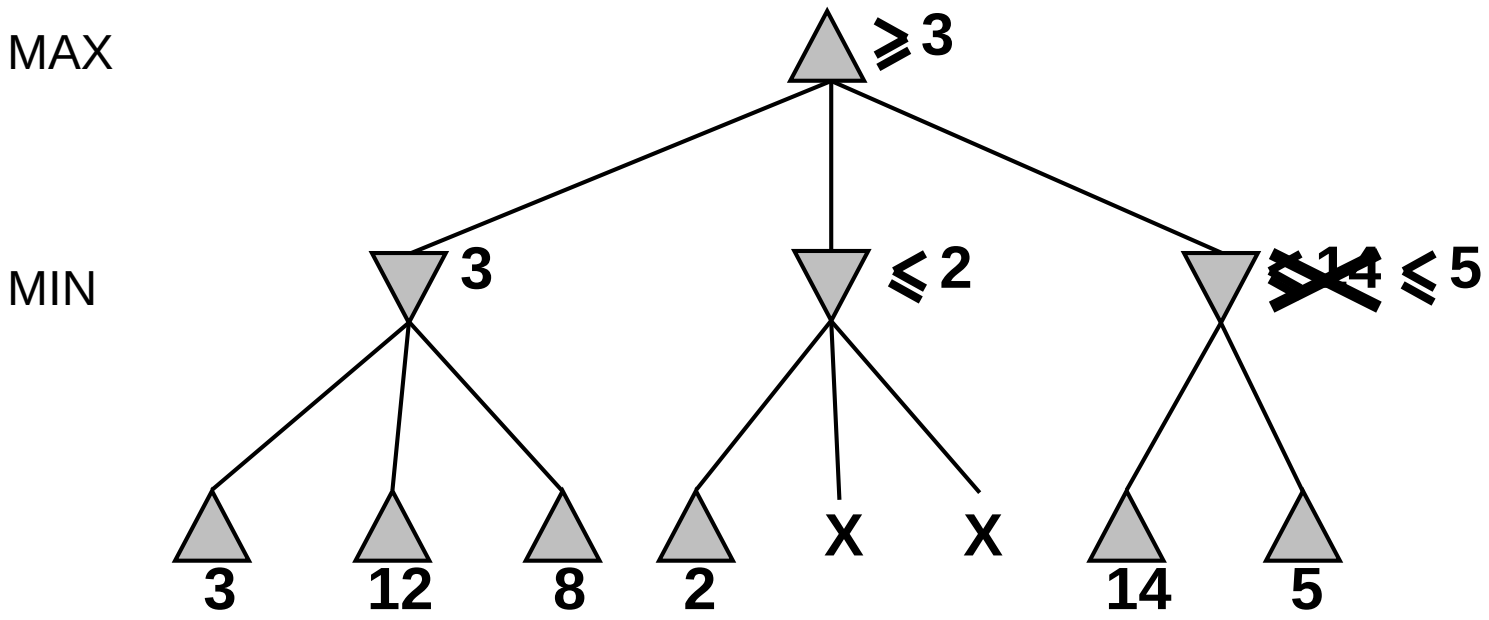
Alpha-Beta-Suche: Beispiel



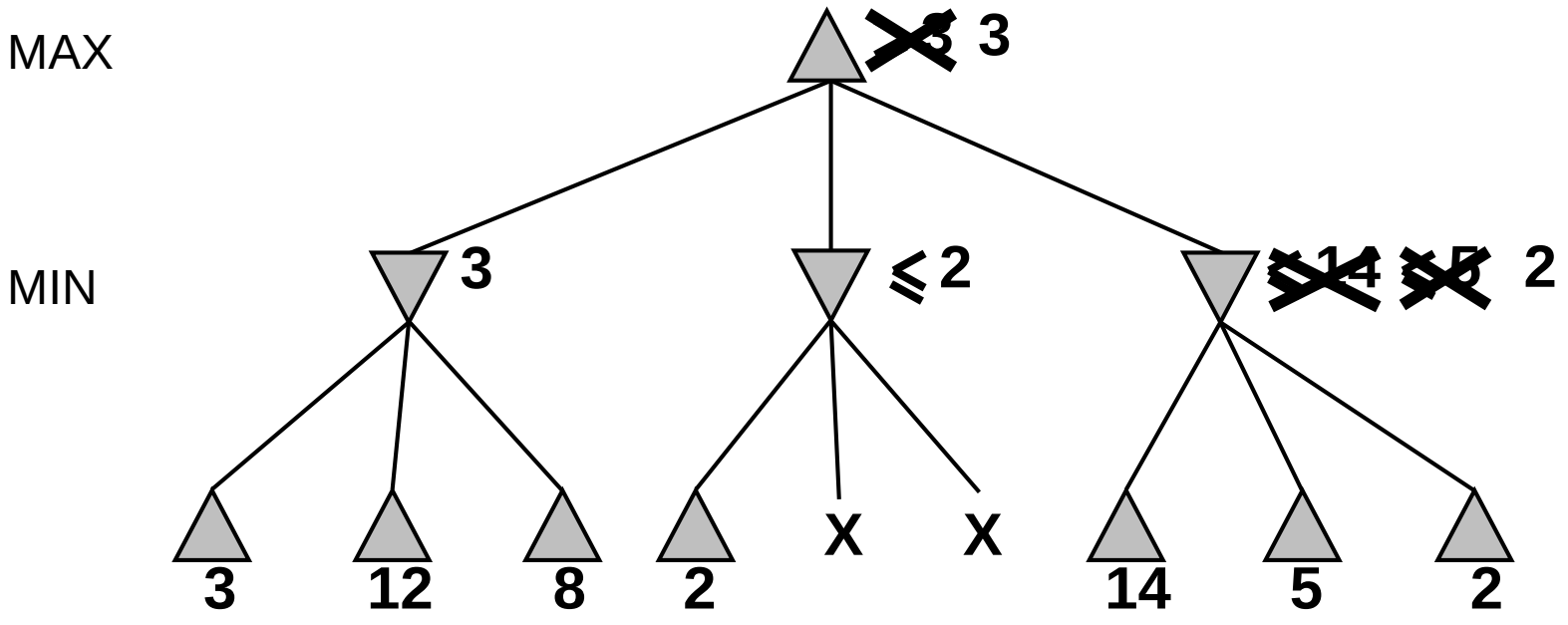
Alpha-Beta-Suche: Beispiel



Alpha-Beta-Suche: Beispiel



Alpha-Beta-Suche: Beispiel



Wie viel bringt Alpha-Beta-Suche?

Annahme: uniformer Spielbaum, Tiefe d, Verzweigungsgrad $b \geq 2$; MAX- und MIN-Positionen immer abwechselnd

- Alpha-Beta-Suche ist am erfolgreichsten, wenn bester Zug für den Spieler am Zug immer als erstes betrachtet wird
  - also ein maximierender Zug bei MAX, ein minimierender Zug bei MIN

$\rightsquigarrow$ Aufwand reduziert sich in diesem besten Fall von $O(b^d)$ (Minimax) auf $O(b^{d/2})$

$\rightsquigarrow$ doppelte Suchtiefe in derselben Zeit möglich

- in der Praxis kommt man oft sehr nah an das Optimum heran

Stand der Wissenschaft

Stand der Wissenschaft

einige bekannte Brettspiele:

- Schach: $\rightsquigarrow$ nächste Folien
- Othello: Logistello besiegt 1997 menschlichen Weltmeister; beste Computer-Spieler deutlich stärker als beste Menschen
- Dame (Checkers): Chinook offizieller Weltmeister (seit 1994); 2007 Unbesiegbarkeit gezeigt (Spiel „gelöst“)
- Go: Die besten Programme (Zen, Mogo, Crazystone) verwenden Monte-Carlo-Techniken (UCT) und sind in etwa auf dem Niveau starker Amateure (1 kyu/1 dan)

Computerschach

Weltmeister Garri Kasparov 1997 durch Deep Blue
in 6 Partien 3,5 : 2,5 besiegt

- spezialisierte Schach-Hardware, 30 Knoten mit je 16 Chips
- Alpha-Beta-Suche (mit Erweiterungen)
- Eröffnungsdatenbank aus Millionen von Schachpartien

Heutzutage spielt normale PC-Hardware auf Weltmeisterniveau.

Computerschach: Zitate

Claude Shannon (1949)

The chess machine is an ideal one to start with, since

1. the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate),
2. it is neither so simple as to be trivial nor too difficult for satisfactory solution,
3. chess is generally considered to require "thinking" for skilful play, [...]
4. the discrete structure of chess fits well into the digital nature of modern computers.

Computerschach: Zitate

Claude Shannon (1949)

The chess machine is an ideal one to start with, since

1. the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate),
2. it is neither so simple as to be trivial nor too difficult for satisfactory solution,
3. chess is generally considered to require "thinking" for skilful play, [...]
4. the discrete structure of chess fits well into the digital nature of modern computers.

Alexander Kronrod (1965)

Chess is the drosophila of Artificial Intelligence.

Computerschach: noch ein Zitat

John McCarthy (1997)

In 1965, the Russian mathematician Alexander Kronrod said, ‘Chess is the Drosophila of artificial intelligence.’

Computerschach: noch ein Zitat

John McCarthy (1997)

In 1965, the Russian mathematician Alexander Kronrod said, 'Chess is the Drosophila of artificial intelligence.'

However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophilae. We would have some science, but mainly we would have very fast fruit flies.

Fazit

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)
- Alpha-Beta-Suche merkt sich, welchen Nutzen beide Spieler anderswo im Baum erzwingen können und vermeidet so viele unnötige Berechnungen
- im besten Fall Aufwand $O(b^{d/2})$ bei uniformen Bäumen $\rightsquigarrow$ doppelt so tiefe Suche wie bei Minimax möglich