

Hauptseminar: Spiele in der Informatik

Adversarial Search

Mario Bittner

19.06.06

Vortrag am 19. Juni 2006

Inhaltsverzeichnis

1	Was ist Adversarial Search	2
2	MinMax Suche	2
3	Alpha-Beta Suche	3
3.1	Tiefen- Zeit- Begrenzung	6
3.1.1	Auswertungsfunktion	6
3.1.2	Abbrechen der Suche	8
4	Zufallskomponenten	9
4.1	Positionsbewertung in Spielen mit Zufallsknoten	10
4.2	Komplexität von <i>ErwartungsMinMax</i>	11

1 Was ist Adversarial Search

Bei Adversarial Search (Konkurrenten Suche) geht es meist um Computerspiele mit einer minimalen Anzahl von 2 Spielern. Mindestens einer der Spieler soll hierbei vom Computer vertreten werden.

Im Folgenden wird es zunächst um den Aufbau und die Auswertung von Suchbäumen mit der MinMax Suche und die verbesserte α - β -Suche gehen. Um Spiele in angemessener Zeit ablaufen zu lassen sind weitere Modifikationen wie der frühere Abbruch der Suche und die Verwendung einer Auswertungsfunktion vonnöten. Logische Fehler die bei der Verwendung von Auswertungsfunktionen auftreten wie der Horizonteffekt, die eine Schwächung des Computers darstellen, werden anschließend behandelt gefolgt von Spielen die ein Zusätzliches Zufallselement besitzen.

Diese Arbeit basiert inhaltlich vorwiegend auf dem Buch „Artificial Intelligence: A Modern Approach“, von Stuart J. Russell und Peter Norvig[1], welches daher im folgenden nicht weiter erwähnt wird.

2 MinMax Suche

Adversarial Search basiert auf dem MinMax Algorithmus, welcher auf der Untersuchung und Auswertung aller legalen Spielzüge des gegebenen Spielfeldes basiert, bis hin zu einem Terminierenden Zustand. Die Aufzeichnung dieser Züge und jeweils resultierenden Zustände nennt man auch Spiel-Baum. Im allgemeinen kann ein Spiel als eine Art Suchproblem mit den folgenden Komponenten eingeordnet werden:

- Der **Ausgangszustand**, welcher den Spielaufbau und den ziehenden Spieler enthält.
- Eine **Nachfolgefunktion**, welche eine Liste von [Zug, Zustand] Paaren mit zugelassenen Zügen und deren jeweils resultierenden Status zurück gibt.
- Einen **Terminaltest**, welcher feststellt wenn ein Spiel zu ende ist. Zustände die ein Spiel beenden werden auch Terminalzustände genannt.
- Eine **Zielfunktion**, welche eine numerische Auswertung eines Terminalzustandes ermittelt. In Schach werden die Endzustände Sieg, Unentschieden und Niederlage mit den Werten +1,0 und -1 unterschieden. Einige Spiele haben auch eine weite Fächerung von Endzuständen, so erstreckt sich der Wertebereich bei Backgammon von +192 bis -192.

Durch die Zielfunktion wird klar, woher der Name MinMax stammt. Der Computer versucht den bei optimaler Spielweise bestmöglichen Zug für sich herauszufinden, indem er seine Züge so wie die des Gegners bewertet und den in aktueller Positionierung bestmöglichen Zug heraus sucht. Hierzu bewertet er die einzelnen Ebene des Spiel-Baums jeweils mit Min und Max, je nach dem, welcher Spieler in dieser Ebene am Zug wäre. Spielt der Gegner nicht optimal, erhöht dies die Chancen des Computers auf einen Sieg. Für Spiele in denen der Gegner nie optimal spielt bzw. spielen kann, sind andere Verfahren effizienter. Dies ist z.B. bei Spielen der Fall, in denen ein Zufallselement mit berücksichtigt werden muss wie Backgammon oder Spielen mit unvollständigen Informationen, wie das bei vielen Kartenspielen der Fall ist.

Für Spiele wie Tic-Tac-Toe oder 4-Gewinnt stellt MinMax die Optimale Spielstrategie dar, es sind so genannte Nullsummenspiele. Diese Spiele besitzen für einen Rechner eine überschaubare Gesamtanzahl von möglichen Zuständen, das Spielfeld wird mit jedem Zug kleiner, weshalb

```

function MINMAX-DECISION(state) returns eine Aktion
  inputs: state, aktueller Zustand im Spiel
   $v \leftarrow \text{MAX-VALUE}(\text{state})$ 
  return die Aktion in NACHFOLGER(state) mit dem Wert v

function MAX-VALUE(state) returns einen Auswertungswert
  if TERMINAL-TEST(state) then return AUSWERTUNG(state)
   $v \leftarrow -\infty$ 
  for a,s in NACHFOLGER(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
  return v

function MIN-VALUE(state) returns einen Auswertungswert
  if TERMINAL-TEST(state) then return AUSWERTUNG(state)
   $v \leftarrow \infty$ 
  for a,s in NACHFOLGER(state) do
     $v \leftarrow \text{MAX}(v, \text{MAX-VALUE}(s))$ 
  return v

```

Abbildung 1: MinMax Algorithmus

der Baum sehr schnell determiniert. Bei komplexeren Spielen die ein fast unbegrenztes Repertoire an Möglichen Spielstellungen und Zugmöglichkeiten bieten (z.B. Schach), wird schnell klar, dass MinMax alleine nicht ausreichen kann. Der Spielbaum ist zu breit gefächert, um alle Pfade bis zu den Determinierungszuständen zu berechnen. Die Anzahl der Zustände ist Exponentiell zur Anzahl der Züge. Das MinMax Verfahren ist sehr vielseitig und kann ebenso gut für Multiplayer Spiele verwendet werden, hierzu müssen zunächst die einzelnen Werte der Knoten durch einen Vektor von Werten ersetzt werden. Bei 3 Spielern ein Vektor der Form $\langle V_a, V_b, V_c \rangle$. Für Endzustände ist der Vektor das Hilfsmittel des Spielstatus aus Sicht des jeweiligen Spielers. (In 2 Spieler Spielen kann der zweiwertige auf einen einwertigen reduziert werden, weil die Werte immer entgegengesetzt sind). Mehrspieler Spiele erfordern andere Spielstrategien, da hier teilweise Allianzen gebildet werden um den Maximalen Gewinn zu erzielen.

3 Alpha-Beta Suche

Offensichtlich kann der Exponent des MinMax Verfahrens nicht eliminiert werden, jedoch kann man die Anzahl im besten Fall auf die Hälfte reduzieren. Hierzu muss eine Anzahl von Zweigen bei der Suche ausgelassen werden. Tatsächlich ist dies mit der Alpha-Beta Suche möglich. Sie liefert in kürzerer Zeit den selben Zug wie das MinMax Verfahren. Hierfür lässt es Zweige in der Verarbeitung aus, die keinen Einfluss auf die Entscheidung des Zuges haben. Für diesen Zweck werden 2 Variablen benötigt, welche zur Beschreibung des Worst-Case-Szenarios Werte aufnehmen und weiterreichen (α und β). Der α -Wert ist das Ergebnis, das Spieler A mindestens erreichen wird, der β -Wert ist das Ergebnis das Spieler B mindestens erreichen wird. Besitzt ein maximierender Knoten (von Spieler A) einen Zug, dessen Rückgabe den β -Wert überschreitet, wird die Suche in diesem Knoten abgebrochen. (Beta-Cutoff, denn Spieler B würde erst gar nicht diese Variante wählen, weil sie ein zu gutes Ergebnis für Spieler A liefern würde). Liefert

der Zug stattdessen ein Ergebnis, das den α -Wert übersteigt wird dieser entsprechend nach oben angehoben. Analog gilt für die minimierenden Knoten, wobei bei Werten kleiner als α abgebrochen wird (Alpha-cutoff) und der β -Wert nach unten angepasst wird.“[2].

```

function ALPHA-BETA-SEARCH(state) returns eine Aktion
  inputs: state, aktueller Zustand im Spiel
   $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$ 
  return die Aktion in NACHFOLGER(state) mit dem Wert v

function MAX-VALUE(state,  $\alpha, \beta$ ) returns einen Auswertungswert
  inputs: state, aktueller Zustand im Spiel
            $\alpha$ , der Wert der besten Alternative für MAX entlang des Pfads zu state
            $\beta$ , der Wert der besten Alternative für MIN entlang des Pfads zu state
  if TERMINAL-TEST(state) then return AUSWERTUNG(state)
   $v \leftarrow -\infty$ 
  for a,s in NACHFOLGER(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v

function MIN-VALUE(state,  $\alpha, \beta$ ) returns einen Auswertungswert
  inputs: state, aktueller Zustand im Spiel
            $\alpha$ , der Wert der besten Alternative für MAX entlang des Pfads zu state
            $\beta$ , der Wert der besten Alternative für MIN entlang des Pfads zu state
  if TERMINAL-TEST(state) then return AUSWERTUNG(state)
   $v \leftarrow +\infty$ 
  for a,s in NACHFOLGER(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
    if  $v \leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v

```

Abbildung 2: Alpha-Beta Algorithmus

Dieses Verfahren kann auf Bäume jeder Tiefe angewendet werden, oft werden ganze Unterbäume vom verfahren abgetrennt und nicht mehr weiter betrachtet, was einen erheblichen Geschwindigkeitszuwachs zur Folge hat. Den Namen hat das Verfahren von den zwei Parametern, welche die aktuellen Grenzen des Suchpfades abstecken.

α = der Wert der besten Wahl, die bis jetzt an irgendeinem Auswahl Punkt entlang des Weges für MAX gefunden wurde.

β = der auf dem bisherigen Pfad beste gefundene Wert (i.A. der niedrigste Wert) für MIN.

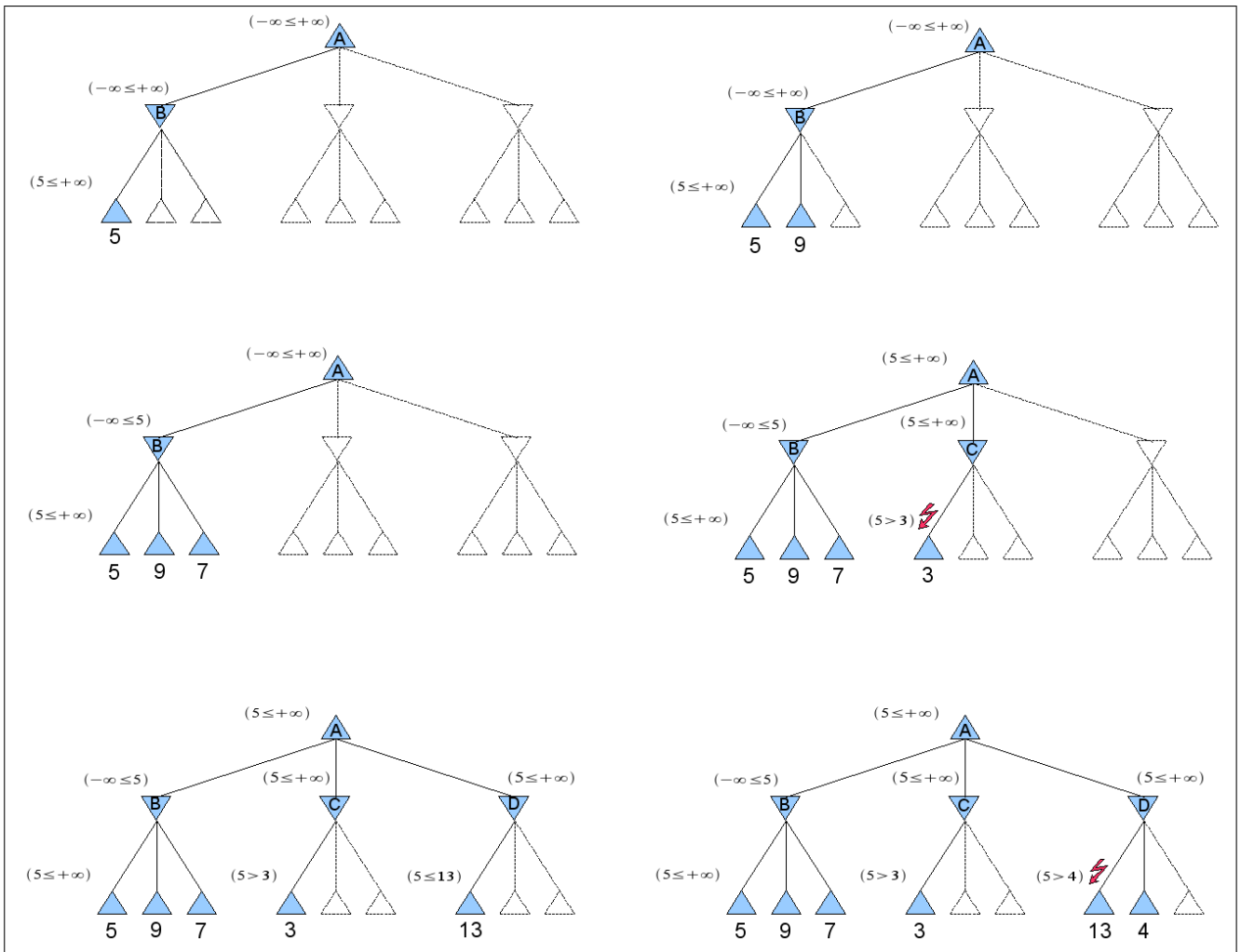


Abbildung 3: Alpha-Beta Suche

Alpha-Beta Suche aktualisiert die Werte von α und β und geht die zur Verfügung stehenden Wege solange ab, bis diese schlechter als α oder β für MAX bzw. MIN sind.

Die Effizienz vom Alpha-Beta Verfahren ist stark von der Anordnung der Unterbäume abhängig, wäre z.B. in Abbildung 3 der Unterbaum von D mit Wertigkeit 4 als erstes erzeugt worden hätte man einen Unterbaum mehr abtrennen können. Dies lässt erkennen, dass es lohnenswert sein kann zuerst alle Unterbäume zu untersuchen die voraussichtlich die besten Ergebnisse erzielen. Wenn man einmal unterstellt, dass dies möglich ist ergibt sich, dass Alpha-Beta nur $O(b^{\frac{d}{2}})$ Knoten betrachten muss um den besten Zug heraus zu finden. Im Gegensatz zu Minimax welches $O(b^d)$ hierfür benötigt ein beachtlicher Zugewinn. Leider ist dies nicht zu 100% Möglich, ansonsten könnte man die Sortierfunktion ein perfektes Spiel spielen lassen. Fügt man dynamische Sortierfunktionen hinzu, welche beispielsweise zuerst die Züge analysieren, die sich beim letzten Spielen als die Besten heraus gestellt haben, bringt einen dies schon sehr nahe an die Theoretische Grenze heran.

In Spielen tauchen gleiche Spielaufbauten regelmäßig durch Umstellung von verschiedenen Permutationen der Zug Sequenz auf. Zum Beispiel, wenn Weiß einen Zug a_1 machen kann, den Schwarz mit b_1 entgegnet und Weiß als nächstes einen bezugslosen Zug a_1 auf der anderen Seite des Spielfeldes ausführt welcher von Schwarz mit b_2 beantwortet wird, dann enden die Sequenzen $[a_1, b_1, a_2, b_2]$ und $[a_1, b_2, a_2, b_1]$ in der selben Spielaufstellung.

Es ist also lohnenswert sich dies Auswertung beim ersten auftauchen in einer Hashtabel-

le zu speichern, so dass sie bei Auftreten nicht ein zweites Mal berechnet werden muss. Eine Hashtabelle mit vorhergesehenen Positionierungen wird traditionell eine „Transpositionstabelle“ genannt. Eine Transpositionstabelle kann zuweilen eine gigantische Auswirkung zur Folge haben. In manchen Fällen kann sie die maximale mögliche Suchtiefe im Schach verdoppeln, andererseits ist es nicht ratsam alle Positionierungen zu speichern, da dies bei Millionen von untersuchten Knoten nicht sehr Praktikabel ist. Es gibt eine Vielzahl von Strategien, welche dieses Problem lösen sollen, diese sollen herausfinden welche Züge sich am meisten lohnen abzuspeichern.

3.1 Tiefen- Zeit- Begrenzung

Bisher wird der Baum immer noch bis zu den Terminierenden Zuständen aufgebaut, was für den Computer durchaus zu den besten Ergebnissen führt, jedoch immer noch in einer unzumutbar langen Zeit. In Spielen, wie Schach ist bereits im Reglement festgelegt dass ein Spieler pro Zug nicht mehr als 3 Minuten benötigen darf[3]. Ähnlich sollte es auch beim Computer sein, sogar noch schneller, denn aus Sicht eines Menschlichen Gegners wäre das Spiel, durch die langen Wartezeiten, unspielbar.

Die Zeit muss, also begrenzt werden, ohne ein all zu hohes Maß an Spielstärke zu verlieren. Hierfür muss bei auf Min-Max basierenden Verfahren (also auch alpha-beta), der Terminaltest durch einen Abbruchtest ersetzt werden. Dies kann im einfachsten Fall eine einfach Suchtiefenbegrenzung sein. Wird die Suchtiefe begrenzt, so müssen nichtterminale Zustände in terminale Zustände überführt werden. Wie sollen aber nun diese Zustände mit echten terminalen Zuständen verglichen werden? Zu diesem Zweck wird eine heuristische Auswertungsfunktion eingeführt, die eine Stellung möglichst gut beurteilt und so einen Vergleichswert erzeugt.

3.1.1 Auswertungsfunktion

Eine Auswertungsfunktion gibt einen Schätzwert, der an einem Punkt zu erwarten ist, zurück. Schachspieler befassen sich schon seit Jahrhunderten mit solchen Spielfeldbewertungen, sie selbst haben keinerlei Möglichkeit alle Spielvariationen durchzugehen, und überlegten sich daher schon sehr früh Auswertungsfunktionen hierfür.

Es sollte klar sein, dass die Leistung eines Spielprogramms maßgeblich von der Qualität seiner Auswertungsfunktion abhängt. Eine schlechte Funktion könnte den Computer zu einer Position führen, von der aus nur noch verlorene Terminalzustände erreicht werden können. Gute Auswertungsfunktionen müssen folgendes berücksichtigen:

1. Die Funktion sollte die Terminalzustände genau so einordnen wie es die original Zielfunktion machen würde. Sonst könnten suboptimale Züge gewählt werden, selbst wenn bereits alle Möglichkeiten zu einem Endzustand einzusehen sind.
2. Die Berechnung darf nicht all zu viel Zeit beanspruchen, schließlich soll ein zeitlicher Vorteil durch diese Maßnahme entstehen.
3. Für Nichtterminalzustände sollte die Auswertungsfunktion die Wahrscheinlichkeit eines Erfolgs möglichst genau widerspiegeln.

Schach ist zwar kein Spiel mit Zufallselementen aber dadurch, dass die Suche an Nichtterminalzuständen Abgekürzt werden muss, ist der Algorithmus ungenau in Hinblick auf den tatsächlichen Wert des Zustands. Konkret sollten Auswertungsfunktionen mit der Berechnung von verschiedenen Eigenschaften arbeiten z.B. mit der Anzahl der Bauern die jeder Spieler in einem

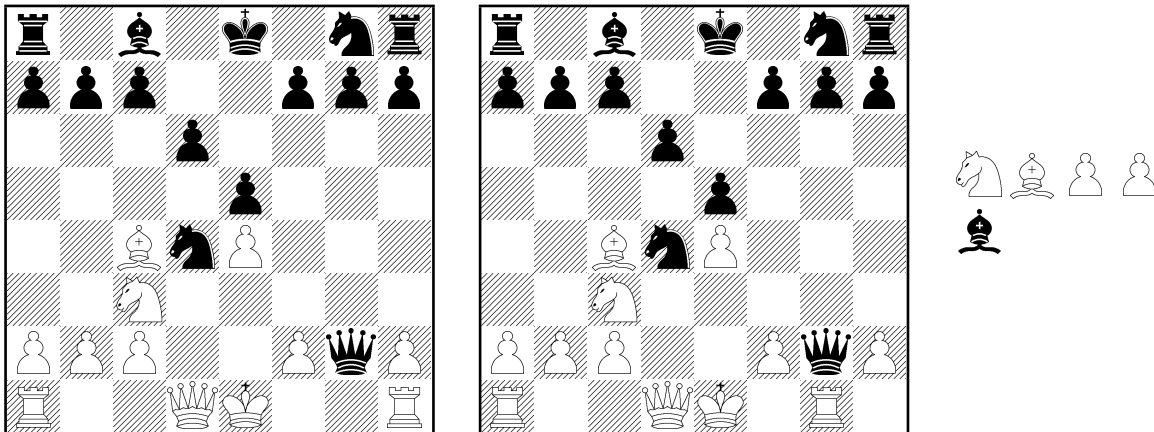


Abbildung 4: 2 leicht unterschiedliche Schachpositionen

Schachspiel besitzt. Die Eigenschaften zusammen genommen, definieren verschiedene Kategorien oder Äquivalenzklassen von Zuständen: Die Zustände jeder Kategorie besitzen die selben Werte für alle Eigenschaften. Jede gegebene Kategorie, enthält im allgemeinen einige Zustände die zum Sieg, zu einem Unentschieden oder zu einer Niederlage führen. Die Auswertungsfunktion kann nicht wissen von welcher Art ein Zustand ist, sie kann aber einen Wert zurückgeben, der den Anteil der Zustände mit jedem Resultat widerspiegelt. Zum Beispiel nehmen wir an, dass 72% der Zustände, die in der Kategorie angetroffen werden zu einem Gewinn (+1), 20% zu einer Niederlage (-1) und 8% zu einem Unentschieden (0) führen. Dann entspricht der Erwartungswert dem Durchschnittswert der Gewichtungen: $(0,72 \times +1) + (0,20 \times -1) + (0,08 \times 0) = 0,52$ was einer angemessenen Auswertung der Zustände entspricht. Prinzipiell kann der Erwartungswert für jede Kategorie, in einer Auswertungsfunktion die in jedem Zustand funktioniert, festgestellt werden. Wie die Terminalzustände auch, braucht die Auswertungsfunktion nicht die tatsächlichen erwarteten Werte zurückgeben, solange die Einordnung der Zustände die Selbe ist.

In der Praxis benötigt diese Art von Analyse jedoch zu viele Kategorien und setzt zu viele Erfahrungswerte voraus. Stattdessen kombinieren die meisten Auswertungsfunktionen unterschiedliche numerische Beiträge von allen Eigenschaften zu einem Gesamtwert zusammen. Einleitende Schachbücher geben z.B. einen ungefähren Materialwert der einzelnen Figuren im Schach wieder: jeder Bauer wird mit 1 bewertet, ein Springer oder Läufer mit 3, ein Turm mit 5, und die Dame mit 9. Andere Eigenschaften wie eine „guter Bauern Aufbau“ oder „Sicherheit des Königs“ zählen wie ein halber Bauer. Diese Eigenschaftswerte werden dann, für die Auswertung der Stellung, ganz einfach zusammen Addiert. Ein sicherer Vorteil von einem Bauern sollte die Wahrscheinlichkeit eines Sieges erhöhen, ein Vorteil von drei Bauern, sollte fast schon einen bestimmten Sieg ergeben (Abbildung 4). Mathematisch wird diese Art der Auswertungsfunktion eine „gewichtete lineare Funktion“ genannt, weil sie wie folgt ausgedrückt werden kann:

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum_{i=1}^n w_i f_i(s)$$

Wobei jedes w_i für eine Gewichtung (weight) und jedes f_i für eine Funktionsmerkmal (feature) der Position i steht. Für Schach könnte f_i die Anzahl der einzelnen Figurenarten auf dem Brett sein und w_i könnten die Werte der Figuren sein (1 für einen Bauern, 3 für einen Läufer, etc.). Diese Funktion unterstellt allerdings, dass der Beitrag jedes Funktionsmerkmal unabhängig von den Werten der anderen Funktionsmerkmalen ist. Dem Springer wird beispielsweise der Wert 3 zugewiesen, damit wird aber die Tatsache ignoriert, dass Läufer im Endspiel, bei einem relativ

leeren Brett einen höheren Wert besitzen. Aus diesem Grund verwenden heutige Programme für Schach und auch andere Spiele auch nichtlineare Kombinationen von Funktionsmerkmalen. Ein Läuferpaar könnte hier beispielsweise einen höheren Wert bekommen als zwei einzelne, sowie ein Läufer im Endspiel mehr wert sein kann als am Anfang.

3.1.2 Abbrechen der Suche

Als nächstes muss die Sucher erst einmal an einer bestimmten Stelle abgebrochen werden, sonst kann die EVAL-Funktion nicht ausgeführt werden und wäre sinnlos. Also muss ermittelt werden wann es Sinnvoll ist die Suche abbrechen und die EVAL-Funktion aufzurufen. Der Terminaltest wird dann durch die EVAL-Funktion ersetzt. Am einfachsten geschieht dies bei einer festen Suchtiefe d . Die Tiefe d wird so gewählt, dass die aufgewendete Zeit die Spielregeln nicht verletzt.

Dass der Ansatz mit einer Bewertungsfunktion Fehler produzieren kann ist offensichtlich, da die Funktion nur Schätzungen vornehmen kann. Betrachtet man noch einmal die Material behaftete Bewertungsfunktion für Schach: Das Programm sucht bis zur angegebenen Tiefenbegrenzung und erreicht die in Abbildung 4 gezeigte Position. Schwarz ist mit einem Springer und zwei Bauern im Vorteil. Der Heuristische Wert würde dies mit einem wahrscheinlichen Gewinn für Schwarz widerspiegeln. Im nächsten Zug schlägt Weiß jedoch die schwarze Dame ohne Ausgleich für den Verlust. Damit entpuppt sich die Positionierung als eigentlicher Vorteil für Weiß. Die Funktion kann dies nur sehen, wenn sie eine Schicht weiter voraussieht.

Ein komplexerer Abbruchttest ist hier von Nöten, die Bewertungsfunktion sollte nur auf Positionen angewendet werden die sich in **Ruhe** befinden, d.h. welche die in der näheren Zukunft keine wilden Wertumschwünge aufweisen wird. Positionen in denen vorteilhafte Figuren geschlagen werden können, sind für Bewertungsfunktionen, die nur Material zählen, nicht in Ruhe. Nicht in Ruhe befindliche Positionen können weiter expandiert werden, bis in Ruhe befindliche Positionen erreicht werden. Diese zusätzliche Suche ist die sogenannte **Ruhensuche** („**unruhige Stellungen**“). Ab und an ist diese nur auf bestimmte Zugarten beschränkt, wie z.B. auf Züge mit denen Figuren geschlagen werden, die schnell die Unsicherheiten in der Position auflösen.

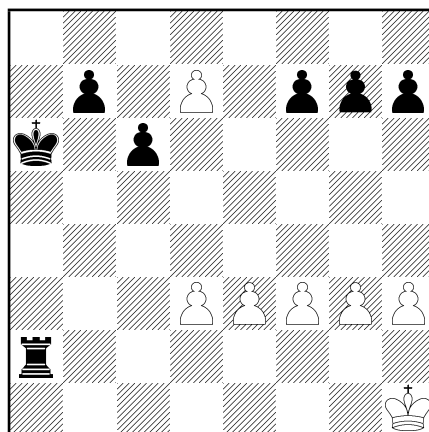


Abbildung 5: Der Horizonteffekt

Der **Horizonteffekt** ist schwieriger zu eliminieren. Dieser tritt auf, wenn der Computer einen gegnerischen Zug erwartet, der ihm selbst im Spiel ernsthaft schadet aber nicht mehr ver-

hindert werden kann. Das Schachspiel in Abbildung 5 betrachtet, ergibt, dass Schwarz Materiell im Vorteil ist. Wenn es aber Weiß gelingt den Bauern in Reihe 7 nach Reihe 8 zu bewegen, wird aus diesem eine Dame, was für den Rechner einen Gewinn für Weiß darstellt. Schwarz kann das Dilemma zwar mit dem Zug des Turms zum Schach vereiteln, letztlich ist der Dame Zug aber nicht zu verhindern. Der Computer schiebt beim Berechnen das Problem über den eigenen Horizont (seine maximale Suchtiefe) hinaus, somit stellt sich für ihn die Zukunft noch einigermaßen rosig dar. Das Problem bei der festen Tiefensuche ist, dass der Computer glaubt, dass er durch seinen vermeidenden Turmzug den Damezug verhindert habe, dabei ist er nur an eine Position außerhalb seines Horizonts verschoben worden, wo er nicht mehr erkannt wird. Durch die laufende Hardwareverbesserung sind immer weitere Suchtiefen möglich, wodurch der Horizonteffekt seltener auftritt. Sehr lang verzögerte Folgen sind relativ selten. Eine Verwendung von **singulären Erweiterungen** ist ebenfalls eine Effiziente Art um einen Horizonteffekt zu vermeiden oder zumindest zu mindern ohne hohe Suchkosten zu verursachen. Eine singuläre Erweiterung ist ein Zug, der „deutlich besser“ als andere Züge in einer Feldaufstellung ist. Eine Singuläre-Erweiterungs-Suche, geht über den normalen Tiefenbeschränkungswert hinaus, ohne viel kosten zu verursachen, da der Verzweigungsfaktor lediglich 1 ist. (Man kann sich die Ruhensuche als eine Variante der singulären Erweiterungen vorstellen) In Abbildung 5 findet eine Singuläre-Erweiterungs-Suche den dameerzeugenden Zug, vorausgesetzt, dass der Schachzug von Schwarz und der Königszug von Weiß als „deutlich besser“ als die Alternativen erkannt werden können.

Bisher wird die Suche in einer bestimmten Tiefe abgebrochen und mit Alpha-Beta noch zu verkürzen. Es ist außerdem noch möglich eine **Vorabkürzung** vorzunehmen, d.h bestimmte Knoten werden sofort gekürzt, ohne sie weiter in Betracht zu ziehen. Menschliche Schachspieler können nur ein paar Züge von jeder Position aus vorausdenken (zumindest bewusst). Der Ansatz ist jedoch sehr risikoreich, da es keine Garantie dafür gibt, dass nicht der beste Zug weggekürzt wird. Dies kann in Wurzelnähe zu einer Katastrophe führen, weil so das Programm häufig *offensichtliche* Züge verfehlt. Das Vorabkürzen kann in einigen speziellen Zügen sicher angewandt werden, z.B. wenn Züge symmetrisch oder anderweitig äquivalent sind. So muss nur einer davon berücksichtigt werden.

4 Zufallskomponenten

Was ist nun mit Spielen die ein unvorhersehbare externe Komponente beinhalten, wie z.B. einen Würfel? Der Computer weiß bei solch einem Spiel wohl, welche Züge er in seiner Situation vollziehen kann, weiss aber nicht was der Gegner für Zufallswerte bekommt und somit auch nicht welche erlaubten Züge seinem Gegner zur Verfügung stehen. Das bedeutet, er kann keinen Standardspielbaum der Art erstellen wie es bei Schach oder Tic-Tac-Toe der Fall wäre. Ein Spielebaum z.B. in Backgammon muss daher neben den Max-Min-Knoten auch **Zufallsknoten** enthalten. Die Verzweigungen, die von jedem Zufallsknoten aus abgehen, kennzeichnen die möglichen Würfelaugen und jede davon ist mit dem Würfelauge und der Wahrscheinlichkeit, dass sie auftritt, beschriftet. Bei Backgammon gibt es 2 Würfel, das bedeutet es gibt 36 mögliche Würfelergebnisse, die alle gleich Wahrscheinlich sind. Weil aber Kombinationen wie 3-4 und 4-3 im selben Ergebnis enden gibt es nur 21 verschiedene Würfe. Pasche (1-1 bis 6-6) haben die Wahrscheinlichkeit $1/36$, die restlichen 15 Würfe die Wahrscheinlichkeit von $1/18$.

Als nächstes muss man verstehen, wie eine korrekte Entscheidung getroffen wird. Der Computer soll immernoch den Zug der besten Positionierung auswählen, die Positionen haben jedoch keinen definierten Minimax-Wert. Stattdessen kann nur der **erwartete Wert** berechnet werden, wobei die Erwartung über alle möglichen Würfelzahlen geht, die auftreten können. Das bedeu-

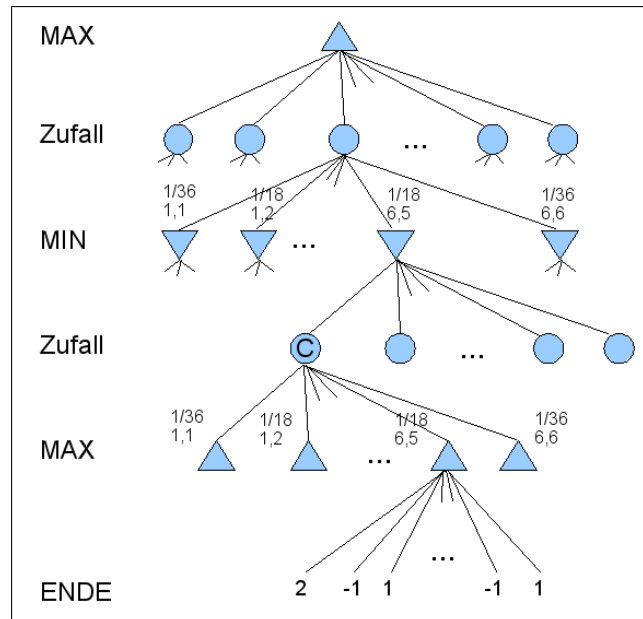


Abbildung 6: Skizze eines Spielbaums für eine Backgammon-Position

tet, der **Minmax-Wert** für deterministische Spiele muss zu einem **erwarteten Minmax-Wert** für Spiele mit Zufallskomponenten verallgemeinert werden. Endknoten sowie Max- und Min-Knoten (für die die Würfelzahlen bekannt sind) arbeiten weiter wie bisher. Zufallsknoten werden bewertet, indem der gewichtete Durchschnitt der Werte ermittelt wird, der aus allen möglichen Würfeln resultiert.(Abbildung 7)

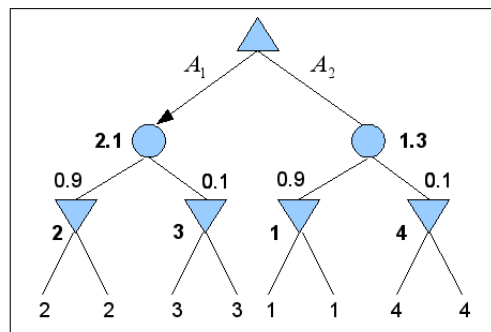


Abbildung 7: Baum mit erwarteten MinMax-Werten

Dabei vergrößert die Nachfolgerfunktion für einen Zufallsknoten n einfach den Zustand n mit jedem möglichen Wurf, um jeden möglichen Nachfolger s zu erzeugen, und $P(s)$ ist die Wahrscheinlichkeit, dass der Wurf stattfindet. Die Gleichungen können bis zur Wurzel des Baums rekursiv aktualisiert werden, so wie in Minimax.

4.1 Positionsbewertung in Spielen mit Zufallsknoten

Bei *ErwartungsMinMax* ist wie bereits in *MinMax* die offensichtliche Abschätzung, die Suche an irgend einem Punkt abubrechen und auf jedes existierende Blatt eine Bewertungsfunktion anzuwenden. Auf den ersten Blick möchte man meinen die Bewertungsfunktionen von Spielen wie Backgammon müssten sich im Wesentlichen nicht von welchen in Spielen wie Schach unterscheiden, sie müssten bessere Positionen einfach nur höher Werte zuteilen. In Wirklichkeit

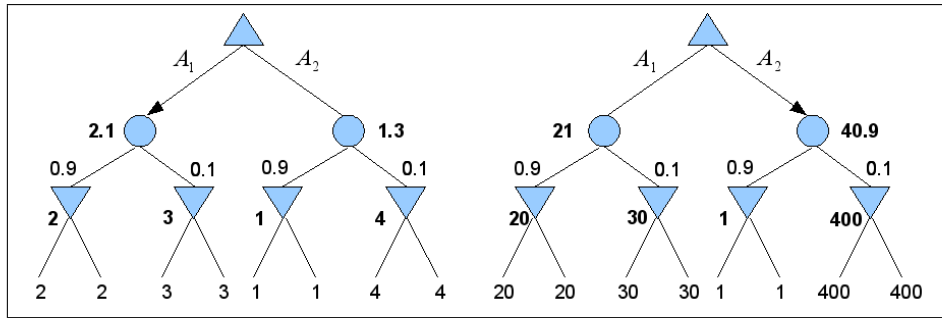


Abbildung 8: Eine die Reihenfolge beibehaltende Transformation der Blattwerte ändert den besten Wert

bedeutet das Vorliegen von Zufallsknoten jedoch, dass noch sorgfältiger geprüft werden muss was die einzelne Bewertung bedeutet. Abbildung 8 zeigt was passiert: Bei einer Bewertungsfunktion, die den Blättern die Werte $[1,2,3,4]$ zuweist, ist der Zug A_1 der beste; bei den Werten $[1,20,30,400]$ ist der Zug A_2 der beste. Das Programm verhält sich bei einer Skalierung der Bewertungswerte also komplett anders. Es hat sich gezeigt, dass die Bewertungsfunktion, wenn diese Instabilitäten vermeiden soll, eine positive lineare Transformation der Wahrscheinlichkeit, von einer Position aus zu gewinnen, sein muss (allgemeiner gesagt, von dem erwarteten Nutzen der Position). Dies ist eine allgemeine Eigenschaft von Situationen, an denen Unsicherheiten beteiligt sind.

4.2 Komplexität von ErwartungsMinMax

Bei Vorauskenntnis aller Würfelresultate, ist die Lösung eines Spiels mit Würfeln gleich der Lösung eines Spiels ohne Würfel, was Minimax in $O(b^m)$ erledigt. Durch die Berücksichtigung aller möglichen Würfelreihenfolgen, benötigt ErwartungsMinMax $O(b^m n^m)$ mit $n = \text{Anzahl der verschiedenen Würfel}$.

Selbst wenn die Suchtiefe auf einen kleinen Wert d beschränkt ist, macht es die zusätzlichen Kosten im Vergleich zu Minimax unrealistisch, bei den meisten Glücksspielen sehr weit vorausszusehen. Beim Backgammon ist $n = 21$, und b liegt normalerweise bei 20, in einigen Situationen kann es aber bis zu 4000 betragen - für Paschwürfe. Drei Schichten ist das höchste, was bewältigt werden kann und auch Sinn macht.

In Spielen mit Würfeln gibt es keine wahrscheinlichen Zugfolgen, was die Anwendung von $\alpha - \beta$ -Suche unmöglich macht, da dort Situationen ausgelassen werden, die beim best möglichen Spiel keine Rolle spielen. Dies ist durch die Unkenntnis des folgenden Würfelresultates nicht möglich. Dies ist ein allgemeines Problem wenn Unsicherheiten mit ins Spiel kommen. Die Möglichkeiten werden extrem multipliziert, und die detaillierte Planung von Aktionen wird sinnlos, da die Welt dabei vielleicht nicht mitspielt.

Es gibt allerdings Lösungen, die dem $\alpha - \beta$ ähneln. Hierbei bleibt die Analyse für Min- und Max-Knoten unverändert. Mit etwas Einfallsreichtum können auch Zufallsknoten gekürzt werden. Man betrachte den Zufallsknoten C in Abbildung 6 und was mit seinem Wert passiert, während man seine Söhne untersucht und bewertet. Ist es möglich eine Obergrenze für den Wert von C zu finden, bevor wir alle seine Söhne betrachtet haben? (Genau das ist was $\alpha - \beta$ benötigt um einen Knoten und seinen Unterbaum zu kürzen.) Auf den ersten Blick scheint dies unmöglich, weil der Wert von C der Mittelwert der Werte seiner Söhne ist. Bevor das Würfelresultat nicht bekannt ist, kann dieser Mittelwert irgendetwas sein, weil die noch unbetrachteten Knoten beliebige Werte haben können. Werden nun aber Grenzen für die möglichen Werte der

Nutzenfunktion festgelegt, gelangt man auch zu Grenzen für den Mittelwert. Wenn man z.B. sagt alle Nutzenwerte liegen zwischen +3 und -3, ist der Wert von Blattknoten begrenzt, und man kann eine Obergrenze für den Wert eines Zufallsknoten angeben, ohne alle seine Söhne zu betrachten.

Literatur

- [1] Stuart J. Russell, Peter Norvig: „Artificial Intelligence: A Modern Approach “
- [2] Wikipedia <http://www.wikipedia.org/>
- [3] <http://www.computerschach.de>