Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Fallbasiertes Lernen von Bewertungsfunktionen

Diplomarbeit

vorgelegt von: Matthias Neumeister
Betreuer: Prof. Dr. Gerhard Brewka
Dr. Steffen Lange

Leipzig, Oktober 1998

Nach dem Spiel ist vor dem Spiel.
Sepp Herberger

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Einleitung

Als eine der vielversprechendsten Disziplinen der Informatik zieht die Künstliche Intelligenz dauerhaft Aufmerksamkeit auf sich. Das Teilgebiet der wissensbasierten Systeme findet dabei sowohl in der Forschung als auch in der Industrie reges Interesse.

Im Jahr 1952[1] begann Samuel mit einem von ihm erstellten Computerprogramm zu experimentieren, welches selbständig lernte, besser Dame zu spielen[2]. Das erste von ihm beschriebene Lernverfahren nannte Samuel *Rote Learning* (dt. *Auswendiglernen*). Dabei speicherte sein Programm die beim Durchsuchen der Zugmöglichkeiten berechneten Bewertungen der Stellungen, um sie in späteren Spielen erneut zu verwenden. Samuels Arbeiten gelten als Pionierleistung auf dem Gebiet des maschinellen Lernens.

Diese Arbeit greift den Ansatz des Auswendiglernens wieder auf. Sie stellt einen um die Methoden des fallbasierten Schließens erweiterten Baumsuche-Algorithmus vor, mit dessen Hilfe eine verbesserte Bewertungsfunktion erlernt wird. Im Rahmen der Arbeit wurde untersucht, wie sich Änderungen an verschiedenen Parametern des Verfahrens auf das Lernverhalten auswirken. Besonders ist dabei die Anwendung unterschiedlicher Auswahlstrategien hervorzuheben. Andere Parameter sind die Suchtiefe des zugrundeliegenden Baumsuche-Algorithmus und die Bedingungen für das Heranziehen von Fällen. Bei letzterem waren die Fragen zu beantworten, zu welchem Zeitpunkt der Baumsuche Fälle in der Fallbasis gesucht werden sollen und mit Hilfe welcher Ähnlichkeitsrelation entschieden werden soll, ob ein Fall in einer gegebenen Situation

[1] [RN95], S. 18

[2] vgl. [Sam59] und [Sam67]

anwendbar ist. Nach der Darlegung der unterschiedlichen Varianten des Lernverfahrens, die aus den Parameteränderungen resultieren, werden ihre Ergebnisse verglichen und diskutiert.

Methoden der Künstlichen Intelligenz werden häufig an abstrakten Spielen untersucht. Zu begründen ist das einerseits damit, daß Spiele mit ausreichend hoher Komplexität anspruchsvolle Aufgaben für intelligente Problemlösungsmethoden darstellen. Andererseits sind sie frei von vielen äußeren Einflüssen und Details, die bei realen Problemen die Bewertung der Resultate erschweren.

Als Anwendungsbeispiel dient in dieser Arbeit das Spiel Kalah[3], das bereits vor mehreren tausend Jahren in Ägypten gespielt wurde. Es gehört zu der großen Familie der Bohnenspiele, die sich über weite Teile Afrikas und Asiens verbreiteten und als Mancala bezeichnet werden. Während diese Spiele in Europa wenig bekannt sind, werden sie in vielen Gebieten Afrikas gespielt und als Schulung strategischen Denkens geschätzt. Wegen seiner Bedeutung wird Mancala als das „Nationalspiel der Völker Afrikas“[4] bezeichnet[5]. Kalah bietet für die in dieser Arbeit vorgenommenen Untersuchungen mehrere Vorteile. Zum einen sind seine Regeln[6] trotz der hohen Komplexität[7] des Spiels einfach. Zum anderen ergibt sich die Bestimmung des Vor- oder Nachteils eines Spielers direkt aus den Regeln und dem Ziel des Spiels[8].

In den folgenden Abschnitten dieses Kapitels wird eine kurze Übersicht über das fallbasierte Schließen und die Spieltheorie gegeben. Anschließend werden für das Verständnis der Arbeit notwendige Begriffe erklärt.

Das Kapitel 2 beschäftigt sich mit dem Minimax-Algorithmus, der die Grundlage aller weiteren in dieser Arbeit behandelten Baumsuche-Algorithmen ist. Zuerst wird ein Algorithmus hergeleitet, der perfekte – d. h. die objektiv besten – Entscheidungen erlaubt. Dieser Algorithmus ist für komplexe Spiele nicht in vertretbarer Zeit ausführbar (siehe Abschnitt 2.1). Deshalb wird danach das Baumstutzen beim Minimax-Algorithmus beschrieben. Dabei wird sowohl die Beschränkung der Suchtiefe als auch

[3] Mein Interesse an Kalah bestand schon vor dem Beginn dieser Diplomarbeit. Deshalb implementierte ich ein Computerspiel, das seitdem unter der folgenden Adresse verfügbar ist:
`http://stinfwww.informatik.uni-leipzig.de/~neumeist/kalah.html`.

[4] [Mac84], S. 68

[5] Andere Quellen zu diesem Thema sind [SV93] und [Thi88].

[6] vgl. dazu den Anhang A

[7] vgl. dazu Abschnitt 1.4

[8] vgl. Abschnitt 2.1

das sogenannte α-β-Stutzen behandelt. Der letzte Abschnitt des Kapitels widmet sich Unzulänglichkeiten des tiefenbeschränkten Minimax-Algorithmus und Ansätzen zu ihrer Behebung.

Schwerpunkte der Arbeit sind die Kapitel 3 und 4.

Im Kapitel 3 wird ein Algorithmus vorgestellt, der an Samuels Auswendiglernen angelehnt ist und mit Hilfe des fallbasierten Schließens zur Verbesserung der Bewertungsfunktion beiträgt. Das Kapitel ist in Abschnitte zur Beschreibung des fallbasierten Schließens, seiner Methoden und der Ansätze, die es für das Lernen bietet, gegliedert. In den einzelnen Abschnitten folgen auf einen theoretischen Teil die Umsetzungen beim Anwendungsbeispiel.

Im Kapitel 4 werden die durchgeführten Experimente beschrieben und ihre Ergebnisse diskutiert. Es enthält Informationen über die Experimentierumgebung und Details der Implementation. Schließlich wird eine Auswahl aller möglichen Kombinationen der verschiedenen Verfahrensparameter vorgestellt, zu denen Testreihen durchgeführt wurden.

Kapitel 5 enthält die Zusammenfassung und gibt einen Ausblick auf mögliche Verbesserungen und Erweiterungen.

1.2 Fallbasiertes Schließen

Seit dem Ende der siebziger Jahre wird mit dem fallbasierten Schließen eine neue Problemlösungsmethode untersucht. Ihre Entwicklung war zweifach motiviert. Auf der einen Seite sollte die Art und Weise untersucht werden, wie Menschen ihre Erfahrungen zur Lösung von ähnlichen Problemen einsetzen. Dafür wurden Modelle für die Verarbeitung situationsbezogenen Wissens erstellt. Auf der anderen Seite sollte eine theoretische Basis für Analogie beim Bau intelligenter Systeme helfen.

Beim fallbasierten Schließen wird versucht, ein System Probleme lösen zu lassen, indem es Lösungen bereits bearbeiteter ähnlicher Probleme der neuen Situation anpaßt. Die Problembeschreibung, eine mögliche Lösung und ihre Konsequenzen werden dabei in Form eines Falls gespeichert. Zentrale Fragestellungen beim fallbasierten Schließen sind, wie erinnernswertes Wissen repräsentiert werden kann, welche Fälle abgerufen und wie diese dazu indiziert werden sollen und schließlich, welche Möglichkeiten es gibt, die Informationen der abgerufenen Fälle dem zu lösenden Problem anzupassen.

Die Bereiche, in denen fallbasiertes Schließen heute in der Praxis angewandt wird,

sind vielfältig. Darunter befinden sich Anwendungen in der medizinischen Diagnose (CASEY, IVY, MEDIC), in der Bauplanung (ARCHIE, CADSYN), in der Kundenbetreuung bei Compaq (The Compaq SMART System) oder für die Produktionsplanung bei der Herstellung von Verbundwerkstoffen für Flugzeugteile bei Lockheed (CLAVIER)[9].

1.3 Spieltheorie

Die Spieltheorie ist ein Teilgebiet der Entscheidungstheorie. Sie untersucht diejenigen Entscheidungsprobleme, bei denen nicht nur die Aktionen des entscheidenden Agenten und zufällige Ereignisse die Bewertung einer Entscheidung beeinflussen, sondern auch die Aktionen anderer Agenten. Probleme mit dieser Eigenschaft heißen *Spiele*, die handelnden Agenten *Spieler*.

In der Entscheidungstheorie wird ein Agent genau dann als *rationaler Entscheider* bezeichnet, wenn er die Aktion auswählt, die nach den ihm zur Verfügung stehenden Informationen zu dem für ihn günstigsten Resultat führt. Für die Bewertung der Aktionen wird eine *Bewertungsfunktion* verwendet. Alle Aussagen in dieser Arbeit setzen voraus, daß die Spieler rational entscheiden. Die Entscheidungen eines rationalen Agenten werden *perfekte Entscheidungen* genannt, wenn er alle notwendigen Informationen besitzt, um die für ihn vorteilhafteste Aktion auszuwählen. Da ein Spieler in der Regel nicht mit Sicherheit weiß, wie ein Spiel verlaufen wird, gibt es bei Spielen im allgemeinen keine perfekten Entscheidungen.

Als *Nullsummenspiele* werden Spiele bezeichnet, bei denen ein Vorteil eines Spielers für die anderen Spieler einen gleichgroßen Nachteil bedeutet. Kalah ist ein solches Nullsummenspiel. Kooperative Spiele, bei denen die Spieler gemeinsame Ziele verfolgen, sind keine Nullsummenspiele. Es gibt z. B. Patiencen – Kartenlegespiele – für mehrere Spieler, deren Ziel es ist, alle Karten eines Kartenstapels nach bestimmten Regeln auszulegen. Gelingt dies, dann „geht die Patience auf“ und das Spiel ist gewonnen.

Das in dieser Arbeit benutzte Anwendungsbeispiel Kalah ist ein *Zwei-Personen-Spiel*. Beide Spieler sind zu jeder Zeit vollständig über den Spielstand, d.h. die Situation, in der sich das Spiel befindet, informiert. In der Spieltheorie wird ein solches Problem ein *Spiel mit perfekter Information* genannt. Neben Kalah sind abstrakte Brettspiele wie Schach und Dame hierfür Beispiele. Im Gegensatz dazu sind bei vielen Kartenspielen die Karten der Gegner verdeckt. Der Spielverlauf hängt bei Kalah nicht

[9] [Kol93], S. 581ff

von Zufallselementen, sondern ausschließlich von den Entscheidungen und Aktionen der Spieler ab.

1.4 Begriffe und Notation

Für das Verständnis dieser Arbeit sind Erläuterungen weiterer Begriffe der Spieltheorie und grundlegender Zusammenhänge notwendig, auf die in den folgenden Kapiteln häufig Bezug genommen wird.

Mancala-Spiele sind Brettspielen sehr ähnlich. Allerdings gibt es Unterschiede bei der Benennung des Spielmaterials und der Ausführung der Züge. Bei Brettspielen ziehen die Spieler Spielsteine von einem Spielfeld auf ein anderes. Wenn sich die Steine, wie z. B. beim Schach, qualitativ unterscheiden, dann werden sie Figuren genannt. Bei Mancala-Spielen heißen die Felder *Mulden* und die Spielsteine *Bohnen*. In den Mulden können mehrere Bohnen liegen. Im Gegensatz zu Brettspielen werden die Bohnen nicht danach unterschieden, welchem Spieler sie gehören. Vielmehr wählt ein Spieler eine ihm gehörende Mulde aus und verteilt die in ihr liegenden Bohnen, indem er jeweils eine Bohne in verschiedene Mulden legt. Wie die Bohnen zu verteilen sind und welche Konsequenzen ein Zug hat, entscheiden die Regeln des jeweiligen Mancala-Spiels. Das Ziel von Kalah ist, im Verlauf des Spiels mehr Bohnen als der Gegner in die eigene Gewinnmulde füllen zu können. Die vollständigen Spielregeln von Kalah sind im Anhang A aufgeführt.

In der Spieltheorie heißt der zuerst ziehende Spieler eines Zwei-Personen-Spiels *Anziehender*, der andere *Nachziehender*.

Als *Position* oder *Stellung* wird, insbesondere bei Brettspielen, die vollständige Beschreibung des Spielstands bezeichnet. Bei Kalah gehören zu einer Position p die Inhalte aller Mulden und die Information, welcher Spieler am Zug ist:

$$p = ((m_{A1}, \ldots, m_{A6}, gm_A, m_{N1}, \ldots, m_{N6}, gm_N), s) \quad \text{mit} \quad s \in \{A, N, E\}.$$

Dabei sind die m_{Ai} bzw. m_{Ni} die Mulden des Anziehenden bzw. Nachziehenden, gm_A und gm_N ihre Gewinnmulden, während $s = A$ bzw. $s = N$ bedeutet, daß der Anziehende bzw. der Nachziehende am Zug ist, und $s = E$ dafür steht, daß das Spiel beendet ist.

Die Menge P aller im Spiel möglichen Positionen ist die Vereinigung der Menge aller *Endpositionen* P_E, in denen das Spiel beendet ist und der Mengen der Positionen P_A

und P_N, in denen der Anziehende bzw. der Nachziehende am Zug ist. Die *Startposition*, mit der das Spiel beginnt, gehört z.B. zur Menge der Positionen des Anziehenden.

Ein *Halbzug* ist die Gesamtheit aller Aktionen eines Spielers. Unterschieden werden dabei die Halbzüge H_A des Anziehenden und die Halbzüge H_N des Nachziehenden. Halbzüge des Anziehenden überführen eine Position p des Anziehenden in eine Position p', in der der Nachziehende am Zug ist oder das Spiel beendet ist:

$$H_A \subset P_A \times (P_N \cup P_E) \quad \text{mit} \quad (p, p') \in H_A.$$

Auf analoge Weise sind Halbzüge H_N des Nachziehenden definiert. Die Mengen der Halbzüge des Anziehenden und des Nachziehenden werden zur Menge H aller Halbzüge zusammengefaßt:

$$H = H_A \cup H_N.$$

Ein Zug besteht aus einem Halbzug des Anziehenden und, wenn das Spiel noch nicht beendet ist, dem darauffolgenden Halbzug des Nachziehenden.

Bei Kalah kann ein Halbzug aus mehreren Teilen bestehen. Die Menge der *Teilzüge* T enthält die kleinsten Aktionen, die eine Position p in eine andere Position p' überführen. Wie bei den Halbzügen werden auch hier Teilzüge T_A des Anziehenden

$$T_A \subset P_A \times P \quad \text{mit} \quad (p, p') \in T_A$$

und analog definierte Teilzüge T_N des Nachziehenden unterschieden. Es gilt ebenfalls:

$$T = T_A \cup T_N.$$

Eine Position p' heißt *durch einen Halbzug erreichbare Folgeposition von* p, wenn $(p, p') \in H$. Analog läßt sich der Begriff der Folgeposition auf der Grundlage von Teilzügen definieren. Im folgenden bezieht er sich, wenn nicht anders beschrieben, auf durch Halbzüge erreichbare Positionen.

Der *Spielbaum* eines Spiels ist ein gerichteter Graph[10]. Seine Wurzel entspricht der Startposition des Spiels. Eine Kante (k, k') verbindet zwei Knoten k und k', die den Positionen p bzw. p' entsprechen, genau dann, wenn (p, p') ein Halbzug ist. Analog läßt sich ein Spielbaum definieren, dessen Kanten Teilzüge sind. Der Spielbaum ist ein Wurzelbaum, d. h. kein Knoten läßt sich, ausgehend von der Wurzel, über verschiedene Kanten erreichen. Damit gibt es für jede Stellung, die Folgeposition mehrerer Stellungen ist, auch mehrere Knoten. Für jede Kante (k, k') heißt k *Vater von* k' und k' *Sohn*

[10] zu den hier verwendeten graphentheoretischen Begriffen vgl. [NS96]

von k. Ein *Terminalknoten* ist ein Knoten ohne Sohn. Jeder Terminalknoten im Spielbaum entspricht einer Endposition. Die *Tiefe des Spielbaums* ist die durchschnittliche Länge der Spiele. Als *Verzweigungsbreite des Spielbaums* wird die durchschnittliche Anzahl der Folgepositionen der Stellungen, in denen der Anziehende oder der Nachziehende am Zug ist, bezeichnet. Sie kann entweder als Konstante oder als Funktion der bereits gespielten Halbzüge angegeben werden.

Die Komplexität eines Spiels läßt sich auf verschiedene Arten definieren. Dabei werden grundsätzlich *state-space complexity* (dt. *Zustandsraumkomplexität*) und *game-tree complexity*[11] (dt. *Spielbaumkomplexität*) unterschieden.

Zustandsraumkomplexität ist die Anzahl der verschiedenen im Spiel erreichbaren Stellungen. Eine obere Schranke für dieses Maß ist bei Kalah die Aufteilung der 72 Bohnen auf die 14 Mulden ohne Unterscheidung der Bohnen jeweils für beide Spieler. Die Aufteilung der Bohnen entspricht einer Kombination 72. Ordnung von 14 Elementen mit Wiederholung[12]. Damit ergibt sich für die Anzahl der Positionen (für den Anziehenden und den Nachziehenden zuzüglich von maximal 73 theoretisch möglichen Endpositionen):

$$2 \cdot \binom{14+72-1}{72} + 73 = 2 \cdot \frac{(14+72-1)!}{72! \cdot (14-1)!} + 73 = 2 \cdot \frac{85!}{72! \cdot 13!} + 73 \approx 1,5 \cdot 10^{15}.$$

Spielbaumkomplexität ist die Anzahl der Terminalknoten im Spielbaum bzw. die Anzahl der möglichen verschiedenen Spiele. Die Bestimmung einer Näherung dieses Maßes ist für komplexe Spiele nur experimentell möglich. Hierzu ist die Tiefe und die Verzweigungsbreite des Spielbaums zu ermitteln. Die Tiefe des Spielbaums von Kalah liegt bei 40 Halbzügen. Seine Verzweigungsbreite ist rund 10, wobei die Anzahl der Folgepositionen je nach Stellung stark schwanken kann, wodurch die folgende Berechnung sehr unzuverlässig ist. Die Ergebnisse der Experimente, die zu den hier genannten Werten führten, sind im Anhang B.1 dargestellt. Die Anzahl der Terminalknoten des Spielbaums beträgt damit näherungsweise 10^{40}.

Eine Position p wird als *Gewinnposition für den Anziehenden* bezeichnet, wenn entweder

- $p \in P_E$ und der Anziehende gewonnen hat,

[11] vgl. [All94], S. 158ff
[12] vgl. [RS92] S. 465

- $p \in P_N$ und jede Folgeposition von p eine Gewinnposition für den Anziehenden ist oder
- $p \in P_A$ und mindestens eine Folgeposition von p eine Gewinnposition für den Anziehenden ist.

In einer Gewinnposition für den Anziehenden kann der Nachziehende bei fehlerlosem Spiel des Anziehenden dessen Sieg nicht verhindern. Für den Nachziehenden ist dieser Begriff analog definiert.

In spielenden Computerprogrammen heißt der Teil des Algorithmus, der für eine Position p alle Folgepositionen p' mit $(p, p') \in H$ bestimmt, *Halbzuggenerator*. Analog wird der *Teilzuggenerator* definiert.

Ein weiterer Teil des Algorithmus wird *Stellungsbewerter* genannt. Er berechnet die *Bewertungsfunktion*, die die *Bewertung* w einer Position p angibt: $w = f(p)$. Die Bewertungsfunktionen werden in dieser Arbeit als Formel mit mehreren alternativen Bedingungen angegeben. Zur einfacheren Notation wurde darauf verzichtet, daß diese Bedingungen sich gegenseitig ausschließen müssen. Sie sind vielmehr so von oben nach unten zu lesen, daß eine Alternative nur dann berechnet wird, wenn die vorher angegebenen Bedingungen nicht gelten.

Während eines Spiels stehen beide Spieler vor der Aufgabe, einen Halbzug auszuführen. Wenn ein menschlicher Spieler zu diesem Zweck die Stellung analysiert, geht er häufig intuitiv vor, erinnert sich an ähnliche Stellungen und bezieht sein Wissen über den Gegner ein. Ein Computerprogramm hingegen führt im allgemeinen die folgenden zwei Schritte durch:

1. Es benutzt die Bewertungsfunktion f, um alle Folgepositionen p' von p zu bewerten.
2. Es wählt mit Hilfe einer *Auswahlstrategie* eine dieser Folgepositionen und führt die Aktionen aus, die zu ihr führen.

Kapitel 2

Der Minimax-Algorithmus

Das Ziel eines rational entscheidenden Spielers ist es, das Spiel möglichst hoch oder so schnell wie möglich zu gewinnen. Den Sieg kann er erreichen, wenn die aktuelle Stellung eine Gewinnposition ist. Dazu muß er lediglich den Halbzug auswählen, der wieder zu einer Gewinnposition führt. Diese existiert, weil ansonsten die aktuelle Stellung keine Gewinnposition wäre.

Die Schwierigkeit besteht in der Praxis darin zu entscheiden, welche Position eine Gewinnposition ist. Für komplexe Spiele kann diese Entscheidung im allgemeinen nicht mit Sicherheit getroffen werden. Deshalb soll die Bewertungsfunktion das Computerprogramm dabei unterstützen, den wahrscheinlichen Spielausgang, ausgehend von den Folgepositionen, vorherzusagen.

Eine gute Bewertungsfunktion eines Zwei-Personen-Spiels besitzt folgende Eigenschaften:

- Sie ist dazu in der Lage, mehrere Folgepositionen nach der Größe des Vorteils des Anziehenden bzw. des Nachziehenden zu unterscheiden. Das wichtigste Kriterium ist dabei, welcher Spieler das Spiel wahrscheinlich gewinnen wird. Untergeordnete Merkmale können die voraussichtliche Höhe des Sieges und die Anzahl der Züge bis zum Spielende sein.

- Sie sollte einfach berechenbar sein, weil den Spielern nur eine begrenzte Zeit – in jedem Fall aus praktischen Gründen, in Turnierspielen mitunter auch exakt festgelegt – für ihre Entscheidung zur Verfügung steht.

In diesem Kapitel wird mit dem Minimax-Algorithmus ein Algorithmus zur Berechnung einer Bewertungsfunktion beschrieben, der von der Voraussetzung rational

entscheidender Spieler ausgeht. Er bildet die Grundlage des im Kapitel 3 vorgestellten fallbasierten Algorithmus.

2.1 Perfekte Stellungsbewertung

Für Endpositionen kann direkt entschieden werden, welcher Spieler gewonnen hat und wie hoch der Sieg ausgefallen ist. Für sie ist durch die Regeln des jeweiligen Spiels die *Standardbewertungsfunktion* f_S mit folgender Eigenschaft definiert:

$f_S(p) > 0$ für alle $p \in P_E$, in denen der Anziehende gewonnen hat,
$f_S(p) = 0$ für alle $p \in P_E$, in denen das Spiel unentschieden ausgegangen ist und
$f_S(p) < 0$ für alle $p \in P_E$, in denen der Nachziehende gewonnen hat.

Um Stellungen perfekt zu bewerten, in denen das Spiel noch nicht beendet ist, müssen deren Folgepositionen betrachtet werden. Wenn diese Stellungen auch keine Endpositionen sind, dann müssen deren Folgepositionen einbezogen werden. Dieser Vorgang muß solange fortgesetzt werden, bis alle Spielausgänge ausgewertet wurden. In den folgenden Ausführungen wird hierfür der Begriff *Suche im Spielbaum* oder *Baumsuche* verwendet. Die Anzahl der – ausgehend von der aktuellen Position – zu durchsuchenden Halbzüge heißt *Suchtiefe*. Wenn der Spielbaum, wie oben beschrieben, vollständig bis zu allen Terminalknoten durchsucht wird, dann entspricht der Zeitaufwand für diesen Algorithmus der Spielbaumkomplexität des Spiels. Bei einer Verzweigungsbreite b des Spielbaums und einer Suchtiefe t, die gleich der Tiefe des Spielbaums ist, hat der Algorithmus eine Zeitkomplexität der Ordnung $O(b^t)$.

Als rational entscheidender Spieler wird der Anziehende bei der Auswahl der Folgepositionen versuchen, die Bewertungsfunktion zu maximieren, während der Nachziehende versuchen wird, sie zu minimieren. Daher wird dieser Algorithmus *Minimax-Algorithmus*[1] (2.1) genannt.

Der Minimax-Algorithmus ermöglicht damit die Auswahl desjenigen Halbzugs, der bei bestmöglicher Spielweise des Gegners zum besten Ergebnis für den ziehenden Spieler führt.

Dafür berechnet er die *perfekte Bewertungsfunktion* $f_P(p)$ einer Position $p \in P$:

$$f_P(p) = \begin{cases} f_S(p) & \text{, falls } p \in P_E, \\ \max(\{f_P(p_i) | (p, p_i) \in H_A\}) & \text{, falls } p \in P_A, \\ \min(\{f_P(p_i) | (p, p_i) \in H_N\}) & \text{, falls } p \in P_N. \end{cases}$$

[1] vgl. [Gin93] S. 88ff

Schritt 1: Sei k_0 ein Knoten im Spielbaum, der der zu bewertenden Stellung p_0 entspricht. Weiterhin soll im folgenden p stets diejenige Position sein, die dem jeweiligen Knoten k entspricht.

Setze $k := k_0$, $t := 1$ und $w[1] := \begin{cases} f_S(p) & \text{, falls } p \in P_E, \\ -\infty & \text{, falls } p \in P_A, \\ +\infty & \text{, falls } p \in P_N. \end{cases}$

Schritt 2: Hat k einen unmarkierten Sohn k'?

Ja: Setze $k := k'$, $t := t + 1$ und $w[t] := \begin{cases} f_S(p) & \text{, falls } p \in P_E, \\ -\infty & \text{, falls } p \in P_A, \\ +\infty & \text{, falls } p \in P_N. \end{cases}$

Nein: Markiere k.

Wenn $t > 1$ und k' der Vater von k ist, dann setze $k := k'$, $t := t - 1$ und $w[t] := \begin{cases} \max(\{w[t], w[t+1]\}) & \text{, falls } p \in P_A, \\ \min(\{w[t], w[t+1]\}) & \text{, falls } p \in P_N. \end{cases}$

Wenn k_0 markiert ist, dann setze $f_P(p_0) := w[1]$ und beende die Berechnung.

Schritt 3: Gehe zu Schritt 2.

Algorithmus 2.1: Der Minimax-Algorithmus zur perfekten Bewertung einer Position

Wenn es mehrere Folgepositionen der aktuellen Position mit der besten Bewertung gibt, dann sollte der Spieler unter diesen zufällig auswählen. Andere Auswahlstrategien erscheinen mangels zusätzlicher Informationen nicht sinnvoll.

Anwendung

In die Standard-Bewertungsfunktion sollen außer der Information über den Sieger des Spiels weitere Kriterien eingehen. Im Schach würde vielleicht ein Sieg durch ein schnelles Matt bevorzugt werden. Bei Kalah ist die Höhe des Sieges oder der Niederlage, die sich in der Differenz der Gewinnmuldeninhalte ausdrückt, die interessanteste Information.

Für Endpositionen sind der Sieger und die Höhe des Sieges bereits eindeutig bestimmt. Eine sinnvolle Standard-Bewertungsfunktion für eine solche Position

$$p = ((0, \ldots, 0, gm_A, 0, \ldots, 0, gm_N), s),$$

in der bis auf die Gewinnmulden alle Mulden leer sind, ist die Gewinnmuldendifferenz:

$$f_S(p) = gm_A - gm_N \quad \text{für} \quad p \in P_E.$$

2.2 Minimax mit beschränkter Suchtiefe

Der zeitliche Aufwand für die oben beschriebene Suche im Spielbaum steigt exponentiell mit der Suchtiefe an. Für die meisten Spiele ist daher keine perfekte Stellungsbewertung möglich. Der einfachste Ansatz, den Aufwand zu begrenzen, besteht darin, die Suchtiefe auf die ersten t_{max} Halbzüge zu beschränken.

Die Standardbewertungsfunktion wurde bisher nur für Endpositionen erklärt. Um sie auf alle Positionen in der maximalen Suchtiefe t_{max} anwenden zu können, muß ihr Definitionsbereich auf alle Positionen $p \in P$ erweitert werden. Damit sie zum Bewerten von Stellungen verwendet werden kann, in denen das Spiel noch nicht beendet ist, muß sie aussagekräftig genug sein, um das wahrscheinliche Spielergebnis vorhersagen zu können. Ihre Definition für Positionen des Anziehenden und des Nachziehenden ist stark vom betrachteten Spiel abhängig. Bei Schach wird z. B. häufig ein Materialvergleich zur Bewertung verwendet. Dabei erhalten die Figuren unterschiedliche Werte: der König das höchste mögliche, denn mit seinem Verlust ist auch das Spiel verloren, die Dame 9, jeder Turm 5, jeder Läufer und jeder Springer 3 und jeder Bauer 1[2]. Die Differenz des summierten Figurenwerte des Anziehenden und des Nachziehenden ergibt die Bewertung der Stellung, weil beim Schach ein Materialvorteil auch als vorteilhaft für den Spielsieg angesehen wird.

Der auf diese Weise definierte *tiefenbeschränkte Minimax-Algorithmus* (2.2) berechnet die folgende Bewertungsfunktion:

$$f_T(p,t) = \begin{cases} f_S(p) & \text{, falls } p \in P_E \quad \text{oder} \quad t = t_{max}, \\ \max(\{f_T(p_i, t+1) | (p, p_i) \in H_A\}) & \text{, falls } p \in P_A, \\ \min(\{f_T(p_i, t+1) | (p, p_i) \in H_N\}) & \text{, falls } p \in P_N. \end{cases}$$

Die gleiche Funktion kann auch mit Hilfe von Teilzügen des Anziehenden und des Nach-

[2] vgl. [Why98] S. 54f

Schritt 1: Sei k_0 ein Knoten im Spielbaum, der der zu bewertenden Stellung p_0 entspricht. Weiterhin soll im folgenden p stets diejenige Position sein, die dem jeweiligen Knoten k entspricht. Die maximale Suchtiefe sei t_{max}.

Setze $k := k_0$, $t := 1$ und $w[1] := \begin{cases} f_S(p) & \text{, falls } p \in P_E \quad \text{oder} \quad t = t_{max}, \\ -\infty & \text{, falls } p \in P_A, \\ +\infty & \text{, falls } p \in P_N. \end{cases}$

Schritt 2: Hat k einen unmarkierten Sohn k' und ist $t < t_{max}$?

Ja: Setze $k := k'$, $t := t + 1$ und $w[t] := \begin{cases} f_S(p) & \text{, falls } p \in P_E \text{ oder } t = t_{max}, \\ -\infty & \text{, falls } p \in P_A, \\ +\infty & \text{, falls } p \in P_N. \end{cases}$

Nein: Markiere k.

Wenn $t > 1$ und k' der Vater von k ist, dann setze $k := k'$, $t := t - 1$ und $w[t] := \begin{cases} \max(\{w[t], w[t+1]\}) & \text{, falls } p \in P_A, \\ \min(\{w[t], w[t+1]\}) & \text{, falls } p \in P_N. \end{cases}$

Wenn k_0 markiert ist, dann setze $f_T(p_0, t_{max}) := w[1]$ und beende die Berechnung.

Schritt 3: Gehe zu Schritt 2.

Algorithmus 2.2: Der tiefenbeschränkte Minimax-Algorithmus

ziehenden definiert werden. Auf diese Darstellung wird in Kapitel 3 zurückgegriffen:

$$f_T(p,t) = \begin{cases} f_S(p) & \text{, falls } p \in P_E \quad \text{oder} \quad t = t_{max}, \\ \max(\{f_T(p_i,t) | (p,p_i) \in T_A\}) & \text{, falls } p \in P_A \quad \text{und} \quad p_i \in P_A, \\ \min(\{f_T(p_i,t) | (p,p_i) \in T_N\}) & \text{, falls } p \in P_N \quad \text{und} \quad p_i \in P_N, \\ \max(\{f_T(p_i,t+1) | (p,p_i) \in T_A\}) & \text{, falls } p \in P_A \quad \text{und} \quad p_i \in P_N, \\ \min(\{f_T(p_i,t+1) | (p,p_i) \in T_N\}) & \text{, falls } p \in P_N \quad \text{und} \quad p_i \in P_A. \end{cases}$$

Die in der maximalen Suchtiefe im Spielbaum angetroffenen Knoten sind bezüglich der Suche ebenfalls Terminalknoten.

Anwendung

Aufgrund der Größe des Spielbaums ist es bei Kalah nicht möglich, tatsächlich jede Fortsetzung bis zum Spielende zu betrachten.

Für Positionen, in denen das Spiel noch nicht beendet ist, stellt die Gewinnmuldendifferenz nur ein Zwischenergebnis dar, da nicht bekannt ist, wieviele der restli-

chen Bohnen welcher Spieler gewinnen wird. Dennoch ist es sinnvoll, die Standard-Bewertungsfunktion auf alle Positionen zu erweitern:

$$f_S(p) = gm_A - gm_N \quad \text{für} \quad p \in P.$$

Zum einen sind für die meisten Positionen keine Informationen über den weiteren Spielverlauf bekannt, die in die Funktion einfließen könnten. Zum anderen ist diese einfache Bewertungsfunktion bereits sehr aussagekräftig, da während eines Spiels erreichte Gewinne in den Gewinnmulden erhalten bleiben und nur noch um die Bohnen gespielt wird, die in den anderen Mulden der Spieler verblieben sind.

2.3 Minimax mit α-β-Baumstutzen

Beim Minimax-Algorithmus leisten einige Teilbäume bzw. ihre Stellungsbewertungen keinen Informationsbeitrag. Ursache dafür ist das abwechselnde Maximieren und Minimieren der Bewertungen in aufeinanderfolgenden Suchtiefen. Zur Verringerung des Aufwands ist es daher sinnvoll, die entsprechenden Teilbäume nicht zu durchsuchen. Der Name des *α-β-pruning*[3] (dt. *α-β-Stutzen*) ist von der Bezeichnung für die Vergleichswerte des Anziehenden und des Nachziehenden abgeleitet, die verwendet werden, um zu überprüfen, wann die Suche beschnitten werden kann. Es ist eine Form des *Rückwärtsbeschneidens* und im Vergleich zum tiefenbeschränkten Minimax-Algorithmus ohne Informationsverlust einsetzbar.

Um diese Art des Baumstutzens durchführen zu können, muß eine Halbzugalternative und für eine zweite Halbzugalternative ein darauffolgender Halbzug des Gegners bewertet sein. Wenn die Bewertung dieses Gegenzugs bereits so gut ist, daß die zweite Halbzugalternative mit Sicherheit schlechter bewertet werden wird als die erste Halbzugalternative, dann erübrigt sich die Bewertung der restlichen Gegenzüge. Abbildung 2.1 illustriert diesen Vorgang.

Für das Baumstutzen ist es nicht notwendig, daß der Halbzug des Gegners direkt auf die zweite zu bewertende Halbzugalternative folgt. Abbildung 2.2 zeigt einen sogenannten *deep cutoff*[4] (dt. *tiefer Schnitt*) zwei Halbzüge tiefer im Spielbaum.

Der Minimax-Algorithmus mit α-β-Baumstutzen wird als Algorithmus 2.3 dargestellt. Er berechnet die gleiche Bewertungsfunktion wie der tiefenbeschränkte Minimax-Algorithmus.

[3] vgl. dazu [Gin93] S. 95ff und [RN95] S. 129ff

[4] [Gin93] S. 98

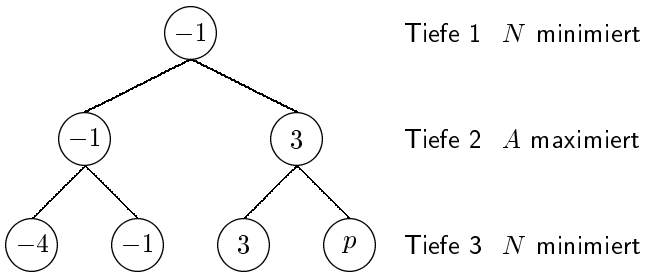

Abbildung 2.1: Beispiel für eine Minimax-Suche mit α-β-Stutzen. Entweder ist $3 > f_T(p)$, dann wählt der Anziehende in Tiefe 2 die mit „3“ bewertete Stellung aus Tiefe 3, indem er maximiert. Oder es ist $f_T(p) \geq 3$, dann wählt der Nachziehende in Tiefe 1 die mit „-1“ bewertete Stellung aus Tiefe 2, indem er minimiert. In beiden Fällen hat die Bewertung von p keinen Einfluß auf die Bewertung der Stellung in Tiefe 1.

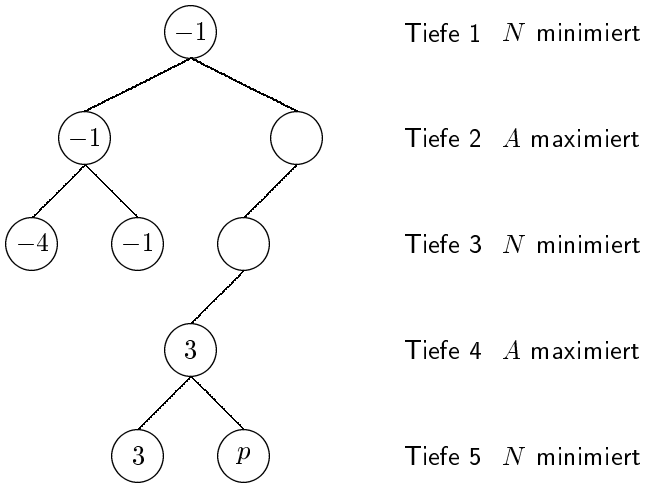

Abbildung 2.2: Beispiel für einen tiefen α-β-Schnitt. Entweder ist $3 > f_T(p)$, dann wählt der Anziehende in Tiefe 4 die mit „3“ bewertete Stellung aus Tiefe 5, indem er maximiert. Oder es ist $f_T(p) \geq 3$, dann wählt der Nachziehende in Tiefe 1 die mit „-1“ bewertete Stellung aus Tiefe 2, indem er minimiert. In beiden Fällen hat die Bewertung von p keinen Einfluß auf die Bewertung der Stellung in Tiefe 1.

Schritt 1: Sei k_0 ein Knoten im Spielbaum, der der zu bewertenden Stellung p_0 entspricht. Weiterhin soll im folgenden p stets diejenige Position sein, die dem jeweiligen Knoten k entspricht. Die maximale Suchtiefe sei t_{max}.
Setze $k := k_0$, $t := 1$, $\alpha := -\infty$, $\beta := +\infty$ und

$$w[1] := \begin{cases} f_S(p) & \text{, falls } p \in P_E \quad \text{oder} \quad t = t_{max}, \\ -\infty & \text{, falls } p \in P_A, \\ +\infty & \text{, falls } p \in P_N. \end{cases}$$

Schritt 2: Hat k einen unmarkierten Sohn k' ist $t < t_{max}$ und gilt $(p \in P_A \rightarrow w[t] < \beta) \wedge (p \in P_N \rightarrow w[t] > \alpha)$?

Ja: Setze $k := k'$, $t := t + 1$ und $w[t] := \begin{cases} f_S(p) & \text{, falls } p \in P_E \text{ oder } t = t_{max}, \\ -\infty & \text{, falls } p \in P_A, \\ +\infty & \text{, falls } p \in P_N. \end{cases}$

Nein: Markiere k.
Wenn $t > 1$ und k' der Vater von k ist, dann setze $k := k'$, $t := t - 1$,

$$w[t] := \begin{cases} \max(\{w[t], w[t+1]\}) & \text{, falls } p \in P_A, \\ \min(\{w[t], w[t+1]\}) & \text{, falls } p \in P_N. \end{cases},$$

$\beta := min(\{+\infty, w[t-2n-1] | n = 0, 1, \ldots \text{ und } t - 2n - 1 > 0\})$, falls $p \in P_A$,
$\alpha := max(\{-\infty, w[t-2n-1] | n = 0, 1, \ldots \text{ und } t - 2n - 1 > 0\})$, falls $p \in P_N$.
Wenn k_0 markiert ist, dann setze $f_T(p_0, t_{max}) := w[1]$ und beende die Berechnung.

Schritt 3: Gehe zu Schritt 2.

Algorithmus 2.3: Der Minimax-Algorithmus mit α-β-Baumstutzen

Der Aufwand für das Baumstutzen ist sehr gering, da wenige Vergleiche in den Knoten genügen, um für oder gegen ein Stutzen zu entscheiden. Sein Nutzen läßt sich durch die Anzahl der Positionen, die bei der Suche ausgelassen werden können, abschätzen. Im schlechtesten Fall müssen alle Positionen durchlaufen werden. Für die Approximation der Zahl der im besten Fall zu durchsuchenden Stellungen sind weitergehende Überlegungen notwendig.

Angenommen, die Verzweigungsbreite b des Spielbaums ist konstant. Um alle theoretisch möglichen Schnitte durchführen zu können, müssen die besten Vergleichswerte zuerst berechnet werden. Das geschieht, wenn die am höchsten bewertete Folgeposition der Stellungen des Anziehenden und die am niedrigsten bewertete Folgeposition der Stellungen des Nachziehenden als erste untersucht werden. Damit der Spielbaum

gestutzt werden kann, muß lediglich eine gute Reaktion des Gegners auf eine Halbzugalternative gefunden werden. Danach können die Knoten der anderen Gegenzüge gestutzt werden. Um diesen einen Halbzug zu bewerten, müssen alle seine Folgepositionen bewertet werden. Dazu reicht es im besten Fall wiederum aus, einen besten Gegenzug je Folgeposition zu finden etc. Dieser beste Fall tritt für die Knoten ein, die als $2., \ldots, b$-ter Sohn eines Knotens besucht werden. Für den zuerst besuchten Sohn eines Knotens müssen hingegen immer alle Folgepositionen besucht werden. Die Anzahl der Söhne je Knoten ist also typischerweise (in den nicht zuerst durchsuchten Teilbäumen) abwechselnd 1 und b (siehe dazu Abbildung 2.3).

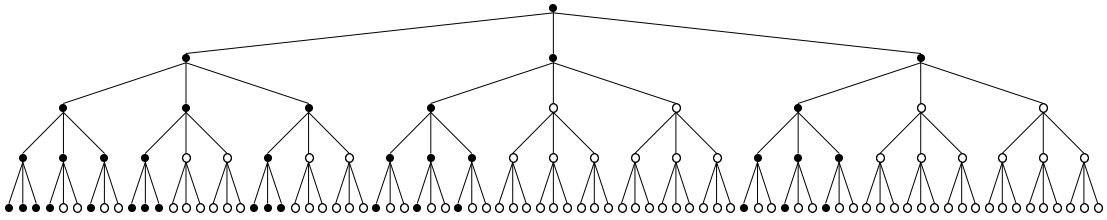

Abbildung 2.3: Beispiel für eine Baumsuche mit allen möglichen α-β-Schnitten bei konstanter Verzweigungsbreite 3 und Suchtiefe 5. Unter der Voraussetzung, daß die Folgepositionen günstig sortiert sind, müssen nur die durch gefüllte Kreise dargestellten Positionen durchsucht werden, während die durch leere Kreise dargestellte Positionen ignoriert werden können.

Damit wächst die Anzahl der Stellungen, die betrachtet werden müssen, nicht mehr mit jedem Halbzug, sondern durchschnittlich nur noch mit jedem zweiten Halbzug auf das b-fache. Anders ausgedrückt, hat der Algorithmus eine Zeitkomplexität der Ordnung $O(b^{\frac{t}{2}})$. Die genaue Zahl k der Knoten in der Tiefe t einer Baumsuche zur Bewertung einer Position in Tiefe 1 beträgt

$$k = \begin{cases} 2 \cdot b^{\frac{t-1}{2}} - 1 & \text{, falls } t \text{ ungerade ist,} \\ b^{\frac{t}{2}} + b^{\frac{t-2}{2}} - 1 & \text{, falls } t \text{ gerade ist.}^5 \end{cases}$$

Bisher wurden alle Folgepositionen der aktuellen Stellung *unabhängig bewertet*. Dabei entsprachen die Folgepositionen den Wurzeln der betrachteten Teilbäume, und die Suchtiefe lief jeweils von 1 bis t_{max}. Beim Minimax-Algorithmus mit α-β-Stutzen ist es aber sinnvoll, eine Baumsuche durchzuführen, die bei der aktuellen Stellung in Tiefe 0

[5] vgl. [SD69]. Ein Beweis für diese Aussage wird auf den S. 201ff geführt.

startet und die Folgepositionen *abhängig bewertet*. Auf diese Weise kann das Baumstutzen statt nach Tiefe 2 bereits nach Tiefe 1 einsetzen. Abbildung 2.4 demonstriert diesen Vorteil.

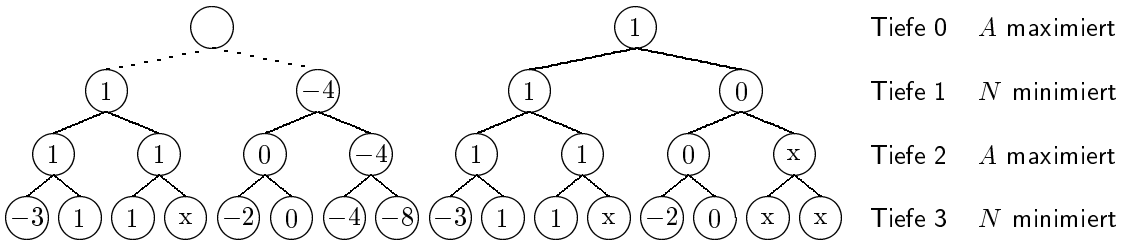

Abbildung 2.4: Beispiel für unabhängiges und abhängiges Bewerten der Folgepositionen der aktuellen Stellung. Links im Bild werden für die beiden Stellungen in Tiefe 1 unabhängige Baumsuchen durchgeführt. Rechts im Bild hingegen wird im gleichen Teilbaum eine Baumsuche in der aktuellen Stellung in Tiefe 0 gestartet. Beim abhängigen Bewerten der Folgepositionen kann der Spielbaum ohne Informationsverlust stärker beschnitten werden. Die Knoten, die nicht durchsucht werden müssen, sind mit einem x gekennzeichnet.

Zu beachten ist hierbei allerdings, daß die Baumsuche nicht mehr nur zur Bewertung der Folgepositionen verwendet wird, sondern zur Auswahl der am besten bewerteten Folgeposition. Deshalb muß die Bedingung für das Baumstutzen geringfügig verschärft werden: Söhne von gegnerischen Stellungen dürfen nur dann beschnitten werden, wenn für einen anderen Sohn dieser gegnerischen Stellung eine für den Gegner bessere Bewertung ermittelt wurde als für eine bereits bewertete gegnerische Position. Anders als beim unabhängigen Bewerten der Folgepositionen ist bei Übereinstimmung dieser Vergleichswerte das Stutzen nicht erlaubt, wie Abbildung 2.5 illustriert.

Um die Anzahl der möglichen Schnitte beim α-β-Stutzen zu erhöhen, ist es sinnvoll, die Knoten zu sortieren, so daß die aussichtsreichsten Knoten zuerst bewertet werden. Die dafür notwendigen Informationen, die Bewertungen der Positionen, werden erst durch die Baumsuche erhalten. Dennoch kann mit Hilfe des *iterative deepening*[6] (dt. *iteratives Absteigen*) sortiert werden. Bei diesem Verfahren werden mehrere Baumsuchen mit wachsender Suchtiefe durchgeführt (siehe Abbildung 2.4). Für das Baumstutzen beim Minimax-Algorithmus ergibt sich damit die Möglichkeit, die Positionen mit Hilfe ihrer Bewertungen aus der vorangegangenen Suche zu sortieren.

Der Einsparung an Rechenzeit, die durch das iterative Absteigen beim α-β-Baum-

[6] vgl. [SD69] S. 195ff

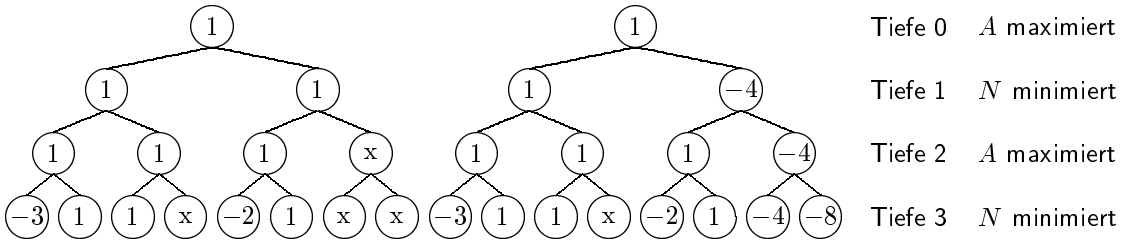

Abbildung 2.5: Beispiel für das α-β-Stutzen beim abhängigen Bewerten. Dieser Teilbaum ist im Vergleich zu Abbildung 2.4 nur leicht verändert. Links im Bild wird mit der Bedingung für Baumstutzen beim unabhängigen Bewerten getestet, ob die Suche beschnitten werden darf. Das führt dazu, daß beide Halbzugalternativen in Tiefe 1 irrtümlich gleich gut bewertet werden. Rechts im Bild wird dagegen mit der für das abhängige Bewerten modifizierten Bedingung getestet. Deshalb wird der mit -4 bewerteten Knoten in Tiefe 2 erkannt. Die mit einem „x“ gekennzeichneten Knoten wurden bei den Suchen beschnitten.

Schritt 1: Wähle eine Anfangstiefe t_{min} mit $t_{min} + n \cdot s = t_{max} \quad (n = 1, 2, \ldots)$. Setze $t = t_{min}$.

Schritt 2: Führe eine Baumsuche mit Tiefe t durch.

Schritt 3: Erhöhe t um die Schrittweite s. Wenn $t > t_{max}$ ist, dann beende die Berechnung.

Schritt 4: Wenn schon eine Suche durchgeführt wurde, dann sortiere die Knoten des Spielbaums nach ihren Stellungsbewertungen der letzten Suche. Sortiere Folgepositionen von Stellungen des Anziehenden nach der Größe, Folgepositionen von Stellungen des Nachziehenden umgekehrt.

Schritt 5: Gehe zu Schritt 2.

Algorithmus 2.4: Minimax mit iterativem Absteigen. Um bei einer Baumsuche mit der maximalen Suchtiefe t_{max} eine möglichst große Anzahl von α-β-Schnitten vornehmen zu können, wird die Reihenfolge, in der die Knoten während der Baumsuche betrachtet werden, aufgrund ihrer Bewertungen aus einer Suche mit niedrigerer Tiefe festgelegt.

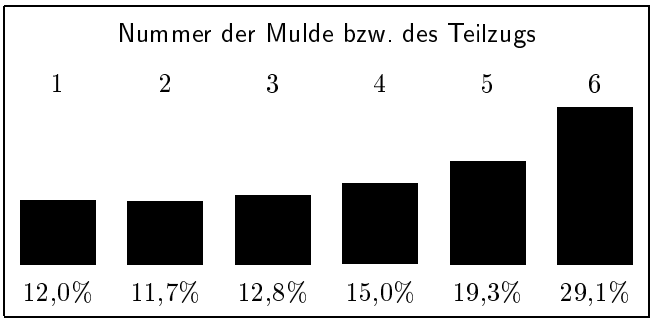

Abbildung 2.6: Häufigkeit, mit der die verschiedenen Teilzüge ausgewählt werden. Die Werte wurden mit Hilfe einer Testreihe ermittelt, deren Daten in Tabelle B.2 im Anhang B.1 aufgeführt sind.

stutzen erreicht wird, stehen die Kosten für die Baumsuchen mit niedrigerer Tiefe und für das Sortieren gegenüber. Um mit geringem Aufwand aussagekräftige Bewertungen für das Sortieren zu erhalten, sind die Fragen zu beantworten, welche Anfangstiefe und welche Schrittweite am sinnvollsten sind.

Anwendung

Eine Anwendung des Baumstutzens ist auch bei Kalah unerläßlich. Der Spielbaum von Kalah hat jedoch keine konstante Verzweigungsbreite. Vielmehr entsteht die durchschnittlich Breite von 10 durch sehr unterschiedlich viele Folgepositionen der Stellungen. In jeder Position gibt es höchstens 6 Teilzüge, weil ein Teilzug der Auswahl einer der 6 Mulden entspricht. Aber einige Halbzüge bestehen aus mehreren Teilzügen. Je größer die Anzahl der Teilzüge eines Halbzuges ist, desto größer ist auch die Verzweigungsbreite. Da mit jedem Teilzug, der einen weiteren eigenen Teilzug nach sich zieht, ein Gewinn von einer Bohne einhergeht, und außerdem Mulden geleert werden, was Gewinnzüge ermöglichen kann, sind diese sehr breiten Abschnitte des Spielbaums häufig sehr gut bewertet und damit kaum zu beschneiden.

Kalah ermöglicht jedoch auch eine Vorsortierung in Form einer *statischen Reihenfolge*, um die Anzahl der α-β-Schnitte zu erhöhen. Die Häufigkeit, mit der die einzelnen Mulden als Teilzüge gewählt werden, zeigt signifikante Abweichungen von der Gleichverteilung. Abbildung 2.6 verdeutlicht diesen Fakt.

Die Tendenz, daß die letzten Mulden häufiger gewählt werden als die ersten, besteht unabhängig von der Tiefe der Baumsuche. Wenn die Teilzüge bei der Baumsuche in der Reihenfolge ihrer Häufigkeit ausgewertet werden, sollte sich die Anzahl der zu

durchsuchenden Positionen stark verringern. Der Erfolg bleibt bei diesem Ansatz jedoch aus, weil – wie oben beschrieben – die breiten Teilbäume oft gute Bewertungen enthalten.

Bisher wurde nur versucht, die Anzahl der möglichen Schnitte zu erhöhen. Bei Kalah ist es aber nötig, die breiten Teilbäume des Spielbaums zu beschneiden. Eine solche *heuristische Sortierung* ist eine Form der dynamischen Bestimmung der Reihenfolge, in der die Knoten durchsucht werden. Bei Kalah empfiehlt sich folgende Sortierung: Zuerst sind Teilzüge zu untersuchen, die einen Gewinn nach sich ziehen, danach diejenigen Teilzüge, die dazu führen, daß der Spieler einen weiteren Teilzug ausführen darf und zuletzt alle anderen Teilzüge. Die Teilzüge, die zu Gewinnen führen, sind nicht notwendigerweise die besten Teilzüge. Diese Reihenfolge ist aber deshalb sinnvoll, weil durch sie schon gute Vergleichswerte für die danach durchzuführenden Suchen in den besonders breiten Teilbäumen existieren. Das ermöglicht ein sehr effektives α-β-Stutzen.

Eine zweite Form der dynamischen Sortierung ist das iterative Absteigen. Seine Ergebnisse bleiben jedoch hinter denen der heuristischen Sortierung zurück. Selbst wenn beide Verfahren auf die Weise miteinander kombiniert werden, daß die einzelnen Baumsuchen beim iterativen Absteigen das heuristische Sortieren nutzen, müssen deutlich mehr Terminalknoten durchsucht werden, wie Tabelle 2.1 zeigt. Die Ursache dafür ist, daß das iterative Absteigen zum Sortieren nur die Bewertung der Knoten, nicht jedoch die Anzahl der zu durchsuchenden Positionen berücksichtigt. Bei den Experimenten wurde für die Schrittweite zur Erhöhung der Suchtiefe 2 und als Anfangstiefe 1, falls t_{max} ungerade ist bzw. 2, falls t_{max} gerade ist, gewählt.

In Anbetracht der guten Ergebnisse und des im Vergleich zum iterativen Absteigen erheblich niedrigeren Aufwands wird das α-β-Stutzen mit heuristischem Sortieren auch in dem im Kapitel 3 vorgestellten fallbasierten Algorithmus verwendet.

2.4 Erweiterungen des Minimax-Algorithmus

Zur Bewertung der Halbzugalternativen eines Spielers wurde der Minimax-Algorithmus eingeführt. Für komplexe Spiele ist es jedoch nicht möglich, den Spielbaum vollständig zu durchsuchen. Die Baumsuche in einer festen maximalen Suchtiefe von t_{max} Halbzügen abzuschneiden, wie in Abschnitt 2.2 vorgeschlagen, führt allerdings zu Problemen, die zusammen mit Lösungsvorschlägen in diesem Abschnitt behandelt werden.

Wahrscheinlich sind nicht alle Zweige des Spielbaums gleich interessant. Die interes-

Suchtiefe in Halbzügen (Anzahl der Spiele)	ohne α-β-Stutzen	mit α-β-Stutzen			
		unabhängiges Bewerten der Folge-positionen	abhängiges Bewerten der Folgepositionen		
			statische Reihenfolge	dynamische Reihenfolge	
				heuristisch	iteratives Absteigen
1 (10000)	10,0	10,0	10,0	10,0	10,0
2 (10000)	79,2	79,2	55,3	46,3	46,2
3 (1000)	1171,3	555,5	430,4	347,1	362,2
4 (1000)	8616,2	3021,7	2143,6	1365,0	1582,5
5 (1000)	149340,5	26589,5	17732,8	9507,5	11317,2
6 (100)	894983,2	101465,4	71843,5	28187,0	38019,1

Tabelle 2.1: Durchschnittlich während der Baumsuche durchsuchte Terminalknoten

santeren Spielfortsetzungen sollten deshalb tiefer durchsucht werden. Dazu müssen die Positionen erkannt werden, bei denen die Suche nicht abgebrochen werden soll. Spielsituationen, in denen die Spieler durch ihre Halbzüge jeweils Vorteile erzielen können, was sich in einer stark schwankenden Bewertung äußert, sind typisch für dieses Problem. Ein Beispiel für eine solche Situation ist ein Schlagabtausch beim Schach, bei dem mit jedem Halbzug eine Figur geschlagen wird. Die Suche mitten in einem solchen taktischen Vorgang abzubrechen, würde dem Spieler, dessen Halbzug zuletzt betrachtet wurde, einen Vorteil zusprechen, der nicht dauerhaft ist. Bei der *quiescence search*[7] (dt. *Suche bis in beruhigte Abschnitte*) wird versucht, dieses Problem zu lösen, indem die Suche so lange fortgesetzt wird, bis sich die Bewertung stabilisiert hat.

Eine zweite Bedingung dafür, die Suchtiefe in einigen Zweigen des Baums zu erhöhen, wird von der Technik der *singular extensions*[8] (dt. *einmalige Erweiterungen*) verwendet. Dabei werden Teilbäume bis in größere Tiefen durchsucht, deren Verzweigungsbreite dadurch sehr gering ist, daß wenige Halbzugalternativen sehr viel besser bewertet sind als alle anderen. Die wenigen Möglichkeiten, beim Schachspiel auf ein Schachgebot zu reagieren, stellen ein gutes Beispiel hierfür dar. Das gilt insbesondere deshalb, weil die gesuchte Reaktion in solchen Situationen wahrscheinlich entscheidend für den weiteren Spielverlauf ist, was ein zusätzlicher Grund ist, die Suchtiefe zu

[7] vgl. [Gin93] S. 93f und [RN95] S. 129
[8] vgl. [Gin93] S. 94

erhöhen.

Es gibt weitere Abweichungen vom Minimax-Algorithmus, die sich mit dem sogenannten *horizon effect*[9] (dt. *Horizonteffekt*) beschäftigen. Dieser Effekt führt dazu, daß eine Halbzugalternative gegenüber einer anderen bevorzugt wird, weil ein guter Zug des Gegners bei ihr hinter dem Suchhorizont, d. h. in einer Tiefe von $t > t_{max}$ Halbzügen liegt, während er in die Bewertung der zweiten Halbzugalternative eingeht. Die Auswirkungen dieses Effekts liegen oft darin, daß sinnlose Züge gewählt werden, die nur den Spielverlauf verzögern, anstatt eine Antwort auf einen drohenden gegnerischen Zug zu finden. Die betrachteten Teilbäume des Suchbaums enthalten dann nur die Gegenzüge auf die sinnlosen Halbzüge, während sich der gute gegnerische Zug in einer Suchtiefe $t > t_{max}$ befindet und damit nicht erkannt wird.

Eine naheliegende aber trotzdem wenig hilfreiche Möglichkeit, den Horizonteffekt abzuschwächen, besteht in einer tieferen Suche im vielversprechendsten Teilbaum. Das Problem ist, daß der Horizonteffekt auf diese Weise möglicherweise zwar erkannt werden kann, aber nicht notwendigerweise aufgelöst wird. Im ungünstigsten Fall, wenn der drohende gegnerische Zug unabwendbar ist, führt dieser Lösungsansatz nur dazu, daß für jede Zugfolge eine tiefere Suche durchgeführt werden muß.

Erfolgversprechender erscheint in diesem Zusammenhang das Anwenden der sogenannten *killer heuristic*[10] (dt. *Verhinderungsheuristik*). Hierbei werden als gut erkannte Züge des Gegners bereits in niedrigen Suchtiefen betrachtet, um eventuell vorhandene Anworten darauf ebenfalls vor dem Suchhorizont zu finden.

In den im Rahmen dieser Arbeit durchgeführten Experimenten spielten die in diesem Abschnitt besprochenen Erweiterungen des Minimax-Algorithmus keine Rolle. Vielmehr lag das Hauptaugenmerk entsprechend der Zielsetzung der Arbeit auf dem Verbessern der Bewertungsfunktion mit Hilfe des fallbasierten Schließens, worauf im nächsten Kapitel eingegangen wird. Allerdings lassen sich durch das dort beschriebene Verfahren ebenfalls Verbesserungen gegenüber dem tiefenbeschränkten Minimax-Algorithmus erreichen.

[9] vgl. [Gin93] S. 94f und [RN95] S. 129
[10] vgl. [Gin93] S. 95

Kapitel 3

Fallbasiertes Lernen von Bewertungsfunktionen

Prinzipiell ermöglicht der im Kapitel 2 behandelte Minimax-Algorithmus eine perfekte Stellungsbewertung. Der Zeitaufwand für eine Suche nach dem besten Zug unter Einbeziehung aller Gegenzüge steigt exponentiell mit der Tiefe der Spielbaumsuche, d. h. mit der Anzahl der aufeinanderfolgenden Halbzüge, an. Die Suche wurde deshalb auf eine Tiefe von wenigen Halbzügen beschränkt. Im Abschnitt 2.4 wurden einige Techniken beschrieben, die sich den dadurch auftretenden Problemen zuwenden. Diese Techniken basierten zum größten Teil darauf, ausgewählte Teilbäume des Spielbaums tiefer zu durchsuchen.

Im folgenden Kapitel wird ein Algorithmus vorgestellt, der diese Probleme auf eine andere Weise zu lösen vermag. Anstatt tiefer zu suchen, verwendet er während der Baumsuche Stellungsbewertungen, die er in der Vergangenheit berechnet hat. Dazu wird der in Abschnitt 2.2 beschriebene tiefenbeschränkte Minimax-Algorithmus entsprechend erweitert. Die Stellungen werden dadurch so bewertet, als ob in einigen Teilbäumen tiefer gesucht würde.

In den einzelnen Abschnitten des Kapitels wird das zugrundeliegende Konzept des fallbasierten Schließens und seine Methoden beschrieben und anschließend die konkrete Umsetzung am Anwendungsbeispiel Kalah erläutert.

3.1 Fälle und fallbasiertes Schließen

Fallbasiertes Schließen ermöglicht es, auf neue Probleme durch Anpassung von Lösungen alter Probleme zu reagieren. In einer ähnlichen Situation ist ein vergleichbares Herangehen, das den Besonderheiten der neuen Situation angepaßt wurde, erfolgversprechend.

Zu einem Fall werden beim fallbasierten Schließen die zu einer Problemsituation gehörenden Informationen zusammengefaßt. Dadurch können einmal gelöste Probleme miteinander oder mit neuen Situationen verglichen werden, was es erlaubt, die Informationen der Fälle auf ihre Verwendbarkeit zu überprüfen.

3.1.1 Aufbau eines Falls

Zu einem *Fall* c gehören

- das zu bearbeitende Problem,
- eine mögliche Lösung und
- die Konsequenzen dieser Lösung.

Er schafft einen Zusammenhang zwischen einer Information und dem Kontext, aus dem sie stammt und in dem das erworbene Wissen anwendbar ist. Je nach Anwendung können Fälle sehr unterschiedlich strukturiert sein. Ihr Aufbau ist davon anhängig, welche Informationen sie bereithalten und wie der Kontext beschrieben werden kann. Grundsätzlich sollten diejenigen Daten in den Fällen enthalten sein, die für die Funktion des fallbasierten Systems notwendig sind. Sie sollten aber den Fall nicht unnötig aufblähen oder seine Struktur ungerechtfertigt verkomplizieren.

Fälle erfassen insbesondere Ausnahmesituationen, die vom zu Erwartenden abweichen. Damit ermöglichen sie neben der Ableitung von Verhalten, wie es auch auf der Grundlage abstrakter Regeln möglich wäre, die Konstruktion von Lösungen, die auf die jeweilige spezielle Situation zugeschnitten sind.

Die Fälle werden in der *Fallbibliothek* oder *Fallbasis FB* gesammelt. Eine Möglichkeit zur Erleichterung des Auffindens verwendbarer Fälle ist, die Anzahl der Fälle in der Fallbibliothek zu begrenzen. Das kann zum Beispiel erreicht werden, indem nicht nur *individuelle Fälle* in der Fallbasis gespeichert werden, sondern *generalisierte Fälle*. Zu einem generalisierten Fall können ähnliche Fälle zusammengefaßt werden, indem von den unterscheidenden Merkmalen abstrahiert wird.

Anwendung

Im Anwendungsbeispiel Kalah gilt für alle Fälle $c \in FB$:

$$c = (p, w, k, u).$$

Die Elemente eines Falls sind:

- eine Stellung $p = ((m_{A1}, \ldots, m_{A6}, gm_A, m_{N1}, \ldots, m_{N6}, gm_N), s)$ mit der Anzahl der Bohnen in jeder Mulde und dem Spieler, der am Zug ist,
- die ganzzahlige Bewertung $w \in \mathbf{Z}$ der Stellung p,
- die Anzahl der Positionen bzw. Knoten des Spielbaums k, die in die Berechnung der Bewertung w eingegangen sind und
- die Anzahl der Verwendungen u des Falls.

Welche Fälle in die Fallbasis aufgenommen werden und auf welche Weise deren Daten gewonnen werden, wird in den folgenden Abschnitten beschrieben.

3.1.2 Verwendung von Fällen

Die naheliegendste Vorstellung von der Fallnutzung ist das Vorschlagen von Lösungen zu einem gegebenen Problem. Im fallbasierten Schließen werden Fälle aber nicht nur dazu benutzt. Zur Evaluierung dieser Lösungen, zum Vorhersagen möglicher Unzulänglichkeiten der Vorschläge und zum Erklären und Beseitigen von Fehlern, die bei der Anwendung der Lösungen auftreten, werden ebenfalls häufig Fälle herangezogen. Schließlich werden interessant erscheinende Fälle in der Fallbibliothek gespeichert, nachdem sie gegebenenfalls mit anderen schon vorhandenen Fällen verglichen und zu generalisierten Fällen kombiniert wurden.

Abbildung 3.1 zeigt schematisch den Zyklus der Prozesse beim fallbasierten Schliessen.

Anwendung

Die Aufgabe eines Spielers ist, in einer gegebenen Stellung p unter mehreren möglichen Halbzügen (p, p_i) zu wählen. Dazu werden alle Folgepositionen p_i durch Baumsuche bewertet.

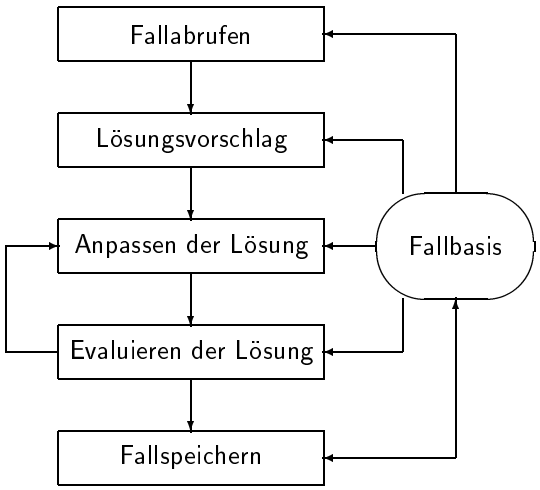

Abbildung 3.1: Der Zyklus des fallbasierten Schließens

Der im Kapitel 2 beschriebene Minimax-Algorithmus wird nun so erweitert, daß dabei auf Fälle und die in ihnen gespeicherten Stellungsbewertungen zurückgegriffen werden kann. Wenn im Verlauf der Baumsuche eine Stellung p erreicht und in der Fallbibliothek ein verwendbarer Fall $c = (p', w, k, u)$ gefunden wird, kann die im Minimax-Algorithmus vorgesehene Bewertung durch die im Fall gespeicherte – wenn nötig den Besonderheiten der Stellung p angepaßte – Bewertung ersetzt werden.

Die Benutzung eines Falls wirkt sich auf die Bewertungen so aus, als ob in den betroffenen Knoten neue Baumsuchen mit der Tiefe t_{max} gestartet würden. Wenn die Stellung p in der maximalen Suchtiefe t_{max} erreicht wird, dann wird die Standardbewertung ersetzt, ansonsten die weitere rekursive Bewertung der Stellung.

Aufgrund der Nutzung von Fällen ist dieser Algorithmus dazu in der Lage, nicht nur die Bewertungsfunktion f_T des tiefenbeschränkten Minimax-Algorithmus zu berechnen, sondern eine veränderte Bewertungsfunktion f_F.

Um bei der Baumsuche so wenig Knoten wie möglich betrachten zu müssen, sucht der fallbasierte Algorithmus nicht erst am Ende eines Halbzugs nach Fällen, sondern nach jedem Teilzug. Um diesen Vorgang darstellen zu können, werden für die Definition der Bewertungsfunktion f_F Teilzüge anstelle von Halbzügen benutzt. Ihre einfachste

Variante f_{F0} lautet:

$$f_{F0}(p,t) = \begin{cases} w & \text{, falls } (p,w,k,u) \in FB, \\ f_S(p) & \text{, falls } p \in P_E \quad \vee \quad t = t_{max}, \\ \max(\{f_{F0}(p_i,t) | (p,p_i) \in T_A\}) & \text{, falls } p \in P_A \quad \wedge \quad p_i \in P_A, \\ \min(\{f_{F0}(p_i,t) | (p,p_i) \in T_N\}) & \text{, falls } p \in P_N \quad \wedge \quad p_i \in P_N, \\ \max(\{f_{F0}(p_i,t+1) | (p,p_i) \in T_A\}) & \text{, falls } p \in P_A \quad \wedge \quad p_i \in P_N, \\ \min(\{f_{F0}(p_i,t+1) | (p,p_i) \in T_N\}) & \text{, falls } p \in P_N \quad \wedge \quad p_i \in P_A. \end{cases}$$

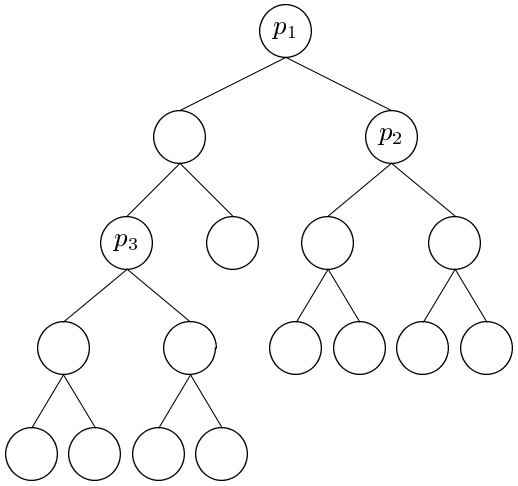

Abbildung 3.2: Verwendung von Fällen während der Baumsuche

Abbildung 3.2 zeigt ein Beispiel für die Benutzung von Fällen während der Baumsuche. Seien c_1, c_2 und c_3 Elemente der Fallbasis:

$$\begin{aligned} c_1 &= (p_1, w_1, k_1, u_1) \in FB, \\ c_2 &= (p_2, w_2, k_2, u_2) \in FB \quad \text{und} \\ c_3 &= (p_3, w_3, k_3, u_3) \in FB. \end{aligned}$$

Anstatt die Baumsuche bis zu $t_{max} = 2$ fortzusetzen, wird die gespeicherte Bewertung w_2 der Stellung p_2 verwendet. In der Position p_3 wird w_3 anstelle der Standardbewertung $f_S(p_3)$ benutzt. Nachfolgend wird der für die Position p_1 neu berechnete Wert w_1' und die neue Knotenzahl k_1' in c_1' gespeichert. Letztere ergibt sich aus der Summe der Anzahlen der durchsuchten Knoten (3) und der Knoten $k_2 = 7$ und $k_3 = 7$, die bei der Bewertung von p_2 und p_3 durchsucht worden sind. Für c_2' und c_3' werden die

Verwendungszähler u_2 und u_3 jeweils um den Wert 1 erhöht. Die so veränderten Fälle

$$\begin{aligned} c_1' &= (p_1, w_1', 3 + k_2 + k_3, u_1), \\ c_2' &= (p_2, w_2, k_2, u_2 + 1) \quad \text{und} \\ c_3' &= (p_3, w_3, k_3, u_3 + 1) \end{aligned}$$

ersetzen die alten Fälle in der Fallbasis:

$$FB' = FB \setminus \{c_1, c_2, c_3\} \cup \{c_1', c_2', c_3'\}.$$

Im folgenden Abschnitt wird beschrieben, wie einzelne Methoden des fallbasierten Schließens die Verwendung von Fällen bei der Baumsuche beeinflussen.

3.2 Methoden des fallbasierten Schließens

3.2.1 Fallspeichern und Indizieren

Die Fälle, die das fallbasierte System verwenden soll, können ihm auf zwei verschiedene Arten zur Verfügung gestellt werden. Sie können vom Nutzer ausgewählt und in die Fallbibliothek eingegeben oder vom fallbasierten System gesammelt werden.

Wenn das System selbständig Fälle zusammentragen soll, stellt sich die Frage, welche der angetroffenen Situationen als Fall gespeichert werden sollen.

Nutzen eines Falls Auf der einen Seite ist jede von der Norm abweichende Information erinnernswert, denn sie ermöglicht eine konkret auf sie zugeschnittene Reaktion. Auf der anderen Seite ist die Anzahl der Situationen bei genügend komplexen Problemen sehr groß. So gibt es in praktischen Anwendungen häufig extrem viele Erlebnisse, die sich nur minimal unterscheiden. Jede dieser Situationen aufzunehmen, würde zu einer sehr großen Fallbibliothek und damit sowohl zu einem hohen Speicherplatzbedarf als auch zu einem hohen Zeitaufwand für das Finden und die Auswahl zu verwendender Fälle führen.

Fälle sollen situationsspezifische Reaktionen ermöglichen. In diesem Sinn kann vom Speichern derjenigen Fälle abgesehen werden, die keine außergewöhnlichen Informationen beinhalten. Diese Fälle können trotzdem sinnvoll sein, wenn sie mühsames Neuberechnen von Lösungen zu bekannten Problemen vermeiden.

Generalisierung Eine Information kann durch verschiedene gespeicherte Fälle repräsentiert sein. Dann ist es möglich, diese Fälle durch Abstraktion von Unterschieden zusammenzufassen. Statt der individuellen Fälle wird ein generalisierter Fall gespeichert.

Neben der Vereinfachung des Fallabrufs führt Generalisierung auch dazu, daß Änderungen der Informationen in den Fällen häufigere Veränderungen des Systemverhaltens nach sich ziehen. Das ist insbesondere für Anwendungen wichtig, deren Ziel das Lernen mit Hilfe der Fälle ist.

Indizierung Ein weiteres Mittel zur Vereinfachung des Fallabrufs ist das *Indizieren* der Fälle.

Die Auswahl der Informationen, die als Index eines Falls dienen sollen, kann manuell oder automatisch erfolgen. Auf automatische Verfahren zur Indizierung soll hier nicht näher eingegangen werden, da sie in den für diese Arbeit durchgeführten Experimenten nicht zur Anwendung kamen.[1]

Indizes müssen verschiedene Eigenschaften besitzen, damit sie die Suche nach verwendbaren Fällen möglichst gut unterstützen:

- Sie müssen aussagekräftig sein. Der Index sollte die Informationen des Falls enthalten, die diesen von nicht verwendbaren Fällen unterscheiden. Das läßt eine Identifizierung der Teilmenge der Fallbasis zu, die die anwendbaren Fälle enthält und schließt eine große Anzahl von Fällen von der weiteren Überprüfung aus.
- Sie müssen abstrakt genug sein, um alle relevanten Fälle zu beschreiben, einschließlich derjenigen, die zukünftig noch zu speichern sein werden.
- Auf der anderen Seite müssen sie konkret genug sein, um ohne großen Aufwand mit dem gesuchten Fall vergleichbar zu sein.
- Sie sollten für ihre unterschiedlichen Verwendungen im Zyklus des fallbasierten Schließens geeignet sein.

Anwendung

Nutzen eines Falls In der untersuchten Anwendung werden ausschließlich die durch den Lernenden während der Spiele angetroffenen Stellungen mit ihren Bewertungen, der Anzahl der durchsuchten Knoten und der Benutzungshäufigkeit als Fall

[1] Zu dieser Thematik sei auf Kolodner [Kol93] S. 257ff verwiesen.

gespeichert. Alle anderen nur bei der Baumsuche bewerteten Stellungen werden ignoriert. Dafür gibt es verschiedene Gründe:

- Die Anzahl der durchsuchten Stellungen steigt exponentiell mit der Suchtiefe an. Daher würde die Zahl der zu speichernden Fälle schnell die Grenzen des Handhabbaren überschreiten, wenn für jede Position ein Fall angelegt würde.
- Die tatsächlich in den Spielen aufgetretenen Positionen wurden durch die Spieler im Verlauf der Spiele ausgewählt. Da die Spieler rational handeln, müssen ihnen diese Positionen erfolgversprechend erschienen sein. Eine genauere Bewertung dieser Stellungen – wie sie mit Hilfe des fallbasierten Schließens erreicht werden soll – ist damit wichtiger als bei nicht tatsächlich gewählten Stellungen.

Nur die wirklich aufgetretenen Positionen als Fall zu speichern, hat einen weiteren positiven Effekt: Die Bewertungen aller gespeicherten Positionen beruhen auf Baumsuchen mit der gleichen Tiefe. Das ist von Bedeutung, weil Stellungen durch Baumsuchen unterschiedlicher Tiefen verschieden bewertet werden. Besonders groß ist der Unterschied beim Vergleich der Bewertungen mit ungerader und mit gerader Suchtiefe, wie Abbildung 3.3 illustriert. Zu erklären sind diese Differenzen damit, daß eine Position einem

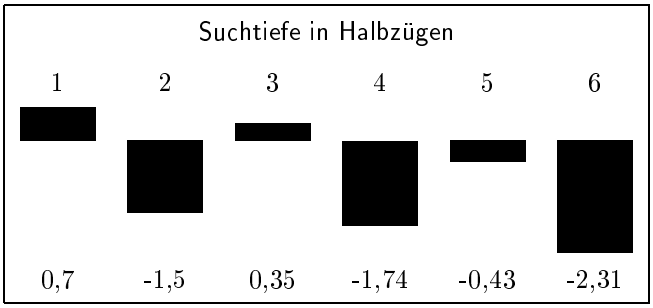

Abbildung 3.3: Durchschnittliche Stellungsbewertungen bei Baumsuchen unterschiedlicher Tiefen

Spieler günstiger erscheint, wenn sein Halbzug als letzter in die Bewertung einging.

Nicht alle angetroffenen Stellungen werden mit den zugehörigen Informationen als Fall gespeichert. Das Kriterium für den Nutzen eines Falls ist die Abweichung der Bewertung einer Stellung $f_F(p,t)$ von der Standardbewertung $f_S(p)$. Wenn gilt, daß

$$f_F(p,t) \neq f_S(p),$$

dann wird der Fall in der Fallbibliothek gespeichert. Ein Fall wird jedoch wieder aus der Fallbibliothek gelöscht, falls diese Bedingung für seine Stellung nicht mehr gilt,

nachdem diese während eines Spiels erneut erreicht wurde und sich ihre Bewertung bei der dabei erfolgten Neuberechnung geändert hat.

Generalisierung Kalah ermöglicht zwei Generalisierungen, die einfach und zudem ohne Informationsverlust durchführbar sind. Zu ihrer Erläuterung sind Definitionen der Mengen P'_A und P'_N hilfreich, die die Positionen des Anziehenden und des Nachziehenden in der Form enthalten, wie sie in der Fallbasis gespeichert werden sollen:

$$P_A \subset P'_A = \{((m_1, \ldots, m_{14}), A)\} \quad \text{und}$$
$$P_N \subset P'_N = \{((m'_1, \ldots, m'_{14}), N)\},$$

wobei gilt: $\sum_{i=1}^{14} m_i = \sum_{i=1}^{14} m'_i = 72$. Die neu definierten Mengen enthalten also nicht nur tatsächlich in Spielen auftretende Positionen, sondern auch nicht von der Startposition aus erreichbare Stellungen.

Spieler-Generalisierung: Wie schon im Kapitel 1 erwähnt, ist Kalah ein Nullsummenspiel. Der Vorteil des einen Spielers ist so groß, wie der Nachteil des anderen Spielers. Alle in dieser Arbeit beschriebenen Bewertungsfunktionen basieren auf der Standardbewertungsfunktion f_S, die diesen Fakt widerspiegelt. Eine Position des Anziehenden p hat die negierte Bewertung der durch Vertauschung der Mulden der beiden Spieler entstehenden *umgekehrten Position* p', in der der Nachziehende am Zug ist. Für

$$p = ((m_{A1}, \ldots, m_{A6}, gm_A, m_{N1}, \ldots, m_{N6}, gm_N), A) \quad \text{und}$$
$$p' = ((m_{N1}, \ldots, m_{N6}, gm_N, m_{A1}, \ldots, m_{A6}, gm_A), N)$$

gilt

$$\begin{aligned} f_S(p) &= gm_A - gm_N \\ &= -(gm_N - gm_A) \\ &= -f_S(p'). \end{aligned} \tag{3.1}$$

Zur Generalisierung der Stellungen wird die Funktion $\delta_{p_S} : P'_A \cup P'_N \to P'_A$ definiert, die Stellungen des Nachziehenden in – nicht notwendigerweise erreichbare – Stellungen des Anziehenden überführt. Das Bild jeder Position $p \in P'_A$ ist dabei p, während es für alle Positionen $p \in P'_N$ die umgekehrte Position p' ist:

$$\delta_{p_S}(p) = \begin{cases} p & \text{, falls } p \in P'_A. \\ p' & \text{, falls } p \in P'_N. \end{cases}$$

Eine Definition der Funktion für Endpositionen ist nicht erforderlich, weil für diese keine Baumsuchen durchgeführt werden und sie auch nicht in Fällen gespeichert werden sollen.

Zusätzlich muß für Positionen $p \in P'_N$ die berechnete Bewertung w negiert werden, wie in Gleichung (3.1) hergeleitet wurde. Zu diesem Zweck wird die Funktion $\delta_{w_S} : \mathbf{Z} \times (P'_A \cup P'_N) \to \mathbf{Z}$ wie folgt definiert:

$$\delta_{w_S}(w,p) = \begin{cases} w & \text{, falls } p \in P'_A. \\ -w & \text{, falls } p \in P'_N. \end{cases}$$

Der zu speichernde Fall c für eine Position p, deren Bewertung bereits u mal verwendet wurde und für die unter Berücksichtigung von k Knoten die Bewertung w berechnet wurde, ist bei der Spieler-Generalisierung demnach

$$c = (\delta_{p_S}(p), \delta_{w_S}(w,p), k, u).$$

Mit der Spieler-Generalisierung können daher maximal 2 Fälle zu einem generalisierten Fall zusammengefaßt werden.

Gewinnmulden-Generalisierung: Der weitere Verlauf des Spiels ist aufgrund der Spielregeln von Kalah nicht von den Gewinnmuldeninhalten abhängig, sondern nur von den anderen 12 Mulden. Für zwei Stellungen p und p', die sich nur in den Gewinnmulden unterscheiden, ist somit die Differenz ihrer Bewertung und ihrer Standardbewertung, die der Gewinnmuldendifferenz entspricht, gleich:

$$f_F(p,t) - f_S(p) = f_F(p',t) - f_S(p'). \tag{3.2}$$

Anstatt für mehrere Positionen, die sich nur in den Gewinnmulden unterscheiden, jeweils einen Fall anzulegen, genügt es, einen generalisierten Fall zu speichern.

Wie bei der oben beschriebenen Spieler-Generalisierung wird je eine Überführungsfunktion $\delta_{p_G} : P'_A \cup P'_N \to P'_A$ für Positionen und $\delta_{w_G} : \mathbf{Z} \times (P'_A \cup P'_N) \to \mathbf{Z}$ für deren Bewertungen definiert.

Durch δ_{p_G} wird jede Position $p = ((m_{A1}, \ldots, m_{A6}, gm_A, m_{N1}, \ldots, m_{N6}, gm_N), s)$ so auf die Position $p' = ((m_{A1}, \ldots, m_{A6}, gm_A + gm_N, m_{N1}, \ldots, m_{N6}, 0), s)$ abgebildet, daß die Inhalte der Gewinnmulden dem Anziehenden zugerechnet werden:

$$\delta_{p_G}(p) = p'.$$

Statt der Bewertung w der Position p muß die Bewertung der Position p' gespeichert werden, die durch Umstellen der Gleichung (3.2) erhalten wird:

$$\begin{aligned}\delta_{w_G}(w,p) = f_F(p',t) &= f_F(p,t) - f_S(p) + f_S(p') \\ &= w - (gm_A - gm_N) + ((gm_A + gm_N) - 0) \\ &= w + 2 \cdot gm_N.\end{aligned}$$

Bei der Gewinnmulden-Generalisierung wird der zu speichernde Fall c für eine Position p, deren Bewertung w während Baumsuchen u mal verwendet wurde, wobei zu ihrer Berechnung k Knoten betrachtet wurden, wie folgt gebildet:

$$c = (\delta_{p_G}(p), \delta_{w_G}(w,p), k, u).$$

Auf diese Weise können bei der Gewinnmulden-Generalisierung mehrere (jedoch auch theoretisch nicht mehr als 72) Fälle zu einem generalisierten Fall kombiniert werden.

Indizierung Zum schnelleren Auffinden eines Falles wurde eine einfach zu berechnende Indizierung gewählt. Durch sie wird die Fallbasis in mehrere Teilmengen untergliedert. Zwei Fälle c und c' liegen in der gleichen Teilmenge, falls für ihre Positionen

$$\begin{aligned}p &= ((m_{A1}, \ldots, m_{A6}, gm_A, m_{N1}, \ldots, m_{N6}, gm_N), s) \quad \text{und} \\ p' &= ((m'_{A1}, \ldots, m'_{A6}, gm'_A, m'_{N1}, \ldots, m'_{N6}, gm'_N), s)\end{aligned}$$

gilt:

$$\sum_{i=1}^{6} m_{Ai} = \sum_{i=1}^{6} m'_{Ai} \quad \text{und} \quad \sum_{i=1}^{6} m_{Ni} = \sum_{i=1}^{6} m'_{Ni}$$

Eine Indizierung, bei der die Positionen in einer Klasse liegen, wenn die Gewinnmuldeninhalte gleich sind, ist noch einfacher:

$$gm_A = gm'_A \quad \text{und} \quad gm_N = gm'_N.$$

Allerdings würde sich diese Bedingung durch die beschriebene Gewinnmulden-Generalisierung zu

$$gm_A + gm_N = gm'_A + gm'_N$$

verändern. Dadurch würde der Index zu allgemein, weil die Anzahl der theoretisch möglichen Klassen von $73 \cdot 73 = 5329$ auf 73 fiele. Aus diesem Grund wurde diese Indizierung im Verlauf der Entwicklung des Testprogramms verworfen und stattdessen die zuerst beschriebene verwendet.

3.2.2 Fallabrufen und Ähnlichkeit von Fällen

Um die Informationen in der Fallbasis nutzen zu können, müssen zu einem gegebenen Problem alle relevanten Fälle aus der Fallbibliothek ausgelesen werden.

Im Abschnitt 3.2.1 wurde das Prinzip der Indizierung bereits vorgestellt. Als Indizes für die Fallbasis dienen dabei typischerweise bestimmte Eigenschaften der Problemsituation. Mit ihrer Hilfe wird eine Vorauswahl aus der Gesamtmenge der gespeicherten Fälle getroffen. Dabei ist die richtige Wahl der Elemente des Problemfalls entscheidend. Es soll eine nicht zu große Teilmenge von Fällen identifiziert werden, die einer näheren Begutachtung unterzogen wird. Relevante Fälle sollen dadurch natürlich nicht von der weiteren Überprüfung ausgeschlossen werden.

Aus der Fallbasis sind ein oder mehrere Fälle auszuwählen, die tatsächlich angewandt werden sollen. Dazu müssen Fälle in der Fallbasis gesucht und ihre Ähnlichkeit zum Problemfall festgestellt werden. Die jeweiligen Methoden hierfür sind eng mit der Struktur der Fallbasis verbunden.

sequentielle Suche: Die einfachste Struktur für eine Fallbibliothek ist ein Feld oder eine Liste mit allen Fällen. In einer solchen Struktur ist es notwendig, alle Fälle daraufhin zu untersuchen, wie ähnlich sie dem Problemfall sind, um schließlich den oder die ähnlichsten Fälle auszulesen.

Der Vorteil dieser Methode liegt zum einen darin, daß die ganze Fallbibliothek durchsucht wird und der Suche somit kein verwendbarer Fall entgeht. Zum anderen ist die Aufnahme eines neuen Falles einfach, denn er muß lediglich am Ende der Liste hinzugefügt werden. Der Nachteil besteht darin, daß bei einer großen Fallbasis die Suche durch den linear mit der Fallanzahl steigenden Aufwand sehr lange dauert. Der lineare Suchaufwand kann bei sequentieller Suche zwar nicht vermieden, aber abgeschwächt werden, indem die Fallbasis partitioniert wird und dadurch nur noch in einem Teil der Fallbasis gesucht werden muß.

Suche in hierarchischen Strukturen: Eine Fallbibliothek kann auch als *shared feature network*[2] oder *discrimination network*[3] aufgebaut sein. Dabei sind die Fälle so gruppiert, daß sich ähnliche Fälle in den gleichen Teilbäumen der als Baum strukturierten Fallbasis befinden. In den Terminalknoten dieser Bäume sind die Fälle angeordnet. Bei shared feature networks enthalten die Nichtterminalknoten

[2] vgl. [Kol93] S. 295ff
[3] vgl. [Kol93] S. 300ff

die Eigenschaften der Fälle in den darunterliegenden Teilbäumen. Bei discrimination networks entsprechen sie den Merkmalen, nach deren Ausprägung sich die Fälle in den verschiedenen Teilbäumen unter dem Knoten unterscheiden lassen. Während der Fallsuche wird, ausgehend von der Wurzel des Baums, in die jeweiligen Teilbäume abgestiegen. Bei shared feature networks werden die Teilbäume gewählt, deren Eigenschaften denen des Problemfalls entsprechen. Bei discrimination networks wird in die Teilbäume verzweigt, deren Fallmerkmale genauso ausgeprägt sind wie für den Problemfall. Die Unterteilung der Fallbasis kann so optimiert werden, daß die Suchzeiten minimal sind.

Der entscheidende Vorteil hierarchischer Strukturen ist, daß Fälle schneller abgerufen werden können, da ein großer Teil der Fälle nicht auf seine Ähnlichkeit hin überprüft werden muß. Es gibt allerdings mehrere Nachteile. Erstens können Fälle in Teilbäume eingefügt worden sein, in denen sie als nicht anwendbar eingestuft werden, obwohl sie es sind. Zweitens ist das Hinzufügen aufwendig, wobei zusätzliche Schwierigkeiten dadurch entstehen, daß der Baum auch nach dem Einfügen eines neuen Falls für schnelle Fallabrufe optimiert sein sollte.

Zur Bestimmung der Ähnlichkeit zweier Fälle gibt es wiederum mehrere Möglichkeiten. Darunter sind insbesondere diejenigen Methoden zu unterscheiden, die quantitativ und qualitativ vergleichen.

Ein Verfahren, das mit einer numerischen Funktion arbeitet, ist *nearest-neighbor matching*[4] (dt. *nächster-Nachbar-Vergleich*). Es berechnet eine Ähnlichkeitsfunktion

$$\sigma(c_1, c_2) = \frac{\sum_{i=1}^{n} w_i \cdot \overline{\sigma}(a_{i1}, a_{i2})}{\sum_{i=1}^{n} w_i}$$

zweier Fälle $c_1 = (a_{11}, \ldots, a_{n1})$ und $c_2 = (a_{12}, \ldots, a_{n2})$. Dabei mißt die Funktion $\overline{\sigma}$ die Ähnlichkeit einzelner Falleigenschaften, deren Wichtigkeit für die Ähnlichkeit der Fälle durch die Gewichte w_i angeben wird. Eine Schwierigkeit ist, die Ähnlichkeiten der Eigenschaften zu bestimmen. Ein noch größeres Problem besteht darin, deren Gewichte korrekt einzustellen. Als anwendbar gelten bei diesem Verfahren der Fall oder die Fälle aus der Fallbibliothek, deren Ähnlichkeit mit dem Problemfall am größten ist.

Der nächster-Nachbar-Vergleich kann zur Unterscheidung von Fällen sowohl nach quantitativen als auch nach qualitativen Gesichtspunkten herangezogen werden. Er bietet sich jedoch besonders für quantitative Vergleiche an, weil diese das Finden und Anwenden der Funktion $\overline{\sigma}$ erleichtern.

[4] vgl. [Kol93] S. 354ff

Für qualitative Vergleiche empfehlen sich Suchen in hierarchischen Strukturen. Allerdings erfordert dieser Ansatz Wissen darüber, welche Merkmale die Fälle qualitativ unterscheiden und wie aussagekräftig das Übereinstimmen einzelner Merkmale ist. Ohne eine ausreichende Theorie oder Expertenwissen ist ein manuelles Umsetzen dieses Ansatzes nicht erfolgversprechend. Als automatische Verfahren zur Klassifizierung und Merkmalsextraktion empfehlen sich Methoden zur Konstruktion eines Entscheidungsbaums[5] oder die Anwendung neuronaler Netze[6].

Anwendung

Fallabruf während der Baumsuche Wenn fallbasiertes Schließen zur Stellungsbewertung eingesetzt werden soll, ist zu entscheiden, wann in der Baumsuche Fälle abgerufen werden sollen. Der Zweck der Baumsuche ist, die Folgepositionen der aktuellen Stellung zu bewerten, um eine von ihnen auszuwählen. Ein Fallabruf bereits in der aktuellen Stellung ist deshalb unzweckmäßig. Sinnvolle Alternativen sind hingegen:

Fallabruf nur in den eigenen Knoten: Diese Möglichkeit ist die naheliegendste, denn es werden nur eigene Stellungen mit ihren Bewertungen in der Fallbasis gespeichert. Ohne Spieler-Generalisierung können in der Fallbibliothek keine Positionen des Gegners gefunden werden. Deshalb ist die Suche nach ihnen ohne diese Generalisierung nicht sinnvoll.

Fallabruf in allen Knoten: Für die Anwendung der Spieler-Generalisierung ist es nicht nur sinnvoll, sondern auch notwendig, den Fallabruf auch in Stellungen des Gegners zuzulassen. Bezweckt wird damit, daß Fälle häufiger verwendet werden können. Die Verwendung von Bewertungen aus Baumsuchen mit geraden und ungeraden Suchtiefen führt allerdings zu Problemen beim Vergleich der Folgepositionen. Dieser Effekt wurde bereits in Abbildung 3.3 in Abschnitt 3.2.1 beschrieben.

Fallabruf nur in den Terminalknoten: Erst in der maximalen Suchtiefe Fälle abzurufen, ist ohne Spieler-Generalisierung nur für gerade maximale Suchtiefen sinnvoll. Dann stammen alle Bewertungen in der Fallbasis aus Suchen mit geraden Tiefen. Ebenso werden die Fälle nur in geraden Tiefen verwendet. Damit

[5] vgl. [Hut94] S. 172ff
[6] vgl. [Law92]

tritt der oben beschriebene nachteilige Effekt durch Bewertungen aus unterschiedlichen Tiefen nur in wesentlich abgeschwächter Form auf. Auf der anderen Seite muß eine vollständige Suche bis in die maximale Suchtiefe durchgeführt werden, d.h. die Suche wird durch Finden eines verwendbaren Falls nicht verkürzt. Ein zweiter und größerer Nachteil ist, daß nach vielen Fällen nicht gesucht werden wird. Am deutlichsten ist dieser Effekt für Fälle, deren Positionen sich im Spielbaum in einer Tiefe von weniger als t_{max} Halbzügen befinden, wobei t_{max} die maximale Suchtiefe ist. Sie können nie in den Knoten ausgelesen werden, in denen ihre Bewertung berechnet wurde.

Suche in der Fallbasis Die Fallbasis des in dieser Arbeit untersuchten Anwendungsbeispiels hat folgende Struktur:

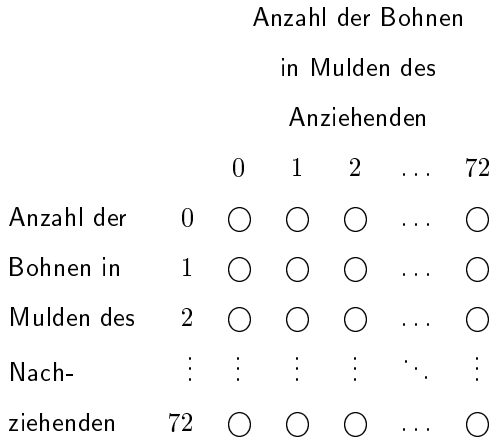

Abbildung 3.4: Die Struktur der Fallbasis im Anwendungsbeispiel

Sie wurde entsprechend der Indizierung partitioniert, so daß sie aus mehreren linearen Listen besteht. Dabei existiert für jede theoretisch mögliche Kombination aus den Anzahlen der Bohnen in den Mulden des Anziehenden und des Nachziehenden eine Liste. Wenn eine Stellung

$$p = ((m_{A1}, \ldots, m_{A6}, gm_A, m_{N1}, \ldots, m_{N6}, gm_N), s)$$

fallbasiert bewertet werden soll, dann wird nur diejenige Liste durchsucht, die alle Fälle mit den Positionen

$$p' = ((m'_{A1}, \ldots, m'_{A6}, gm'_A, m'_{N1}, \ldots, m'_{N6}, gm'_N), s)$$

enthält, für die gilt:

$$\sum_{i=1}^{6} m_{Ai} = \sum_{i=1}^{6} m'_{Ai} \quad \text{und} \quad \sum_{i=1}^{6} m_{Ni} = \sum_{i=1}^{6} m'_{Ni}.$$

Aufgrund dieser Partitionierung muß nur ein Teil der Fallbasis durchsucht werden.

Eine erfolglose sequentielle Suche in einer linearen Liste mit n Elementen erfordert den Vergleich mit allen Listenelementen. Eine erfolgreiche sequentielle Suche benötigt durchschnittlich $\frac{n}{2}$ Vergleiche, wenn alle Elemente gleich oft gesucht werden.

Wenn die relativen Häufigkeiten, mit denen nach den einzelnen Listenelementen gesucht wird, nicht gleich groß sind, kann die Suche unter der Annahme beschleunigt werden, daß in der Vergangenheit häufig gesuchte Elemente auch in der Zukunft häufig gesucht werden. Der Zeitaufwand kann verringert werden, indem in der Vergangenheit häufig gesuchte Fälle an den Beginn der Liste sortiert werden. Wenn dafür ein Zähler für jedes Listenelement geführt wird, der angibt, wie oft der Fall bisher benutzt wurde, dann wird diese Technik als Suchen in *selbstanordnenden Listen*[7] bezeichnet. Ein ohne Zähler einsetzbarer Algorithmus ist das *selbstorganisierende Suchen*[8]. Es funktioniert auf die Weise, daß ein benutzter Fall aus seiner Liste entfernt und an ihren Anfang gesetzt wird. Häufig ausgelesene Fälle bleiben so immer im vorderen Bereich der Liste.

Diese Techniken ermöglichen eine Verringerung des Zeitaufwands für erfolgreiche Suchen. Im Anwendungsbeispiel treten aber weit mehr erfolglose Suchen auf. *Binäres Suchen*[9] in einer Liste mit n Elementen erfordert sowohl für eine erfolgreiche als auch für eine erfolglose Suche nur rund $log_2 n$ Vergleiche. Es ist damit besonders bei langen Listen und einem großen Anteil von erfolglosen Suchen die geeignetste Suchmethode, weshalb sie auch beim Anwendungsbeispiel verwendet wird. Die Voraussetzung für binäres Suchen ist, daß die Listen sortiert sind. Dafür wird über der Menge der Fälle die Ordnungsrelation[10] $\preceq$ definiert. Sie beschreibt zum Vergleich der in den Fällen gespeicherten Stellungen die lexikografische Ordnung auf $\mathbf{N}^{14}$: Für zwei Fälle $c = (p, w, k, u)$ und $c' = (p', w', k', u')$ mit den Positionen $p = ((m_1, \ldots, m_{14}), s)$ und $p' = ((m'_1, \ldots, m'_{14}), s')$ gilt:

$$c \preceq c' \quad \leftrightarrow \quad \forall i \; (1 \leq i \leq 14 \quad \wedge \quad m_i > m'_i \quad \rightarrow \quad \exists j \; (1 \leq j < i \quad \wedge \quad m_j < m'_j)).$$

Mit Hilfe von $\preceq$ können $\succeq$ und die Gleichheit über der Menge der Fälle definiert

[7] [OW93] S. 191
[8] [Sed92] S. 236
[9] [Sed92] S. 236ff
[10] [RS92] S. 43

werden:

$$\begin{aligned} c \succeq c' \quad &\leftrightarrow \quad c' \preceq c \quad \text{und} \\ c = c' \quad &\leftrightarrow \quad c \preceq c' \quad \wedge \quad c' \preceq c \\ &\leftrightarrow \quad \forall i \; (1 \leq i \leq 14 \quad \wedge \quad m_i = m'_i) \end{aligned}$$

Die Fälle können anhand der in ihnen gespeicherten Positionen verglichen werden, da für eine Position nie ein zweiter Fall angelegt wird. Die definierte Relation erlaubt das Sortieren der Fälle innerhalb der Listen der Fallbasis.

Die Fallabfrage mit binärer Suche unterteilt das Suchintervall – zu Beginn die gesamte Liste – in zwei möglichst gleichgroße Teilintervalle und entscheidet, in welchem sich der gesuchte Fall befindet. Mit dem Teilintervall wird solange in der gleichen Weise verfahren, bis der gesuchte Fall gefunden wird. Algorithmus 3.1 zeigt die binäre Suche.

Schritt 1: Sei x das gesuchte Element und $a[1], \ldots, a[n]$ die Listenelemente. Setze $k = 1$ und $g = n$.

Schritt 2: Falls $k > g$, dann beende die Berechnung, denn das gesuchte Element ist nicht in der Liste.

Schritt 3: Setze $m = (k + g)$ div 2.

Schritt 4: Falls $x \preceq a[m]$ und $x \neq a[m]$, dann setze $g = m - 1$ und gehe zu Schritt 2.
Falls $x = a[m]$, dann ist $a[m]$ das gesuchte Element. Beende die Berechnung und gib m bzw. $a[m]$ zurück.
Falls $x \succeq a[m]$ und $x \neq a[m]$, dann setze $k = m + 1$ und gehe zu Schritt 2.

Algorithmus 3.1: Binäres Suchen in einer Liste. k ist die Nummer des kleinsten Elements im Suchintervall, g die Nummer seines größten Elements. Das Element $a[m]$ liegt in der Mitte des Intervalls und wird mit dem gesuchten Element x verglichen. Nach jedem Vergleich wird abhängig vom Ergebnis das Suchintervall etwa halbiert.

Bestimmung der Ähnlichkeit von Fällen Die Verwendung der Ordnungsrelation $\preceq$ und der binären Suche in einer Liste mit Fällen ist nur mit der Identität als Ähnlichkeitsrelation einsetzbar. Die in Abschnitt 3.2.1 beschriebenen Generalisierungen können allerdings verwendet werden, weil sie bereits beim Fallspeichern durchgeführt werden und damit nur ein generalisierter Fall ausgelesen wird. Ähnlichkeitsre-

lationen, mit denen potentiell mehrere Fälle aus der Fallbibliothek ausgelesen werden könnten, sind mit dem Verfahren zum Fallabruf in dieser Form nicht kombinierbar.

Für die Bestimmung anwendbarer Fälle wurden in den in Kapitel 4 beschriebenen Experimenten folgende Ähnlichkeitsrelationen untersucht, die mit Hilfe der in Abschnitt 3.2.1 erklärten Überführungsfunktionen definiert werden:

Identität ohne Generalisierung: Diese feinste Ähnlichkeitsrelation läßt keine Unterschiede zwischen der zu bewertenden Stellung und der Stellung im anwendbaren Fall zu. Fälle sind damit nur dann in mehreren Knoten anwendbar, wenn deren Positionen identisch sind. Für $c = (p, w, k, u)$ und $c' = (p', w', k', u')$ gilt:

$$sim_I(c, c') \quad \leftrightarrow \quad p = p'.$$

Identität mit Spieler-Generalisierung: Zusätzlich zu identischen Positionen lassen sich bei der Spieler-Generalisierung die umgekehrten Positionen, in denen der Gegenspieler am Zug ist, auslesen. Für zwei Fälle $c = (p, w, k, u)$ und $c' = (p', w', k', u')$ gilt:

$$sim_S(c, c') \quad \leftrightarrow \quad \delta_{p_S}(p) = \delta_{p_S}(p').$$

Identität mit Gewinnmulden-Generalisierung: Bei der Gewinnmulden-Generalisierung wird von den im Verlauf des Spiels erreichten Gewinnen abstrahiert. Dadurch lassen sich mehrere Positionen mit den zu ihnen gehörenden Informationen als ein generalisierter Fall behandeln, was zu einer Erhöhung der Anzahl der Fallabrufe führt. Für zwei Fälle $c = (p, w, k, u)$ und $c' = (p', w', k', u')$ gilt:

$$sim_G(c, c') \quad \leftrightarrow \quad \delta_{p_G}(p) = \delta_{p_G}(p').$$

Es wäre wünschenswert, neben den genannten auch weniger restriktive Ähnlichkeitsrelationen zu untersuchen. Die einfache Struktur der Stellungen bei Kalah könnte vermuten lassen, daß die Betrachtung quantitativer Unterschiede der Stellungselemente, d. h. Differenzen der Bohnenanzahlen in den einzelnen Mulden, für eine aussagekräftige Ähnlichkeitsrelation ausreicht. Experimente mit nächster-Nachbar-Ähnlichkeit, bei denen zur Berechnung der Ähnlichkeit zweier Fälle lediglich die Differenzen der Muldeninhalte herangezogen wurden, führten jedoch bei sehr geringen quantitativen Abweichungen zu unbefriedigenden Ergebnissen. Die Ähnlichkeitsrelation war bei diesen Versuchen für zwei Fälle $c = (p, w, k, u)$ und $c' = (p', w', k', u')$ mit den Stellungen

$p = ((m_1, \ldots, m_{14}), s)$ und $p' = ((m'_1, \ldots, m'_{14}), s')$ wie folgt definiert:

$$sim_{nn}(c, c') \quad \leftrightarrow \quad \sum_{i=1}^{14} |m_i - m'_i| < k \quad \wedge \quad s = s',$$

wobei $|m_i - m'_i|$ der vorzeichenlose Abstand der Muldeninhalte m_i und m'_i und k eine vorgegebene Schranke ist, mittels derer die Feinheit der Relation verändert werden kann. In den Experimenten zeigten Testreihen bereits für kleine $k > 0$, z. B. für $k = 2$ oder $k = 3$ eine starke Verschlechterung der Spielergebnisse aus Sicht des Lernenden.

Deshalb ist es für eine Erweiterung der Ähnlichkeitsrelation notwendig, qualitative Stellungsmerkmale einzubeziehen. Auch dabei erbrachten Experimente mit den folgenden identifizierten Merkmalen sehr schlechte Ergebnisse:

- Gewinnmuldeninhalte des Anziehenden und des Nachziehenden,
- Anzahl der nicht leeren Mulden des Ziehenden (Anzahl der möglichen Teilzüge),
- Anzahl der Mulden, deren Auswahl zu einem Bonuszug führen würde,
- Anzahl der angegriffenen Mulden des Gegners,
- Anzahl der Bohnen in den angegriffenen Mulden des Gegners,
- Anzahl der angegriffenen Mulden des Ziehenden,
- Anzahl der Bohnen in den angegriffenen Mulden des Ziehenden,
- Anzahl der Mulden, deren Auswahl die gegnerischen Mulden verändern würde,
- Anzahl der höchstens die gegnerischen Mulden beeinflussenden Bohnen,
- Anzahl der mindestens die gegnerischen Mulden beeinflussenden Bohnen.

Zu Kalah war keine für diese Arbeit verwendbare Theorie verfügbar. Eine solche Theorie für eine Stellungsanalyse zu entwickeln, stand außerhalb der Zielsetzung dieser Arbeit. Deshalb konnte die fallbasierte Stellungsbewertung nicht in der Form, wie sie z. B. für das Schachspiel möglich ist[11], umgesetzt werden.

[11] vgl. [Ker95]. In dieser Arbeit wird ein Entscheidungsbaum für verschiedene qualitative Merkmale einer Stellung beim Schach vorgeschlagen. Zu seiner Erstellung verwendete der Autor mehrere Werke aus der Schachliteratur und arbeitete mit einem internationalen und einem FIDE-Schachmeister zusammen.

Für eine automatische Merkmalsextraktion und Klassifizierung gegebener Stellungen ist eine Verwendung neuronaler Netze zum Auffinden von Gesetzmäßigkeiten beim Vergleich von Stellungen und den resultierenden Spielergebnissen erfolgversprechend. Auf der anderen Seite könnte eine Klassifizierung der Fälle durch die Konstruktion eines Entscheidungsbaums direkt eine hierarchische Struktur für den Aufbau der Fallbasis liefern.

Der dafür zu betreibende Aufwand entspricht jedoch der Aufgabenstellung für eine eigenständige Arbeit. Aufgrund des hohen zeitlichen Aufwands für Implementation und Durchführung der Experimente und wegen des ohnehin großen Arbeitsumfangs konnten diese Methoden im Rahmen dieser Arbeit nicht praktisch realisiert werden.

3.2.3 Anpassen und Evaluieren von Lösungen

Wenn entschieden wurde, welcher Fall zur Lösung des Problems beitragen soll, stellt sich die Frage, wie die gefundene Lösung beim Problemfall angewandt werden kann.

Bei praktischen Problemen ist es sehr unwahrscheinlich, das der abgerufene Fall und der Problemfall identisch sind. Die Anpassung der Lösung aus dem abgerufenen Fall muß

1. die Unterschiede zwischen den Fällen identifizieren und
2. entsprechend dieser Abweichungen Änderungen an der Problemlösung vornehmen.

Es gibt zwei prinzipiell verschiedene Arten, eine Problemlösung anzupassen:[12]

Strukturelle Anpassung ist die häufiger gebrauchte Form der Anpassung. Hierbei werden Elemente der Lösung aus dem verwendeten Fall mit Hilfe von Anpassungsregeln verändert. Insbesondere werden nicht zur Verfügung stehende oder unpassende Elemente ersetzt.

Nachahmende Anpassung nutzt die Algorithmen, Methoden und Regeln, die zur Konstruktion der abgerufenen Lösung verwendet wurden, und erstellt mit ihnen eine Lösung des neuen Problems. Dazu muß außer der Lösung auch der Lösungsweg in den Fällen der Fallbibliothek gespeichert werden.

[12] vgl. [Wat95] S. 7f

Nach dem Erstellen einer Lösung für das neue Problem wird ihre Verwendbarkeit im Problemfall beurteilt. Dieser Schritt der Evaluierung trägt dazu bei, Gründe dafür zu finden, warum eine Lösung nicht zum Problem paßt. Er hat auf diese Weise Anteil an der Verbesserung des fallbasierten Systems. Außerdem hilft er sicherzustellen, daß eine Lösung ausreichend angepaßt wurde. Ansonsten kann der Schritt der Anpassung wiederholt werden.

In dem in dieser Arbeit verwendeten Anwendungsbeispiel werden aufgrunddessen, daß die Ähnlichkeitsrelationen beim Fallabruf ohne Informationsverlust verwendet werden können, nur selten Lösungen angepaßt. Weiterhin ist deshalb der Schritt der Evaluierung nicht notwendig. Aus diesem Grund wird hier nur eine kurze Einführung gegeben.[13]

Anwendung

Bei Kalah werden Anpassungen nur durchgeführt, um die Informationen in den generalisierten Fällen verwenden zu können. Dazu müssen die Umkehroperationen der Überführungsfunktionen auf die Bewertung im ausgelesenen Fall angewandt werden.

Spieler-Generalisierung: Bei der Spieler-Generalisierung muß zur fallbasierten Bewertung einer Position $p \in P'_A \cup P'_N$ die Funktion $\overline{\delta}_{w_S} : \mathbf{Z} \times (P'_A \cup P'_N) \to \mathbf{Z}$ auf die im Fall gespeicherte Bewertung w und die zu bewertende Position angewandt werden:

$$\overline{\delta}_{w_S}(w,p) = \begin{cases} w & \text{, falls } p \in P'_A, \\ -w & \text{, falls } p \in P'_N. \end{cases}$$

Gewinnmulden-Generalisierung: Für die fallbasierte Bewertung einer Position $p = ((m_{A1}, \ldots, m_{A6}, gm_A, m_{N1}, \ldots, m_{N6}, gm_N), s)$ wird auf die im gefundenen Fall gespeicherte Bewertung w und die Position p die Funktion $\overline{\delta}_{w_S} : \mathbf{Z} \times (P'_A \cup P'_N) \to \mathbf{Z}$ angewandt:

$$\overline{\delta}_{w_G}(w,p) = w - 2 \cdot gm_N.$$

Alle hier beschriebenen Anpassungen sind strukturell. Die benutzten Anpassungsregeln kehren lediglich die Operationen um, die bei der Fallspeicherung zur Generalisierung angewandt wurden.

[13] Auf die verschiedenen Aspekte der Anpassung und der Evaluierung geht [Kol93] S. 393ff tiefer ein.

3.3 Fallbasiertes Lernen

Ein fallbasiertes System trifft Entscheidungen aufgrund der Informationen, die ihm in der Form von Fällen zur Verfügung stehen. Um zu lernen, muß es entweder die Fallbasis oder ihre Verwendung ändern.

Eine Fallbibliothek kann durch

- die Aufnahme eines neuen Falls,
- die Änderung der Informationen in gespeicherten Fällen oder
- das Entfernen eines Falls

verändert werden.

Wenn ein fallbasiertes System bei gleicher Fallbasis anders reagieren soll, dann müssen in der fraglichen Situation andere Fälle benutzt werden. Das kann erreicht werden, indem

- die Indizierung geändert wird und somit andere Fälle genauer auf ihre Anwendbarkeit geprüft werden oder
- die Ähnlichkeitsrelation der Fälle anders definiert wird.

In den durchgeführten Experimenten wurden keine veränderlichen Indizierungen oder Ähnlichkeitsmaße verwendet. Deshalb soll hier nur auf das Verändern der Fallbasis eingegangen werden.

3.3.1 Aufnahme neuer Fälle in die Fallbibliothek

Die Aufnahme neuer Fälle in die Fallbibliothek führt dazu, daß mehr Informationen zu konkreten Situationen verfügbar sind. Dadurch ist es dem fallbasierten System möglich, differenzierter auf spezielle Situationen einzugehen.

Aus der alten Fallbasis FB entsteht eine neue Fallbasis FB', indem ein bisher nicht gespeicherter Fall c aufgenommen wird:

$$FB' = FB \cup \{c\} \quad \text{mit} \quad c \notin FB.$$

Anwendung

In der untersuchten Anwendung Kalah werden die während der Spiele erreichten Stellungen des Lernenden zusammen mit ihren Bewertungen gespeichert. In folgenden Baumsuchen wird die Fallbasis nach verwendbaren Fällen durchsucht. Wird ein solcher Fall gefunden, dann wird die durch den tiefenbeschränkten Minimax-Algorithmus definierte Bewertung durch die im Fall gespeicherte Bewertung ersetzt. Mit der Aufnahme eines neuen Falls in die Fallbibliothek kann sich die Bewertungsfunktion $f_F(p,t)$ ändern, da der Vorgang des Ersetzens häufiger stattfindet als vor Aufnahme des neuen Falls.

3.3.2 Verbesserung von gespeicherten Informationen

Die in den entsprechenden Fällen abgespeicherten Informationen werden benutzt, um auf die angetroffenen Situationen angemessen zu reagieren. Je genauer die Fälle das zu behandelnde Problem beschreiben, desto besser kann es gelöst werden.

Wiederholtes Antreffen einer Situation erlaubt es, diese genauer zu beschreiben. In der Vergangenheit gewählte Reaktionen können bewertet und Alternativen untersucht werden. Hierzu ist es möglich, andere Fälle aus der Fallbibliothek zum Vergleich heranzuziehen.

Wenn sich Informationen zu einem Fall ändern, dann ersetzt der geänderte Fall c' den ursprünglichen Fall c in der Fallbasis FB:

$$FB' = FB \setminus \{c\} \cup \{c'\} \quad \text{mit} \quad c \in FB, c' \notin FB.$$

Anwendung

In der untersuchten Anwendung werden Fälle zur Bewertung von Stellungen während der Baumsuche verwendet. Dadurch können sich die Bewertungen einer Position von den vorher berechneten unterscheiden. Dieser Vorgang kann in verschiedene Phasen unterteilt werden, die an einem Beispiel erläutert werden. Zur einfacheren Darstellung sollen folgende Bedingungen gelten:

- Der Spielbaum ist ein vollständiger Binärbaum, der bis in die Tiefe 2 durchsucht wird.
- Als Ähnlichkeitsrelation wird die Identität verwendet.

- Alle berechneten Bewertungen unterscheiden sich von der Standardbewertung, so daß jede bewertete Position mit ihrer Bewertung als Fall gespeichert wird.

Fallbasiertes Lernen Phase I: Wenn die Fallbasis leer ist oder keine verwendbarer Fall gefunden wird, entspricht jede berechnete Bewertung derjenigen, die bei einer Baumsuche ohne fallbasiertes Schließen berechnet würde. Für jede Stellung wird ein Fall mit ihrer Bewertung gespeichert.

Im Beispiel aus Abbildung 3.5 findet der Lernende die Position p_1 zum ersten Mal vor. Die Fallbasis sei leer:

$$FB = \emptyset.$$

Deshalb ist das Ergebnis der Baumsuche das gleiche wie ohne Anwendung von

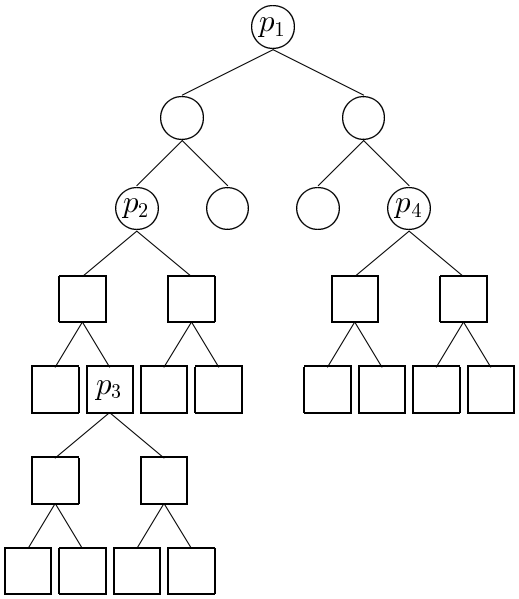

Abbildung 3.5: Fallbasiertes Lernen Phase I. Die Fallbasis enthält keine anwendbaren Fälle. Während der Spiele werden Fälle mit Positionen und ihren Bewertungen in die Fallbibliothek eingetragen, die denen einer Baumsuche ohne fallbasiertes Schließen entsprechen. Die zur Bewertung von p_1 herangezogenen Stellungen sind als Kreis dargestellt, die nicht betrachteten als Quadrat.

Fällen. Danach wird ein Fall c_1 mit der Stellung p_1 und ihrer Bewertung w in die Fallbibliothek eingetragen. Angenommen, die Spieler wählen die Spielfortsetzung so, daß der Lernende die Stellungen p_2 und p_3 antrifft. Dann führt er in diesen Stellungen wieder Baumsuchen durch und speichert die Fälle c_2 und c_3 mit p_2

und p_3 und ihren Bewertungen analog zu c_1 ab. Die Fallbasis enthält danach Fälle mit aufeinanderfolgenden Stellungen des Lernenden:

$$\{c_1, c_2, c_3\} \subset FB.$$

Dabei gilt:

$$\begin{aligned}
c_1 &= (p_1, f_F(p_1, 2), 7, 0) \quad \text{mit} \quad f_F(p_1, 2) = f_T(p_1, 2) \neq f_S(p_1), \\
c_2 &= (p_2, f_F(p_2, 2), 7, 0) \quad \text{mit} \quad f_F(p_2, 2) = f_T(p_2, 2) \neq f_S(p_2) \quad \text{und} \\
c_3 &= (p_3, f_F(p_3, 2), 7, 0) \quad \text{mit} \quad f_F(p_3, 2) = f_T(p_3, 2) \neq f_S(p_3).
\end{aligned}$$

Fallbasiertes Lernen Phase II: Bereits das erstmalige Verwenden von Fällen kann zu einer veränderten Stellungsbewertung führen. Sie würde berechnet werden, wenn nicht der durch den tiefenbeschränkten Minimax-Algorithmus definierten Berechnungsvorschrift gefolgt würde, sondern für die fallbasiert bewertete Stellung eine neue Baumsuche gestartet würde. Die so neu gewonnenene Bewertung überschreibt die bisher im Fall gespeicherte Bewertung in der Fallbibliothek.

In Abbildung 3.6 ist die Situation dargestellt, daß der Lernende die Position p_1 in einem weiteren Spiel zum zweiten Mal antrifft. Während der Baumsuche findet er den Fall c_2 in der Fallbasis, der die Stellung p_2 mit ihrer Bewertung enthält. Durch Anwendung dieses Falls entspricht die neue Bewertung für p_1 derjenigen, die durch eine in p_2 erweiterte Baumsuche ermittelt werden würde. In diesem Spiel soll als Fortsetzung die Halbzugfolge zu der Position p_4 gewählt werden. Nachdem diese Stellung bewertet wurde, enthält die Fallbibliothek einen Fall c_4 mit p_4:

$$\{c_1, c_2, c_3, c_4\} \subset FB,$$

Für die Fälle gilt im einzelnen:

$$\begin{aligned}
c_1 &= (p_1, f_F(p_1, 2), 13, x) \quad \text{mit} \quad f_F(p_1, 2) \neq f_S(p_1), \\
c_2 &= (p_2, f_F(p_2, 2), 7, 1) \quad \text{mit} \quad f_F(p_2, 2) = f_T(p_2, 2) \neq f_S(p_2), \\
c_3 &= (p_3, f_F(p_3, 2), 7, 0) \quad \text{mit} \quad f_F(p_3, 2) = f_T(p_3, 2) \neq f_S(p_3) \quad \text{und} \\
c_4 &= (p_4, f_F(p_4, 2), 7, 0) \quad \text{mit} \quad f_F(p_4, 2) = f_T(p_4, 2) \neq f_S(p_4).
\end{aligned}$$

Wie oft c_1 verwendet wurde, wird in diesem Beispiel nicht ersichtlich, weshalb für seine Verwendungshäufigkeit ein x eingetragen wurde.

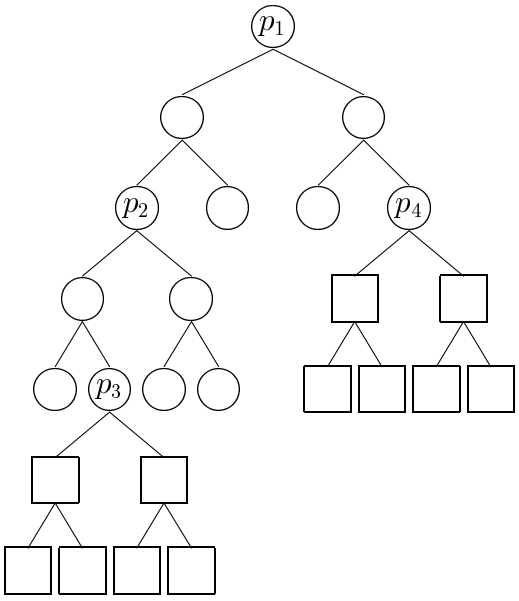

Abbildung 3.6: Fallbasiertes Lernen Phase II. Die Bewertungen der in der Fallbasis gespeicherten Fälle wurden durch Baumsuchen ohne Anwendung von Fällen erhalten. Gespeichert werden in dieser Phase Fälle, für deren Stellungen eine davon abweichende Bewertung erlernt wurde. Die zur Bewertung von p_1 herangezogenen Stellungen sind als Kreis dargestellt, die nicht betrachteten als Quadrat.

Fallbasiertes Lernen Phase III: Die veränderten Fälle werden wiederum zur Berechnung der Stellungsbewertung benutzt. Da einige Positionen unter Verwendung von Fällen bewertet wurden, kann sich ein erneut veränderter Wert ergeben. Wenn in einer Stellung mehrere Male der gleiche Zug gewählt wird, entsteht eine Bewertung, die einer tieferen Suche im jeweiligen Teilbaum entspricht. Bei Auswahl verschiedener Zugalternativen in einer Stellung entsteht hingegen eine Bewertung, die einer in mehreren Teilbäumen weniger tief erweiterten Suche entspricht.

Abbildung 3.7 zeigt die Situationen, in denen der Lernende in zwei darauffolgenden Spielen die Position p_1 weitere Male antrifft. Wenn p_1 das nächste Mal bewertet wird, werden die beiden Fälle c_2 und c_4 gefunden, welche die Stellungen p_2 und p_4 enthalten. Nachdem p_1 bewertet wurde, soll das Spiel über p_2 und nicht über p_3 fortgesetzt werden. Nach der Aktualisierung der Daten in den Fällen c_1 bis c_4 hat sich die Fallbasis wieder verändert. Zwar sind nicht notwendigerweise

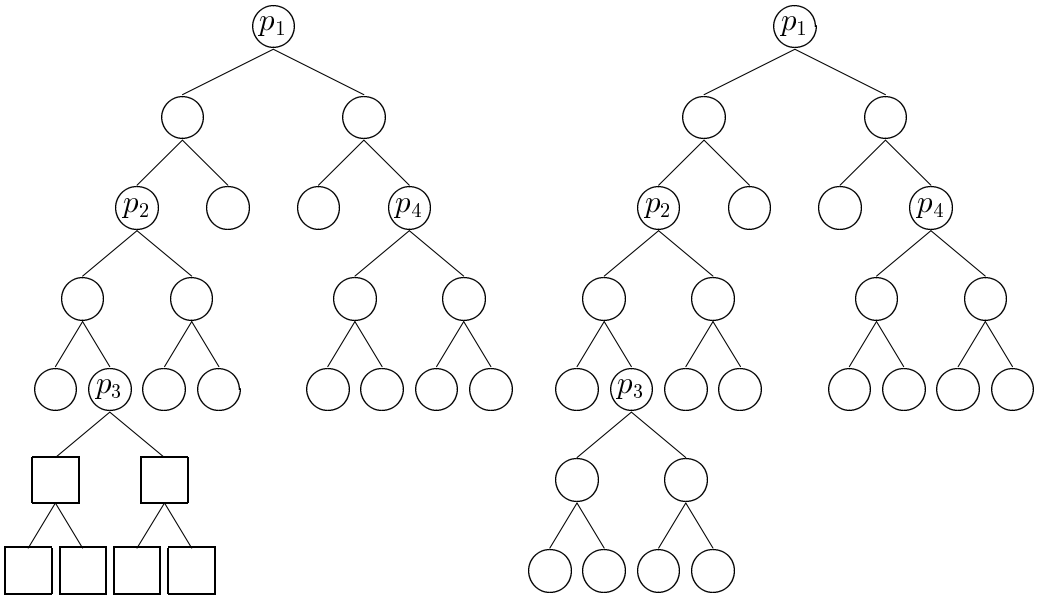

Abbildung 3.7: Fallbasiertes Lernen Phase III. Ein Teil der Fälle in der Fallbasis enthält Positionen, deren Bewertung von derjenigen abweicht, die durch den tiefenbeschränkten Minimax-Algorithmus berechnet wird. Die zur Bewertung von p_1 herangezogenen Stellungen sind als Kreis dargestellt, die nicht betrachteten als Quadrat.

neue Fälle hinzugekommen:

$$\{c_1, c_2, c_3, c_4\} \subset FB,$$

für die Fälle gilt aber:

$$\begin{aligned}
c_1 &= (p_1, f_F(p_1, 2), 19, x) \quad \text{mit} \quad f_F(p_1, 2) \neq f_S(p_1), \\
c_2 &= (p_2, f_F(p_2, 2), 13, 2) \quad \text{mit} \quad f_F(p_2, 2) \neq f_S(p_2), \\
c_3 &= (p_3, f_F(p_3, 2), 7, 1) \quad \text{mit} \quad f_F(p_3, 2) = f_T(p_3, 2) \neq f_S(p_3) \quad \text{und} \\
c_4 &= (p_4, f_F(p_4, 2), 7, 1) \quad \text{mit} \quad f_F(p_4, 2) = f_T(p_4, 2) \neq f_S(p_4).
\end{aligned}$$

Die neue Bewertung von p_2 ergibt sich aus einer in p_3 erweiterten Baumsuche. Sie muß damit nicht mehr gleich der Bewertung aus einer Baumsuche ohne Verwendung von Fällen sein. Die Benutzungshäufigkeit von c_4 hat sich bei der Bewertung von p_1 erhöht, die von c_3 bei der Bewertung von p_2.

Wenn der Lernende nun noch einmal die Stellung p_1 antrifft, dann findet und verwendet er die Fälle c_2 und c_4, also sowohl die Bewertung für p_2, die unter Verwendung der Bewertung von p_3 berechnet wurde als auch die Bewertung von

p_4, die ohne Verwendung eines Falls berechnet wurde. Die auf diese Weise für p_1 gefundene Bewertung entspricht einer durch eine Baumsuche berechnete, die in mehreren Stellungen bis in unterschiedliche Tiefen erweitert wurde.

Auswahlstrategie Bisher wurde beschrieben, wie fallbasiertes Schließen die Bewertung einer Position verändern kann. Nach ihrer Bewertung muß eine der Folgepositionen ausgewählt und der entsprechende Halbzug ausgeführt werden. Dazu wird eine Auswahlstrategie verwendet. Beim Minimax-Algorithmus wird die am besten bewertete Folgeposition gewählt. Wenn es mehrere am besten bewertete Folgepositionen gibt, wird unter ihnen zufällig ausgewählt.

Durch das fallbasierte Schließen steht dem lernenden Spieler ein weiteres Unterscheidungsmerkmal zur Verfügung. Ein Teil der Positionen ist ihm schon bekannt, während andere bisher nicht in den Spielen auftraten. Die bekannten Positionen sind als Fall in der Fallbasis gespeichert. Die Auswahlstrategie unterstützt oder vermeidet die Wiederholung von Halbzügen. Auf diese Weise wird das oben erläuterte Ändern der Fallbasis und somit der Lernprozeß beeinflußt.

Neben der zufälligen Auswahl wurden in den Experimenten verschiedene Auswahlstrategien untersucht:

„bekannte“ und „bekannteste“: Ziel dieser Auswahlstrategien ist, durch das Bevorzugen bekannter Positionen die Benutzungshäufigkeit der gespeicherten Fälle zu erhöhen. Sie führen dazu, daß auf alten Entscheidungen beharrt wird, bis sich diese als schlechter herausstellen als in der ursprünglichen Bewertung. Fraglich ist, ob nicht auch die Wiederholung von ungünstigen Entscheidungen unterstützt wird. Die Schwäche dieser konservativen Haltung besteht darin, daß sie dem eigentlichen Ziel des Lernens entgegensteht, Verhalten zu verbessern und damit zu verändern.

Die beim Lernen veränderten Bewertungen werden häufiger Werte annehmen, die bei der beschriebenen Durchführung von in wenigen Teilbäumen erweiterten Baumsuchen erhalten werden würden. Diese Strategie durchsucht den Spielbaum in bekannten Stellungen verstärkt in größere Tiefen.

Der Unterschied der beiden Varianten besteht darin, daß „bekannte“ lediglich Folgepositionen bevorzugt, bei deren Bewertung Fälle verwendet wurden, während „bekannteste“ die Folgeposition bevorzugt, bei deren Bewertung der Fall mit dem größten Verwendungszähler gefunden wurde.

„unbekannte“ und „unbekannteste“: Die gegenteilige Strategie, unbekannte Stellungen zu bevorzugen, geht das Risiko ein, gute Entscheidungen in Frage zu stellen. Dabei wird sie aber wahrscheinlich Alternativen, die überraschenderweise besser sind, schneller entdecken.

Der Spielbaum wird mit dieser Strategie verstärkt mehrere alternative Teilbäume in größere Tiefen durchsuchen.

Wie bei der oben beschriebenen Strategie gibt es auch hier eine verschärfte Variante, die die Folgeposition bevorzugt, bei deren Bewertung der Fall mit dem kleinsten Verwendungszähler gefunden wurde.

„bekannte und unbekannte“ und „bekannteste und unbekannteste“: Eine Kombination der beiden oben beschriebenen Strategien bevorzugt bekannte Stellungen, deren Bewertung einen Vorteil verspricht, und unbekannte Stellungen, wenn die Bewertung auf einen Nachteil schließen läßt. Die Motivation für diese Strategie besteht darin, daß ein vermeintlicher Vorteil nicht durch großes Risiko verspielt werden soll, aber ein zu erwartender nachteiliger Verlauf des Spiels durch Veränderung bisheriger Verhaltensweisen möglicherweise abgewendet werden kann. Diese Strategie erinnert damit an die durch verschiedene Lernverfahren bekannte Technik der Belobigung guten Verhaltens und Bestrafung schlechten Verhaltens.

Auch diese Strategie kann durch die erwähnte Variation verschärft werden.

3.4 Vor- und Nachteile des fallbasierten Lernens

In diesem Abschnitt werden Gründe, die zu Lernerfolgen bzw. zu schlechtem Lernverhalten beim fallbasierten Lernen der Bewertungsfunktion führen könnten, aufgezählt. Dabei wurden die ermittelten Testergebnisse nicht einbezogen. Vielmehr sind diese Listen als Thesen für zu erwartende Resultate zu verstehen.

3.4.1 Vorteile

- Das Verfahren ist weitgehend unabhängig vom betrachteten Anwendungsbeispiel. Voraussetzung der Nutzbarkeit ist nur, daß zum Problemlösen Baumsuchen zur Berechnung von Bewertungsfunktionen durchgeführt werden. Lediglich für die

Generalisierungen bzw. Definitionen der Ähnlichkeitsrelationen sind von Anwendung zu Anwendung prinzipielle Änderungen nötig.

- Positionen, für die in der maximalen Suchtiefe t_{max} ein Fall gefunden wird, können statt mit der Standard-Bewertungsfunktion mit Hilfe des Falles bewertet werden.
- Für Positionen, zu denen in einer Tiefe $t < t_{max}$ ein Fall gefunden wird, muß keine Baumsuche mit Tiefe $t_{max} - t$ bewertet werden, was zu einer Zeitersparnis führt.
- Positionen, für die in einer Tiefe $t < t_{max}$ ein Fall gefunden wird, gehen nicht mit einer Bewertung aus einer Suche mit $t_{max} - t$ in die Berechnung ein, sondern mit der gespeicherten Bewertung, die durch eine Suche mit Tiefe t_{max} ermittelt wurde.
- Die in den Fällen gespeicherten Bewertungen sind durch weitere Suchen und Anwendung von Fällen verbesserbar, insbesondere führt Wiederholung von Zügen zu einer wiederholten Veränderung der Bewertungen der entsprechenden Positionen des Lernenden.
- Gespeicherte Positionen sind wichtig in dem Sinne, daß sie tatsächlich von den Spielern gewählt wurden und es damit wahrscheinlich ist, daß sie
  - während folgender Baumsuchen verwendet werden,
  - wieder gewählt werden und dadurch die entsprechenden Bewertungen verbessert werden.

3.4.2 Nachteile

- Der Zeitaufwand für die Suche nach einem passenden Fall ist hoch.
- Der Speicherplatzbedarf für die Fallbibliothek des Lernenden ist hoch.
- Bei der Verwendung von feinen Ähnlichkeitsrelationen wird nur für einen sehr geringen Anteil der während der Baumsuche durchsuchten Positionen ein anwendbarer Fall gefunden werden.
- Ein qualitativer Vergleich von Positionen ist ohne eine Theorie des Spiels sehr schwierig und wurde in den Experimenten nicht praktisch umgesetzt.

- Das Verfahren bezieht keine Informationen über den Ausgang vorangegangener Spiele ein.
- Der Algorithmus bezieht keine Informationen über die Spielweise des Gegners ein – höchstens in Form der gespeicherten Fälle, in denen sich bei wiederholter Auswahl einer Stellung deren Bewertung ändern kann.
- Es gibt keine Gewißheit, daß der Lernvorgang konvergiert. Gute Halbzugalternativen werden nicht notwendigerweise erkannt, wenn ihre nicht fallbasierte Bewertung schlechter als die einer anderen Zugmöglichkeit ist.
- Der fallbasierte Algorithmus vergleicht Stellungsbewertungen, die aus Suchen mit unterschiedlichen Tiefen stammen. Daher wird er möglicherweise Positionen häufig über- bzw. unterschätzen.

Kapitel 4

Beschreibung und Ergebnisse der Experimente

Für die Untersuchung der in Kapitel 3 vorgestellten Varianten des fallbasierten Algorithmus zum Lernen einer verbesserten Bewertungsfunktion wurden im Rahmen dieser Arbeit umfangreiche Experimente durchgeführt.

Der Umfang der Versuche lag insgesamt bei weit mehr als 400 000 Spielen, bei denen verschiedene Algorithmen gegeneinander spielten. Neben dem Aufwand für die Experimente, die der Planung der Testreihen dienten, betrug der Rechenzeit-Aufwand für die Testreihen zur Überprüfung der Leistungsfähigkeit des Lernverfahrens über 1 500 Stunden.[1]

4.1 Beschreibung der Experimentierumgebung

4.1.1 Verwendete Hard- und Software

Die Tests wurden auf IBM-kompatiblen PCs mit unterschiedlichen Konfigurationen durchgeführt. Darunter befand sich ein PC mit einer AMD 486/DX4 100 MHz CPU, mit 8 MB RAM Arbeitsspeicher und Microsoft Windows 3.x. Alle anderen PCs hatten Intel-Prozessoren vom Typ Pentium und Pentium II oder AMD K6-Prozessoren mit Taktfrequenzen zwischen 133 MHz und 300 MHz und liefen mit Windows 95 oder Windows NT. Für den Bedarf an Arbeitsspeicher ist die Größe der Fallbibliotheken

[1] An dieser Stelle möchte ich mich für die Bereitstellung von Rechenzeitkapazität, die gute Zusammenarbeit und das Entgegenkommen bei der Firma smartec GmbH, Leipzig, und ihren Mitarbeitern sehr herzlich bedanken.

maßgeblich. Bei allen im Rahmen dieser Arbeit beschriebenen Testreihen waren etwa 5 MB ausreichend.

Das Testprogramm wurde unter Borland Pascal 7.0 für Microsoft Windows 3.x erstellt, läuft jedoch auch unter Windows 95 und Windows NT. Es benötigt die Borland-Laufzeitbibliothek `bwcc.dll`.

4.1.2 Vorstellung des Testprogramms

Mit dem Testprogramm ist es möglich, für den Anziehenden und den Nachziehenden jeweils einen Algorithmus auszuwählen und die Anzahl der gewünschten Spiele festzulegen und die damit bestimmte Testreihe durchführen zu lassen. Nach dem Ablauf oder bei einer Unterbrechung der Testreihe kann sie gespeichert und zu einem späteren Zeitpunkt fortgesetzt werden.

Die eine Testreihe beschreibenden Daten beinhalten:

- Informationen über die spielenden Algorithmen, gegebenenfalls einschließlich der maximalen Suchtiefe, der verwendeten Form des α-β-Baumstutzens, der Auswahlstrategie, der verwendeten Ähnlichkeitsrelation und der Information, wann während der Baumsuche Fallabrufe erlaubt sind,
- Informationen zum verwendeten Spielbrett, d. h. zur Anzahl der Mulden je Spieler und zur Anzahl der Bohnen je Mulde in der Startposition,
- statistische Daten zu den Gewinnen beider Spieler nach gewonnenen Bohnen und gewonnenen Spielen, zur Halb- und Teilzugwahl, zur verstrichenen Zeit und zu den im Teilzug-Spielbaum insgesamt und in der maximalen Suchtiefe durchsuchten Knoten,
- eine Statistik über die Häufigkeit der verschiedenen Spielergebnisse,
- der Verlauf der Gewinne beider Spieler nach Bohnen und Spielen während der Testreihe,
- gegebenenfalls die Fallbasen und Statistiken zum Fallabruf, einschließlich
  - der Fallbibliotheken der fallbasiert bewertenden Spieler,
  - der Anzahl der Fallabrufe, aufgeschlüsselt nach dem abrufenden Spieler und den Suchtiefen, in denen die Fälle abgerufen wurden,

– dem Verlauf der Fallabrufe insgesamt und nach Suchtiefen aufgeschlüsselt, jeweils für beide Spieler.

Neben der damit geschaffenen Möglichkeit, Testreihen durchzuführen und zu verwalten, erlaubt es das Programm ebenfalls, grundlegende Auswertungen vorzunehmen. Hierzu wurden verschiedene Ansichten implementiert, die z. B. den Verlauf der Spielergebnisse und der Fallabrufe grafisch darstellen. Im folgenden wird eine Übersicht der Funktionen des Programms anhand seiner Menüpunkte gegeben:

Testreihe|Neue Testreihe: Ruft Dialoge auf, in denen das Spielbrett und die agierenden Spieler definiert werden, und erstellt damit eine neue Testreihe.

Testreihe|Laden...: Lädt eine gespeicherte Testreihe.

Testreihe|Speichern unter...: Speichert eine Testreihe.

Testreihe|Beenden: Beendet das Programm.

Optionen|Brett...: Wird automatisch beim Erstellen einer neuen Testreihe aufgerufen. Hier kann die Anzahl der Mulden je Spieler und die Anzahl der Bohnen je Mulde festgelegt werden.

Abbildung 4.1: Das Optionsfenster zur Auswahl der Spielbretteigenschaften.

Wenn bereits eine Testreihe definiert ist, dann werden die Einstellungen lediglich angezeigt.

Optionen|Spieler...: Wird automatisch beim Erstellen einer neuen Testreihe aufgerufen. Hier kann festgelegt werden, welche Spieler agieren sollen.

Wenn bereits eine Testreihe definiert ist, dann werden die Einstellungen lediglich angezeigt.

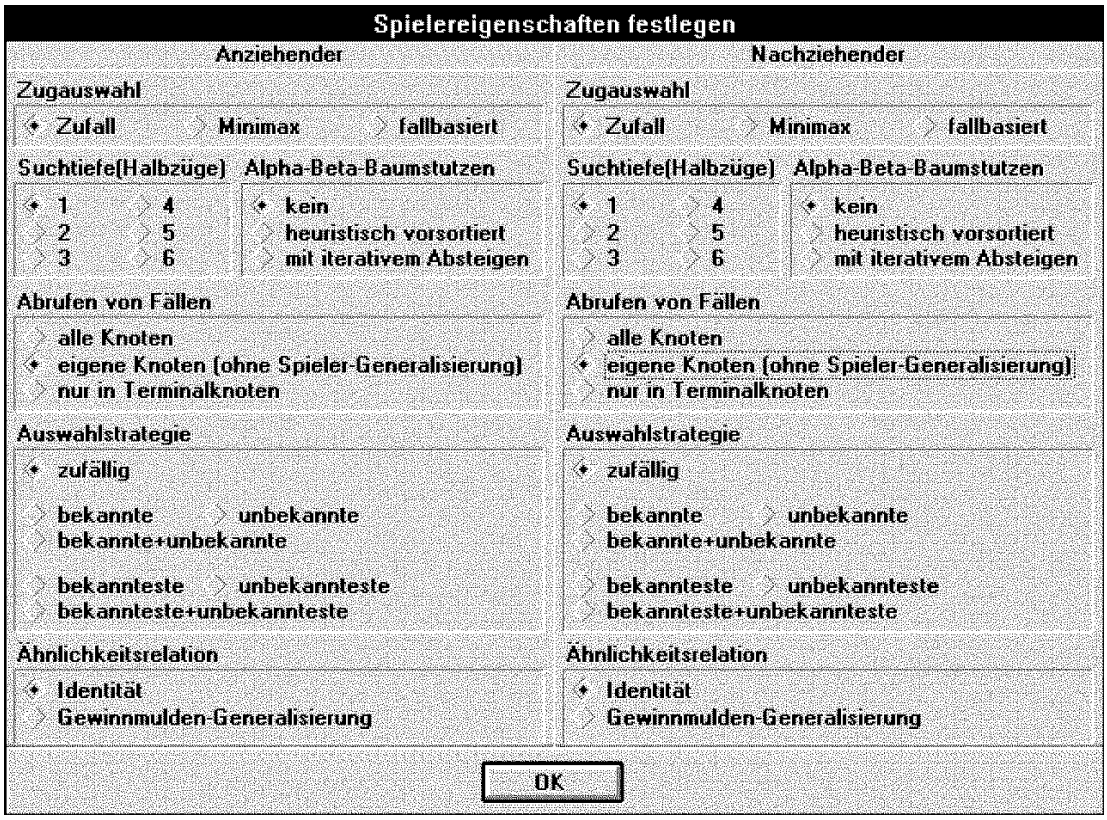

Abbildung 4.2: Das Optionsfenster zur Auswahl der Spielereigenschaften

Start!: Startet eine Testreihe, die vorher neu erstellt oder geladen wurde.

Stop!: Führt das laufende Spiel zuende und hält danach die Testreihe an.

Ansicht|Testreihe: Wechselt die Ansicht, um das Gesamtergebnis der Testreihe, Statistiken über die Anzahl der Züge bzw. verschiedenen Teilzüge und über die Anzahl der durchsuchten Stellungen anzuzeigen. Weiterhin enthält diese Ansicht die verstrichene Zeit und eine grafische Darstellung der Häufigkeitsverteilung der Spielergebnisse.

Ansicht|Ergebnisse: Wechselt die Ansicht, um den Verlauf der Ergebnisse der Testreihe anzuzeigen. Dabei werden sowohl die Anzahl der Siege der Spieler als auch die exakte Höhe des Sieges oder der Niederlage ausgewertet. Mit dem Menüpunkt **Ansicht|Skalierung...** kann eingestellt werden, wieviele Spiele zu einem Zwischenergebnis zusammengefaßt werden sollen.

Ansicht|Fallabrufe: Wechselt die Ansicht, um den Verlauf der Häufigkeit der Fall-

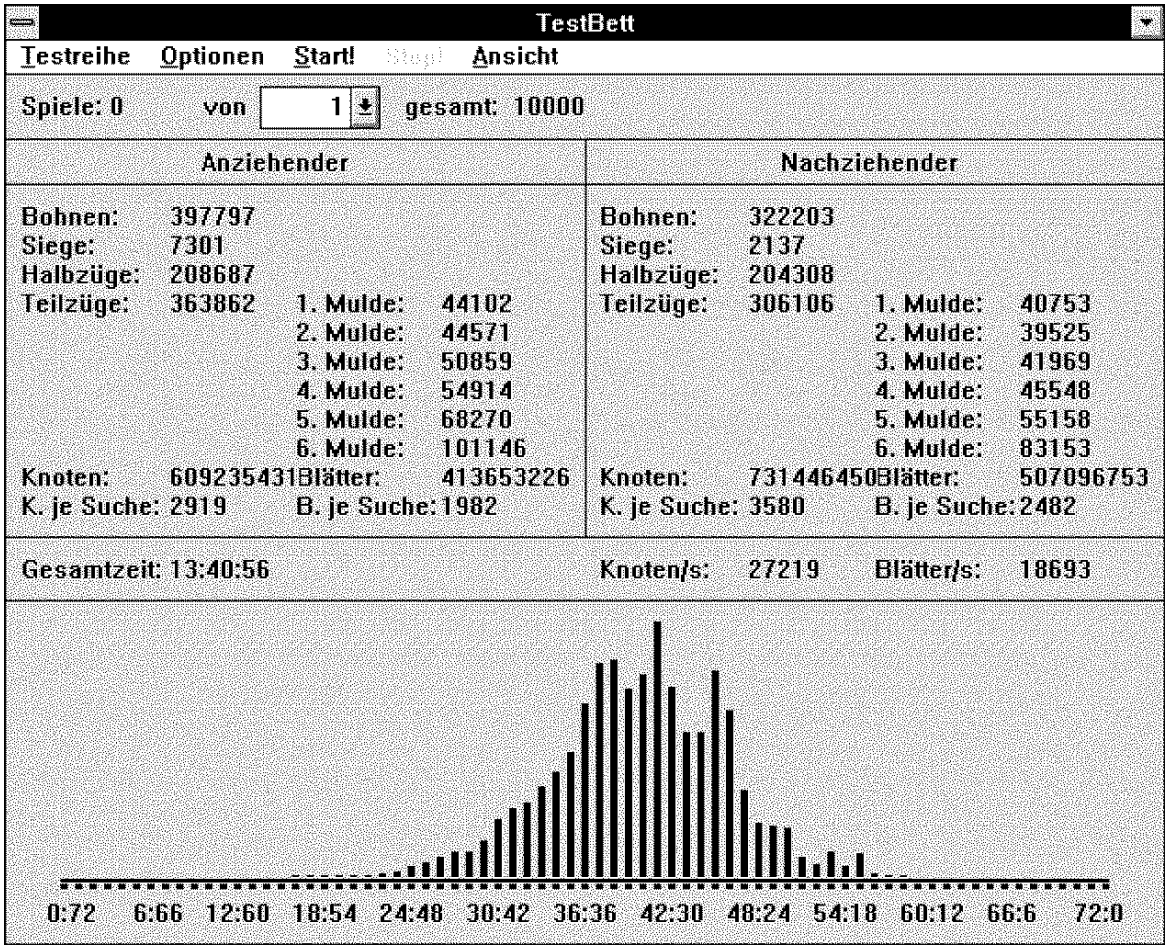

Abbildung 4.3: Die Ansicht „Testreihe“ für eine Beispiel-Testreihe

nutzungen anzuzeigen. Hier wird die Anzahl der Fallnutzungen insgesamt und die Verteilung der Tiefen, in denen der anzuwendende Fall gefunden wurde, dargestellt. Mit **Ansicht|Skalierung...** kann eingestellt werden, wieviele Spiele zu einem Zwischenergebnis zusammengefaßt werden sollen.

Ansicht|Fallbasis: Wechselt die Ansicht, um eine grafische Darstellung der Fallbasen der Spieler anzuzeigen. Dargestellt werden die Anzahl der Fälle je Index, die Häufigkeit der Fallnutzungen je Index und die Anzahl der Stellungen, die zur Bewertung der Stellungen der entsprechenden Fälle herangezogen wurden, je Index.

Ansicht|laufend aktualisieren: Wenn dieser Menüpunkt gewählt ist, dann wird die Anzeige regelmäßig erneuert. Mit **Ansicht|Skalierung...** kann eingestellt wer-

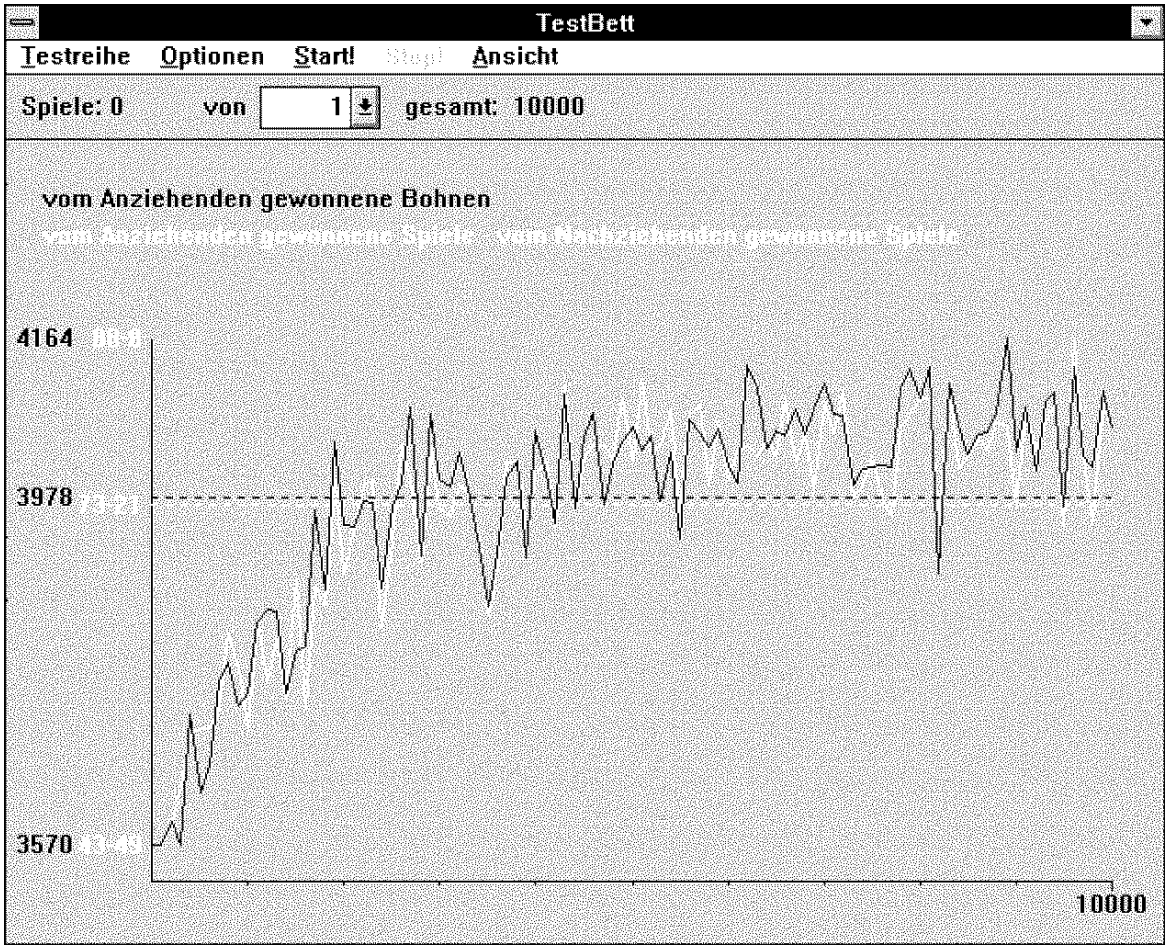

Abbildung 4.4: Die Ansicht „Ergebnisse“ für eine Beispiel-Testreihe

den, wieviele Spiele durchgeführt werden sollen, bis die Anzeige aktualisiert wird.

Ansicht|Skalierung...: Legt fest, aller wieviel Spiele die Ansicht aktualisiert werden soll, falls **laufend aktualisieren** gewählt ist. Gibt außerdem die Anzahl der Spiele an, die bei den Ansichten **Ergebnisse** und **Fallabrufe** zu einem Zwischenergebnis zusammengefaßt werden sollen.

4.2 Planung der Experimente

Anzahl der Spiele je Testreihe Ausschlaggebend für die Entscheidung, in den Testreihen zum Nachweis der Leistungsfähigkeit des Lernverfahrens 10 000 Spiele durchzuführen, waren zwei Gründe. Erstens sind die statistischen Abweichungen bei 10 000 Spielen im Vergleich zu z. B. 100 oder 1 000 Spielen ausreichend gering, wie die

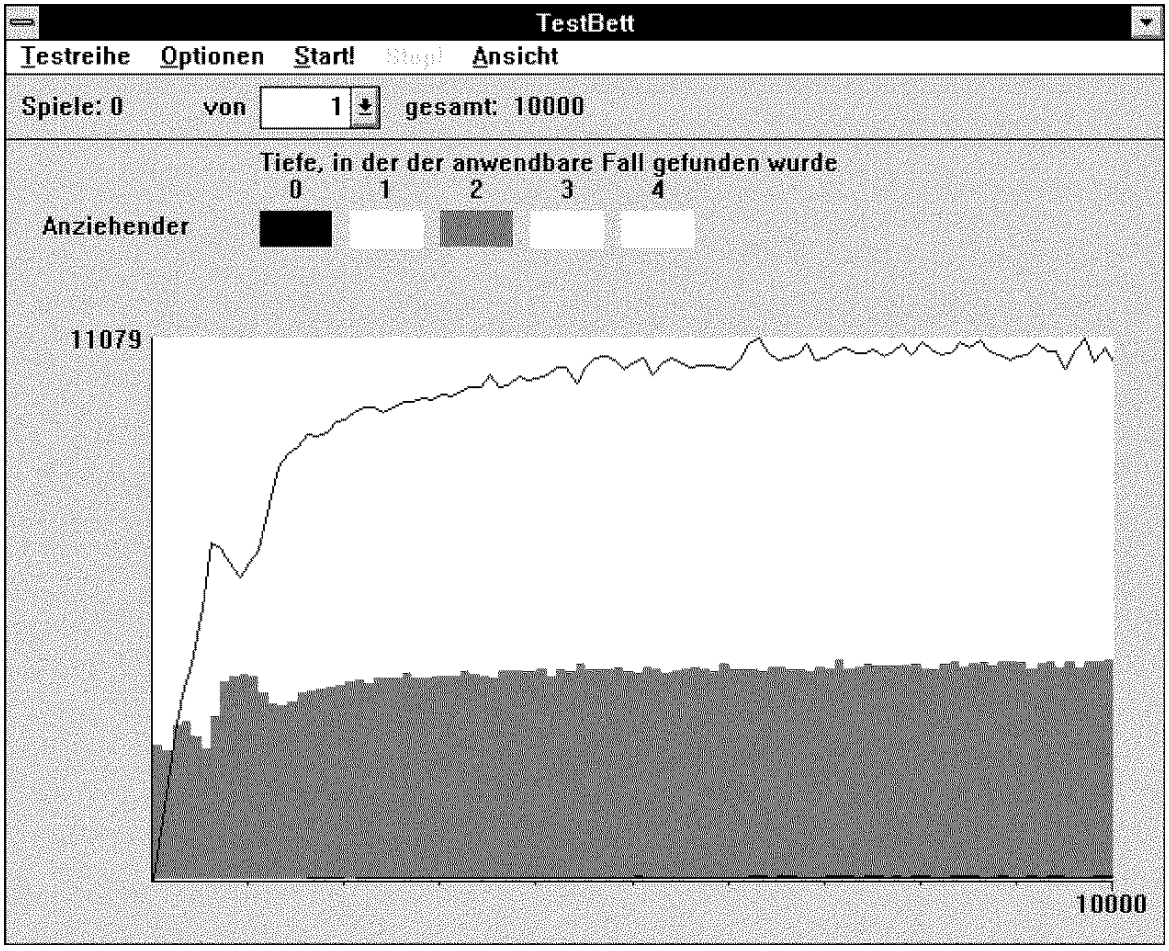

Abbildung 4.5: Die Ansicht „Fallabrufe“ für eine Beispiel-Testreihe

in Abbildung 4.7 dargestellten Häufigkeitsdiagramme illustrieren. Zweitens lassen die in den Experimenten erreichten Lernerfolge für die meisten Testreihen keine wesentlichen Veränderungen auch bei einer größeren Spielanzahl vermuten. Es ist nicht auszuschließen, daß sich bei einigen Testreihen geringe Verbesserungen in weiteren Spielen herausgestellt hätten. Aus Zeitgründen konnten jedoch nicht einheitlich mehr Spiele je Testreihe durchgeführt werden.

Lernende Spieler In den Experimenten wurde ausschließlich das Verhalten des fallbasierten Algorithmus für den Anziehenden untersucht. Um den zeitlichen Aufwand für die Tests nicht unnötig ansteigen zu lassen, wurde von entsprechenden Untersuchungen für den Nachziehenden abgesehen, weil dafür keine prinzipiell abweichenden Ergebnisse zu erwarten waren.

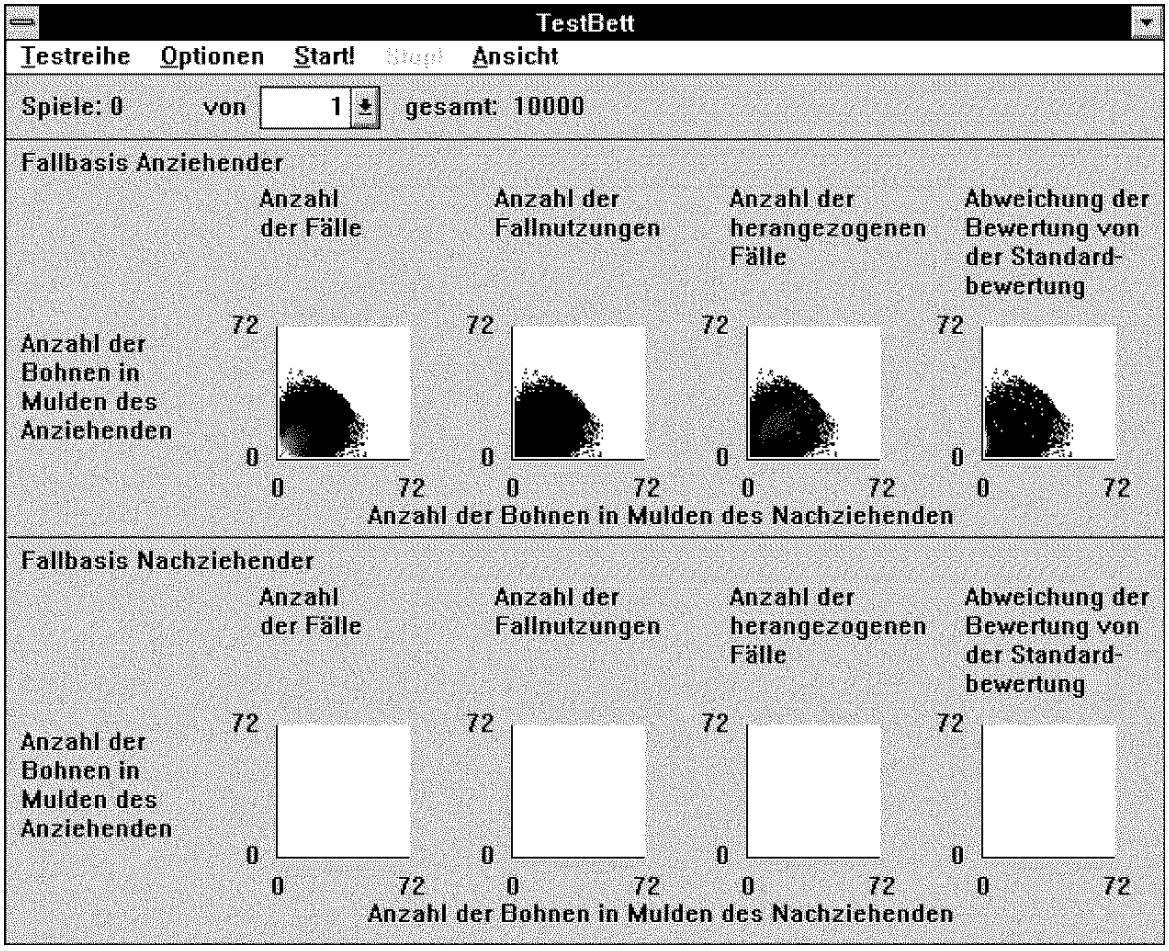

Abbildung 4.6: Die Ansicht „Fallbasis“ für eine Beispiel-Testreihe

Spieler-Generalisierung Die Umsetzung der im Abschnitt 3.2 erklärten Spieler-Generalisierung erfolgte nicht wie dort beschrieben, sondern ergibt sich durch eine Abwandlung des Minimax-Algorithmus automatisch. Dabei werden Baumsuchen des Anziehenden und des Nachziehenden nicht unterschieden. Vielmehr bewerten die Spieler die Positionen jeweils aus ihrer Sicht. Das abwechselnde Maximieren und Minimieren in aufeinanderfolgenden Suchtiefen muß dazu durch Maximieren in jeder Tiefe und anschließendes Negieren ersetzt werden. Auf diese Weise werden die Bewertungen der Positionen des Anziehenden und des Nachziehenden direkt identifiziert, so daß für sie keine weiteren Überführungen notwendig sind. Damit ist es auch nicht mehr nötig, bei den gespeicherten Positionen zwischen Stellungen des Anziehenden und des Nachziehenden zu unterscheiden.

Deshalb muß bei dieser Abwandlung des Algorithmus lediglich der Fallabruf in geg-

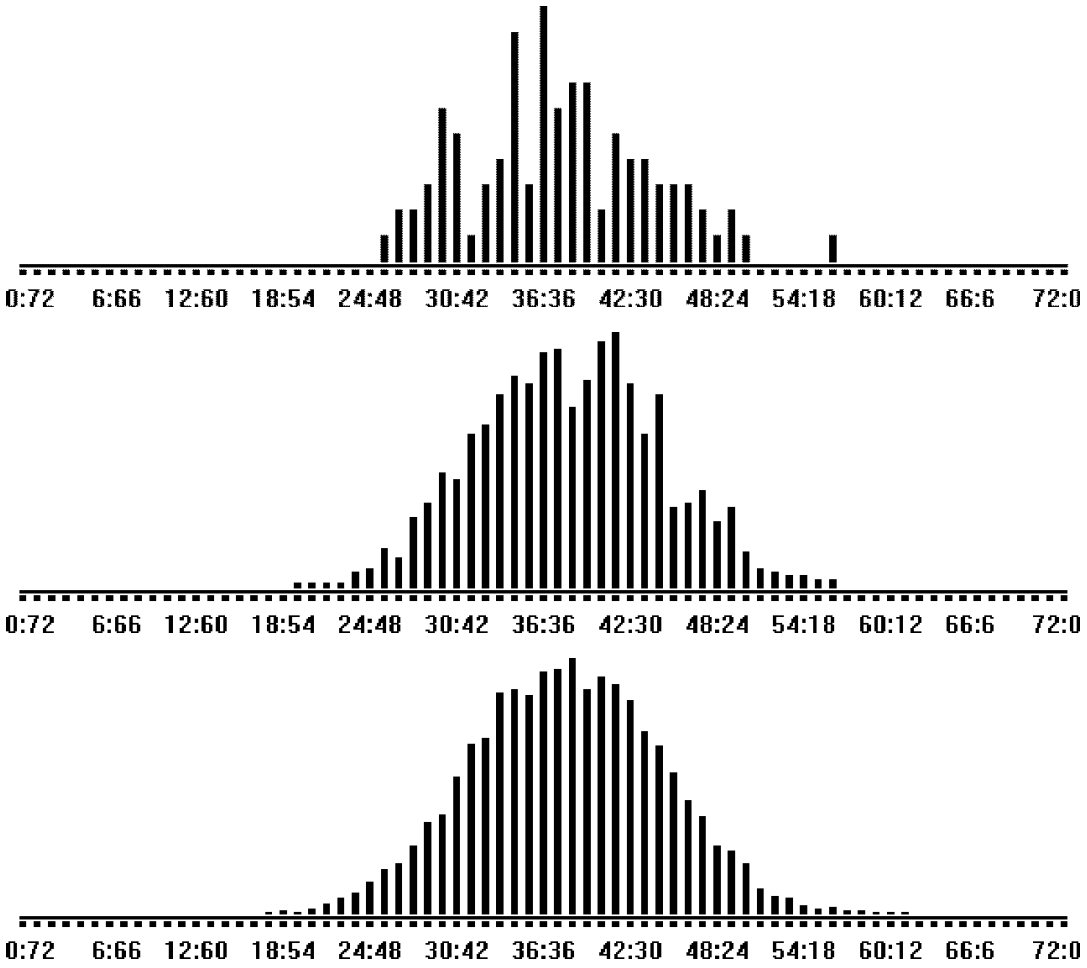

Abbildung 4.7: Häufigkeiten der Spielergebnisse bei unterschiedlich vielen Spielen. Durchgeführt wurden 100, 1 000 bzw. 10 000 Spiele (von oben nach unten) zwischen zwei Spielern mit Minimax-Suche der Tiefe 1.

nerischen Stellungen unterbunden werden, wenn keine Spieler-Generalisierung durchgeführt werden soll. Die Testreihe „Fallabruf in allen Stellungen“ ist die einzige, bei der das nicht geschieht und die damit die bei der Spieler-Generalisierung vorgesehenen Fallabrufe ermöglicht. Es wurde deshalb keine gesonderte Testreihe „Spieler-Generalisierung“ durchgeführt. Die entsprechenden Ergebnisse finden sich vielmehr im Abschnitt der Testreihe „Fallabruf in allen Stellungen“.

Tiefen der Baumsuchen Aufgrund der Tatsache, daß der betrachtete fallbasierte Algorithmus auf dem tiefenbeschränkten Minimax-Algorithmus aufbaut, war zu erwarten, daß die Ergebnisse von Experimenten mit unterschiedlichen Suchtiefen

möglicherweise starke Differenzen aufweisen würden. Deshalb wurden stellvertretend für verschiedene Alternativen die Suchtiefen 2, 4 und 6 Halbzüge untersucht. Dabei wurden für den Anziehenden und den Nachziehenden jeweils die gleichen Suchtiefen gewählt, um einen adäquaten Gegner für den fallbasierten Algorithmus zur Verfügung zu haben. Die Ergebnisse der entsprechenden Testreihen werden in den einzelnen Teilen des folgenden Abschnitts nacheinander vorgestellt.

Tiefere Suchen waren bei der Anzahl von Spielen, die für die Experimente notwendig waren, nicht mit vertretbarem Aufwand möglich. Die Entscheidung für gerade maximale Suchtiefen ist damit zu begründen, daß ohne Spieler-Generalisierung anwendbare Fälle ausschließlich in geraden Suchtiefen gefunden werden können. Daß die Bevorzugung gerader maximaler Suchtiefen sinnvoll ist, ist am augenscheinlichsten bei den Testreihen „Fallabruf nur in Terminalknoten“. Für sie wären bei ungerader maximaler Suchtiefe ohne Spieler-Generalisierung Fallabrufe unmöglich.

α-β-Baumstutzen Für die Durchführung der großen Anzahl von Spielen war es notwendig, bei den Baumsuchen α-β-Stutzen zu verwenden. Wie in Abschnitt 2.3 dargelegt, ist dies ohne Informationsverlust gegenüber dem tiefenbeschränkten Minimax-Algorithmus möglich. Als diejenige Variante, die bei den Voruntersuchungen die besten Resultate erzielte, wurde in den in diesem Kapitel beschriebenen Experimenten das α-β-Stutzen bei abhängigem Bewerten der Folgepositionen mit heuristischer Bestimmung der Reihenfolge der zu betrachteten Spielbaumknoten gewählt.

4.3 Ergebnisse der Experimente

In diesem Abschnitt werden die Ergebnisse der Untersuchungen zur Leistungsfähigkeit des fallbasierten Lernens von Bewertungsfunktionen dargelegt. Zu Beginn werden zu den drei betrachteten Suchtiefen Vergleichstestreihen ohne Lernen angegeben. Anschließend wird die für die weiteren Testreihen benötigte „Standard“-Testreihe definiert und ihre Ergebnisse vorgestellt. Durch Änderungen einzelner in Kapitel 3 erklärter Parameter werden ausgehend von dieser „Standard“-Testreihe die anderen untersuchten Testreihen gebildet, die nachfolgend beschrieben werden.

4.3.1 Vergleichstestreihen ohne Lernen

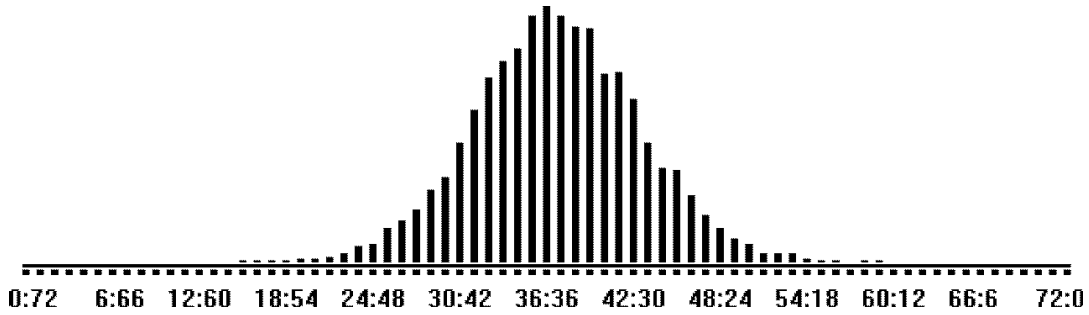

Abbildung 4.8: Häufigkeit der Spielergebnisse Minimax2 gegen Minimax2

Wenn sowohl der Anziehende als auch der Nachziehende die Folgepositionen mit dem tiefenbeschränkten Minimax-Algorithmus bei einer maximalen Suchtiefe von 2 Halbzügen bewerten, ergibt sich nach 10 000 Spielen die in Abbildung 4.8 dargestellte Häufigkeitsverteilung der Spielergebnisse. Statistische Daten zu der Testreihe finden sich in Tabelle B.4.

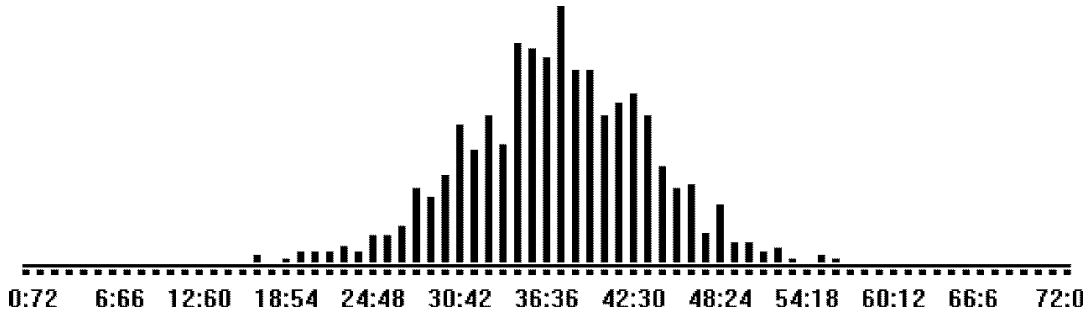

Abbildung 4.9: Häufigkeit der Spielergebnisse Minimax4 gegen Minimax4

Wenn beide Spieler die Folgepositionen mit dem tiefenbeschränkten Minimax-Algorithmus bei einer maximalen Suchtiefe von 4 Halbzügen bewerten, ergibt sich die in Abbildung 4.9 dargestellte Häufigkeitsverteilung der Spielergebnisse. Zu dieser Testreihe wurden aus Zeitgründen nur 1 000 Spiele durchgeführt.

Ebenfalls aus Gründen des hohen Zeitaufwands wurden zu der Testreihe, bei der beide Spieler die Folgepositionen mit dem tiefenbeschränkten Minimax-Algorithmus bei einer maximalen Suchtiefe von 6 Halbzügen bewerten, nur 1 000 Spiele durchgeführt. Im Gegensatz zu den beiden oben vorgestellten Testreihen weicht die in Abbildung 4.10

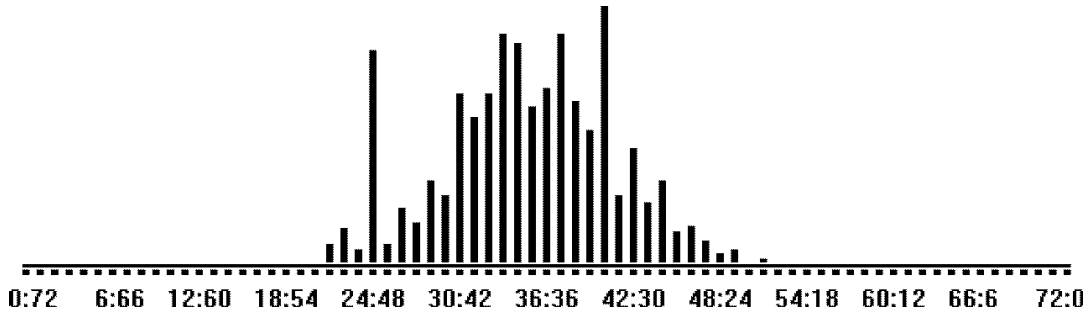

Abbildung 4.10: Häufigkeit der Spielergebnisse Minimax6 gegen Minimax6

dargestellte Häufigkeitsverteilung der Spielergebnisse stark von der zu erwartenden – stetigen, aber der Normalverteilung ähnlichen – Verteilung ab.

4.3.2 Die „Standard“-Testreihen

Die „Standard“-Testreihen haben folgende Eigenschaften:

Fallabruf: nur in eigenen Knoten,

Ähnlichkeitsrelation: Identität ohne Generalisierung,

Auswahlstrategie: zufällige Auswahl.

Die aussagekräftigste Darstellung des Verlaufs der Gewinne und der Fallabrufe ist bei allen in diesem Kapitel vorgestellten Testreihen diejenige, bei der jeweils die Daten von 100 Spielen zusammengefaßt werden. Die entsprechenden Ansichten werden hier nur für die „Standard“-Testreihen angegeben.

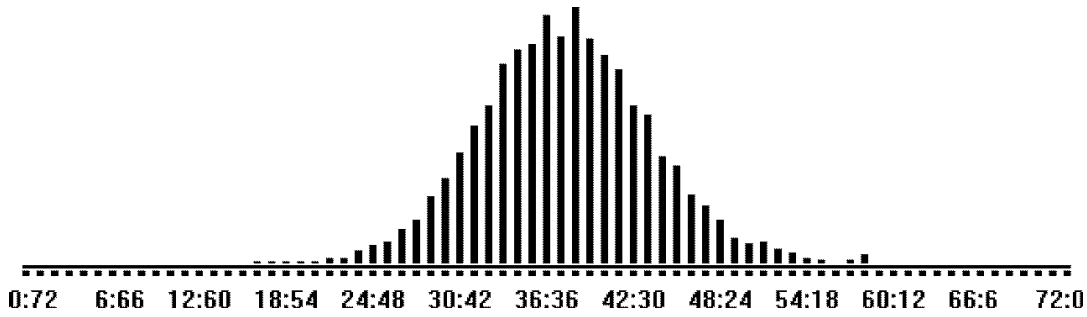

Abbildung 4.11: „Standard-Testreihe“ Tiefe 2. Häufigkeit der Spielergebnisse

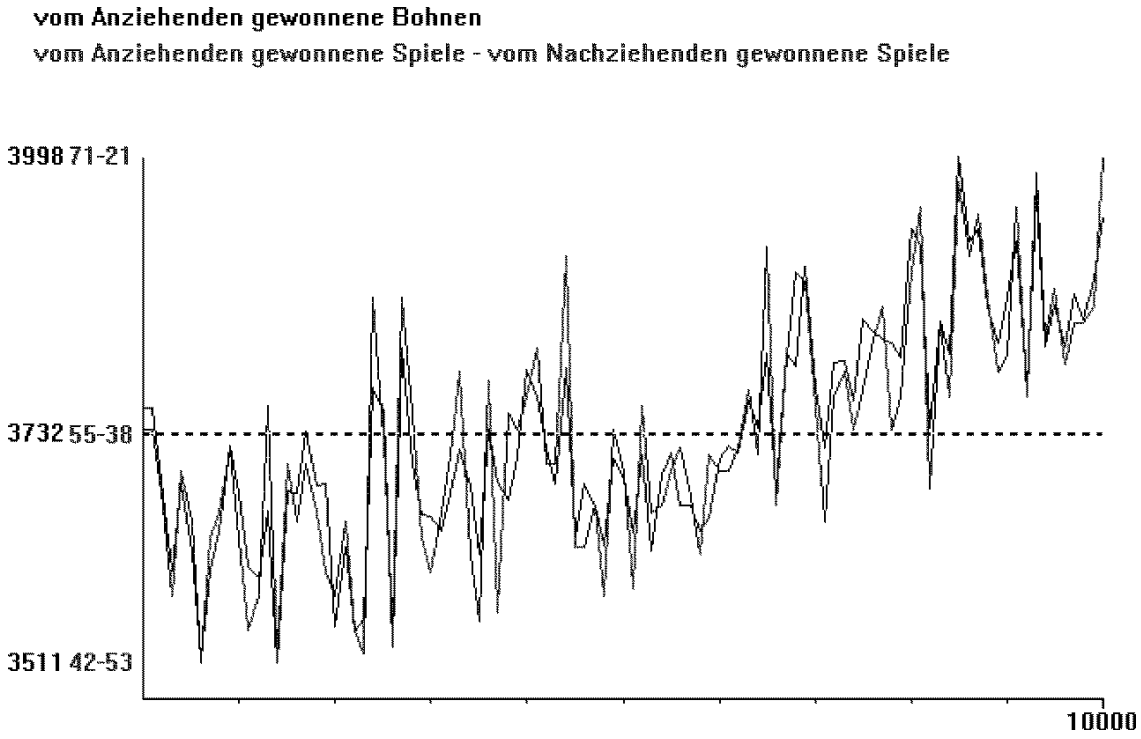

Abbildung 4.12: „Standard-Testreihe“ Tiefe 2. Verlauf der Ergebnisse

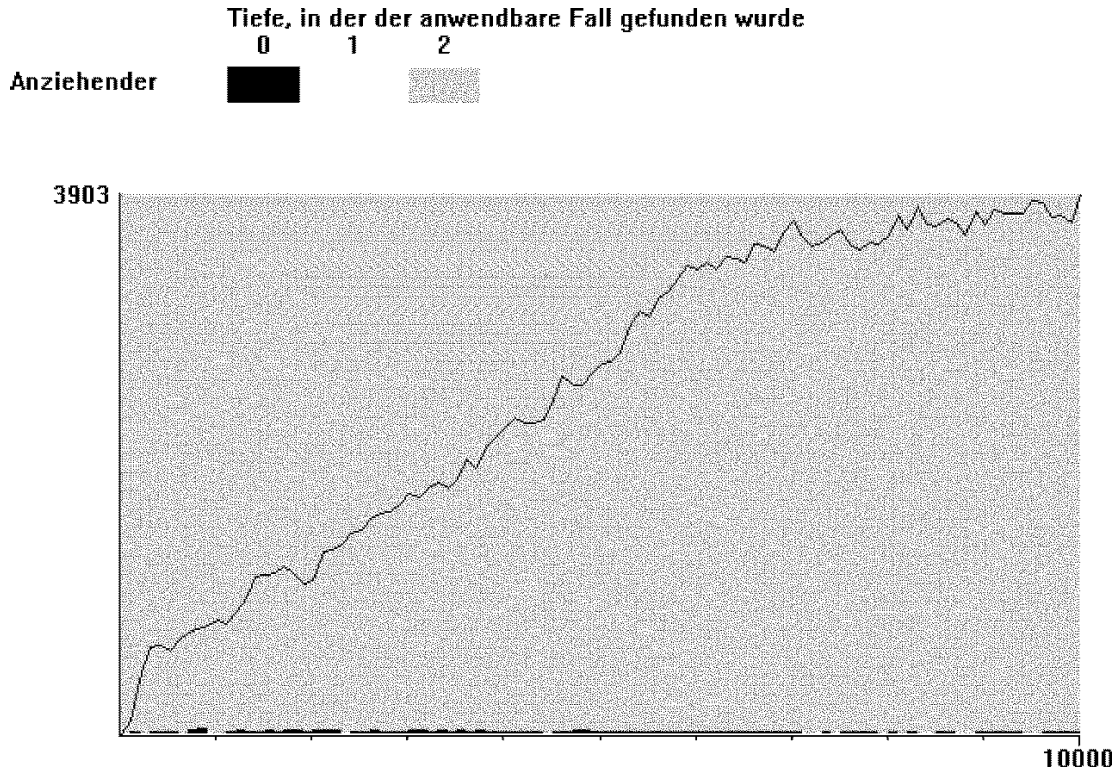

Abbildung 4.13: „Standard-Testreihe“ Tiefe 2. Verlauf der Fallabrufe

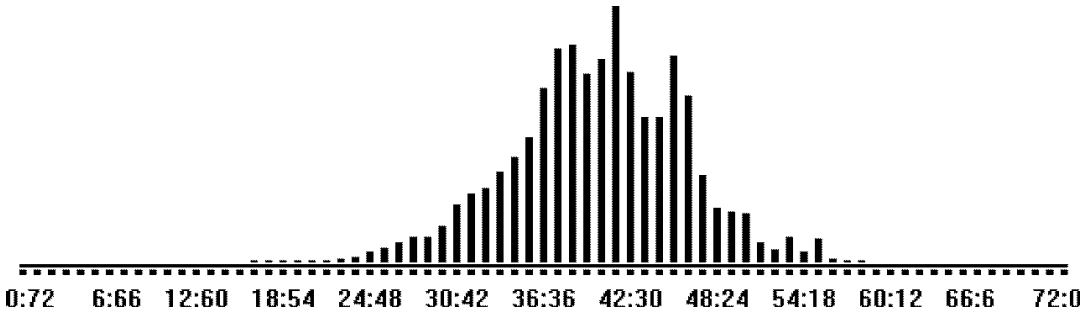

Abbildung 4.14: „Standard-Testreihe“ Tiefe 4. Häufigkeit der Spielergebnisse

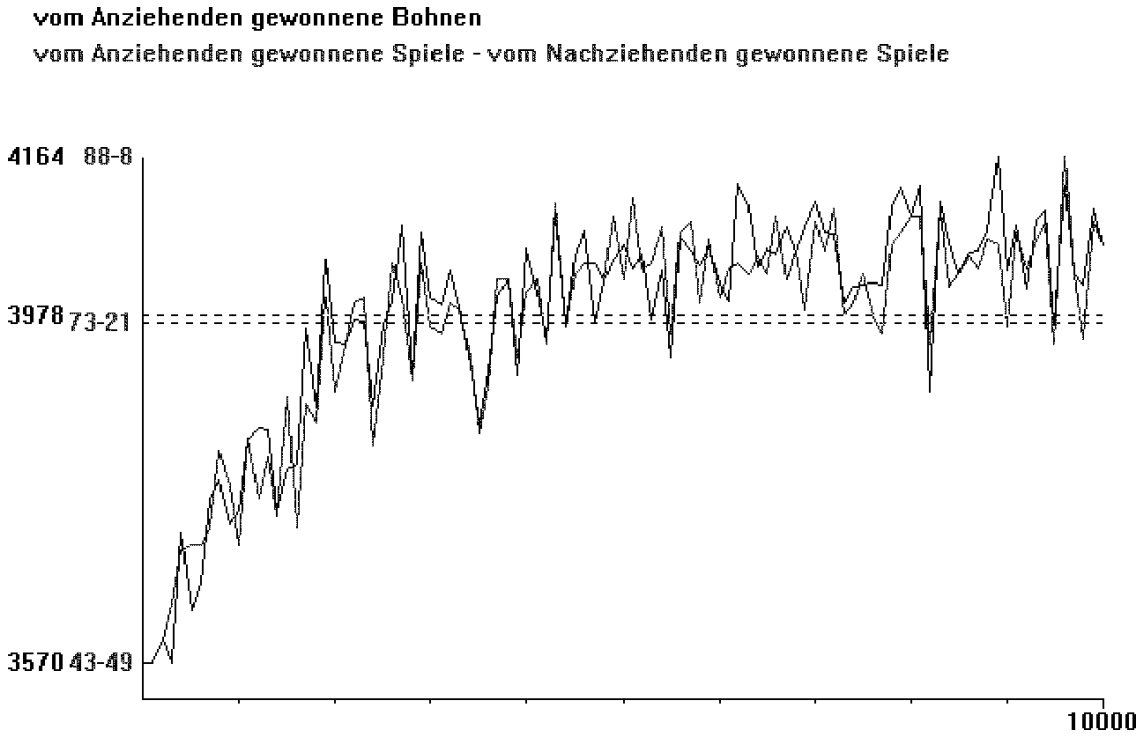

Abbildung 4.15: „Standard-Testreihe“ Tiefe 4. Verlauf der Ergebnisse

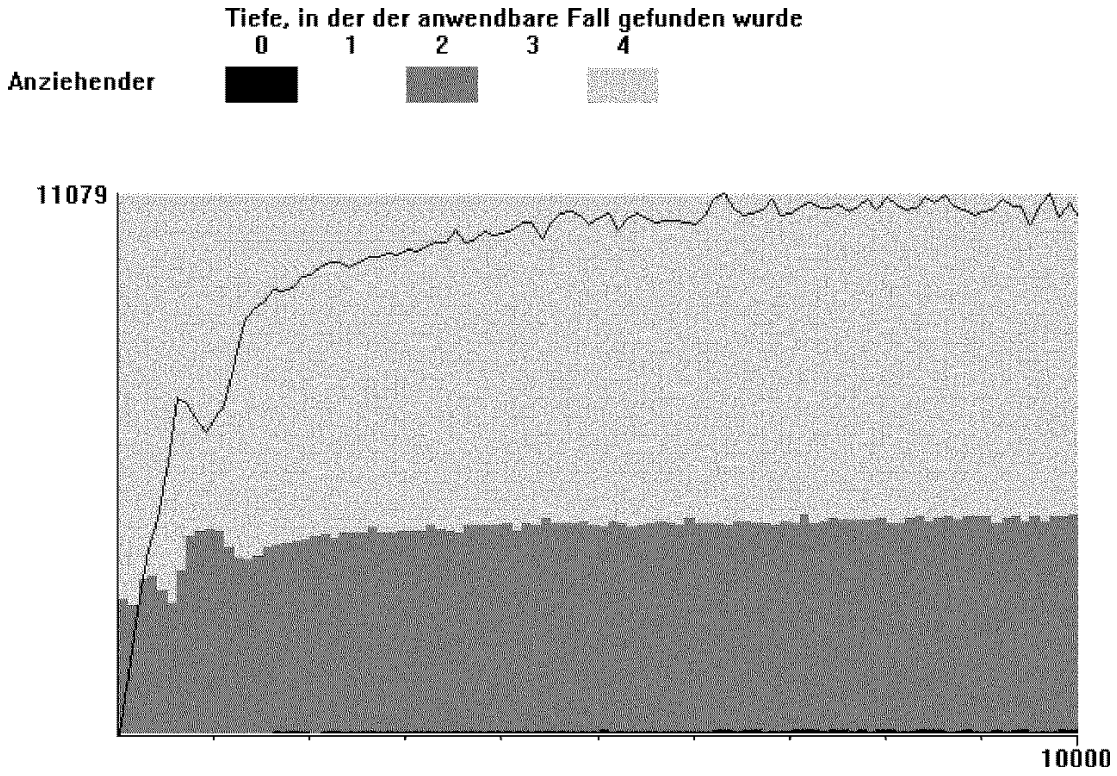

Abbildung 4.16: „Standard-Testreihe“ Tiefe 4. Verlauf der Fallabrufe

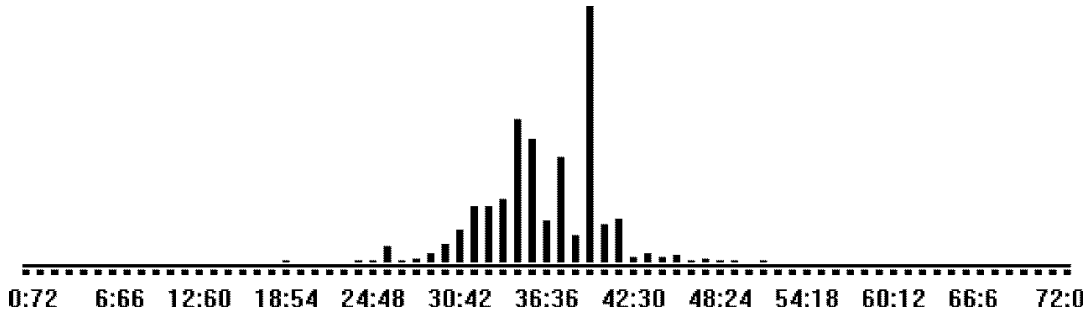

Abbildung 4.17: „Standard-Testreihe“ Tiefe 6. Häufigkeit der Spielergebnisse

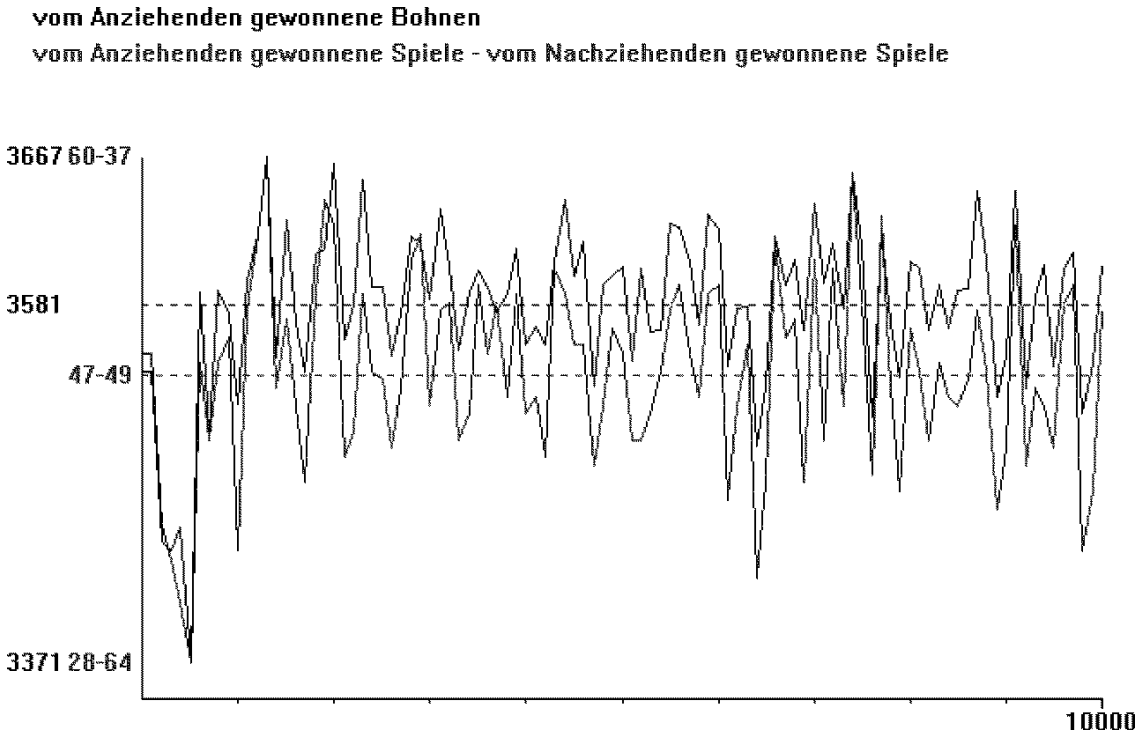

Abbildung 4.18: „Standard-Testreihe“ Tiefe 6. Verlauf der Ergebnisse

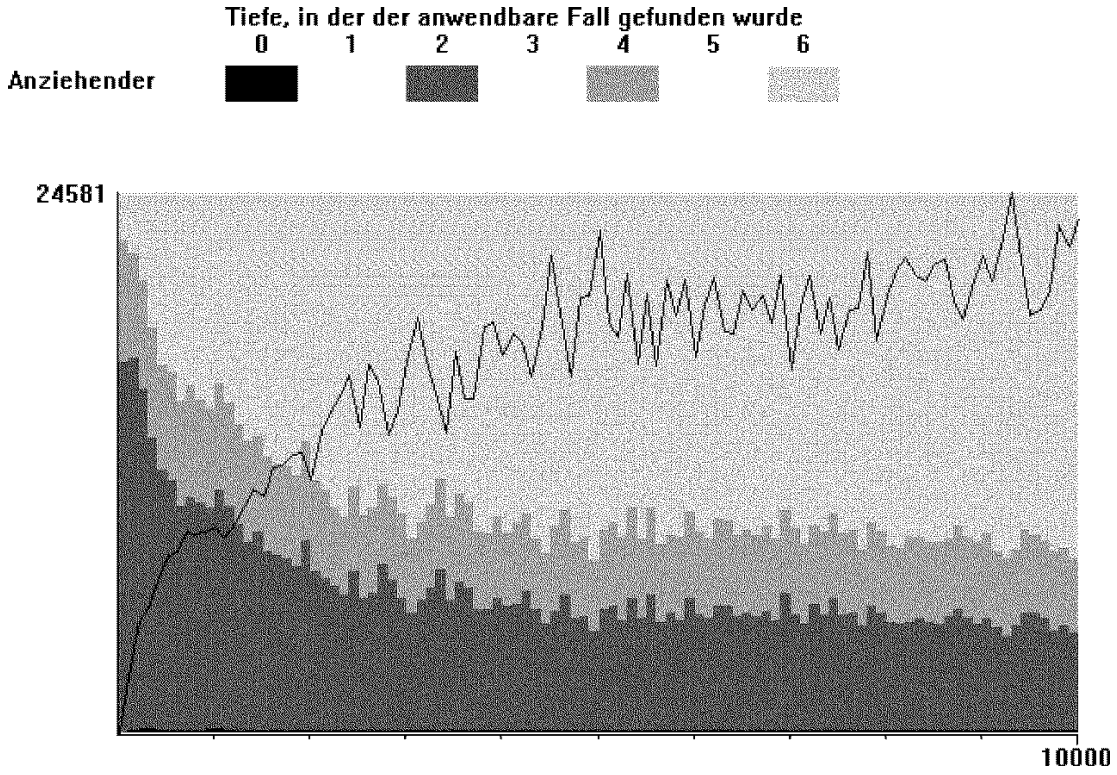

Abbildung 4.19: „Standard-Testreihe“ Tiefe 6. Verlauf der Fallabrufe

4.3.3 Fallabruf während der Baumsuche

Fallabruf in allen Knoten

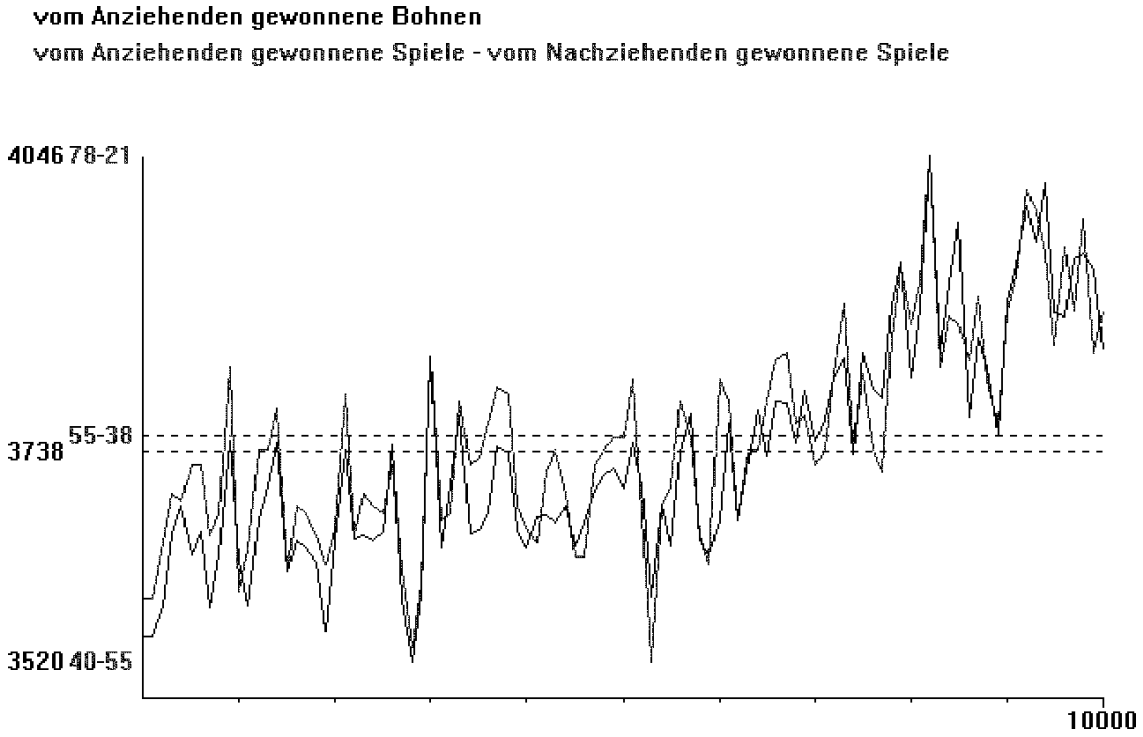

Abbildung 4.20: „Fallabruf in allen Knoten“ Tiefe 2. Verlauf der Ergebnisse

Die Spieler-Generalisierung wurde in der Testreihe „Fallabruf in allen Knoten“ umgesetzt. Die Effekte der Generalisierung waren jedoch wie erwartet gering. Bei den Suchtiefen 2 und 4 war der Anteil von Fallabrufen in gegnerischen Knoten kleiner als zwei Prozent. Nur für die Suchtiefe 6 ergab sich ein Anteil der Fallabrufe in gegnerischen Knoten von fast 20 Prozent und eine relativ hohe Gesamtzahl von Fallabrufen. Diese schlug sich aber nicht in besonders guten Ergebnissen nieder.

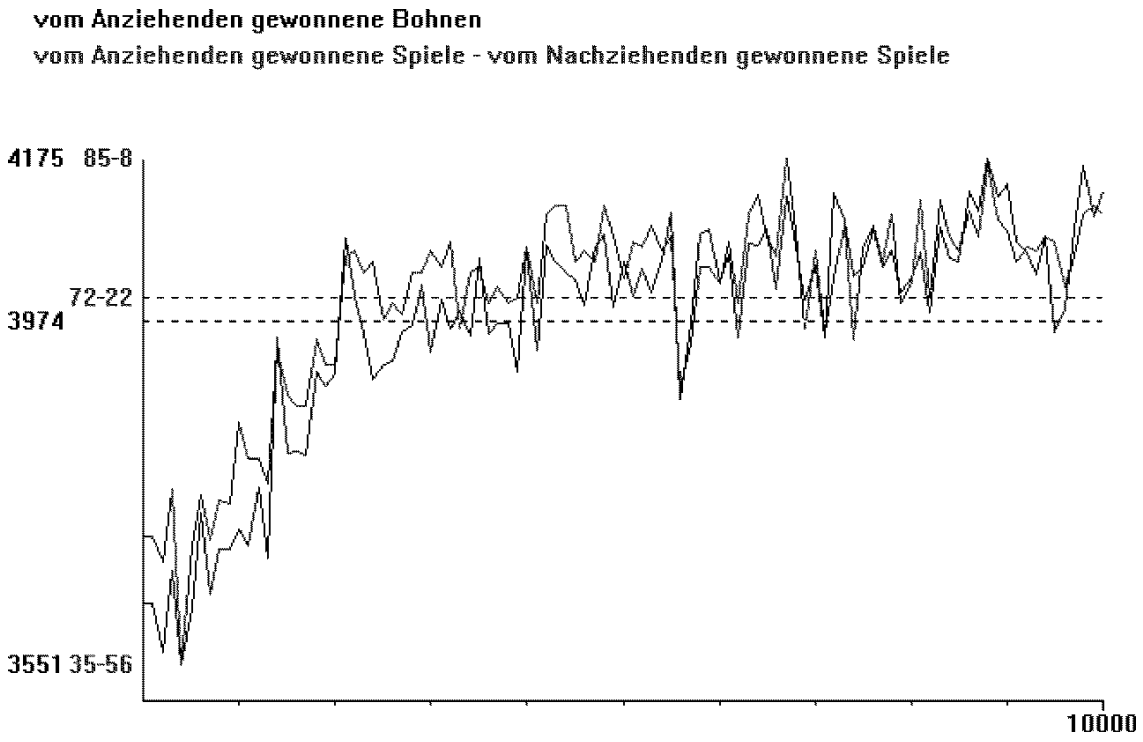

Abbildung 4.21: „Fallabruf in allen Knoten“ Tiefe 4. Verlauf der Ergebnisse

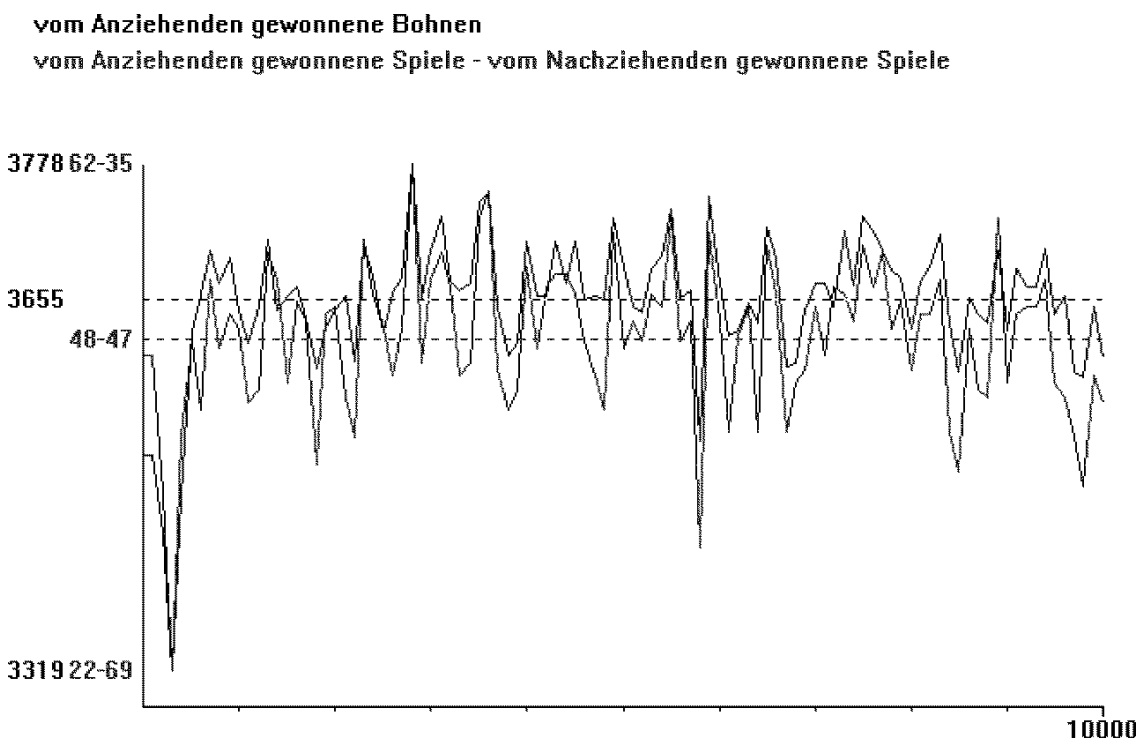

Abbildung 4.22: „Fallabruf in allen Knoten“ Tiefe 6. Verlauf der Ergebnisse

Fallabruf nur in Terminalknoten

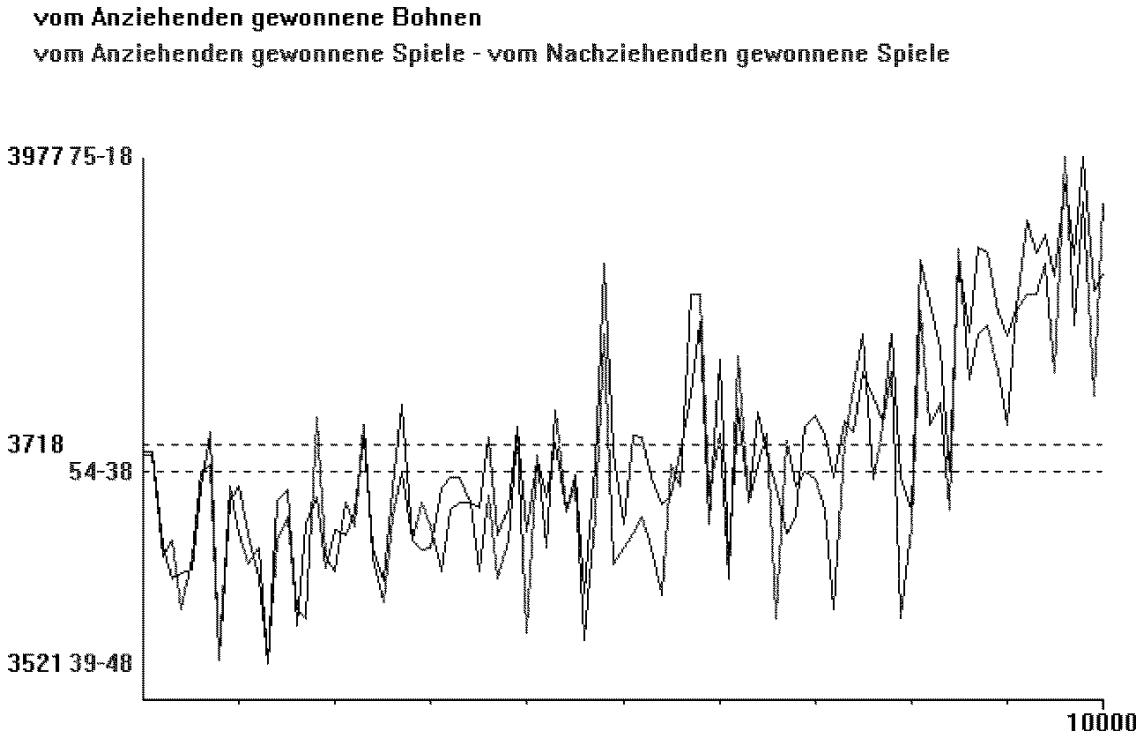

Abbildung 4.23: „Fallabruf nur in Terminalknoten“ Tiefe 2. Verlauf der Ergebnisse

Die Testreihe „Fallabruf nur in Terminalknoten“ wies bei allen betrachteten Suchtiefen die geringsten Unterschiede zu den Ergebnissen der Vergleichstestreihen ohne Lernen auf. Lerneffekte waren erst am Ende der 10 000 Spiele zu erkennen.

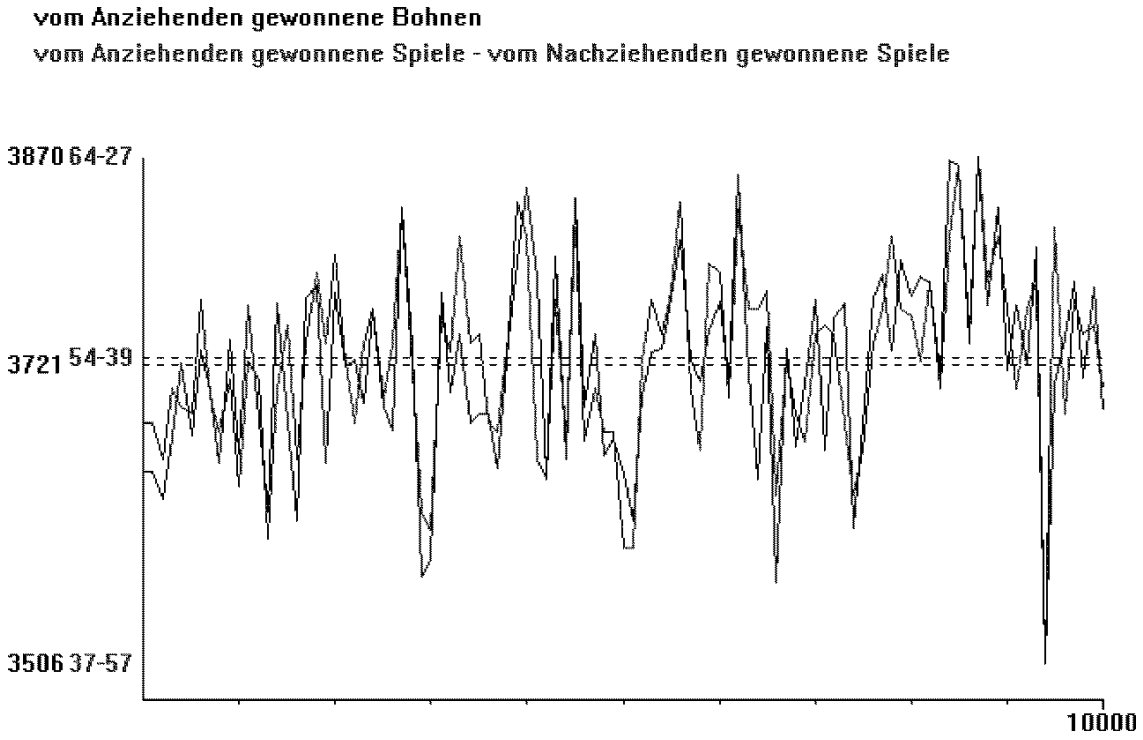

Abbildung 4.24: „Fallabruf nur in Terminalknoten“ Tiefe 4. Verlauf der Ergebnisse

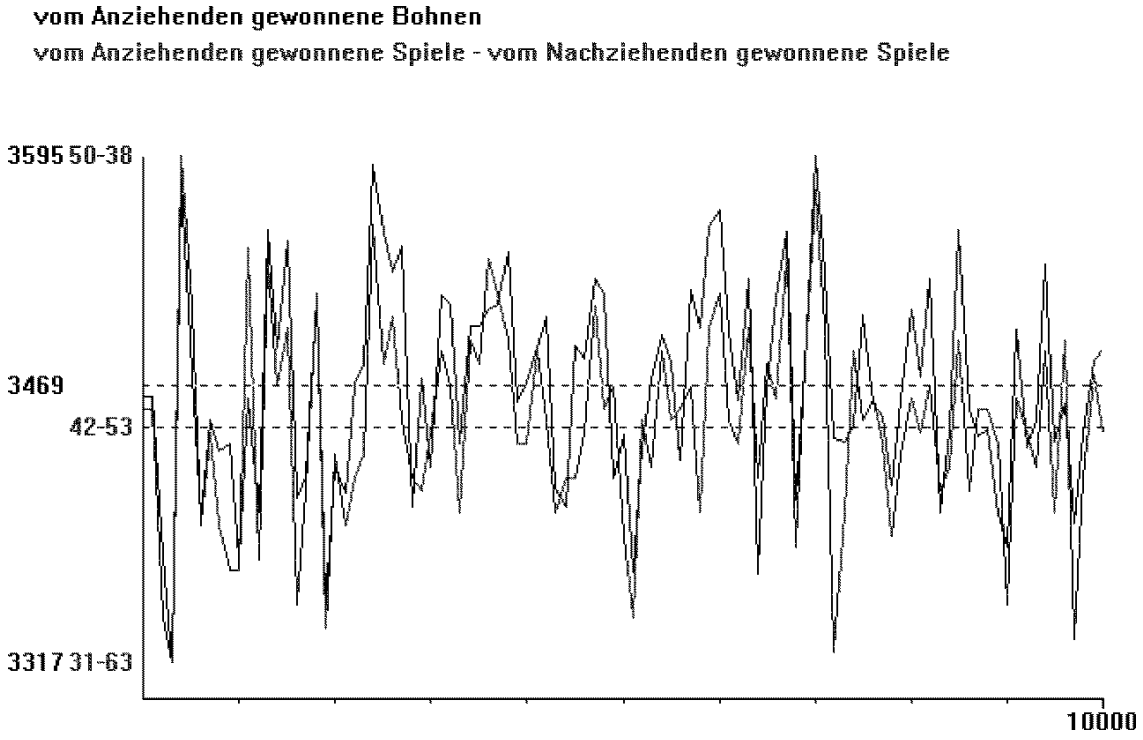

Abbildung 4.25: „Fallabruf nur in Terminalknoten“ Tiefe 6. Verlauf der Ergebnisse

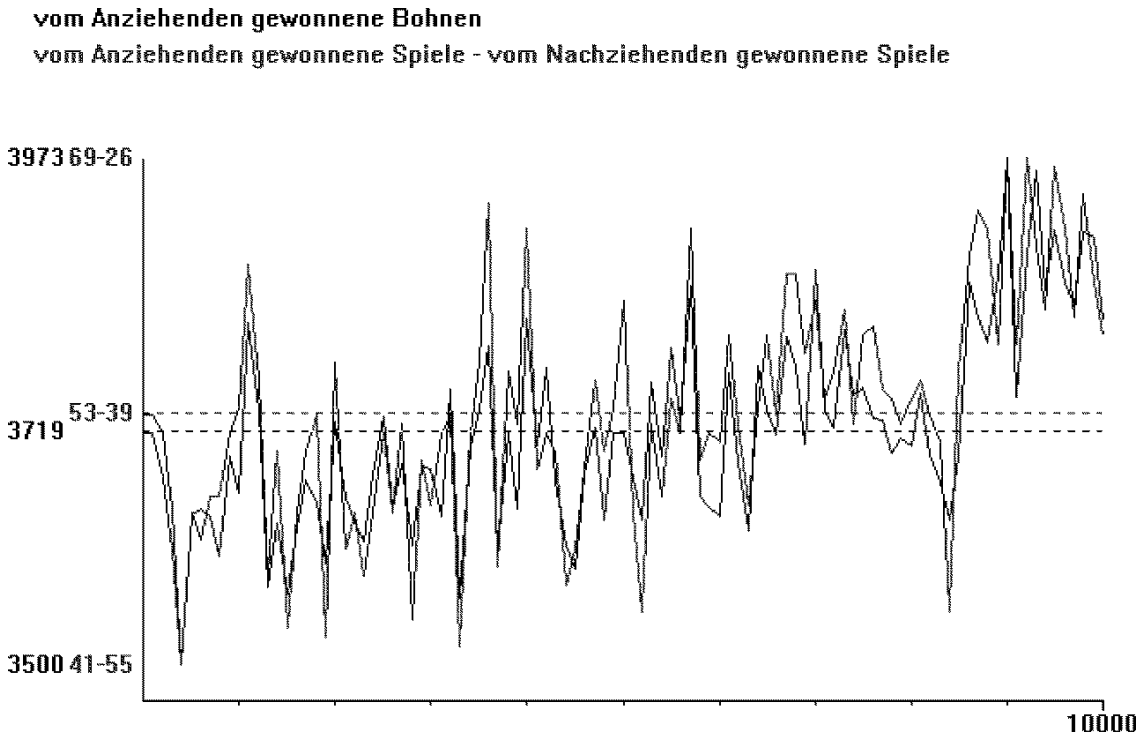

Abbildung 4.26: „Gewinnmulden-Generalisierung“ Tiefe 2. Verlauf der Ergebnisse

4.3.4 Ähnlichkeitsrelation

Spieler-Generalisierung

Wie in Abschnitt 4.2 dargelegt, wurde die Spieler-Generalisierung in der Form der Testreihe „Fallabruf in allen Knoten“ realisiert. Ihre Ergebnisse wurden bereits im Abschnitt 4.3.3 vorgestellt.

Gewinnmulden-Generalisierung

Die Gewinnmulden-Generalisierung führte erwartungsgemäß zu einer wesentlich höheren Anzahl von Fallabrufen. Während diese Tatsache bei den Suchtiefen 2 und 4 noch nicht augenscheinlich war, entspricht bei Suchtiefe 6 die Anzahl der Fallabrufe einem Vielfachen der Abrufe bei anderen Testreihen. Dieser Unterschied bei kleinen und großen Suchtiefen ist damit zu erklären, daß bei niedrigen Suchtiefen häufig bekannte Stellungen bei der Baumsuche bewertet werden und deshalb keine Generalisierung notwendig ist, während die Anzahl bekannter Stellungen im Vergleich zu der Anzahl der bei großen Suchtiefen im Spielbaum zu durchsuchenden Stellungen sehr gering ist. Zur Bewertung der vielen unbekannten Stellungen wird die Generalisierung eingesetzt.

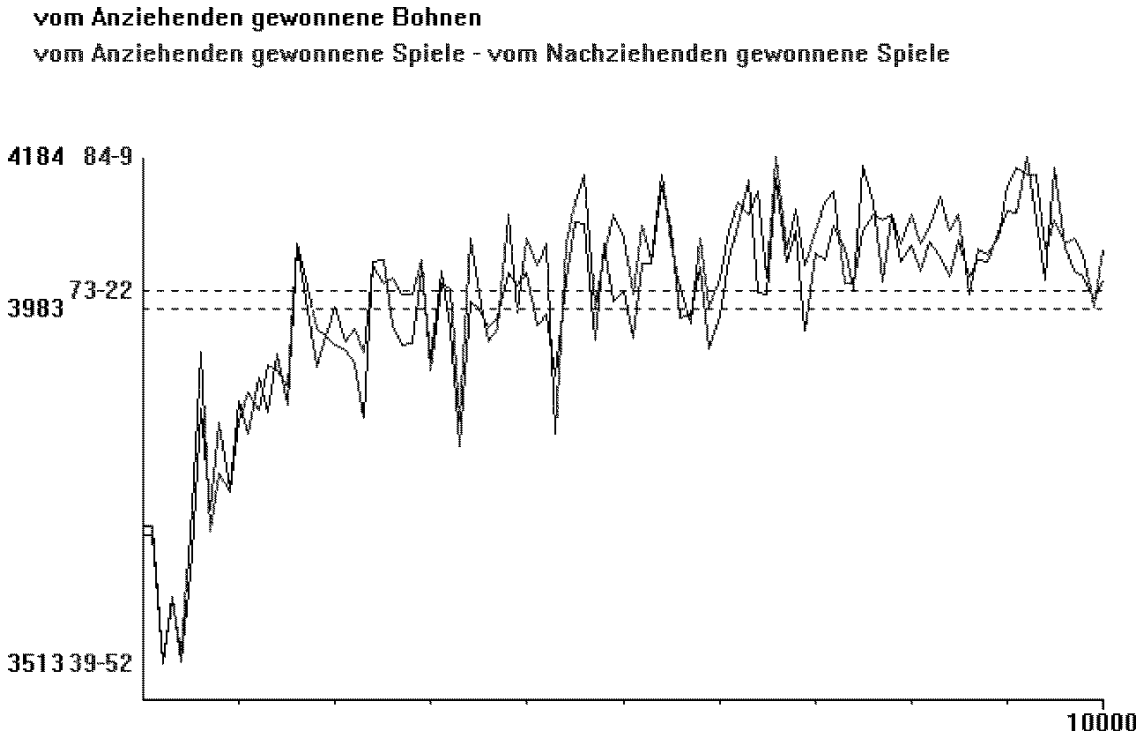

Abbildung 4.27: „Gewinnmulden-Generalisierung“ Tiefe 4. Verlauf der Ergebnisse

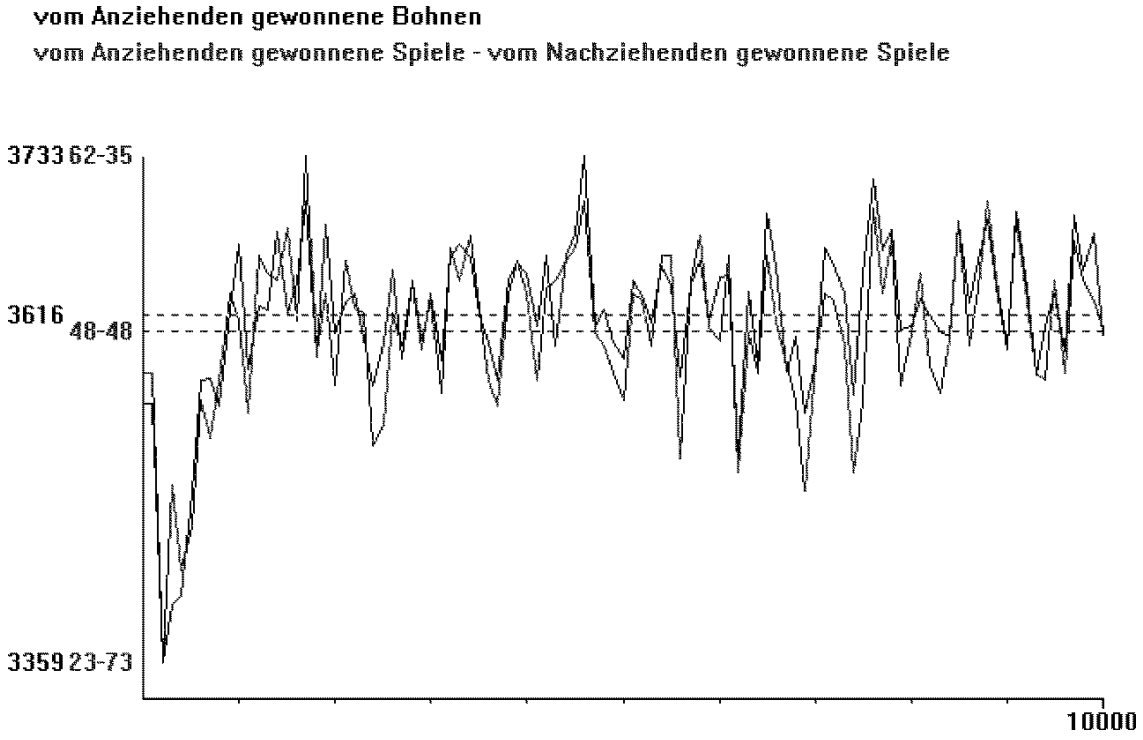

Abbildung 4.28: „Gewinnmulden-Generalisierung“ Tiefe 6. Verlauf der Ergebnisse

4.3.5 Auswahlstrategie

„bekannte“

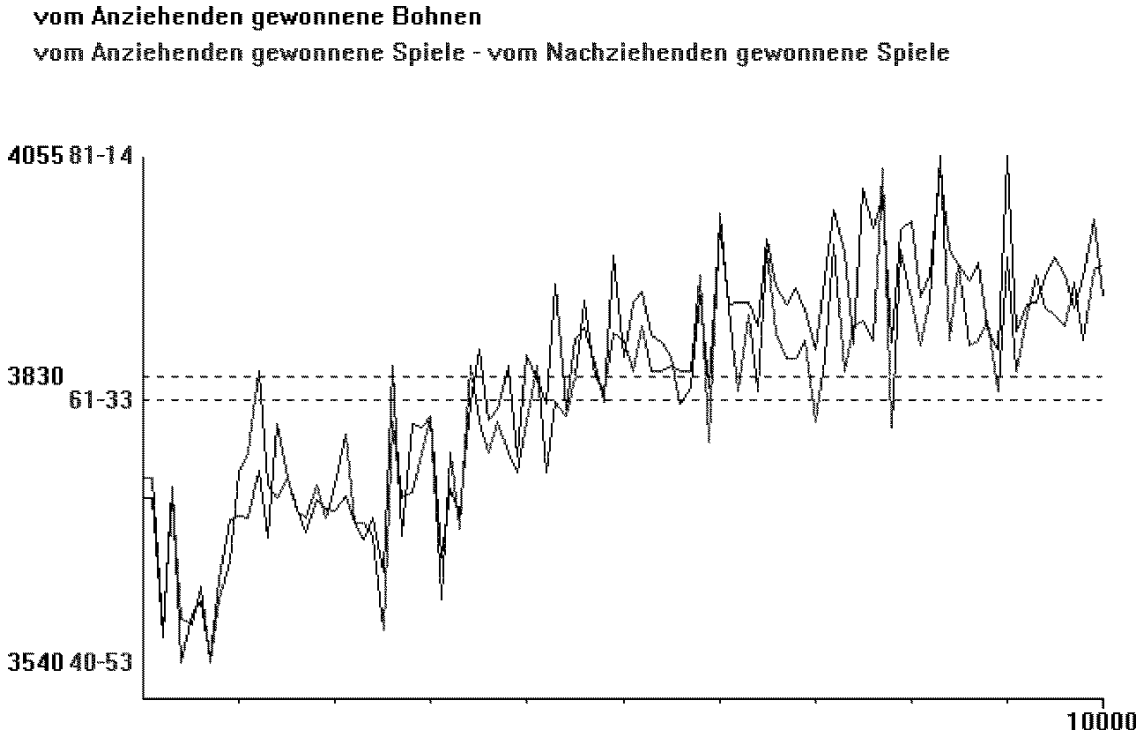

Abbildung 4.29: „bekannte“ Tiefe 2. Verlauf der Ergebnisse

Die Testreihen „bekannte“ und „bekannteste“ erzielen im Vergleich zu den meisten anderen Testreihen bessere Ergebnisse. Besonders ist dabei hervorzuheben, daß die Verbesserungen sehr früh eintreten. Auffällig ist, daß bei Suchtiefe 2 die Spiele sehr häufig mit 58:14 enden. Der Algorithmus hat offensichtlich gelernt, daß dieses Ergebnis außerordentlich gut ist, und wiederholt nun häufig seine Entscheidungen.

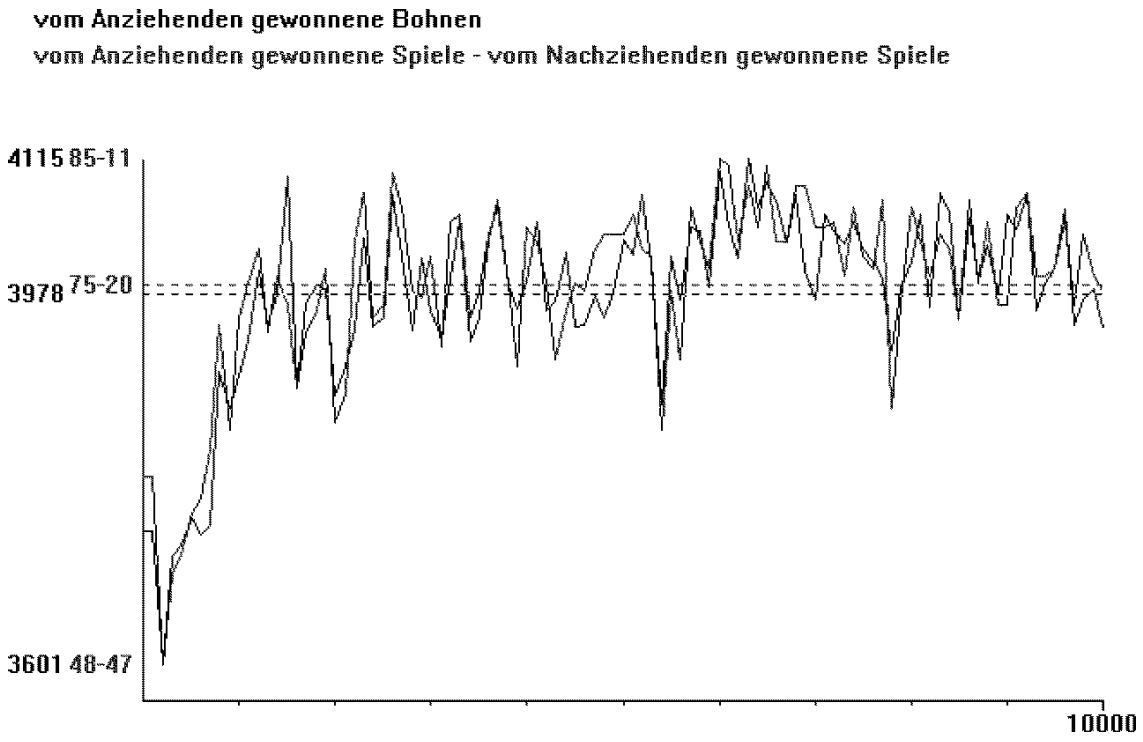

Abbildung 4.30: „bekannte“ Tiefe 4. Verlauf der Ergebnisse

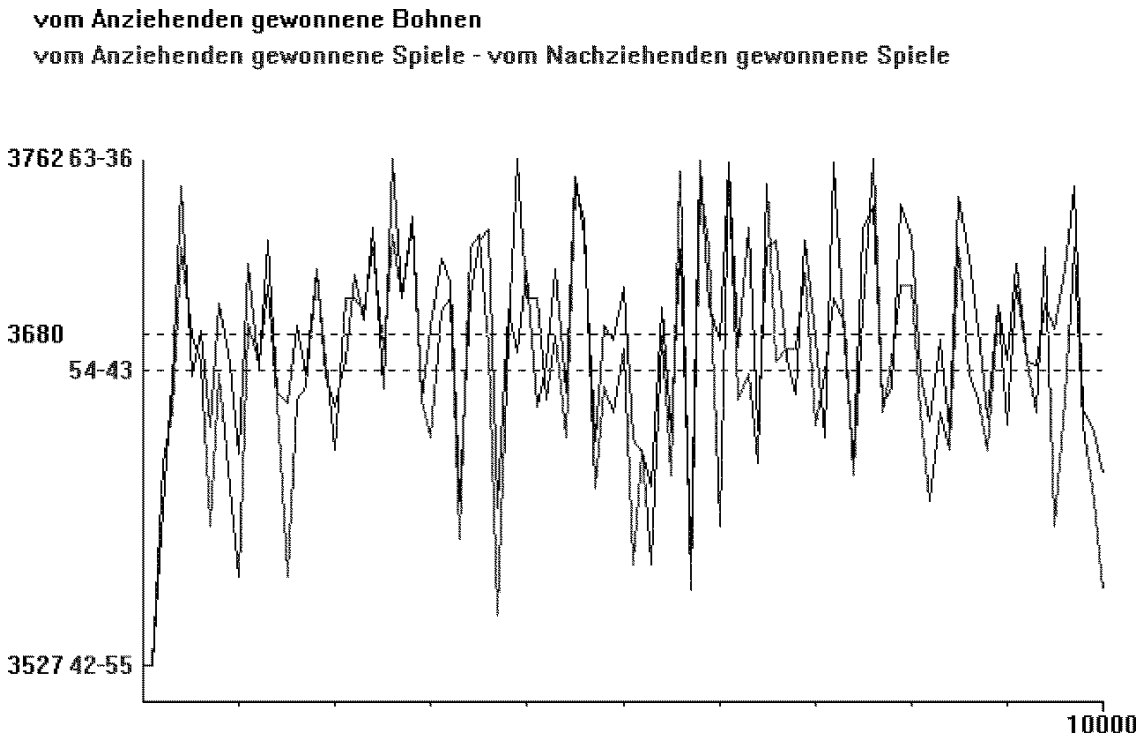

Abbildung 4.31: „bekannte“ Tiefe 6. Verlauf der Ergebnisse

„bekannteste“

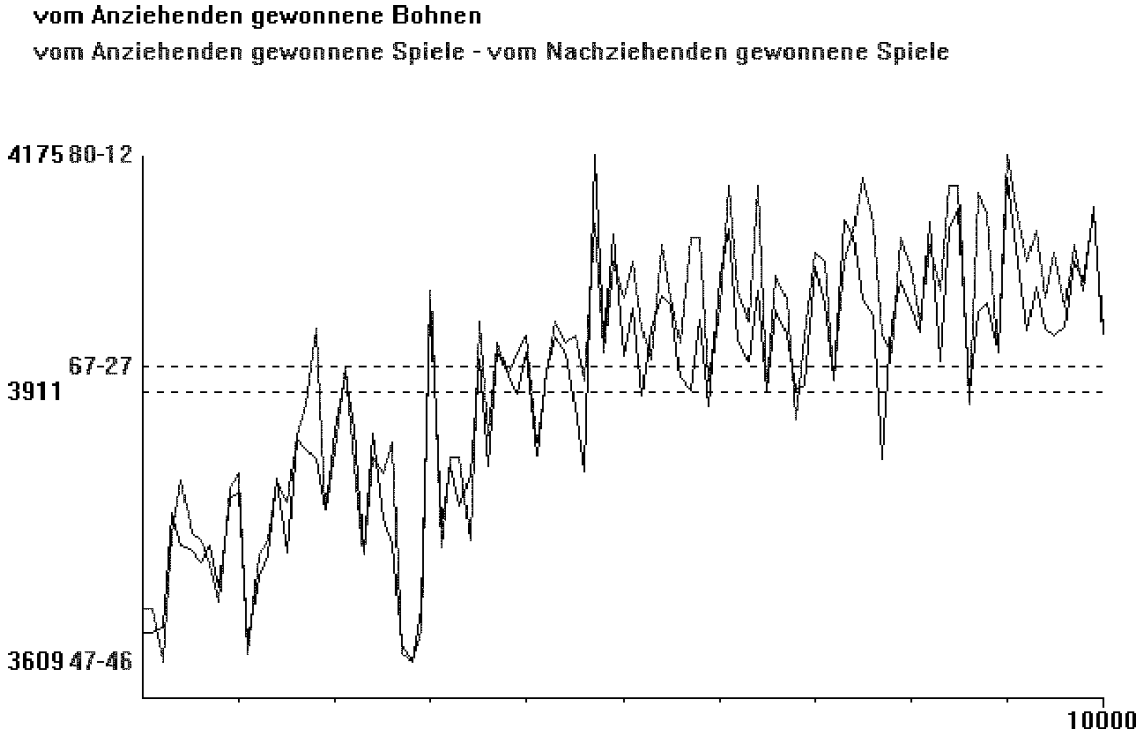

Abbildung 4.32: „bekannteste“ Tiefe 2. Verlauf der Ergebnisse

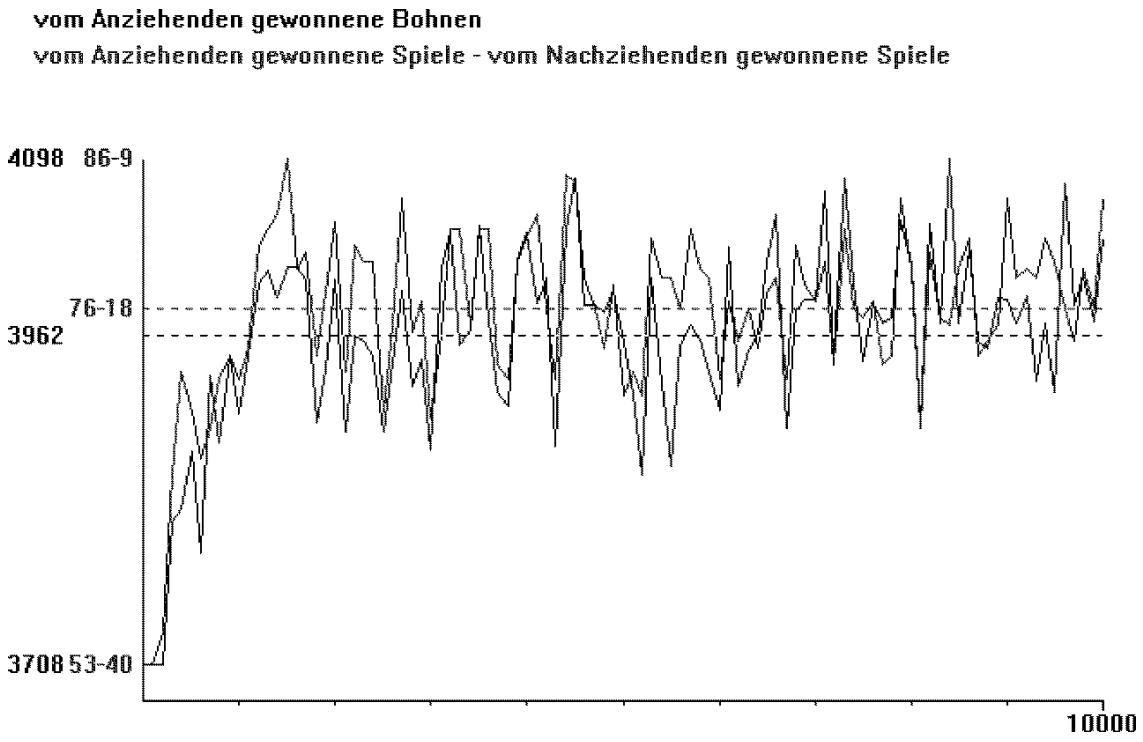

Abbildung 4.33: „bekannteste“ Tiefe 4. Verlauf der Ergebnisse

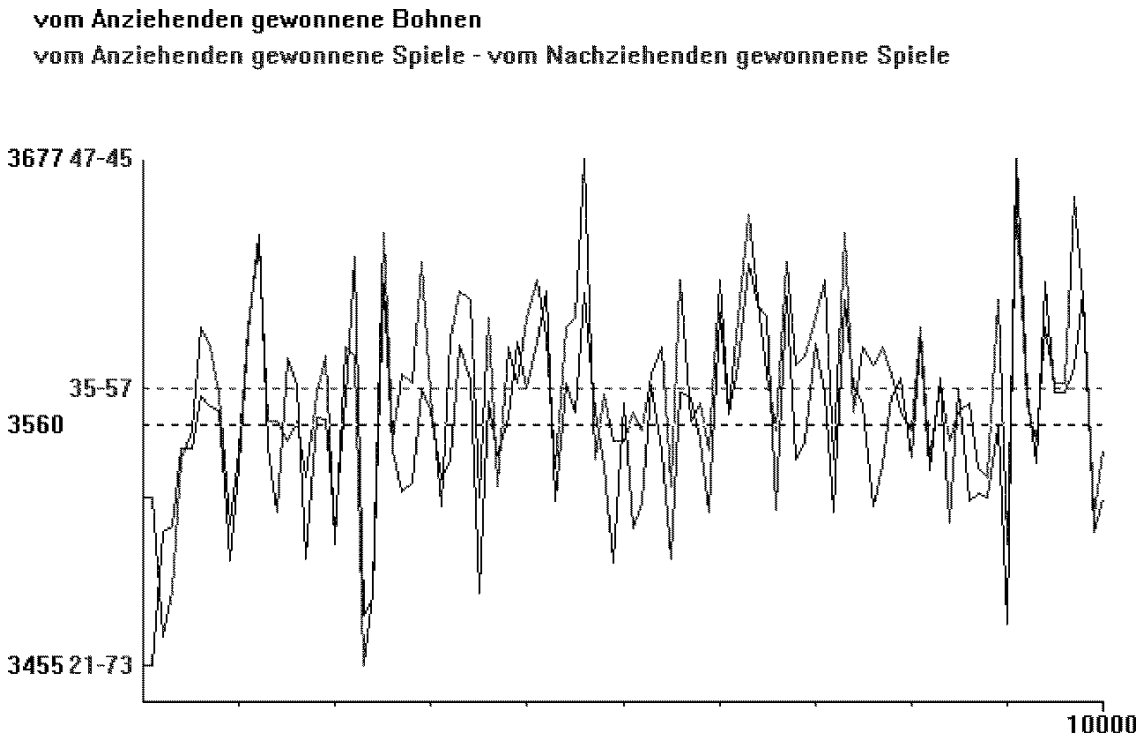

Abbildung 4.34: „bekannteste“ Tiefe 6. Verlauf der Ergebnisse

„unbekannte“

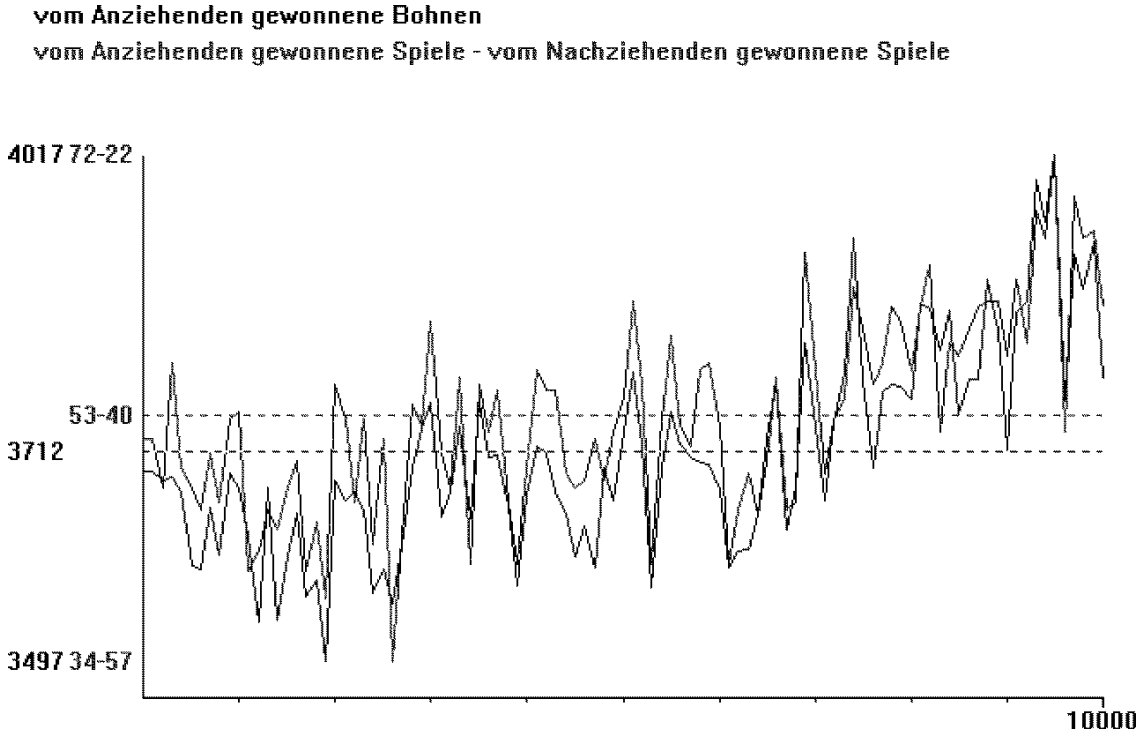

Abbildung 4.35: „unbekannte“ Tiefe 2. Verlauf der Ergebnisse

Die Testreihen „unbekannte“ und „unbekannteste“ erbrachten die ungewöhnlichsten Ergebnisse der Experimente. Während sie bei einer Suchtiefe von 4 Halbzügen die besten Resultate erzielten, kann der Verlauf der Testreihen bei Suchtiefe 6 nur katastrophal genannt werden. Sie sind die einzigen Testreihen überhaupt, die – bei einer Suchtiefe von 6 Halbzügen – zu schlechteren Spielergebnissen führten als ohne Lernen. Von Beginn an sanken die durchschnittlichen Ergebnisse innerhalb weniger hundert Spiele stark ab, ohne daß sie im Verlauf der 10 000 Spiele der Testreihen wieder anstiegen. Im Gesamtergebnis konnte der Lernende weniger als ein Fünftel der Spiele gewinnen. Ursache dafür ist vermutlich eines der grundlegenden Probleme des Verfahrens: Ein in Wirklichkeit guter Halbzug kann durch einmalige fehlerhafte Bewertung als schlecht eingeschätzt werden und verliert diese schlechte Bewertung nicht mehr, weil die fallbasierten Bewertungen seiner Folgepositionen nicht mehr neu berechnet werden. Dieser Effekt wird durch die feine Ähnlichkeitsrelation ermöglicht bzw. verstärkt.

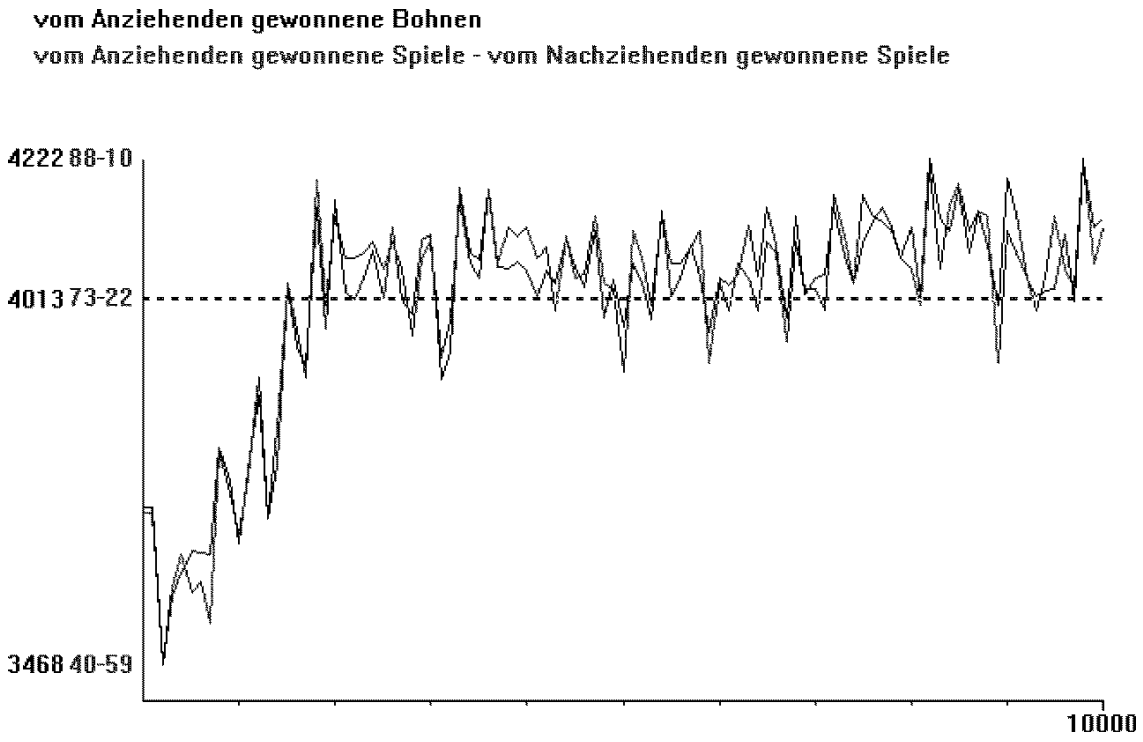

Abbildung 4.36: „unbekannte“ Tiefe 4. Verlauf der Ergebnisse

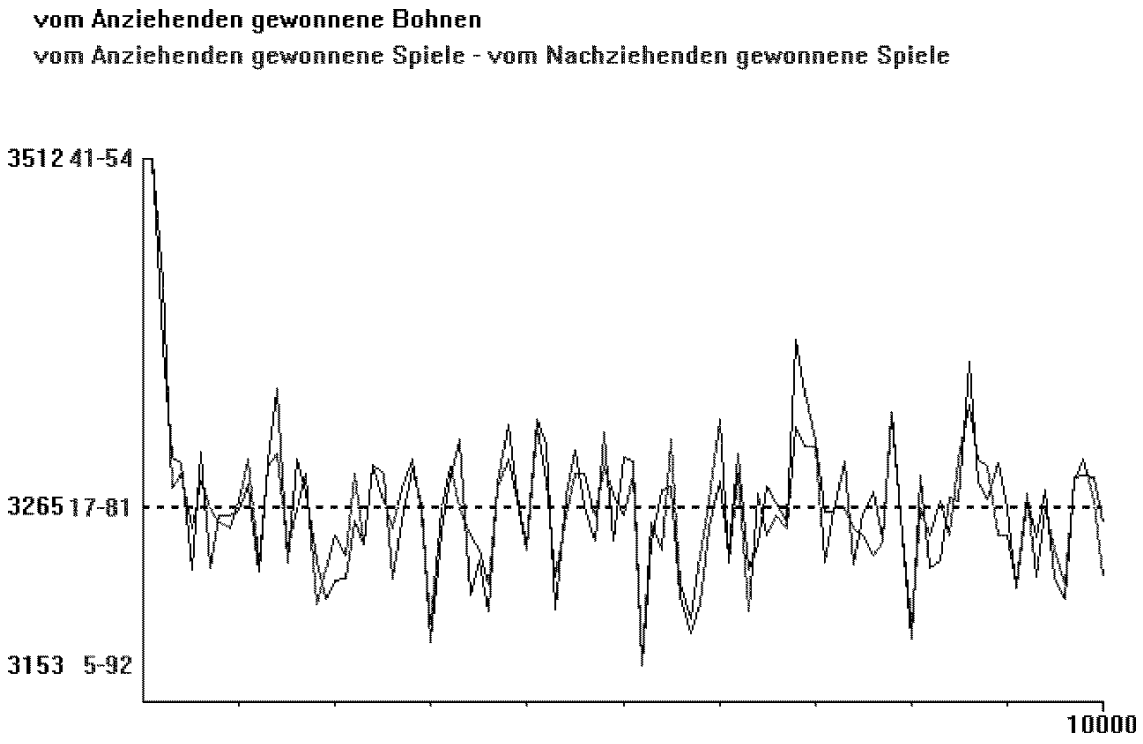

Abbildung 4.37: „unbekannte“ Tiefe 6. Verlauf der Ergebnisse

„unbekannteste“

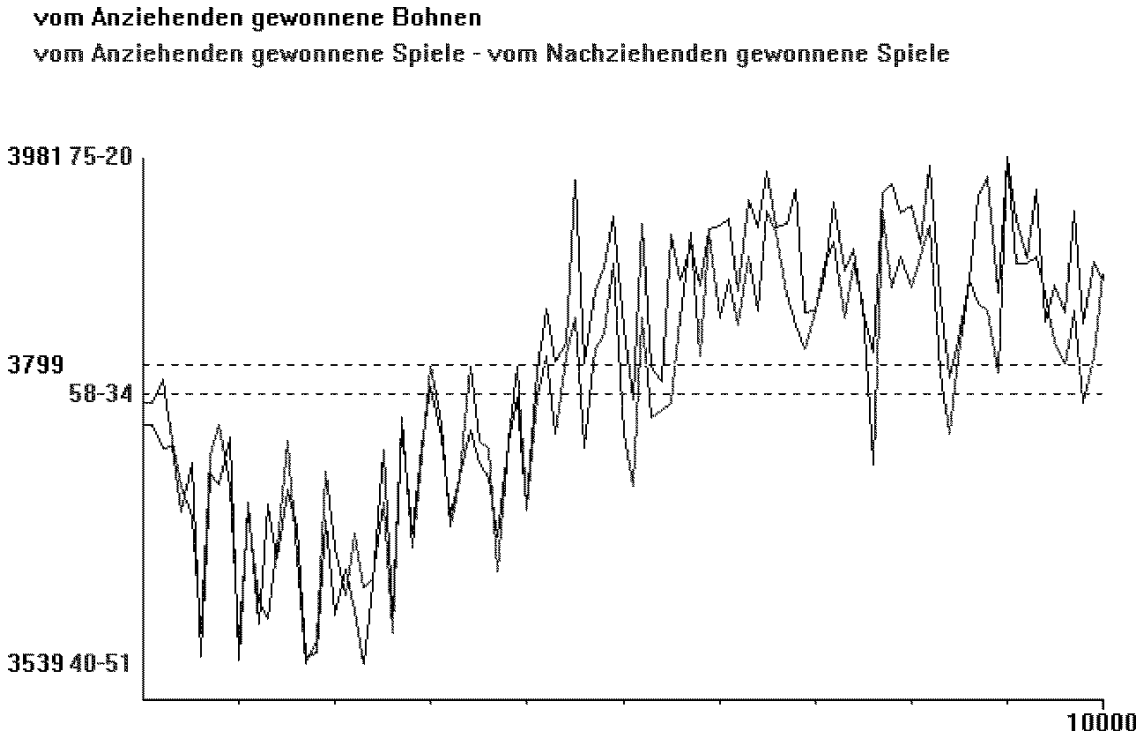

Abbildung 4.38: „unbekannteste“ Tiefe 2. Verlauf der Ergebnisse

Die Ergebnisse dieser Testreihe unterscheiden sich nur minimal von denen der Testreihe „unbekannte“.

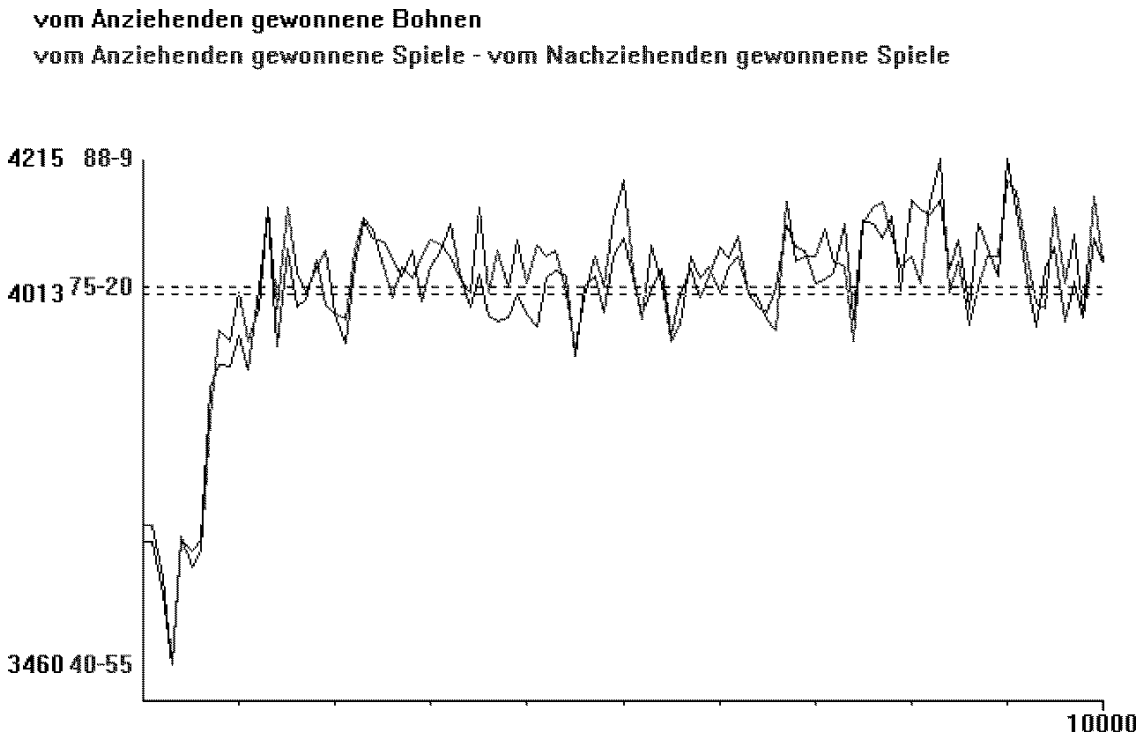

Abbildung 4.39: „unbekannteste“ Tiefe 4. Verlauf der Ergebnisse

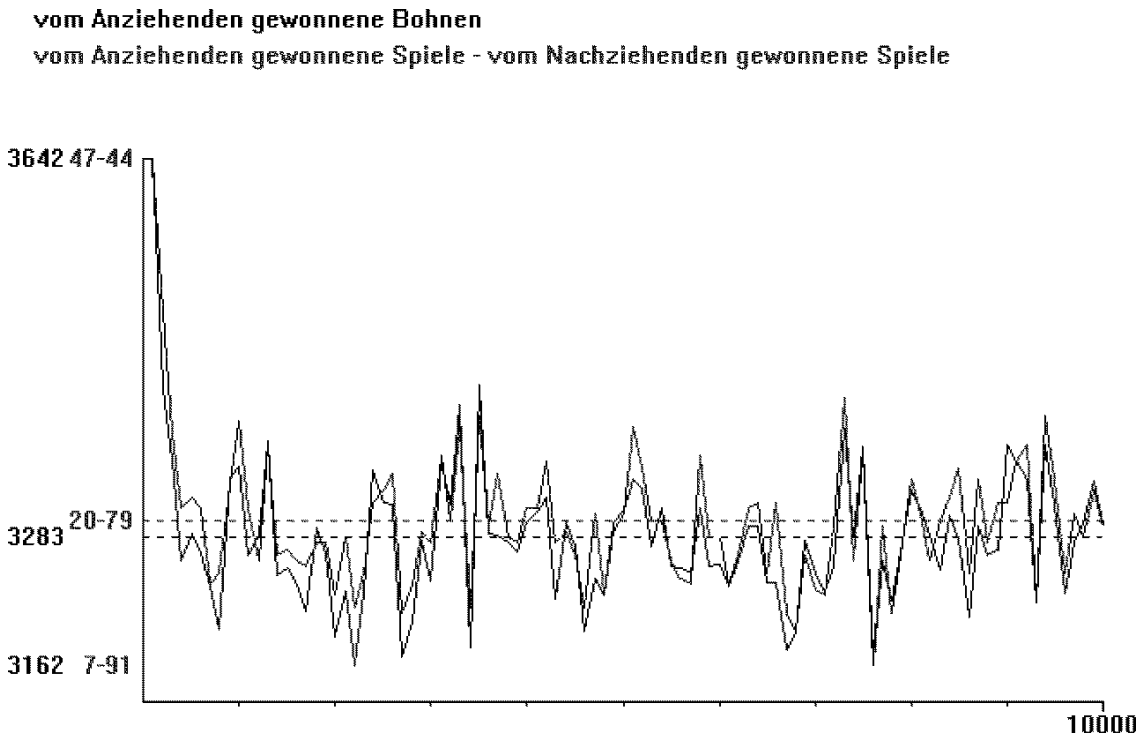

Abbildung 4.40: „unbekannteste“ Tiefe 6. Verlauf der Ergebnisse

„bekannte und unbekannte“

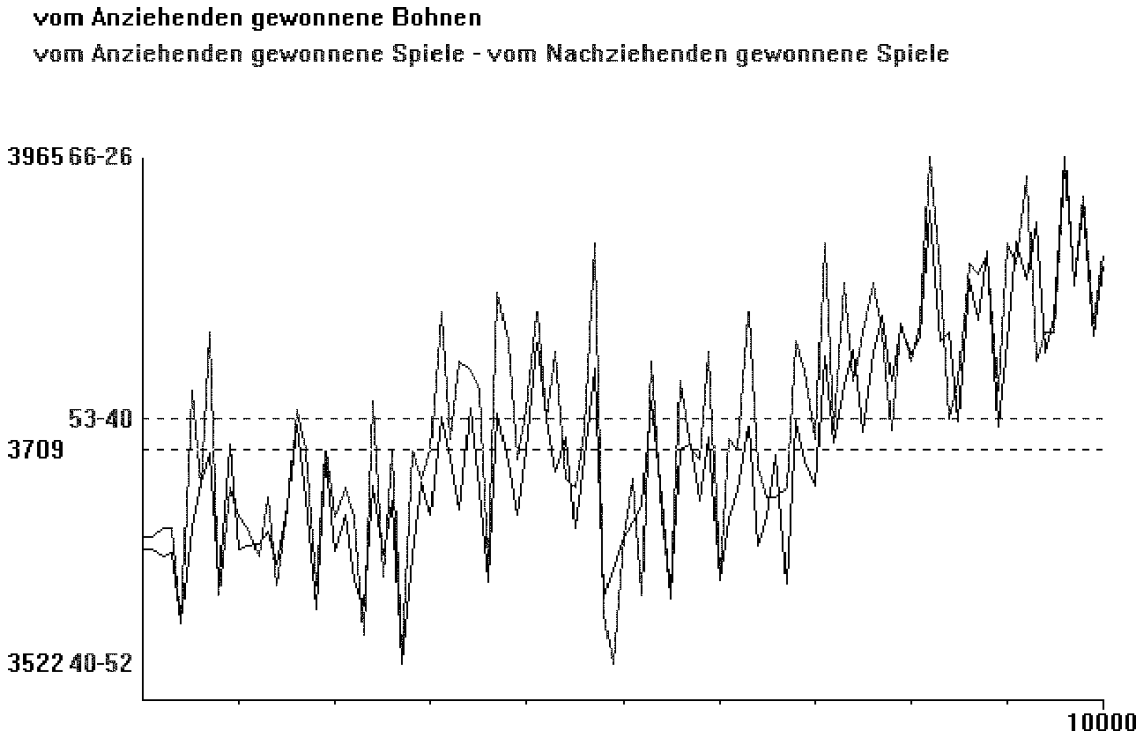

Abbildung 4.41: „bekannte und unbekannte“ Tiefe 2. Verlauf der Ergebnisse

Die Ergebnisse der Testreihen „bekannte und unbekannte“ und „bekannteste und unbekannteste“ enttäuschten die Erwartungen. Die besten Ergebnisse erzielten sie bei Suchtiefe 6.

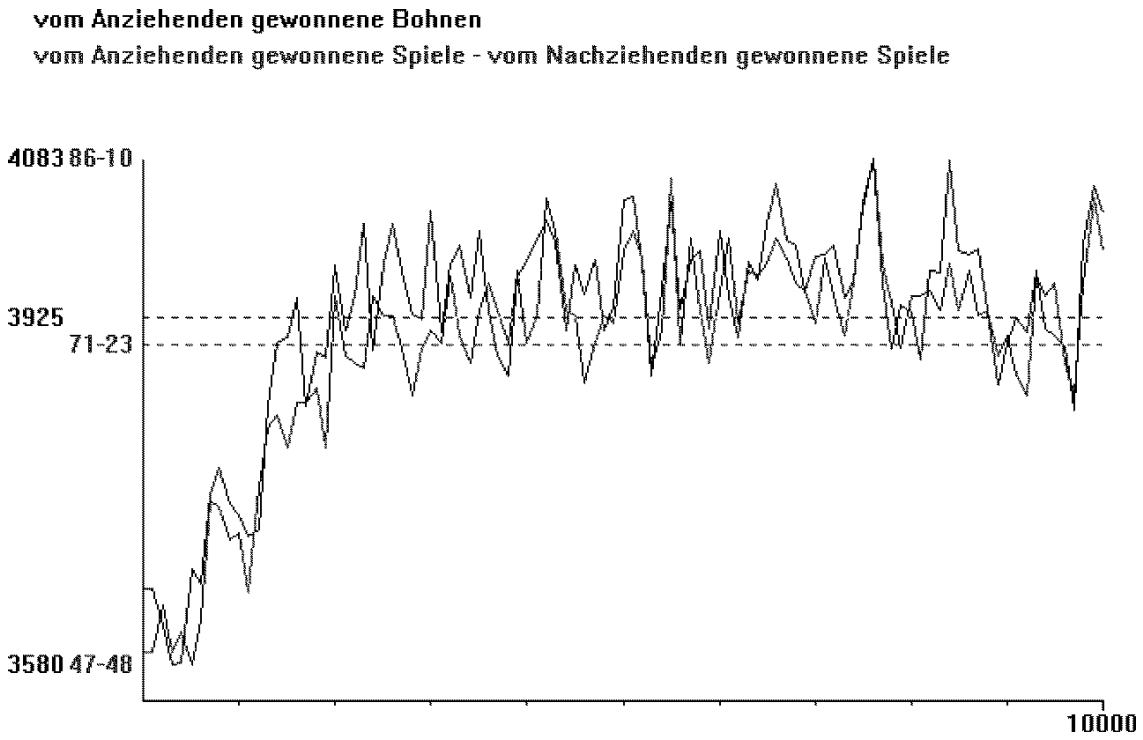

Abbildung 4.42: „bekannte und unbekannte“ Tiefe 4. Verlauf der Ergebnisse

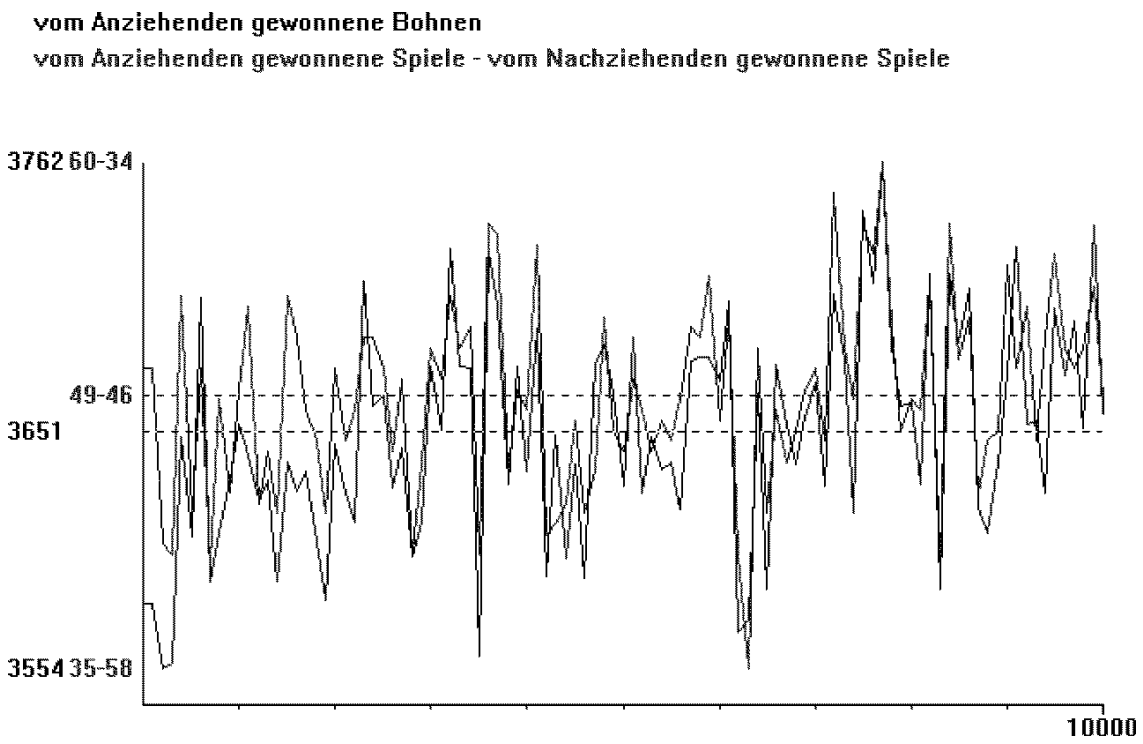

Abbildung 4.43: „bekannte und unbekannte“ Tiefe 6. Verlauf der Ergebnisse

„bekannteste und unbekannteste“

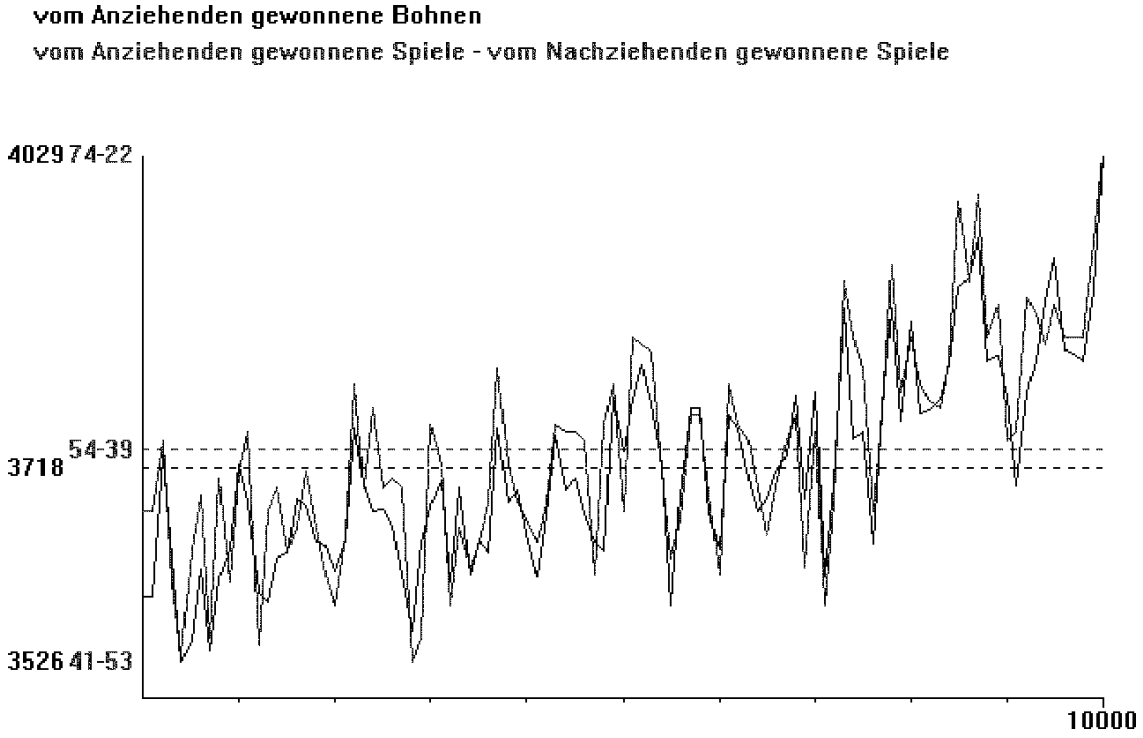

Abbildung 4.44: „bekannteste und unbekannteste“ Tiefe 2. Verlauf der Ergebnisse

Die Ergebnisse dieser Testreihe unterscheiden sich nur minimal von denen der Testreihe „bekannte und unbekannte“.

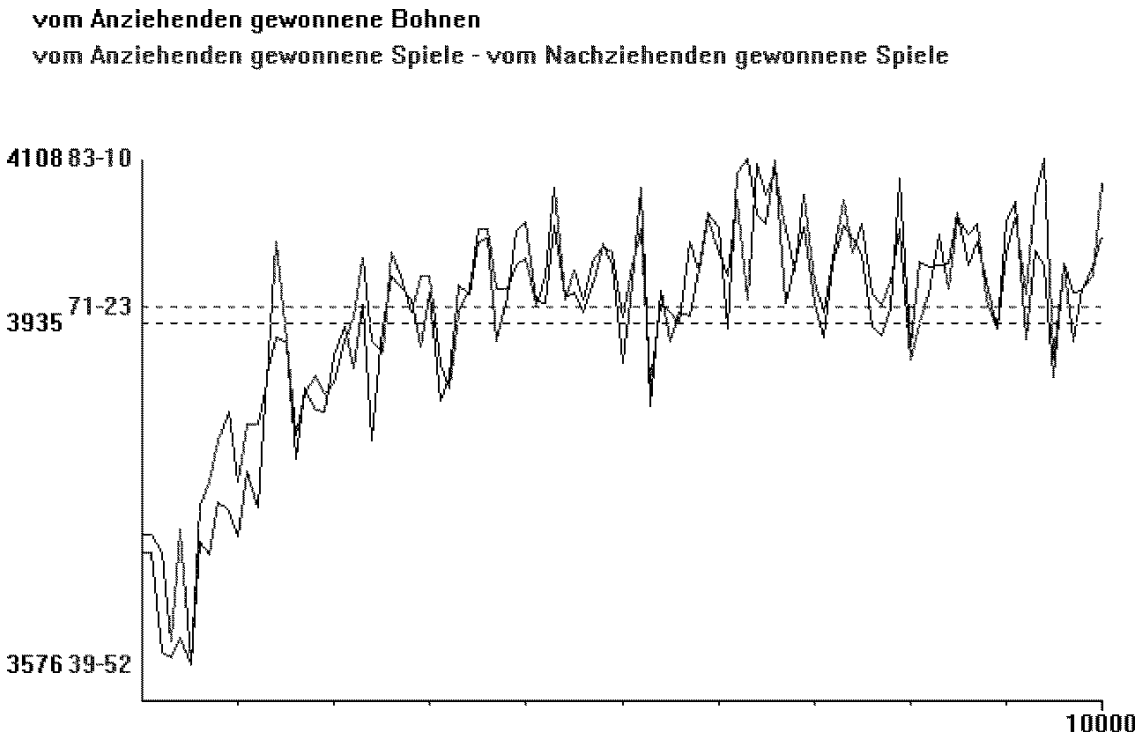

Abbildung 4.45: „bekannteste und unbekannteste“ Tiefe 4. Verlauf der Ergebnisse

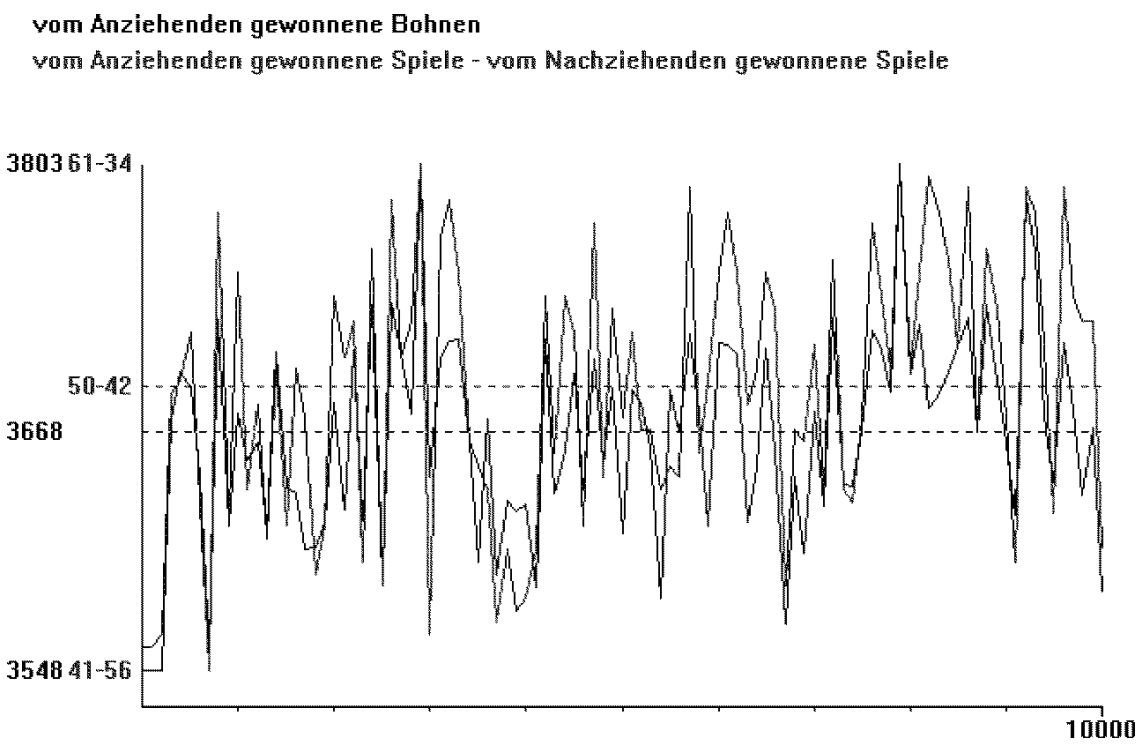

Abbildung 4.46: „bekannteste und unbekannteste“ Tiefe 6. Verlauf der Ergebnisse

4.4 Zusammenfassung der Ergebnisse

In diesem Abschnitt wird eine Zusammenfassung der im Abschnitt 4.3 aufgeführten Ergebnisse der Testreihen in Form eines Vergleichs der erreichten Resultate gegeben. Grafisch dargestellt sind für jede Testreihe die Mittelwerte der Gewinne des Anziehenden.

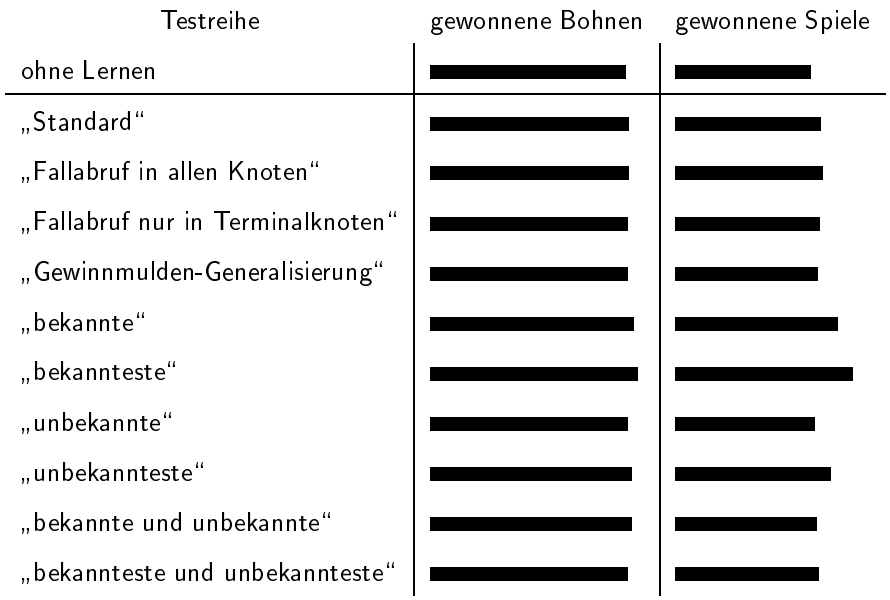

Tabelle 4.1: Vergleich der Testreihen mit Suchtiefe 2.

Testreihe	gewonnene Bohnen	gewonnene Spiele
ohne Lernen		
„Standard“		
„Fallabruf in allen Knoten“		
„Fallabruf nur in Terminalknoten“		
„Gewinnmulden-Generalisierung“		
„bekannte“		
„bekannteste“		
„unbekannte“		
„unbekannteste“		
„bekannte und unbekannte“		
„bekannteste und unbekannteste“		

Tabelle 4.2: Vergleich der Testreihen mit Suchtiefe 4.

Testreihe	gewonnene Bohnen	gewonnene Spiele
ohne Lernen		
„Standard“		
„Fallabruf in allen Knoten“		
„Fallabruf nur in Terminalknoten“		
„Gewinnmulden-Generalisierung“		
„bekannte“		
„bekannteste“		
„unbekannte“		
„unbekannteste“		
„bekannte und unbekannte“		
„bekannteste und unbekannteste“		

Tabelle 4.3: Vergleich der Testreihen mit Suchtiefe 6.

Zusammenfassend ist hervorzuheben, daß die unterschiedlichen Ergebnisse der Testreihen fast vollständig durch schnelles bzw. langsames Lernen der einzelnen Algorithmen entstehen.

Um die Leistungsfähigkeit des fallbasierten Algorithmus auf eine andere Weise darlegen zu können, wurden weitere Testreihen durchgeführt. Bei ihnen sind die Suchtiefen der beiden Gegner nicht gleich groß, weshalb die Vergleichbarkeit zu Testreihen ohne Lernen nicht ohne weiteres gegeben ist.

Dennoch illustrieren diese Testreihen außerordentlich deutlich, wie sehr die Spielstärke des Lernenden wächst. In den Testreihen spielte der fallbasierte Algorithmus – einmal mit den Standard-Einstellungen und einmal mit der Auswahlstrategie des Bevorzugens der bekanntesten Stellungen – gegen Minimax-Suche mit Tiefe 4 und Tiefe 6. Die Ergebnisverläufe dieser vier Testreihen sind in den Abbildungen 4.47 bis 4.50 dargestellt. Obwohl der fallbasierte Algorithmus bei der Baumsuche im Vergleich zu seinem Gegner nur einen Bruchteil der Knoten durchsucht, erreicht er dessen Spielstärke – bzw. bei Suchtiefe 6 die Spielstärke eines gleichwertigen Gegners mit Suchtiefe 6. Ebenfalls deutlich wird der Vorteil der Auswahlstrategie.

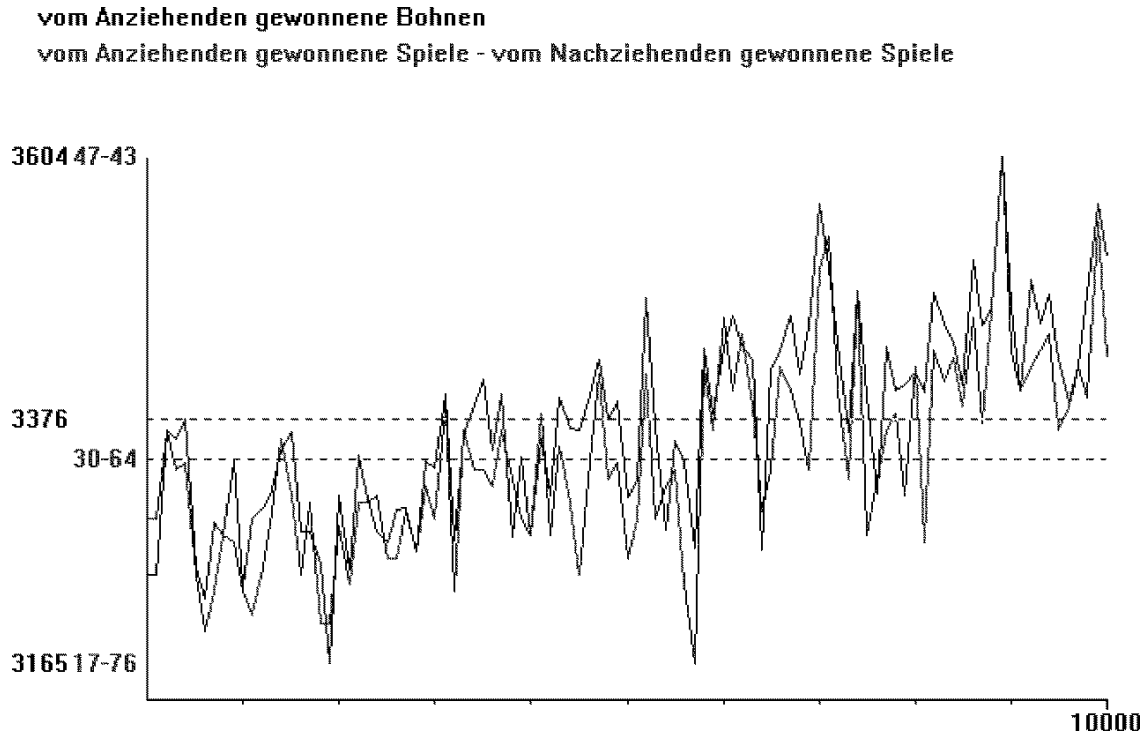

Abbildung 4.47: Fallbasiert Tiefe 2 gegen Minimax Tiefe 4

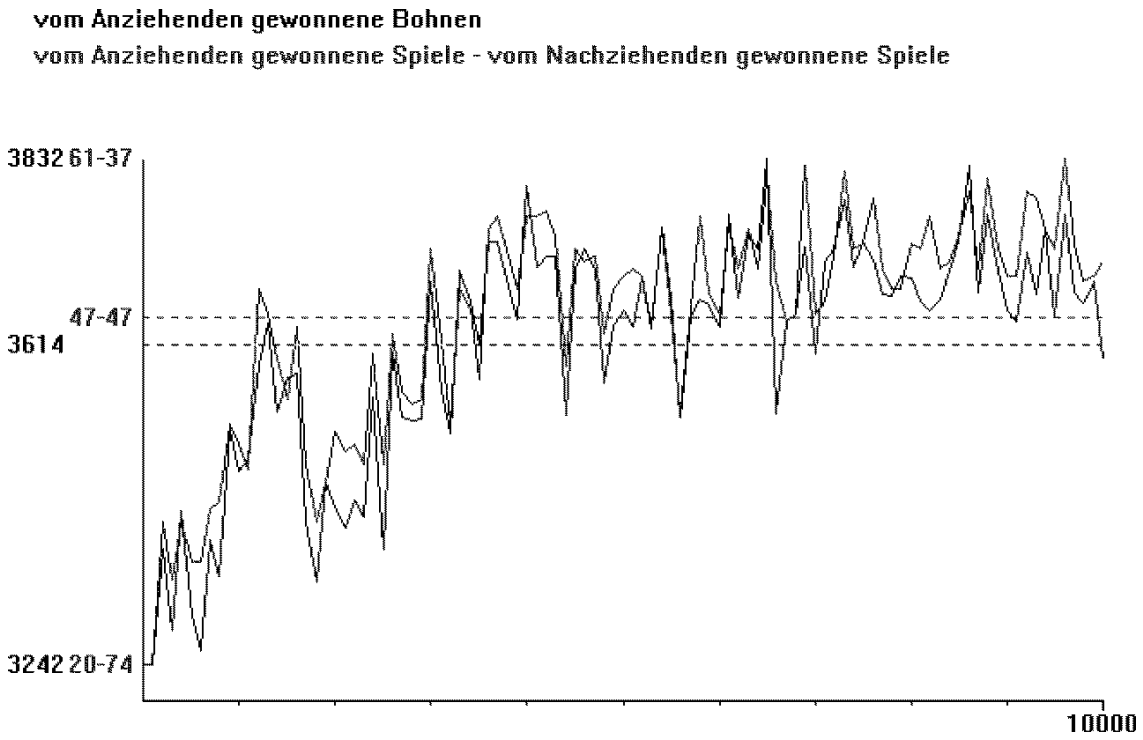

Abbildung 4.48: Fallbasiert bekannteste Tiefe 2 gegen Minimax Tiefe 4

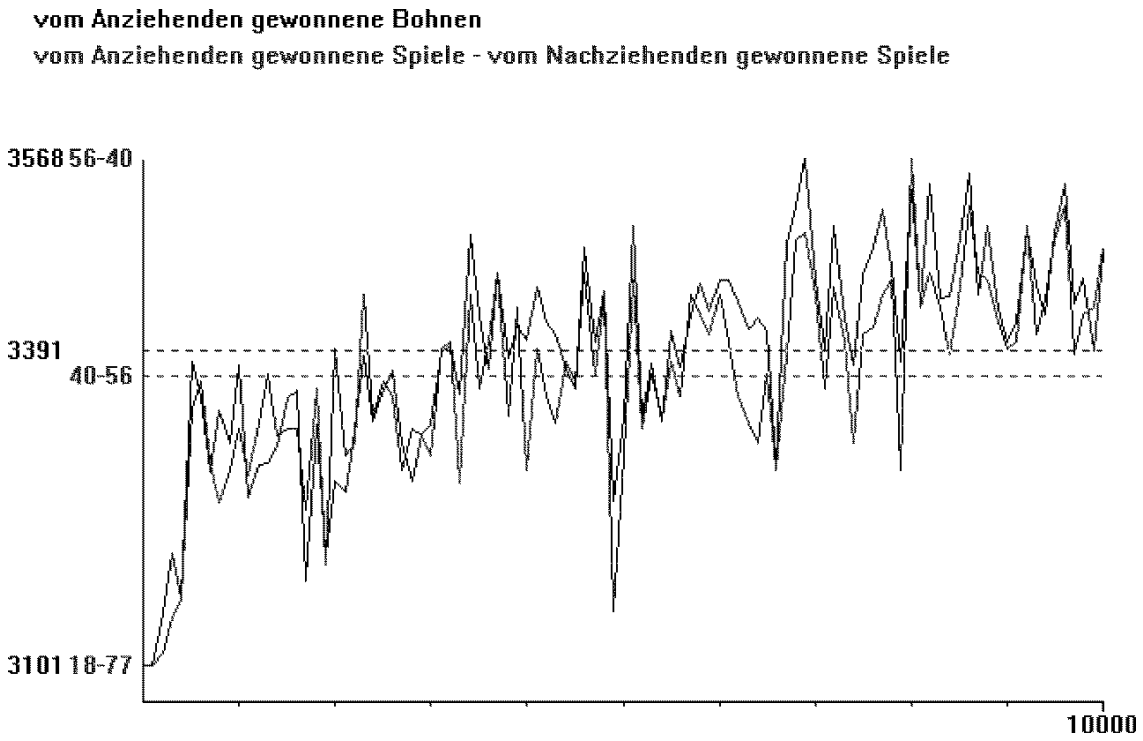

Abbildung 4.49: Fallbasiert Tiefe 2 gegen Minimax Tiefe 6

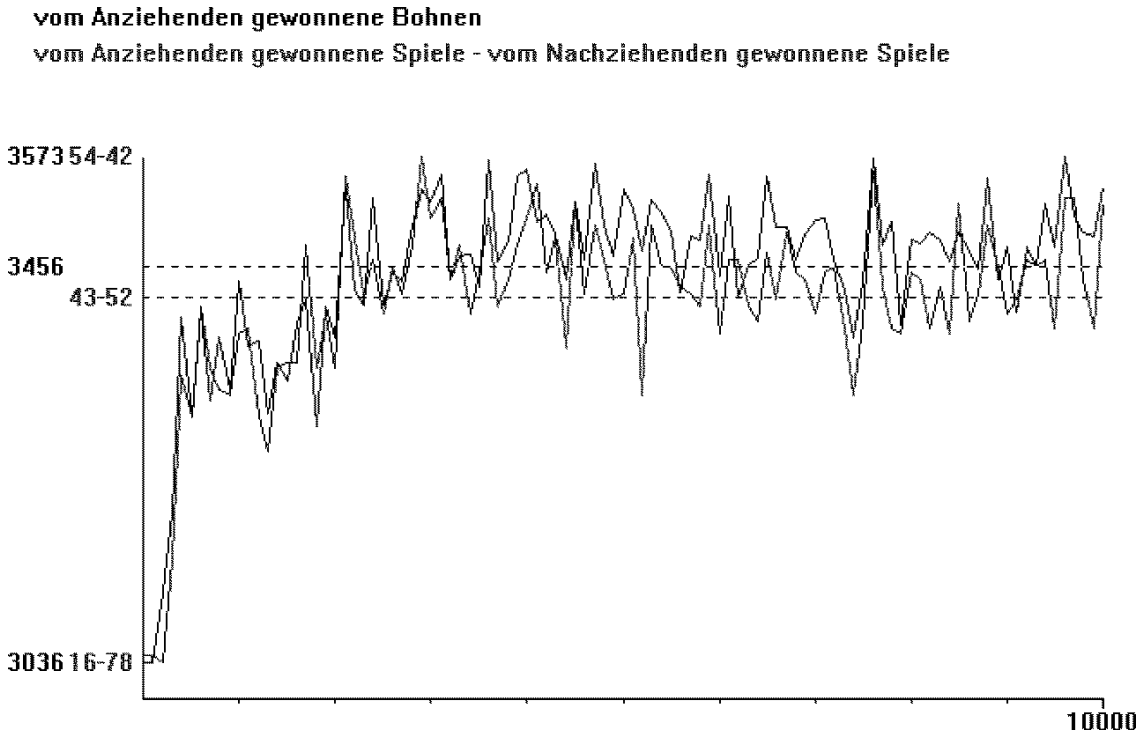

Abbildung 4.50: Fallbasiert bekannteste Tiefe 2 gegen Minimax Tiefe 6

Kapitel 5

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde das Konzept des fallbasierten Schließens für die Verwendung während der Baumsuche bei Spielen vorgeschlagen und ein entsprechender Algorithmus für das Anwendungsbeispiel Kalah implementiert. Dieser Algorithmus sollte in der Lage sein, durch selbständige Aufnahme von Fällen in seine Fallbibliothek und Verändern der in den Fällen enthaltenen Informationen die ursprünglich durch den tiefenbeschränkten Minimax-Algorithmus definierte Bewertungsfunktion zu verbessern.

Zielstellung der Arbeit war zu untersuchen, welchen Einfluß verschiedene Variationen des fallbasierten Algorithmus auf das Lernverhalten haben. Hierzu wurden mehrere Parameter des Algorithmus identifiziert. Zur Steuerung der Baumsuche wurden mehrere Auswahlstrategien entwickelt. Außer der Bewertung der Halbzugalternativen verwenden diese Strategien Fälle bei der Halbzugwahl, um in den Konfliktsituationen, in denen mehrere am besten bewertete Folgepositionen existieren, nicht nur zufällig unter diesen auswählen zu können.

Für die Untersuchung und zum Nachweis ihrer Leistungsfähigkeit wurden die Variationen und verschiedene Auswahlstrategien implementiert und ausführliche Experimente durchgeführt. Bei der großen Anzahl von Spielen war eine Beschränkung der Suche im Spielbaum notwendig. Zusätzlich zu den bekannten Konzepten der Tiefenbeschränkung und des α-β-Stutzens mit tiefen Schnitten wurde in diesem Zusammenhang das Prinzip des abhängigen Bewertens der alternativen Folgepositionen entwickelt und seine Vor- und Nachteile beim α-β-Stutzen dargelegt. Experimentell wurde nachgewiesen, daß es eine wesentliche Verkürzung der Baumsuche ermöglicht. Zu einer weiteren Verbesserung des Baumstutzens könnte bei Spielen wie Kalah, bei denen verschiedene Teilbäume des Spielbaums sehr unterschiedliche Breiten aufweisen, das iterative

Absteigen modifiziert werden. Dazu müßten zur Festlegung der Reihenfolge, in der die jeweiligen Folgepositionen untersucht werden, nicht nur ihre Bewertungen aus der vorhergehenden Suche, sondern auch die Anzahl der zu durchsuchenden Knoten im entsprechenden Teilbaum betrachtet werden. Die Umsetzung dieser Modifizierung läßt eine erneute Verkürzung der Baumsuchen erwarten.

Zur praktischen Umsetzung der beschriebenen Konzepte und zur Durchführung und Auswertung der Experimente wurde ein Testprogramm erstellt. Es ist mit entsprechenden Ergänzungen für weiterführende Untersuchungen geeignet. Nach dafür nötigen Anpassungen ist es als Werkzeug zur Verwaltung und Auswertung auch für andere Anwendungen nutzbar.

Im Verlauf der Experimente konnte die Leistungsfähigkeit der einzelnen Auswahlstrategien nachgewiesen werden. Dabei fielen insbesondere die erfolgreichen Strategien „bekannte“ und „bekannteste“ auf, mit denen das Verfahren sehr schnell lernte. Mit Hilfe der Gewinnmulden-Generalisierung konnten vergleichsweise viele Fallabrufe erzielt werden, was aber nicht zu besseren Ergebnissen führte.

Die betrachteten Ähnlichkeitsrelationen zur Bestimmung anwendbarer Fälle beziehen keine qualitativen Merkmale der Spielstellungen ein. Da zu Kalah in dieser Arbeit keine Theorie verfügbar war, wäre für die Erstellung einer qualitative Merkmale unterscheidenden Ähnlichkeitsrelation eine Analyse des Spiels notwendig gewesen. Das Hauptaugenmerk lag jedoch bei dieser Arbeit entsprechend ihrer Zielstellung auf der Verwendung des fallbasierten Schließens während der Baumsuche. Außerdem hätte sowohl eine manuelle Stellungsanalyse als auch die Implementierung und Anwendung einer automatischen Methode zum Finden und Gewichten der bestimmenden strategischen Merkmale einer Stellung den Rahmen dieser Diplomarbeit gesprengt. Für dieses Problem bieten sich die vielversprechendsten, aber sicher auch am schwierigsten umzusetzenden, Verbesserungsmöglichkeiten des vorgestellten Systems.

Im Zusammenhang mit der Anwendung von qualitativ unterscheidenden Ähnlichkeitsrelationen wird im Gegensatz zu der implementierten Fassung des Algorithmus der zeitsparende Fallabruf mit Hilfe von binärer Suche in der vorgestellten Form nicht möglich sein. Vielmehr ist dafür eine gemäß der Hierarchie der Stellungsmerkmale strukturierte Fallbasis erfolgversprechend.

Für die Untersuchung der Parametereinflüsse war es notwendig, den Algorithmus vorsichtig zu verändern, um Ergebnisänderungen tatsächlich auf eine Ursache eingrenzen zu können. Daher wurde der Minimax-Algorithmus und sein zugrundeliegendes

Prinzip, stets unter den am besten bewerteten Folgepositionen zu wählen, in dieser Arbeit nicht infragegestellt. Während der Minimax-Algorithmus zu perfekten Entscheidungen führt, wenn die Folgepositionen fehlerfrei bewertet worden sind, gibt es andere Algorithmen, die bei fehlerbehafteter Bewertung – wovon in den meisten Fällen ausgegangen werden muß – bessere Resultate erzielen. Insbesondere wären Auswahlstrategien denkbar, die auch schlechter bewertete Folgepositionen wählen, z. B. mit einer von der Differenz zur besten Bewertung abhängigen Wahrscheinlichkeit. Neben einer direkten Änderung des Spielverhaltens ermöglicht dieser Ansatz vermutlich eine höhere Leistungsfähigkeit des fallbasierten Lernens, weil auf diese Weise auch ursprünglich schlechter bewertete Folgepositionen gewählt und in folgenden Spielen fallbasiert bewertet werden können.

Eine grundsätzliche Entscheidung bei der Implementation des Verfahrens war, die Phasen für das Lernen und die Benutzung von Fällen nicht zu trennen. Dafür gab es zwei Hauptgründe: Zum einen ändert sich das Spielverhalten durch das Lernen, wodurch ein Teil der gelernten Fälle bei einer feinen Ähnlichkeitsrelation nicht mehr anwendbar wäre und nur wenige Fälle für die neuen Spielsituationen gelernt worden wären. Zum anderen kommt der Effekt des fallbasierten Lernens von Stellungsbewertungen gerade durch häufiges und abwechselndes Verwenden und Verändern der Fallinformationen zustande. Bei der Arbeit mit einer anderen Ähnlichkeitsrelation sollte diese Entscheidung überdacht werden, weil der zuerst genannte Grund dann möglicherweise nur noch eingeschränkt zutrifft.

Es erscheint sinnvoll, in die Stellungsbewertung oder als Kriterien für Auswahlstrategien bisher nicht verwendete Informationen einzubeziehen. Dazu könnten die tatsächlichen Spielausgänge, die Unterscheidung von perfekten und nicht perfekten Bewertungen und das gegnerische Spielverhalten betrachtet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[All94] L[ouis] Victor Allis: *Searching for Solutions in Games and Artificial Intelligence*. Dissertation (Rijksuniversiteit Limburg te Maastricht), Maastricht, 1994

[Dew88] A[lexander] K[eewatin] Dewdney: *Meisterliche Dameprogramme*. Spektrum der Wissenschaft, Sonderheft Computer-Kurzweil II, S. 19-24 (1988)

[Gin93] Matt[hew L.] Ginsberg: *Essentials of Artificial Intelligence*. Morgan Kaufmann Publishers, San Francisco, 1993

[Hut94] Alan Hutchinson: *Algorithmic Learning*. Clarendon Press, Oxford, 1994

[Ker95] Yaakov Kerner: *Case-Based Evaluation in Computer Chess*. In *Advances in Case-Based Reasoning*. Second European Workshop, EWCBR-94, Chantilly, France, November 1994, Selected Papers, Lecture Notes in Artificial Intelligence Vol. 984, Springer-Verlag, Berlin Heidelberg, 1995

[Kie90] Anders Kierulf: *Smart Game Board: a Workbench for Game-Playing Programs, with Go and Othello as Case Studies*. Dissertation (Diss. ETH Nr. 9135), Zürich, 1990

[Kle77] Eberhard Klein: *Heuristische Auswahlmethoden und Lernansätze bei Computerspielen*. Dissertation (Universität Bonn), Bonn, 1977

[Kol93] Janet [L.] Kolodner: *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, California, 1993

[Kop94] Helmut Kopka: LaTeX – *Einführung Band 1*. Addison-Wesley (Deutschland), 1994

[Law92] Jeanette Lawrence: *Neuronale Netze – Computersimulation biologischer Intelligenz*. Systhema Verlag, München, 1992

[LBBW98] Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, Stefan Wess (Eds.): *Case-Based Reasoning Technology – From Foundations to Applications*. Lecture Notes in Artificial Intelligence Vol. 1400, Springer-Verlag, Berlin Heidelberg, 1998

[LS93] George F. Luger, William A. Stubblefield: *Artificial Intelligence – Structures and Strategies for Complex Problem Solving*. The Benjamin/Cummings Publishing Company, 1993

[Mac84] Heinz Machatschek: *Stein um Stein – Exotik der Brettspiele*. Verlag Neues Leben, Berlin, 1984

[MCM83] Ryszard S[tanislaw] Michalski, Jaime G[uillermo] Carbonell, Tom M[ichael] Mitchell (Eds.): *Machine Learning – An Artificial Intelligence Approach*. Morgan Kaufmann Publishers, 1983

[NS96] Günter Nägler, Friedmar Stopp: *Graphen und Anwendungen*. Reihe „Mathematik für Ingenieure und Naturwissenschaftler“, B. G. Teubner Verlagsgesellschaft, Stuttgart, Leipzig, 1996

[OW93] T[homas] Ottmann, P[eter] Widmayer: *Algorithmen und Datenstrukturen*. BI-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 2. Auflage 1993

[RS92] Fritz Reinhardt, Heinrich Soeder: *dtv-Atlas zur Mathematik, Band 2 – Analysis und angewandte Mathematik*. Deutscher Taschenbuch Verlag, München, 8. Auflage 1992

[RN95] Stuart J. Russell, Peter Norvig: *Artificial Intelligence – A Modern Approach*. Prentice-Hall, 1995

[Sam59] A[rthur] L. Samuel: *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal of Research and Development, 3(3):211-229 (1959)

[Sam67] A[rthur] L. Samuel: *Some Studies in Machine Learning Using the Game of Checkers, II—Recent Progress*. IBM Journal of Research and Development, 11(6):601-617 (1967)

[Sed92] Robert Sedgewick: *Algorithmen*. Addison-Wesley (Deutschland), 1992

[SHHTY89] Mike Sharples, David Hogg, Chris Hutchison, Steve Torrance, David Young: *Computers and Thought – A Practical Introduction to Artificial Intelligence*. MIT Press, London, 1989

[SD69] James R. Slagle, John K. Dixon: *Experiments with some Programs that search Game Trees*. Journal of the Association for Computing Machinery, 16(2):189-207 (1969)

[SV93] Helmut Steuer, Claus Voigt: *Das rororo-Spielbuch*. Rowohlt Taschenbuch Verlag, Reinbek bei Hamburg, 1993

[Thi88] Rüdiger Thiele: *Die gefesselte Zeit: Spiele, Spaß und Strategien*. Urania-Verlag, Leipzig Jena Berlin, 1988

[VA95] Manuela Veloso, Agnar Aamodt (Eds.): *Case-Based Reasoning Research and Development*. First International Conference, ICCBR-95, Sesimbra, Portugal, October 1995, Proceedings, Lecture Notes in Artificial Intelligence Vol. 1010, Springer-Verlag, Berlin Heidelberg, 1995

[Wat95] Ian D. Watson: *An Introduction to Case-Based Reasoning*. In Ian D. Watson (Ed.): *Progress in Case-Based Reasoning*. First United Kingdom Workshop, Salford, UK, January 1995, Proceedings, Lecture Notes in Artificial Intelligence Vol. 1020, Springer-Verlag, Berlin Heidelberg, 1995

[Why98] Ken Whyld: *Schach lernen*. Delius Klasing Verlag, Bielefeld, 1998

Anhang A

Die Spielregeln von Kalah

Kalah wird von zwei Spielern auf einem Brett mit zwei sich gegenüberliegenden Reihen von 6 Mulden und zwei Gewinnmulden (*Kalahs*) gespielt. Die Kalah des Spielers befindet sich rechts neben seinen 6 Mulden. Die Spielsteine heißen *Bohnen*. Zu Beginn liegen in jeder Mulde 6 Bohnen. Die Kalahs sind leer.

0	6	6	6	6	6	6	0
	6	6	6	6	6	6	

Das Ziel von Kalah ist, so viele Bohnen wie möglich in die eigene Kalah zu bekommen. Gewonnen hat der Spieler, der bei Spielende mehr als die Hälfte der Bohnen, also mindestens 37, in seiner Kalah hat.

Ein Spieler zieht, indem er aus einer seiner 6 Mulden alle Bohnen entnimmt und sie einzeln linksherum (entgegen des Uhrzeigersinns) verteilt, wobei er nur die gegnerische Kalah ausläßt.

Ein Teilzug kann folgende Fortsetzungen erfahren:

Bonuszug: Fällt die letzte zu verteilende Bohne in die eigene Kalah, so ist der Spieler erneut am Zug. Er muß noch einmal ziehen, auch wenn es für ihn nicht von Vorteil ist.

Gewinn: Fällt die letzte zu verteilende Bohne in eine eigene leere Mulde und ist die gegenüberliegende (gegnerische) Mulde gefüllt, gibt der Spieler die zuletzt abgelegte und die ihr gegenüberliegenden Bohnen in der gegnerischen Mulde in seine Kalah. Er hat alle diese Bohnen gewonnen. Danach ist der Gegner am Zug.

Spielerwechsel: Tritt keine der oben beschriebenen Situationen ein, dann ist der Gegner sofort am Zug.

Das Spiel ist beendet, wenn alle 6 Mulden eines Spielers leer sind (unabhängig davon, welcher Spieler am Zug ist). Der Gegenspieler erhält dann die in seinen Mulden verbliebenen Bohnen in seine Kalah.

Anhang B

Experimentell ermittelte Daten

B.1 Statistische Daten zu Kalah

	Suchtiefe t in Halbzügen					
	1	2	3	4	5	6
Anzahl der Spiele	10 000	10 000	1 000	1 000	1 000	100
Spiellänge in Halbzügen	31,3	47,7	38,7	42,0	40,7	37,8
Terminalknoten k je Suche	10,0	79,2	1 171,3	8 616,2	149 340,5	894 983,2
Breite $b \approx \sqrt[t]{k}$ des Spielbaums	10,0	8,9	10,5	9,6	10,8	9,8

Tabelle B.1: Spiellänge und Verzweigungsbreite des Spielbaums für verschiedene Suchtiefen

Suchtiefe in Halbzügen	1	2	3	4	5	6
Anzahl der Spiele	10 000	10 000	1000	1000	1000	100
Teilzüge						
1	50 981	103 927	7 646	8 878	8 303	834
2	58 362	95 214	7 995	8 624	8 626	692
3	71 327	99 460	8 815	9 162	8 877	797
4	93 419	111 902	10 023	10 019	10 730	952
5	131 729	140 066	13 250	11 916	13 973	1 168
6	197 268	206 979	21 936	18 281	20 179	1 716

Tabelle B.2: Häufigkeit der verschiedenen Teilzüge

	Suchtiefe in Halbzügen					
	1	2	3	4	5	6
kumulierte Bewertungen	1 272	-2 737	647	-3 190	-787	-4 216
durchschnittliche Bewertungen	0,7	-1,5	0,35	-1,74	-0,43	-2,31

Tabelle B.3: Stellungsbewertungen bei Suchen mit unterschiedlichen Tiefen. Es wurden einhundert Spiele mit Minimax-Algorithmus mit einer Suchtiefe von 6 Halbzügen als Anziehender und Nachziehender durchgeführt. Dabei wurden alle 1 829 tatsächlich erreichten Positionen des Anziehenden durch Suchen mit den Tiefen von 1 bis 6 Halbzügen bewertet.

B.2 Statistische Daten zu den Testreihen

B.2.1 Vergleichstestreihen ohne Lernen

Ergebnisse	
gewonnene Bohnen	367 287:352 713
Mittelwert	36,73
Varianz	32,54
Streuung	5,70
gewonnene Spiele	5 103:4 167

Tabelle B.4: Tiefe 2 ohne Lernen

Ergebnisse	
gewonnene Bohnen	36 708:35 292
Mittelwert	36,71
Varianz	38,11
Streuung	6,17
gewonnene Spiele	522:414

Tabelle B.5: Tiefe 4 ohne Lernen

Ergebnisse	
gewonnene Bohnen	34 633:37 367
Mittelwert	34,63
Varianz	34,93
Streuung	5,91
gewonnene Spiele	403:539

Tabelle B.6: Tiefe 6 ohne Lernen

B.2.2 Die „Standard“-Testreihen

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	373 154:346 846	insgesamt	251 679	Anzahl der Fälle	135 758
Mittelwert	37,32	davon in Tiefe 0	780	Mittelwert Knoten	66,23
Varianz	34,4	davon in Tiefe 1	0	Streuung Knoten	155,99
Streuung	5,83	davon in Tiefe 2	250 899	Mittelwert Abrufe	1,84
gew. Spiele	5 459:3 824	davon in Tiefe 3	0	Streuung Abrufe	96,70
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.7: Standard-Testreihe Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	397 797:322 203	insgesamt	978 320	Anzahl der Fälle	58 850
Mittelwert	39,78	davon in Tiefe 0	3 843	Mittelwert Knoten	2 135,82
Varianz	34,69	davon in Tiefe 1	0	Streuung Knoten	8 997,7
Streuung	5,89	davon in Tiefe 2	368 052	Mittelwert Abrufe	16,52
gew. Spiele	7 301:2 137	davon in Tiefe 3	0	Streuung Abrufe	328,42
		davon in Tiefe 4	606 425		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.8: Standard-Testreihe Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	358 078:361 922	insgesamt	1 691 481	Anzahl der Fälle	8 972
Mittelwert	35,81	davon in Tiefe 0	5 444	Mittelwert Knoten	17 752,77
Varianz	13,66	davon in Tiefe 1	0	Streuung Knoten	118 857,18
Streuung	3,70	davon in Tiefe 2	409 501	Mittelwert Abrufe	187,71
gew. Spiele	4 684:4 931	davon in Tiefe 3	0	Streuung Abrufe	974,14
		davon in Tiefe 4	259 753		
		davon in Tiefe 5	0		
		davon in Tiefe 6	1 016 783		

Tabelle B.9: Standard-Testreihe Tiefe 6

B.2.3 Fallabruf während der Baumsuche

Fallabruf in allen Knoten

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	373 772:346 228	insgesamt	279 413	Anzahl der Fälle	132 090
Mittelwert	37,38	davon in Tiefe 0	1 265	Mittelwert Knoten	89,9
Varianz	35,48	davon in Tiefe 1	4 725	Streuung Knoten	226,87
Streuung	5,96	davon in Tiefe 2	273 423	Mittelwert Abrufe	2,10
gew. Spiele	5 535:3 812	davon in Tiefe 3	0	Streuung Abrufe	101,82
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.10: Fallabruf in allen Knoten Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	397 430:322 570	insgesamt	942 010	Anzahl der Fälle	60 838
Mittelwert	39,74	davon in Tiefe 0	3 828	Mittelwert Knoten	3 013,28
Varianz	36,79	davon in Tiefe 1	1 879	Streuung Knoten	11 862,14
Streuung	6,7	davon in Tiefe 2	379 651	Mittelwert Abrufe	15,39
gew. Spiele	7 243:2 250	davon in Tiefe 3	14 527	Streuung Abrufe	311,12
		davon in Tiefe 4	542 125		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.11: Fallabruf in allen Knoten Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	365 479:354 521	insgesamt	2 396 203	Anzahl der Fälle	9 021
Mittelwert	36,55	davon in Tiefe 0	6 495	Mittelwert Knoten	23 619,32
Varianz	20,43	davon in Tiefe 1	1 838	Streuung Knoten	150 477,49
Streuung	4,52	davon in Tiefe 2	410 566	Mittelwert Abrufe	264,75
gew. Spiele	4 849:4 716	davon in Tiefe 3	12 712	Streuung Abrufe	1 243,12
		davon in Tiefe 4	286 340		
		davon in Tiefe 5	452 194		
		davon in Tiefe 6	1 226 058		

Tabelle B.12: Fallabruf in allen Knoten Tiefe 6

Fallabruf nur in Terminalknoten

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	371 790:348 210	insgesamt	249 784	Anzahl der Fälle	136 112
Mittelwert	37,18	davon in Tiefe 0	0	Mittelwert Knoten	65,59
Varianz	33,47	davon in Tiefe 1	0	Streuung Knoten	169,47
Streuung	5,79	davon in Tiefe 2	249 784	Mittelwert Abrufe	1,81
gew. Spiele	5 441:3 847	davon in Tiefe 3	0	Streuung Abrufe	93,58
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.13: Fallabruf nur in Terminalknoten Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	372 102:347 898	insgesamt	523 208	Anzahl der Fälle	71 513
Mittelwert	37,21	davon in Tiefe 0	0	Mittelwert Knoten	1 664,73
Varianz	39,58	davon in Tiefe 1	0	Streuung Knoten	5 463,62
Streuung	6,29	davon in Tiefe 2	0	Mittelwert Abrufe	7,30
gew. Spiele	5 378:3 942	davon in Tiefe 3	0	Streuung Abrufe	192,94
		davon in Tiefe 4	523 208		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.14: Fallabruf nur in Terminalknoten Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	346 920:373 080	insgesamt	1 688 280	Anzahl der Fälle	25 092
Mittelwert	34,69	davon in Tiefe 0	0	Mittelwert Knoten	18 235,26
Varianz	35,89	davon in Tiefe 1	0	Streuung Knoten	95 294,80
Streuung	5,99	davon in Tiefe 2	0	Mittelwert Abrufe	66,82
gew. Spiele	4 163:5 313	davon in Tiefe 3	0	Streuung Abrufe	341,41
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	1 688 280		

Tabelle B.15: Fallabruf nur in Terminalknoten Tiefe 6

B.2.4 Ähnlichkeitsrelation

Gewinnmulden-Generalisierung

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	371 864:348 136	insgesamt	283 321	Anzahl der Fälle	130 615
Mittelwert	37,19	davon in Tiefe 0	5 315	Mittelwert Knoten	69,66
Varianz	33,2	davon in Tiefe 1	0	Streuung Knoten	176,89
Streuung	5,75	davon in Tiefe 2	278 006	Mittelwert Abrufe	2,15
gew. Spiele	5 337:3 898	davon in Tiefe 3	0	Streuung Abrufe	96,80
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.16: Gewinnmulden-Generalisierung Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	398 330:321 670	insgesamt	1 300 061	Anzahl der Fälle	52 610
Mittelwert	39,83	davon in Tiefe 0	6 228	Mittelwert Knoten	2 416,20
Varianz	37,73	davon in Tiefe 1	0	Streuung Knoten	9 802,61
Streuung	6,14	davon in Tiefe 2	413 556	Mittelwert Abrufe	24,56
gew. Spiele	7 339:2 155	davon in Tiefe 3	0	Streuung Abrufe	329,74
		davon in Tiefe 4	880 277		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.17: Gewinnmulden-Generalisierung Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	361 598:358 402	insgesamt	10 776 397	Anzahl der Fälle	7 772
Mittelwert	36,16	davon in Tiefe 0	10 442	Mittelwert Knoten	27 858,65
Varianz	17,34	davon in Tiefe 1	0	Streuung Knoten	176 974,13
Streuung	4,16	davon in Tiefe 2	456 510	Mittelwert Abrufe	1 385,62
gew. Spiele	4 827:4 788	davon in Tiefe 3	0	Streuung Abrufe	4 615,24
		davon in Tiefe 4	1 044 489		
		davon in Tiefe 5	0		
		davon in Tiefe 6	9 264 956		

Tabelle B.18: Gewinnmulden-Generalisierung Tiefe 6

B.2.5 Auswahlstrategie

„bekannte“

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	382 988:337 012	insgesamt	278 544	Anzahl der Fälle	108 132
Mittelwert	38,30	davon in Tiefe 0	1 137	Mittelwert Knoten	65,86
Varianz	37,95	davon in Tiefe 1	0	Streuung Knoten	181,51
Streuung	6,16	davon in Tiefe 2	277 407	Mittelwert Abrufe	2,56
gew. Spiele	6 097:3 253	davon in Tiefe 3	0	Streuung Abrufe	104,57
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.19: bekannte Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	397 775:322 225	insgesamt	498 414	Anzahl der Fälle	38 760
Mittelwert	39,78	davon in Tiefe 0	4 624	Mittelwert Knoten	2 060,69
Varianz	31,57	davon in Tiefe 1	0	Streuung Knoten	9 335,95
Streuung	5,62	davon in Tiefe 2	355 462	Mittelwert Abrufe	12,80
gew. Spiele	7 539:1 974	davon in Tiefe 3	0	Streuung Abrufe	214,1
		davon in Tiefe 4	138 328		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.20: bekannte Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	368 032:351 968	insgesamt	1 512 974	Anzahl der Fälle	5 190
Mittelwert	36,80	davon in Tiefe 0	5 755	Mittelwert Knoten	18 329,50
Varianz	19,62	davon in Tiefe 1	0	Streuung Knoten	114 531,17
Streuung	4,43	davon in Tiefe 2	317 990	Mittelwert Abrufe	290,69
gew. Spiele	5 373:4 339	davon in Tiefe 3	0	Streuung Abrufe	1 309,71
		davon in Tiefe 4	249 060		
		davon in Tiefe 5	0		
		davon in Tiefe 6	940 169		

Tabelle B.21: bekannte Tiefe 6

„bekannteste“

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	391 092:328 908	insgesamt	274 091	Anzahl der Fälle	95 557
Mittelwert	39,11	davon in Tiefe 0	1 167	Mittelwert Knoten	65,2
Varianz	37,57	davon in Tiefe 1	0	Streuung Knoten	194,56
Streuung	6,13	davon in Tiefe 2	272 924	Mittelwert Abrufe	2,84
gew. Spiele	6 679:2 685	davon in Tiefe 3	0	Streuung Abrufe	102,13
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.22: bekannteste Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	396 180:323 820	insgesamt	542 868	Anzahl der Fälle	24 200
Mittelwert	39,62	davon in Tiefe 0	4 558	Mittelwert Knoten	1 701,29
Varianz	25,95	davon in Tiefe 1	0	Streuung Knoten	7 777,26
Streuung	5,9	davon in Tiefe 2	330 504	Mittelwert Abrufe	22,35
gew. Spiele	7 573:1 768	davon in Tiefe 3	0	Streuung Abrufe	283,60
		davon in Tiefe 4	207 806		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.23: bekannteste Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	356 015:363 985	insgesamt	1 190 790	Anzahl der Fälle	3 936
Mittelwert	35,60	davon in Tiefe 0	4 481	Mittelwert Knoten	16 807,89
Varianz	15,58	davon in Tiefe 1	0	Streuung Knoten	94 862,36
Streuung	3,95	davon in Tiefe 2	341 570	Mittelwert Abrufe	301,96
gew. Spiele	3 451:5 702	davon in Tiefe 3	0	Streuung Abrufe	1 320,64
		davon in Tiefe 4	147 260		
		davon in Tiefe 5	0		
		davon in Tiefe 6	697 479		

Tabelle B.24: bekannteste Tiefe 6

„unbekannte“

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	371 177:348 823	insgesamt	260 945	Anzahl der Fälle	134 530
Mittelwert	37,12	davon in Tiefe 0	1 370	Mittelwert Knoten	65,75
Varianz	34,35	davon in Tiefe 1	0	Streuung Knoten	151,81
Streuung	5,86	davon in Tiefe 2	259 575	Mittelwert Abrufe	1,92
gew. Spiele	5 258:4 008	davon in Tiefe 3	0	Streuung Abrufe	95,87
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.25: unbekannte Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	401 304:318 696	insgesamt	954 044	Anzahl der Fälle	55 123
Mittelwert	40,13	davon in Tiefe 0	4 615	Mittelwert Knoten	2 100,61
Varianz	38,86	davon in Tiefe 1	0	Streuung Knoten	8 264,47
Streuung	6,23	davon in Tiefe 2	387 319	Mittelwert Abrufe	17,22
gew. Spiele	7 332:2 215	davon in Tiefe 3	0	Streuung Abrufe	329,0
		davon in Tiefe 4	562 110		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.26: unbekannte Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	326 545:393 455	insgesamt	1 216 518	Anzahl der Fälle	6 617
Mittelwert	32,65	davon in Tiefe 0	5 234	Mittelwert Knoten	19 676,54
Varianz	19,99	davon in Tiefe 1	0	Streuung Knoten	113 576,49
Streuung	4,47	davon in Tiefe 2	442 801	Mittelwert Abrufe	183,73
gew. Spiele	1 691:8 092	davon in Tiefe 3	0	Streuung Abrufe	1 067,4
		davon in Tiefe 4	175 432		
		davon in Tiefe 5	0		
		davon in Tiefe 6	593 051		

Tabelle B.27: unbekannte Tiefe 6

„unbekannteste“

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	379 907:340 093	insgesamt	261 408	Anzahl der Fälle	112 238
Mittelwert	37,99	davon in Tiefe 0	1 749	Mittelwert Knoten	61,80
Varianz	36,42	davon in Tiefe 1	0	Streuung Knoten	141,55
Streuung	6,3	davon in Tiefe 2	259 659	Mittelwert Abrufe	2,31
gew. Spiele	5 837:3 429	davon in Tiefe 3	0	Streuung Abrufe	92,45
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.28: unbekannteste Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	401 268:318 732	insgesamt	973 611	Anzahl der Fälle	41 727
Mittelwert	40,13	davon in Tiefe 0	5 953	Mittelwert Knoten	1 998,41
Varianz	39,36	davon in Tiefe 1	0	Streuung Knoten	10 075,54
Streuung	6,27	davon in Tiefe 2	362 003	Mittelwert Abrufe	23,27
gew. Spiele	7 543:2 012	davon in Tiefe 3	0	Streuung Abrufe	364,22
		davon in Tiefe 4	605 655		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.29: unbekannteste Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	328 288:391 712	insgesamt	1 121 913	Anzahl der Fälle	6 147
Mittelwert	32,83	davon in Tiefe 0	4 438	Mittelwert Knoten	26 088,56
Varianz	22,43	davon in Tiefe 1	0	Streuung Knoten	147 291,18
Streuung	4,74	davon in Tiefe 2	410 007	Mittelwert Abrufe	182,30
gew. Spiele	1 970:7 889	davon in Tiefe 3	0	Streuung Abrufe	1 059,34
		davon in Tiefe 4	195 422		
		davon in Tiefe 5	0		
		davon in Tiefe 6	512 046		

Tabelle B.30: unbekannteste Tiefe 6

„bekannte und unbekannte“

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	370 945:349 055	insgesamt	251 041	Anzahl der Fälle	136 688
Mittelwert	37,9	davon in Tiefe 0	1 111	Mittelwert Knoten	67,15
Varianz	33,92	davon in Tiefe 1	0	Streuung Knoten	170,84
Streuung	5,82	davon in Tiefe 2	249 930	Mittelwert Abrufe	1,82
gew. Spiele	5 316:4 000	davon in Tiefe 3	0	Streuung Abrufe	93,77
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.31: bekannte und unbekannte Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	392 465:327 535	insgesamt	878 266	Anzahl der Fälle	50 080
Mittelwert	39,25	davon in Tiefe 0	4 954	Mittelwert Knoten	2 170,83
Varianz	30,65	davon in Tiefe 1	0	Streuung Knoten	8 619,89
Streuung	5,54	davon in Tiefe 2	373 055	Mittelwert Abrufe	17,44
gew. Spiele	7 076:2 298	davon in Tiefe 3	0	Streuung Abrufe	322,55
		davon in Tiefe 4	500 257		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.32: bekannte und unbekannte Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	365 111:354 889	insgesamt	1 789 777	Anzahl der Fälle	7 765
Mittelwert	36,51	davon in Tiefe 0	6 684	Mittelwert Knoten	22 160,82
Varianz	19,60	davon in Tiefe 1	0	Streuung Knoten	137 810,37
Streuung	4,43	davon in Tiefe 2	402 299	Mittelwert Abrufe	229,79
gew. Spiele	4 918:4 575	davon in Tiefe 3	0	Streuung Abrufe	1 124,20
		davon in Tiefe 4	274 860		
		davon in Tiefe 5	0		
		davon in Tiefe 6	1 105 934		

Tabelle B.33: bekannte und unbekannte Tiefe 6

„bekannteste und unbekannteste“

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	371 830:348 170	insgesamt	263 419	Anzahl der Fälle	134 917
Mittelwert	37,18	davon in Tiefe 0	1 195	Mittelwert Knoten	66,69
Varianz	34,40	davon in Tiefe 1	0	Streuung Knoten	180,20
Streuung	5,87	davon in Tiefe 2	262 224	Mittelwert Abrufe	1,93
gew. Spiele	5 401:3 928	davon in Tiefe 3	0	Streuung Abrufe	99,88
		davon in Tiefe 4	0		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.34: bekannteste und unbekannteste Tiefe 2

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	393 470:326 530	insgesamt	910 667	Anzahl der Fälle	53 762
Mittelwert	39,35	davon in Tiefe 0	3 592	Mittelwert Knoten	2 210,25
Varianz	34,73	davon in Tiefe 1	0	Streuung Knoten	9 050,78
Streuung	5,89	davon in Tiefe 2	387 089	Mittelwert Abrufe	16,82
gew. Spiele	7 064:2 292	davon in Tiefe 3	0	Streuung Abrufe	317,22
		davon in Tiefe 4	519 986		
		davon in Tiefe 5	0		
		davon in Tiefe 6	0		

Tabelle B.35: bekannteste und unbekannteste Tiefe 4

Ergebnisse		Fallabrufe		Fallbasis	
gew. Bohnen	366 798:353 202	insgesamt	1 535 410	Anzahl der Fälle	5 634
Mittelwert	36,68	davon in Tiefe 0	6 700	Mittelwert Knoten	20 662,83
Varianz	18,14	davon in Tiefe 1	0	Streuung Knoten	115 396,51
Streuung	4,26	davon in Tiefe 2	405 456	Mittelwert Abrufe	272,40
gew. Spiele	5 022:4 170	davon in Tiefe 3	0	Streuung Abrufe	1 277,33
		davon in Tiefe 4	254 336		
		davon in Tiefe 5	0		
		davon in Tiefe 6	868 918		

Tabelle B.36: bekannteste und unbekannteste Tiefe 6

Anhang C

Hinweise zur beigefügten Diskette

Die Diskette enthält das Testprogramm als Pascal-Quelltext und als lauffähiges Programm. Die Dateien sind im einzelnen:

`testbett.exe`	lauffähiges Programm
`testbett.pas`	Quelltext der Hauptdatei
`testunit.pas`	Unit-Quelltext
`gameplay.pas`	Unit-Quelltext
`winstd.pas`	Unit-Quelltext
`minimax.inc`	Quelltext für den Minimax-Algorithmus
`pruning.inc`	Quelltext für den Minimax-Algorithmus mit α-β-Stutzen
`pruning2.inc`	Quelltext für den Minimax-Algorithmus mit iterativem Absteigen
`fallbas.inc`	Quelltext für den fallbasierten Algorithmus
`zufall.inc`	Quelltext für zufällige Zugwahl
`testbett.res`	Ressourcen-Datei

Die Daten zu den vorgestellten Testreihen mitzuliefern war aufgrund deren Größe vom mehr als 300 MB nicht möglich.

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, den 29. Oktober 1998

.....

Matthias Neumeister