Automatisierte Generierung einer Bewertungsfunktion für Schachendspiele

Matthias Lüscher

Betreuer:
Thomas Lincke und Christoph Wirth

ETH Zürich
Februar 2000

Ergänzungen und Korrekturen: März 2000

Abstract. One not yet satisfactory solved problem of the evaluation function construction for chess is the efficient selection of features and the assignment of weights. Michael Buro presented in his paper "From Simple Features to Sophisticated Evaluation Functions" a practical framework for the semi-automatic construction of evaluation functions for games. According to his approach, only some simple domain specific features are hand coded. Complex rules become automatically derived from the simple features and are combined in a linear evaluation model. Each rule has a weight that will be fitted automatically according to a large set of classified game positions. This approach was very successful in the domain of Othello. The aim of the work presented here is therefore to investigate, whether this evaluation function generator can also be used for the domain of computer chess. The following text contains some helpful hints for an efficient implementation of such an evaluation function generator for chess and also the necessary mathematical formulas needed for the weight fitting algorithm.

Inhaltsverzeichnis

1 Einleitung

Schachprogramme können dank der hohen Rechenleistung aktueller Prozessoren schon bei kurzer Bedenkzeit vier und mehr Züge vorausdenken. Damit sind sie taktisch auch sehr guten Schachspielern weit überlegen. Mensch gegen Maschine-Turniere zeigen aber, dass gute Schachspieler gegen gute Schachprogramme keineswegs chancenlos sind. Diese Feststellung lässt sich damit begründen, dass, im Vergleich zu einem guten Schachspieler, das positionelle Wissen eines Programmes relativ bescheiden ausfällt.
Eine der Hauptschwierigkeiten bei der Schachprogrammierung besteht darin, das Schachwissen, welches man aus unzähligen Büchern beziehen kann, in Programmcode umzusetzen. Diese Umsetzung geschieht heute meist durch explizites Einprogrammieren von Regeln und anschliessender Gewichtung der Regeln. Dabei muss der Programmierer ein einigermassen grosses Schachwissen aufweisen und er darf vor wochenlanger Optimierungsarbeit nicht zurückschrecken.
Im folgenden Text soll nun ein Bewertungsverfahren vorgestellt werden, bei welchem sowohl das Erstellen der Regeln als auch deren Gewichtung automatisiert wird.
Das Potential dieses Verfahrens wird aus verschiedenen Gründen hauptsächlich anhand von Schachendspielen erprobt.
Schliesslich soll hier noch festgehalten werden, dass es sich bei diesem Bewertungsverfahren nicht um eine Neuentwicklung handelt, sondern dass ein sehr ähnlicher Ansatz schon in einem der weltbesten Othelloprogramme umgesetzt wurde [1].

2 Funktionsweise der Bewertungsfunktion

Eine Bewertungsfunktion soll einer gegebenen Spielposition einen Wert zuweisen, welcher zum Beispiel über die Siegeschancen der Partei mit Zugrecht Auskunft gibt. Dabei soll diese Bewertungsfunktion rein positionellen Charakter haben. Es ist also nicht die Aufgabe der Bewertungsfunktion, eine Suche im Zustandsraum durchzuführen. Vielmehr wird wie bei Schachprogrammen üblich eine α-β Suche gestartet, wobei an den Astenden des Suchbaumes die Bewertungsfunktion aufgerufen wird und schliesslich deren Wert an die Wurzel zurückpropagiert wird.
Im folgenden wird der Aufbau dieser Bewertungsfunktion erläutert.

2.1 Fakten

Eine Schachposition wird durch Figuren, deren Standorte sowie ein paar weitere Gegebenheiten wie Zugsrecht, en passant Feld und Rochadestatus eindeutig charakterisiert. Diese Angaben stellen aber aus verschiedenen Gründen noch keine sinnvollen Eingangsgrössen für eine Bewertungsfunktion dar. In einem ersten Schritt soll vorerst eine gegebene Schachposition in eine endliche Anzahl von aussagekräftigen Fakten (entsprechen den „Atomic Features“ in [1]) umgewandelt werden, welche dann der Bewertungsfunktion übergeben werden. Der Mensch hat eine gute

Intuition, welche Fakten für einen bestimmten Problembereich von Bedeutung sind. Die einzelnen Fakten müssen einfach sein, in ihrer Gesamtheit sollten sie aber dennoch eine vorhandene Schachposition gut charakterisieren. Um die Diskussion im folgenden zu vereinfachen, nehmen wir nun an, dass bei einer gegebenen Position ein Faktum nur zutreffen kann oder nicht.

2.2 Konfigurationen

Mehrere Fakten werden zu Konfigurationen verknüpft. Eine Konfiguration ist bei einer bestimmten Spielposition aktiv, falls alle deren Fakten zutreffen. Durch diese Verknüpfung von einfachen Fakten zu Konfigurationen ist die Bewertungsfunktion nun in der Lage, komplexe Zusammenhänge darzustellen. Nehmen wir an, dass „der König ist im Zentrum“ ein Faktum darstelle. Es wird sofort klar, dass sich mit diesem einzelnen Faktum noch keine Aussage machen lässt: Manöveriert man den König bei der Eröffnung ins Zentrum, so steht es im allgemeinen schlecht um die eigenen Siegeschancen. Bei einem Endspiel hingegen kann es allenfalls unerlässlich sein, dass man den König Richtung Zentrum bewegt, da er dort wesentlich mehr Einfluss auf den Spielverlauf hat. Hat man als zweites Faktum „es sind wenig Figuren auf dem Spielfeld“, so kann man mit diesen beiden Fakten die Konfiguration {„der König ist im Zentrum“, „es sind wenig Figuren auf dem Spielfeld“} zusammensetzen. Mit dieser Konfiguration lässt sich nun eine sinnvolle Aussage machen: Ist diese Konfiguration aktiv, so hat dies positive Auswirkungen für die betroffene Partei. Dieser Sachverhalt wird so dargestellt, dass jede Konfiguration einen Wert erhält, welcher die Bedeutung der Konfiguration symbolisiert.
Während die Fakten vom Menschen programmiert wurden, überlässt man dem Computer das Erstellen von Konfigurationen sowie deren Bewertung. Schon aus einigen wenigen Fakten lassen sich tausende von Konfigurationen generieren.

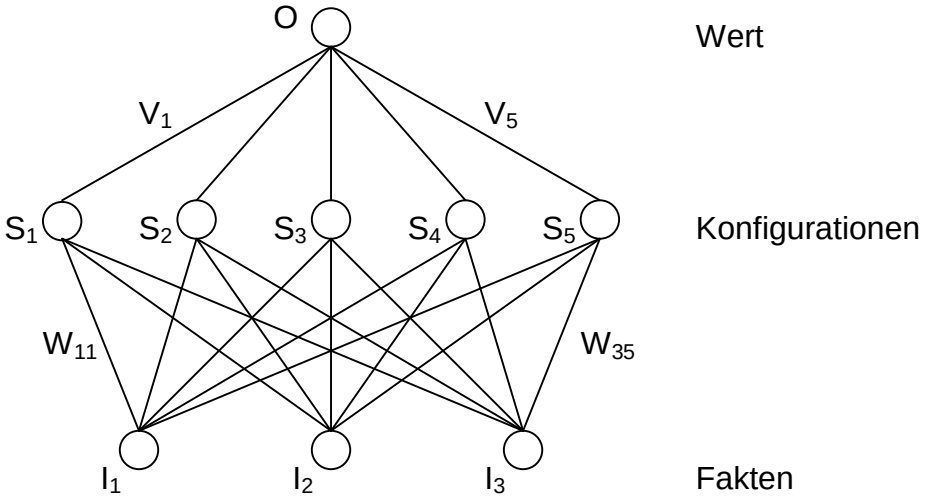
2.3 Auswertung

Bei einer gegebenen Spielposition werden alle zutreffenden Fakten bestimmt und an die Bewertungsfunktion übergeben. Die Bewertungsfunktion entscheidet nun aufgrund der Fakten, welche Konfigurationen aktiv sind. Der Rückgabewert der Bewertungsfunktion ist die Summe der Werte der aktiven Konfigurationen.
Da die Bewertungsfunktion sehr oft aufgerufen wird, ist eine effiziente Bestimmung der zutreffenden Fakten sowie das schnelle Auffinden der aktiven Konfigurationen von grosser Bedeutung. Vorschläge, wie man dieser Anforderung gerecht wird, finden sich im folgenden Kapitel.

2.4 Interpretation als Multilayer Perceptron

Die hier beschriebene Bewertungsfunktion kann sehr einfach als ein Feedforward-Netzwerk mit einem Hidden Layer, auch bekannt unter dem Namen Multilayer-Perceptron, interpretiert werden.
Die Fakten I_1-I_n stellen dabei die Inputs dar. Sei I_i gleich eins für den Fall, dass das i-te Faktum zutreffe, andernfalls sei I_i null. Die Gewichte W_{ij} sind entweder eins oder null, je nachdem ob das Faktum i Teil der Konfiguration j ist oder nicht. S_j wird eins, falls $\Sigma W_{ij} I_i$ gleich der Anzahl der in der Konfiguration enthaltenen Fakten ist, andern-

falls null. Der Rückgabewert der Bewertungsfunktion ist O und berechnet sich als $O=\Sigma S_j V_j$.



Dieser Vergleich mag vorerst etwas erzwungen wirken. Bei einem allgemeinen Multilayer-Perceptron werden die Gewichte meistens durch Error-Backpropagation bestimmt und nehmen somit kontinuierliche Werte an. Es zeigt sich aber [2], dass die hier erfolgte Diskretisierung der Gewichte W_{ij} die Möglichkeiten eines allgemeinen Multilayer-Perceptrons gut approximiert.
Dieser Vergleich ermöglicht somit, von Erfahrungen, welche man mit Multilayer-Perceptrons gemacht hat, zu profitieren.

3 Beschreibung der Implementation

In den folgenden Unterkapiteln sollen einige Teilaspekte der Implementierung näher beschrieben werden. Das ganze Bewertungsverfahren kann in zwei Phasen unterteilt werden: Zuerst muss die Bewertungsfunktion überhaupt konfiguriert werden, das heisst, die Konfigurationen sowie deren Gewichte müssen anhand von bewerteten Schachpositionen bestimmt werden. Erst dann kann die Bewertungsfunktion zur Bewertung einer beliebigen Position eingesetzt werden.
Wie schon erwähnt, hat die Effizienz der Implementierung sehr grosse Bedeutung, da die Bewertungsfunktion sowohl bei der Konfiguration als auch beim Einsatz in einer Brute-Force Suche sehr oft aufgerufen wird. Da die ganze programmtechnische Umsetzung in relativ kurzer Zeit erfolgen musste, ist zu erwarten, dass das Optimierungspotential des ganzen Algorithmus bei weitem noch nicht ausgeschöpft wurde. Dennoch dürfte der folgende Text einige interessante Ansätze liefern.

3.1 Fakten

Nehmen wir an, dass wir insgesamt n verschiedene Fakten haben. Damit ist es zumindest theoretisch möglich 2^n verschiedene Konfigurationen zu generieren. Schon für kleine n hat man so viele Möglichkeiten, dass eine effiziente Verwaltung

der Konfigurationen unmöglich wird - geschweige denn eine Anpassung deren Gewichte. Leider ist es schwierig, im voraus zu bestimmen, ob eine Konfiguration sinnvoll ist oder nicht.
Für diese Bewertungsfunktion wurde daher folgender Ansatz gewählt (vgl. [1]): Alle n Fakten werden in kleinere Gruppen mit k < n Fakten aufgeteilt. Eine Verknüpfung zu Konfigurationen erfolgt nur noch innerhalb dieser kleineren Gruppen. Um die Leistung der Bewertungsfunktion nicht zu stark einzuschränken, sollten natürlich logisch zusammengehörende Fakten in die gleiche Gruppe eingeteilt werden. Im folgenden wird jeder auf dem Schachbrett stehenden Figur eine Gruppe von Fakten zugewiesen. Mit diesem Ansatz kann man figurtypspezifischen Eigenschaften gut gerecht werden.
Jedes Faktum wird als ein Bit codiert – trifft das Faktum bei einer bestimmten Position zu, so ist das Bit für diese Position true, andernfalls false. Die Anzahl Fakten pro auf dem Schachbrett vorhandenen Figuren wird auf 32 beschränkt. Damit lässt sich bei einer gegebenen Schachposition für jede Figur ein Bitvektor mit einer Länge von vier Bytes generieren.

Als Beispiel sei hier die Codierung des Bitvektors eines Königs gegeben:

Bit	Bedeutung
0	Farbe der Figur ist im Vorteil (Figurenvorsprung)
1	Farbe der Figur ist im Nachteil (Figurenrückstand)
	Figur wird angegriffen von:
2,3	Bauer
4	Springer
5	Läufer
6	Turm
7	Dame
8	...
	Figur wird gedeckt von:
9,10	Bauer
11	Springer
12	Läufer
13	Turm
14	Dame
15	...
16	Farbe der Figur ist nicht im Zugsrecht
17-20	Distanz zur Ecke
21-26	Distanz zum gegnerischen König
27	König wurde rochiert
28-31	Standort

3.2 Konfigurationen

Wie im vorhergehenden Unterkapitel erwähnt, werden Fakten nur pro Figur zu Konfigurationen verknüpft und ausgewertet. Jeder Figurtyp – Bauer, Springer, Läufer, Turm, Dame und König – besitzt somit ein Set von Konfigurationen. Die Farbe der Figur wird dabei nicht unterschieden. Die Fakten werden pro Figurtyp so

codiert, dass sie sich bezüglich der Farbe der Figur symmetrisch verhalten. Besondere Vorsicht ist hier beim Bauer angebracht, da dessen Zugeigenschaften stark von der Farbe abhängen.
Mit 32 Fakten pro Figurtyp lassen sich theoretisch immer noch 2^{32} verschiedene Konfigurationen erzeugen. Da 2^{32} Konfigurationen noch immer die Speicherkapazität der meisten Rechner überfordern würden, kommen hier noch zwei Kriterien zur Reduktion der Anzahl Konfigurationen zum Einsatz (vgl. [1]):

- Die maximale Anzahl der zur Konfiguration gehörenden Anzahl Fakten wird beschränkt
- Es wird eine für den Problembereich relevante Auswahl von Schachpositionen ausgewählt. Damit eine bestimmte Konfiguration in das Set von Konfigurationen aufgenommen wird, muss sie in einem bestimmten Prozentsatz der ausgewählten Schachpositionen aktiv sein.

Eine Konfiguration kann nun auch wieder auf einen Bitvektor aufcodiert werden. Dieses Vorgehen ermöglicht ein sehr effizientes Auffinden der aktiven Konfigurationen. Die Bewertungsfunktion erhält von der Figur A einen Bitvektor, auf welchem die zutreffenden Fakten mit true markiert sind. Andererseits hat die Bewertungsfunktion eine Liste der Bitvektoren der möglichen Konfigurationen der Figur A. Mit folgendem simplen Vergleich lässt sich nun feststellen, ob die Konfiguration B bei Figur A aktiv ist:

```
if ( (B.Bitvektor & A.Bitvektor) == B.Bitvektor )
{
      Konfiguration B ist bei Figur A aktiv
}
else
{
      Konfiguration B ist bei Figur A nicht aktiv
}
```

Der Operator & steht hier für einen bitweisen Vergleich der korrespondierenden Elemente der beiden Bitvektoren.
Da die Anzahl möglicher Konfigurationen pro Figurtyp sehr gross werden kann, wird zur weiteren Optimierung pro Konfiguration noch eine Sprungadresse gespeichert. Die Konfigurationen können nämlich in einer Liste so verwaltet werden, dass, sofern eine Konfiguration nicht aktiv ist, eine grosse Anzahl folgender Konfigurationen ohne Test übersprungen werden können, da sie garantiert auch nicht aktiv sind. Trifft zum Beispiel bei einer bestimmten Figur das Faktum C nicht zu, so können alle Konfigurationen, welche Faktum C enthalten, übersprungen werden.

3.3 Auswertung

Als rechentechnische Vereinfachung wird in aktuellen Schachprogrammen meist anstelle des Minimax-Prinzipes die Negamax Variante implementiert [3]. Für die Bewertungsfunktion bedeutet dies, dass die Bewertung immer aus Sicht der Farbe mit Zugrecht geschehen soll.

In einem ersten Schritt werden alle aktiven Konfigurationen der Seite mit Zugrecht gesucht und deren Werte werden zum Wert A aufsummiert. Der gleiche Vorgang wird für die Seite ohne Zugrecht wiederholt, wobei dort die Werte der aktiven Konfigurationen zum Wert B aufsummiert werden. Der Rückgabewert der Bewertungsfunktion berechnet sich dann als A minus B.
Hier bietet sich eine weitere wichtige Optimierungsmöglichkeit: Eine Figur hat im allgemeinen mehrere tausend mögliche Konfigurationen. Das bestimmen der aktiven Konfigurationen sowie das aufsummieren deren Werte nimmt also viel Rechenleistung in Anspruch. Das Resultat dieser Auswertung pro Figur ist einzig vom Bitvektor der Figur – also von den zutreffenden Fakten bei einer gegebenen Schachposition – abhängig. Von nun an wird also, nachdem die aktiven Konfigurationen für eine Figur bestimmt und deren Werte zum Wert C aufsummiert wurden, der Wert C gespeichert. Der Speicher kann ähnlich wie ein Cache Speicher aufgebaut werden. Die Adresse berechnet sich dabei aus dem Bitvektor, gespeichert wird der Wert C und als eindeutiger Schlüssel dient wiederum der Bitvektor der Figur.
Wenn nun in Zukunft wieder die Summe der Werte der aktiven Konfigurationen einer Figur berechnet werden muss, so kann zuerst im Speicher nachgeschaut werden, ob die gleiche Rechnung bereits ausgeführt wurde. Trifft dies zu, so kann der Wert aus dem Speicher gelesen werden und eine Neuberechnung erübrigt sich damit. Wird der Speicher gut organisiert, so kann damit gerechnet werden, dass in über 99% der Speicheranfragen ein Hit zu verzeichnen ist, also der korrekte Wert schon mal berechnet wurde. Dies führt zu einer enormen Beschleunigung der Bewertungsfunktion.

3.4 Berechnung der Werte der Konfigurationen

Wie schon zuvor erwähnt, wird die Bedeutung einer Konfiguration mit einem Wert ausgedrückt. In der Einleitung wurde eine automatische Berechnung dieser Werte versprochen. Voraussetzung dazu ist, dass man für bestimmte Schachpositionen vorgegebene Bewertungen hat, welche durch die Bewertungsfunktion möglichst genau approximiert werden sollen. Bei Endspielen kann eine Schachposition sogar exakt, das heisst in Anzahl Halbzügen bis zum Schachmatt oder zu einer Umwandlung, bewertet werden.

Für die Berechnung der Werte wird nun folgende Notation eingeführt:

- i Index über die Schachpositionen
- k Index über die möglichen Konfigurationen
- $h_{i,k}$ Zähler, wie oft die k-te Konfiguration in der i-ten Position aktiv ist. Konfigurationen, welche bei Figuren der Farbe mit Zugrecht aktiv sind, zählen positiv, die anderen negativ.
- r_i Vorgegebener Wert der i-ten Position
- w_k Gewicht der k-ten Konfiguration
- $\underline{w}$ Gewichte der Konfigurationen in Vektorschreibweise

Die Bewertung der i-ten Position durch die Bewertungsfunktion ist dann gegeben mit

$$e_i(\underline{w}) = \sum_k w_k h_{i,k} \ .$$

Der Sollwert derselben Position ist r_i und wir definieren ein quadratisches Fehlermass zwischen berechnetem Wert und Sollwert. Der quadratische Fehler, aufsummiert über alle i Positionen, beträgt dann

$$f(\underline{w}) = \sum_i (r_i - e_i(\underline{w}))^2 \ .$$

Das Ziel ist es, $f(\underline{w})$ zu minimieren. Es handelt sich hier also um ein quadratisches Optimierungsproblem. Solche Probleme werden in der Regel mit einem konjugierten Gradientenverfahren gelöst. Die für die numerische Lösung des vorliegenden Problems nötigen Formeln werden im folgenden Text angegeben.

Das Verfahren der konjugierten Gradienten besteht aus folgenden Schritten [7]:

a) Da im voraus nichts über die Werte der Konfigurationen bekannt ist, werden im ersten Schritt ($m=1$) alle Werte der Konfigurationen mit 0 initialisiert:
$\underline{w}^1 = \underline{0}$
Als erste Abstiegsrichtung wird der negative Gradient der Fehlerfunktion $f(\underline{w})$ gewählt:
$\underline{d}^1 = -\nabla f(\underline{w}^1)$

b) Es wird ein Schritt mit der Abstiegsrichtung $\underline{d}^m$ ausgeführt. Die Werte der Konfigurationen nach diesem Schritt betragen
$\underline{w}^{m+1} = \underline{w}^m + \alpha_m \underline{d}^m$.
α_m wird so gewählt, dass die Fehlerfunktion entlang der Abstiegsrichtung minimiert wird (aus Übersichtlichkeitsgründen wird im folgenden bei α, $\underline{d}$ und $\underline{w}$ der Index m weggelassen):

$$\alpha = \frac{\sum_i (r_i - \underline{w}^T \underline{h}_i)\underline{d}^T \underline{h}_i}{\sum_i [\underline{d}^T \underline{h}_i]^2}$$

Die neue Abstiegsrichtung lautet:

$$\underline{d}^{m+1} = -\nabla f(\underline{w}^{m+1}) + \mu_m \underline{d}^m \text{ mit } \mu_m = \frac{\nabla f(\underline{w}^{m+1})^T \nabla f(\underline{w}^{m+1})}{\nabla f(\underline{w}^m)^T \nabla f(\underline{w}^m)}$$

c) Wiederholen des Schrittes b), bis eine genügend grosse Genauigkeit erreicht wird.

Bemerkungen:

- Die Konvergenz des hier angegebenen Verfahrens kann mit einer Jacobi-Vorkonditionierung noch verbessert werden.
- Die Komponentenweise Darstellung für $\nabla f(\underline{w})$ lautet:

$$\frac{\partial f(\underline{w})}{\partial w_l} = \sum_i 2(r_i - \sum_k w_k h_{i,k})(-h_{i,l})$$

3.5 Spielphasen

Im Rahmen der Semesterarbeit wurde die hier beschriebene Bewertungsfunktion an verschiedenen Schachproblemen auf ihre Tauglichkeit erprobt. Es wurde schnell einmal festgestellt (vgl. [1]), dass es wenig Sinn macht, alle möglichen Schachpositionen auf einmal erfassen zu wollen. Zwar sind viele Parallelitäten zwischen den vier Endspielen KPK, KPPK, KPKP und KPPKP zu erkennen, zwischen KBBK und einer Eröffnung besteht aber kaum ein Zusammenhang. Diesem Umstand soll Rechnung getragen werden, indem alle denkbaren Schachpositionen in verschiedene Gruppen eingeteilt werden. Die Konfigurationen mitsamt deren Werten werden dann für jede Gruppe einzeln bestimmt. Die Gruppeneinteilung kann aufgrund von verschiedenen Gesichtspunkten erfolgen. Mögliche Entscheidungskriterien sind zum Beispiel Materialbilanzen, Materialdifferenzen, Bauernstellungen, Königssicherheit, Endspieltypen usw.
Wenn die Bewertungsfunktion in einer Brute-Force Suche eingesetzt werden soll, so ist noch eine weitere Überlegung wichtig: Theoretisch müsste vor jedem Aufruf der Bewertungsfunktion bestimmt werden, zu welcher Gruppe die momentan vorliegende Schachposition gehört. Je nach Art der Gruppeneinteilung ist dies aber viel zu aufwendig und es wird daher nur an der Wurzel des Suchbaumes die Gruppenzugehörigkeit der dort vorhandenen Schachstellung ermittelt. Innerhalb der Suche können nun aber andere Gruppen von Schachstellungen vorgefunden werden, wobei auch dort noch eine sinnvolle Bewertung erfolgen sollte. Um im ganzen Suchbaum eine sinnvolle Bewertung zu gewährleisten, sollte in jeder Gruppe von Schachpositionen auch eine Anzahl von weiteren Positionen enthalten sein, welche mit den eigentlichen Positionen aus der Gruppe verwandt sind.
Ein weiterer Grund, dass die Gruppenzugehörigkeit nur an der Wurzel des Suchbaumes bestimmt wird, ist, dass die Bewertungen über die Grenzen von Gruppen hinweg nicht ohne weiteres vergleichbar sind.

4 Wahl des Trainingssets

Mindestens ebenso schwierig wie die Codierung der Fakten gestaltet sich das Zusammenstellen eines guten Trainingssets. Einerseits muss eine für den Problembereich repräsentative Auswahl von Schachpositonen getroffen werden, andererseits müssen alle ausgewählten Schachpositionen bewertet werden. Da tausende von Schachpositionen benötigt werden, kommt nur eine automatische Bewertung in Frage.
Eine perfekte Lösung existiert nur für Endspiele. Dort kann, sofern das Endspiel in einer Datenbank erfasst wurde, jeder Position ein exakter Wert zugewiesen werden. In der bei dieser Arbeit zur Verfügung stehenden Datenbank wurde für jede Schachposition die Anzahl Halbzüge bis zum Schachmatt oder bis zu einer Umwandlung berechnet. Ist die Anzahl Halbzüge ungerade, so bedeutet dies für die Seite mit Zugsrecht, dass sie gewinnen wird. Mit folgender einfachen Abbildung wurden diese Informationen in ein Format umgewandelt, welches sich für ein Trainingsset besser eignet:

```
if ( databaseValue % 2 == 1 )
      newValue = maxValue + OFFSET - databaseValue;
else
      newValue = -(maxValue + OFFSET - databaseValue);
```

maxValue ist dabei die grösste in der Datenbank vorkommende Anzahl Halbzüge.

Schwieriger gestaltet sich das ganze, wenn ein Trainingsset für Eröffnungen beziehungsweise für das Mittelspiel zusammengestellt werden soll. Zufriedenstellende Resultate wurden hier dadurch erzielt, dass ganze Partien betrachtet wurden. Den einzelnen in der Partie enthaltenen Positionen wurden dann aufgrund der gesamten Anzahl Züge, der Anzahl Züge bis zum Schachmatt sowie der über die Partie gemittelten Materialdifferenz je ein Wert zugewiesen.
Auch wenn mit diesem einfachen Ansatz bereits gute Resultate erzielt werden können, sollte man nicht ausser acht lassen, dass für solche und ähnliche Probleme wesentlich raffiniertere Algorithmen existieren, wie beispielsweise Temporal Difference Learning [2], [10].

5 Kompatibilität zu herkömmlichen Algorithmen

Da die hier vorgestellte Bewertungsfunktion letztendlich nur ein Ersatz für eine herkömmlich programmierte Bewertungsfunktion ist, müssen am restlichen Programmcode kaum Änderungen vorgenommen werden. Insbesondere ist diese Bewertungsfunktion kompatibel mit den in aktuellen Schachprogrammen verwendeten Techniken wie α-β, Nullzug, Hashtables usw.
Vorsicht ist geboten, wenn man versucht, eine herkömmliche, handoptimierte Bewertungsfunktion mit der hier vorgestellten Bewertungsfunktion zu kombinieren. Hier sollte zuerst die automatisch generierte Bewertungsfunktion erstellt werden. Erst dann kann die Handoptimierung stattfinden, wobei die automatisch generierte Bewertungsfunktion in die Handoptimierung miteinbezogen werden sollte.
In dieser Arbeit wurde zum Beispiel die automatisch generierte Bewertungsfunktion mit einer Materialbilanz kombiniert. In der Regel wird bei der Materialbilanz der Wert des Springers etwas unterhalb des dreifachen Wertes eines Bauern angesetzt, der Wert des Läufers etwas oberhalb. Dies führt hier aber zu einem Problem: Bei der Eröffnung versucht der Computer mit dieser Einstellung um jeden Preis einen Springer gegen einen Läufer umzutauschen, selbst wenn er sich dabei einen ungünstigen Dopplebauer einhandelt. Dieses Problem kann einfach behoben werden, indem der Wert des Läufers mit dem Wert des Springers gleichgesetzt wird.

6 Resultate

6.1 Endspiele

Im Rahmen dieser Arbeit wurde die neue Bewertungsfunktion intensiv anhand von Endspielen getestet. Endspiele haben zwei gewaltige Vorteile: Erstens ist die Erstellung eines Trainingssets einfach und zweitens kennt man bei Endspielen bei allen Positionen, die in einer Datenbank erfasst wurden, die optimalen Züge.

Die Versuchsanordnung sah wie folgt aus:

1. Mit Hilfe einer Endspieldatenbank wurden für verschiedene Endspiele Trainingssets generiert. Bei einfacheren Endspielen wie KRK wurden alle bei diesem Endspiel legalen Positionen ins Trainingsset integriert, bei komplexeren Endspielen wie KPPKP konnten nur noch ca. 1% der legalen Positionen berücksichtigt werden.
2. Die Bewertungen der einzelnen Positionen wurden gemäss dem in Kapitel 4 beschriebenen Vorgehen angepasst.
3. Die verschiedenen Trainingssets wurden zur Bestimmung der einzelnen Werte der Konfigurationen benutzt.
4. Die Kenndaten der trainierten Bewertungsfunktion wurden in Statistiken erfasst.
5. Für das KBBK Endspiel wurde ein Testset generiert, welches mit dem Trainingsset von KBBK keine Position gemeinsam hat. Die statistischen Kenndaten wurden bei diesem Testset ebenfalls erfasst und mit denjenigen vom Trainingsset verglichen.

Hier noch einige Erläuterungen, zu den in den in den folgenden Statistiken enthaltenen Werten:

Konfigurationen: Pro Endspiel und Figurtyp wird angegeben, wie viele Konfigurationen generiert wurden.

Mittlerer Fehler, mittlerer quadratischer Fehler: Für die Trainingssets wurden vor und nach dem Trainieren der mittlere und der mittlere quadratische Fehler zwischen dem Wert im Trainingsset und dem von der Bewertungsfunktion vorhergesagten Wert ausgerechnet. Vor dem Trainieren wird jede Position mit dem Wert null bewertet.

Gewinn, Unentschieden, Verlust Vergleich: In diesem Vergleich wird die Korrektheit der Voraussage bezüglich des Spielausgangs bei einer gegebenen Position ermittelt.

Halbzugsuche: Bei einer gegebenen Position wird eine Tiefensuche mit der Tiefe von genau einem Halbzug durchgeführt und dadurch der mutmasslich beste Zug ermittelt. Die Materialbilanz wird dabei in der Bewertung berücksichtigt. Bei jedem mutmasslich besten Zug wird untersucht, ob es sich effektiv um einen optimalen Zug handelt oder nicht. Somit erhält man eine Trefferquote wie oft die Halbzugsuche zu einem korrekten Zug führen würde. Zum Vergleich wurde auch die Trefferquote ermittelt, die man erhält, wenn man bei einer gegebenen Position zufällig einen legalen Zug wählt.

6.1.1 KRK Endspiel

Beim König und Turm gegen König Endspiel geht es darum, den einzelnen König an den Rand zu drängen und dort matt zu setzen. Dazu sind ein paar charakteristische Zugfolgen notwendig. Jedes gute Schachprogramm beherrscht dieses Endspiel problemlos. Allerdings konnte festgestellt werden, dass ein mit der neuen Bewertungsfunktion betriebenes Schachprogramm bei diesem Endspiel meist näher an der Optimalvariante spielt, als sehr gute Freeware Programme.

Statistik für das KRK Endspiel (Trainingsset)				
Anzahl Konfigurationen:		König:	798	
		Turm:	659	
Mittlerer Fehler:		Vor Training:	15.6 Halbzuege	
		Nach Training:	2.13 Halbzuege	
Mittlerer quadratischer Fehler:		Vor Training:	286 Halbzuege^2	
		Nach Training:	8.59 Halbzuege^2	
		Bewertungsfunktion		
		Gewinn	Unentschieden	Verlust
Daten-bank	Gewinn	43.9%	0%	0%
	Unentschieden	0%	5.48%	0.108%
	Verlust	0%	0.112%	50.4%
Trefferquote der Bewertungsfunktion: 99.8%				
Halbzugsuche:		Trefferquote mit Bewertungsfunktion:		68.7%
		Trefferquote bei zufälliger Wahl des Zuges:		27.5%

6.1.2 KBBK Endspiel

Das Endspiel, bei dem ein König mit zwei Läufern gegen einen einzelnen König gewinnen muss, wird nach dem Training ebenso erfolgreich beherrscht, wie das viel einfachere König Turm gegen König Endspiel.

Statistik für das KBBK Endspiel (Trainingsset, *Testset*)				
Anzahl Konfigurationen:		König:	810	
		Läufer:	729	
Mittlerer Fehler:		Vor Training:	16.4 Halbzuege	
		Nach Training:	2.47 Halbzuege *2.5 Halbzuege*	
Mittlerer quadratischer Fehler:		Vor Training:	306 Halbzuege^2	
		Nach Training:	14 Halbzuege^2 *14.8 Halbzuege^2*	
		Bewertungsfunktion		
		Gewinn	Unentschieden	Verlust
Daten-bank	Gewinn	41.9% *41.8%*	0% *0%*	0% *0%*
	Unentschieden	0.008% *0.013%*	10.6% *10.5%*	0.406% *0.413%*
	Verlust	0% *0%*	0.205% *0.23%*	46.9% *47%*
Trefferquote der Bewertungsfunktion: 99.4% *99.3%*				
Halbzugsuche:	Trefferquote mit Bewertungsfunktion:		62.3%	*65.5%*
	Trefferquote bei zufälliger Wahl des Zuges:		32.2%	*31.6%*

Vergleicht man die auf dem Trainingsset erhaltenen Daten mit denjenigen vom Testset, so kommt man zur erfreulichen Feststellung, dass sich diese Daten nur wenig unterscheiden. Das Verallgemeinerungsverhalten der Bewertungsfunktion ist – nicht nur bei diesem Endspiel – sehr gut.

6.1.3 KBNK Endspiel

Das Endspiel, in welchem ein König mit einem Läufer und einem Springer den gegnerischen König matt setzen muss, gilt als eines der schwierigsten Endspiele mit vier Figuren. Es kann in zwei Abschnitte unterteilt werden: Zuerst wird der einzelne König an den Rand gedrängt, danach muss er mit vereinten Kräften in eine Ecke getrieben werden, welche vom Läufer erreicht werden kann. Nur in diesen Ecken kann der König matt gesetzt werden.

Statistik für das KBNK Endspiel (Trainingsset)				
Anzahl Konfigurationen:		König:	768	
		Läufer:	546	
		Springer:	528	
Mittlerer Fehler:		Vor Training:	19.6 Halbzuege	
		Nach Training:	4.90 Halbzuege	
Mittlerer quadratischer Fehler:		Vor Training:	495 Halbzuege^2	
		Nach Training:	50.3 Halbzuege^2	
		Bewertungsfunktion		
		Gewinn	Unentschieden	Verlust
Daten-bank	Gewinn	44.1%	0.0032%	0%
	Unentschieden	0.214%	8.27%	1.8%
	Verlust	0%	0.487%	45.1%
Trefferquote der Bewertungsfunktion: 97.5%				
Halbzugsuche:	Trefferquote mit Bewertungsfunktion:		59%	
	Trefferquote bei zufälliger Wahl des Zuges:		26.3%	

Nach dem Training wird der erste Abschnitt des Endspiels problemlos beherrscht. Trotz den guten statistischen Resultaten, welche für dieses Endspiel ermittelt wurden, gelingt es dem Programm aber (noch) nicht, den König anschliessend in die richtige Ecke abzudrängen.

6.1.4 KPPKP Endspiel

Nach dem Training ist der subjektive Spieleindruck auch bei diesem Bauernendspiel sehr gut. Die wichtigen Konzepte von diesem Endspiel werden beherrscht. Wird die Bewertungsfunktion nur mit Positionen vom KPPKP Endspiel trainiert, so kann sie dennoch auch für andere Endspiele, bei welchen nur Könige und Bauern auf dem Spielfeld sind, erfolgreich eingesetzt werden.

Statistik für das KPPKP Endspiel (Trainingsset)				
Anzahl Konfigurationen:		König:	687	
		Bauer:	4099	
Mittlerer Fehler:		Vor Training:	43.4 Halbzuege	
		Nach Training:	18.2 Halbzuege	
Mittlerer quadratischer Fehler:		Vor Training:	2091 Halbzuege^2	
		Nach Training:	537 Halbzuege^2	
		Bewertungsfunktion		
		Gewinn	Unentschieden	Verlust
Daten-bank	Gewinn	50.1%	0.761%	0.975%
	Unentschieden	3.3%	5.11%	6.3%
	Verlust	0.926%	1.2%	31.3%
Trefferquote der Bewertungsfunktion: 86.5%				
Halbzugsuche:		Trefferquote mit Bewertungsfunktion:	55.8%	
		Trefferquote bei zufälliger Wahl des Zuges:	39.8%	

6.2 Eröffnung und Mittelspiel

Nachdem bei den Endspielen gute Resultate mit der neuen Bewertungsfunktion erzielt werden konnten, bestand natürlich auch das Interesse, die Tauglichkeit der Bewertungsfunktion bei Eröffnungen und dem Mittelspiel zu erproben. Zu diesem Zweck wurde beim Schachprogramm Chessterfield (ca. 1900 ELO) die herkömmliche Bewertungsfunktion durch diese neuartige Bewertungsfunktion ersetzt. Über mehrere Stunden hinweg wurde danach abwechslungsweise gegen starke Freeware-Programme (> 2200 ELO) gespielt und anhand dieser Spiele die Bewertungsfunktion trainiert. Nach ca. 500 Spielen konnte folgendes Verhalten festgestellt werden: Wichtige Grundkonzepte von Schach sind erfasst worden und das Programm als ganzes hat seit dem Trainingsbeginn ca. 250 ELO Punkte zugelegt. Damit spielt es bereits etwas über dem Niveau, das es mit der alten Bewertungsfunktion erreicht hat. Zu berücksichtigen ist hier aber, dass die neue Bewertungsfunktion im Gegensatz zu der alten kaum optimiert wurde. Hier kann also durch Laufzeitoptimierung sowie durch Verbesserung der Fakten und des Trainingssets noch sehr viel Potential ausgeschöpft werden.

7 Fazit

Die hier vorliegenden Resultate lassen erahnen, dass diese Art von automatisch generierter Bewertungsfunktion in Zukunft auch vermehrt auf dem Gebiet von Computerschach zum Einsatz kommen könnte.
Die hier und in [1] beschriebene Technik vereint zwei ganz wesentliche Eigenschaften auf sich. Einerseits hat sie nahezu die Mächtigkeit eines Neuronalen Netzwerks, genauer die eines Multilayer-Perceptrons, andererseits ist der rechnerische Aufwand im Vergleich zu einem herkömmlich implementierten Neuronalen Netzwerke [9] äusserst bescheiden. Mit der hier beschriebenen Vorgehensweise kann somit Brute-Force mit intelligenten Lernalgorithmen verbunden werden.
Soweit mir dies bekannt ist – die Bewertungsfunktionen kommerzieller Schachprogramme sind streng geheim – werden heute immer noch die meisten Bewertungsfunktionen in jahrelanger Handarbeit optimiert. Es ist nicht zu erwarten, dass, sofern die Möglichkeit überhaupt theoretisch besteht, diese hochoptimierten Bewertungsfunktionen in kurzer Zeit überboten werden können. Auf dem Gebiet dieser automatisch generierten Bewertungsfunktionen ist noch sehr viel Entwicklungsarbeit zu erledigen, welche sicher noch einige Zeit in Anspruch nehmen wird.

8 Literaturhinweise

[1] M. Buro; From Simple Features to Sophisticated Evaluation Functions; Lecture Notes in Computer Science LNCS 1558; Springer Verlag; 1998
[2] G. Tesauro; Temporal Difference Learning and TD-Gammon; Communications of the ACM; Vol. 38, No. 3, 1995
[3] D. Steinwender, F. A. Friedel; Schach am PC; Markt und Technik Verlag; 1995
[4] C. Wirth; Exhaustive and Heuristic Retrograde Analysis of the KPPKP Endgame; ICCA Journal; Vol. 22, No. 2; 1999
[5] J. Nunn; Taktische Schachendspiele; Falken Verlag; 1985
[6] P. Keres; Praktische Endspiele; Kurt Rattmann Verlag Hamburg; 1973
[7] I. N. Bronstein, K. A. Semendjajew, G. Musiol, H. Mühlig; Taschenbuch der Mathematik; Harri Deutsch Verlag; 4. Auflage, 1999
[8] M. Bain, S. H. Muggleton, A. Srinivasan; Generalising Closed World Specialisation: A Chess End Game Application; 1995
[9] S. Thrun; Learning to Play the Game of Chess; Advances in Neural Information Processing Systems (NIPS) 7; MIT Press; 1995
[10] Jonathan Baxter, Andrew Tridgell, Lex Weaver; KnightCap: A chess program that learns by combining TD(λ) with game-tree search

Anhang

A Software

Endspieldatenbank von Christoph Wirth

Dank der Endspieldatenbank von Christoph Wirth konnte ein Trainingsset mit allen Schachpositionen vom KK bis zum KPPKP Endspiel erstellt werden. Der Wert von jeder Position wird in dieser Datenbank auf ein Byte codiert. In diesem Byte enthalten ist, ob die Position legal ist, und, falls dies zutrifft, die Anzahl Halbzüge bis zum Schachmatt oder bis zu einer Umwandlung. Dank einer ausgeklügelten Zugriffsroutine auf die Datenbank kann von einer beliebigen Position deren Wert erfragt werden. Mehr Informationen zu dieser Datenbank können in [4] gefunden werden.

Die Adresse des Autors lautet:

Christoph Wirth
ETH Zürich, Institut für Theoretische Informatik
CH-8092 Zürich

wirthc@inf.ethz.ch

WinBoard, XBoard

WinBoard bzw. XBoard ist ein grafisches Frontend für zahlreiche, teilweise sehr starke, Schachprogramme wie beispielsweise Crafty. WinBoard ist vor allem bei den Freeware Schach Entwicklern sehr beliebt, da damit das eigene Schachprogramm gegen sehr starke andere Freeware Programme vollautomatisch getestet werden kann. Mittlerweile hat sich das Konzept von WinBoard auch im Profibereich durchgesetzt. Zugute kommt diesem Programm insbesondere auch, dass es für fast alle gängigen Plattformen erhältlich ist.

Programmiert wurde WinBoard von Tim Mann und ist im Internet gratis erhältlich unter:

http://www.research.digital.com/SRC/personal/mann/chess.html

Chessterfield

Chessterfield ist ein einfaches Schachprogramm geschrieben von Matthias Lüscher. Es existiert in zwei verschiedenen Varianten: Einerseits ist es für das Windows Betriebssystem mit einem grafischen Interface erhältlich, andererseits existiert auch eine WinBoard kompatible Kommandozeilenversion. Die Spielstärke des Programmes beträgt ungefähr 1900 ELO Punkte. Das Programm wurde vollständig objektorientiert in der Sprache C++ geschrieben.
Die neuentwickelte Bewertungsfunktion wurde in Chessterfield eingebaut und getestet. Die Bewertungsfunktion benutzt zudem zahlreiche Features von Chessterfield. Das Copyright von Chessterfield liegt bei Matthias Lüscher. Sollte im Rahmen einer weiteren Semester- oder Diplomarbeit Interesse am Sourcecode von Chessterfield bestehen, so kann eine jeweils aktuelle Version des Sourcecodes bei Matthias Lüscher bezogen werden.

Weitere Informationen zu Chessterfield finden sich unter:

http://www.luescher-online.com/computerchess.html

B Verwendete Standards

Portable Game Notation PGN

PGN ist ein Standard, der sich zur Repräsentation von Schachspieldaten durchgesetzt hat. PGN ist so strukturiert, dass es sowohl von Schachprogrammen als auch von Menschen einfach gelesen und geschrieben werden kann.
Der Standard ist gut dokumentiert und dessen Spezifikation kann beispielsweise unter folgender Adresse gefunden werden:

http://www.research.digital.com/SRC/personal/mann/Standard

Forsyth-Edwards Notation FEN

FEN wird zur Beschreibung von Schachpositionen benutzt. In FEN sind alle Informationen enthalten, welche zur eindeutigen Beschreibung einer Schachposition notwendig sind. Die FEN Spezifikation ist in der PGN Spezifikation enthalten.

C Klassenhierarchie von Chessterfield (Ausschnitt)