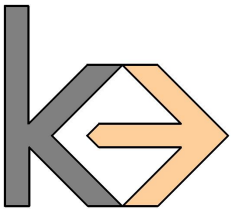
# Lernen unterschiedlich starker Bewertungsfunktionen aus Schach-Spielprotokollen



Diplomarbeit
8. Juli 2009

**Philip Paulsen**

Prof. Johannes Fürnkranz

Knowledge Engineering

TU Darmstadt

# Erklärung

Hiermit versichere ich, diese Arbeit ohne unzulässige Hilfsmittel und nur unter Verwendung der angegebenen Literatur erstellt zu haben.

Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Darmstadt, 8. Juli 2009 Philip Paulsen

# Danksagung

Ich danke allen die mir bei der Anfertigung dieser Arbeit mit Rat und Kritik zur Seite gestanden haben für ihre Unterstützung. Ganz besonderer Dank gebührt Philipp Lies, ohne den ich im Laufe des Studiums sicher mehrfach verrückt geworden wäre und meiner Freundin Josephine dafür, dass sie mich unterstützt und meine Launen ertragen hat, wenn es gerade einmal nicht gut lief. Ich danke ebenso meinen Eltern die mir durch ihr Vertrauen und ihre finanzielle Unterstützung mein Studium erst ermöglicht haben. Auch danke ich Prof. Dr. Johannes Fürnkranz für die Vergabe der Diplomarbeit sowie für seine Vorschläge und seine moralische Unterstützung. Des weiteren gilt mein Dank den Mitarbeitern des Fachgebietes Knowledge Engineering die mir bei Fragen zur Seite standen.

Der Mensch hat drei Wege, klug zu handeln:
Erstens durch Nachdenken: Das ist der edelste.
Zweitens durch Nachahmen: Das ist der leichteste.
Drittens durch Erfahrung: Das ist der bitterste.
- Konfuzius

# Kurzfassung

Aufgabe dieser Arbeit war die Anwendung von Methoden des maschinellen Lernens, um auf der Basis von Schach-Spielprotokollen Spieler unterschiedlicher Stärke zu lernen. Dazu wurde ein herkömmliches Open Source Schach-Programm um eine Komponente erweitert, die es erlaubte, die Evaluierungsfunktion anzupassen. Das Lernen erfolgte über eine Ranking Support Vector Machine, welche Gewichte für die einzelnen Parameter bestimmte. Ziel war es, den von einem Spieler der jeweiligen Stärke gewählten Zug am höchsten zu reihen. Die gelernten Funktionen wurden anschließend vergleichend interpretiert. Anschließend wurde deren Spielstärke sowohl durch Turniere gegeneinander, als auch durch ihre Spielstatistik auf einem Internet Schachserver bewertet. Es war auf diesem Weg möglich brauchbare Evaluierungsfunktionen zu erzeugen. Das Ziel mit diesen Funktionen verschiedene Spielstärken zu simulieren konnte jedoch nicht erreicht werden.

# Inhaltsverzeichnis

# Tabellenverzeichnis

# Abbildungsverzeichnis

# Kapitel 1

# Einführung

## 1.1 Motivation

Das Erlernen von Bewertungsparametern, die das Spielverhalten unterschiedlich starker Spieler widerspiegeln, stellt den Schwerpunkt dieser Arbeit dar. Diesem Ziel liegt die Annahme zugrunde, dass der Einfluss der Bewertungsfunktion allein groß genug ist um deutliche Unterschiede in der Spielstärke zu bewirken. Im Rahmen dieser Arbeit wird untersucht, ob sich mit einer Ranking-SVM aus Schach-Spielprotokollen aussagekräftige Parameter lernen lassen und in wie fern sich die auf diese Art, von Spielern unterschiedlicher Stärke, gelernten Evaluierungsfunktionen unterscheiden.

## 1.2 Aufbau der Arbeit

Zuerst wird in den Kapiteln zwei und drei das nötige Grundwissen für das Verständnis der später durchgeführten Verarbeitungsschritte und Versuche vermittelt. Das zweite Kapitel beinhaltet dabei eine Vielzahl von Themen rund um das Schachspiel und dessen Umsetzung auf dem Computer während das dritte Kapitel die verwendeten Methoden des Maschinellen Lernens behandelt. Der vierte Abschnitt erklärt wie aus den Schach-Spielprotokollen verwendbare Trainingsdaten gewonnen werden und wie diese durch die Support Vector

Machine verarbeitet werden um unterschiedliche Bewertungsfunktionen zu lernen. Das fünfte Kapitel befasst sich mit der Analyse der gelernten Funktionen sowie einer Reihe von Experimenten um deren Spielstärken, sowohl im Spiel gegeneinander als auch gegen menschliche Kontrahenten, zu ermitteln und zu vergleichen. Im abschließenden sechsten Abschnitt folgt eine Zusammenfassung und Begutachtung der Resultate.

## 1.3 Verwandte Arbeiten

Es gibt eine Reihe von Arbeiten, in denen sich einer Vielzahl von Methoden bedient wird um bestmögliche Evaluierungsfunktionen für das Schachspiel zu lernen.

So verwendet man in [12] und [1], sowie darauf aufbauend in [7] und [21], das sogenannte Temporal Difference Learning bei dem die Bewertungsfunktionen durch aktives Schachspiel gelernt und optimiert werden. An anderer Stelle [20] wird die Methode des Temporal Difference Learning verwendet um Bewertungsfunktionen offline aus hochklassigen Spielprotokollen zu lernen.

Ebenfalls gibt es eine Reihe von Ansätzen die genetische [23] [26] [27] oder evolutionäre [24] [14] [15] Algorithmen für die Suche nach guten Gewichten für einzelne Parameter oder auch komplexeren Bewertungsmustern [17] einsetzen.

Auch andere (Optimierungs-)Verfahren wie Simplex [2] oder Kendalls-$\tau$ [3] [4] wurden im Laufe der Jahre zum Lernen von Schach Evaluierungsfunktionen eingesetzt.

Die Methoden zur Reduzierung der Spielstärke oder der Anpassung an die Stärke des Gegenspielers beschränken sich jedoch in der Regel auf das Reduzieren der Suchtiefe oder der Bedenkzeit.

# Kapitel 2

# Grundlagen

Für das Verständnis dieser Arbeit sind einige Grundlagen zwingend notwendig. Diese werden in den folgenden Abschnitten erläutert. Bei entsprechenden Vorkenntnissen können die einzelnen Themen übersprungen werden.

## 2.1 Das Schachspiel

Das Schachspiel (von persisch: Schah, für König) wanderte, in immer wieder veränderter Form, ausgehend von Indien oder China, entlang der Seidenstrasse zu den Arabern und kam dann etwa im 9. Jahrhundert nach Europa. Dort gehörte es seit dem Beginn des 13. Jahrhunderts, neben Reiten, Schwimmen, Bogenschießen, Boxen, Jagen und Verse schreiben, zu den sieben Tugenden der Ritter. Auch in Europe entwickelte sich das Spiel weiter und gegen Ende des 16. Jahrhunderts kam es zu einer letzten großen Reform der Spielregeln. Zu diesem Zeitpunkt wurde das Spiel, welches wir heute unter dem Namen *Schach* kennen, geboren. [6]

Beim Schach stehen sich auf einem quadratischen Spielfeld mit 8 waagerechten Reihen und 8 vertikalen Linien, die „Armeen“ der beiden Spieler gegenüber. Die Figuren beider Seiten (je 8 Bauern, 2 Türme, 2 Springer, 2 Läufer, 1 Dame, 1 König) werden, wie in Abbildung 2.1 gezeigt, aufgestellt. Das Spielbrett wird

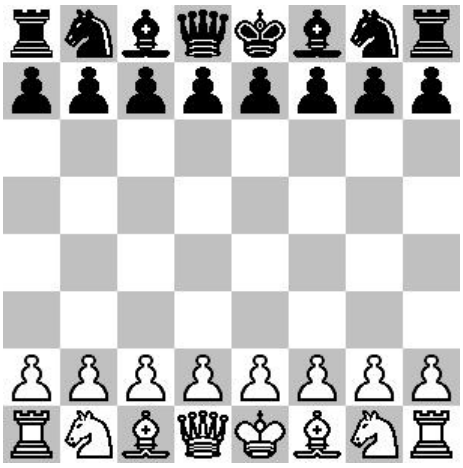


Abbildung 2.1: Schach Anfangsstellung

dabei so platziert, dass das unterste linke Feld beider Spieler schwarz gefärbt ist. Auf der untersten Reihe, der so genannten *Grundreihe* des jeweiligen Spielers, stehen nun die Türme auf den beiden Eckfeldern und anschließend von außen nach innen die Springer und Läufer. Auf den beiden mittleren Feldern findet man den König und die Dame. Die weiße Dame auf einem weißen Feld und ihr gegenüber die schwarze. Die acht Felder der zweiten Reihe werden allesamt von Bauern besetzt. Der Spieler der weißen Figuren beginnt die Partie mit dem ersten Zug. Im weiteren Verlauf der Partie müssen die Spieler abwechselnd eine ihrer Figuren bewegen, es besteht also ein Zug-Zwang.[1]

Die verschiedenen Figuren haben hierbei je nach Typ unterschiedliche Zugmöglichkeiten, genannt *Gangarten*, diese werden im weiteren Verlauf des Abschnittes beschrieben. Betritt eine Figur ein Feld in dem sich bereits ein Spielstein des Gegners befindet so wird dieser *geschlagen* und aus dem Spiel entfernt. Bedroht ein Spieler den König des Gegners, indem er eine seine Figuren in eine Position bewegt die es ihm erlauben würde den König im nächsten Zug zu schlagen so nennt man dies *Schach bieten*. Der Gegenspieler ist gezwungen die Drohung abzuwehren indem er entweder seinen König bewegt, mit einer anderen Figur den Weg des Angreifers blockiert oder die drohende Figur schlägt.

[1] FIDE Handbook E.I.01A. Laws of Chess

Das Ziel beider Spieler ist es den gegnerischen König so zu bedrohen, dass der Gegner keine Möglichkeit mehr hat das *schlagen* seines Königs mit einem regelkonformen Zug zu vermeiden. Der Spieler, der dies erreicht, hat den Gegner *mattgesetzt* und das Spiel gewonnen. Ergibt sich jedoch eine Situation in der keinem der beiden Spieler das *mattsetzen* mehr möglich ist so endet das Spiel unentschieden (*Remis*). Dies ist auch der Fall wenn einer der Spieler keinen legalen Zug mehr Ausführen kann, ohne das sein Gegner Schach geboten hat, oder aber wenn eine Stellung mit identischen Zugmöglichkeiten zum dritten Mal innerhalb der Partie eintritt.

Die weiße Seite befindet sich prinzipiell in der angenehmen Situation, dem Gegner das Spiel aufzwingen zu können. Weiß agiert und Schwarz reagiert. Durch gutes Spiel ist es dem Spieler der schwarzen Steine zwar möglich diesen so genannten *Anzugsvorteil* aufzuheben, doch werden etwa 54 Prozent aller Spiele von der weißen Seite gewonnen, zählt man die Unentschiedenen nicht dazu. Einige Spieler sehen diesen Unterschied als so schwerwiegend an, dass sie mit Weiß auf den Sieg und mit Schwarz von Anfang an auf ein Remis hinarbeiten.

***Der Bauer:*** ♙

Der Bauer kann sich pro Zug nur um ein einziges Feld nach vorne bewegen. Es ist ihm nicht möglich zurück zu weichen. Befindet sich der Bauer in seiner Ausgangsstellung, auf der 2. b.z.w 7. Reihe, so ist es ihm einmalig gestattet zwei Felder vorwärts zu ziehen. Eine weitere Besonderheit des Bauern ist es das er Figuren nur auf den beiden Feldern diagonal vor sich schlagen kann, damit ist er der einzige Spielstein der in eine andere Richtung als seine Zugrichtung schlägt. Auch wenn in den offiziellen Regeln nicht zwischen Bauer und Figur unterschieden wird so wird der Bauer umgangssprachlich häufig nicht als Figur bezeichnet.

***Der Springer:*** ♘

Der Springer bewegt sich zwei Felder in der senkrechten oder waagerechten und dann ein Feld im rechten Winkel dazu. Oder anders formuliert erst ein

Feld geradeaus und dann eines diagonal in dieselbe Richtung. Er darf dabei als einzige Schachfigur andere Steine überspringen und daher kann keine Figur zum Schutz gegen einen Springerangriff in den Weg gezogen werden.

***Der Läufer:*** ♗

Die Läufer ziehen und schlagen diagonal beliebig weit. Sie können daher keine Felder betreten die eine andere Farbe haben als ihr Ausgangsfeld. Man spricht deshalb auch vom weißfeldrigen und vom schwarzfeldrigen Läufer.

***Der Turm:*** ♖

Die Türme bewegen sich senkrecht oder waagerecht beliebig weit. Sie sind zu Beginn der Partie nur sehr begrenzt einsetzbar doch gewinnen sie an Stärke umso leerer das Spielfeld wird. Der Turm ist neben der Dame die einzige Figur die im Endspiel nur mit Unterstützung des eigenen Königs die Partie gewinnen kann.

***Die Dame:*** ♕

Die Dame vereint in sich die Wirkung eines Turmes und eines Läufers. Sie darf also in jede Richtung (horizontal, vertikal und diagonal) beliebig weit ziehen und schlagen. Die Dame ist damit die stärkste Figur des Spieles und sollte nicht leichtfertig riskiert werden.

***Der König:*** ♔

Der König darf auf jedes seiner Nachbarfelder ziehen und schlagen. Er darf jedoch keine Felder betreten die durch einen Stein des Gegners bedroht werden. Beide Könige können daher niemals nebeneinander stehen.

***Besondere Züge:***

*Bauernumwandlung:* Wenn ein Bauer die gegnerische Grundreihe erreicht so wird er sofort in eine andere Figur mit Ausnahme des Königs umgewandelt. Die Eigenschaften der neuen Figur treten augenblicklich in Kraft so, dass durch die Umwandlung ein Schach oder gar Schachmatt herbeigeführt werden kann.

*En passant:* Eine weitere Besonderheit des Bauern ist das Schlagen „im Vorbeigehen“ (franz. *en passant*). Dies Bezeichnet die Möglichkeit einen gegnerischen Bauern auch dann zu schlagen, wenn dieser von der Ausgangsstellung heraus durch einen Doppelschritt über den Schlagbereich des eigenen Bauern hinaus gezogen ist. Der Zug wird dann so behandelt als hätte der Bauer nur einen einzelnen Schritt nach vorne getan und wäre auf diesem Feld geschlagen worden. Das Schlagen *en passant* ist nur unmittelbar nach dem Doppelschritt des gegnerischen Bauern zulässig. Wird diese Gelegenheit nicht wahrgenommen so verfällt sie.

*Rochade:* Die Rochade erlaubt es als einziger Zug zwei Figuren zugleich zu Bewegen. Bei einer Rochade zieht der König zwei Felder in Richtung eines der beiden Türme und der jeweilige Turm springt auf das Feld welches der König überquert hat. Jede Seite hat also zwei Möglichkeiten zu *rochieren*, zum einen die kurze Rochade nach c1 b.z.w c8 und die lange Rochade nach g1/g8. Damit eine Rochade zulässig ist muss jedoch eine Reihe von Bedingungen erfüllt sein.

- Der Weg muss frei sein
- Der König und der Turm dürfen im Verlauf des Spieles noch nicht bewegt worden sein
- Der König darf vor und nach Ausführung der Rochade nicht im Schach stehen
- Das Feld das der König überschreitet darf nicht bedroht sein

## 2.2 Spielprotokolle

Obwohl das Schachspiel schon seit langer Zeit Gegenstand vieler Analysen - auch in schriftlicher Form - ist, dauerte es sehr lange bis sich eine einfache und effiziente Notation fand um die einzelnen Spielzüge einer Partie zu beschreiben. In der Mitte des 18. Jahrhunderts findet die heute verwendete algebraische Notation zum ersten mal Verwendung, und es dauerte bis in die 1980er Jahre bis sie sich weltweit durchgesetzt hatte.

Bei der Beschreibung eines Spieles werden jeweils eine weiße und eine schwarze Figurenbewegung zu einem Zug zusammengefasst. Die Aktionen der einzelnen Spieler nennt man einen Halbzug (engl. *ply*). Den Feldern werden, aus Sicht des Spielers mit den weißen Figuren, eindeutige Koordinaten zugeordnet. Dabei werden ausgehend von der linken unteren Ecke die Linien mit den Buchstaben a bis h (links nach rechts) und die Reihen mit den Zahlen 1 bis 8 (unten nach oben) gekennzeichnet.

Ein Zug wird durch die Angabe des bewegten Spielsteines sowie des Ausgangs- und Zielfeldes beschrieben. Die unterschiedlichen Figuren werden hierbei durch einzelne Großbuchstaben dargestellt (K = King/König, Q = Queen/Dame, R = Rook/Turm, N = Knight/Springer, B = Bishop/Läufer). Der Bauer wird durch das fehlen eines Buchstaben gekennzeichnet. Start- und Zielfeld werden durch einen Bindestrich `-` getrennt, dieser wird durch ein `x` ersetzt wenn auf dem Zielfeld eine Figur geschlagen wird. Eine kurze Rochade wird als `0-0` notiert und eine lange als `0-0-0`. Wird dem Gegner Schach geboten so symbolisiert man dies durch ein `+` am Ende des Zuges. Führt der Zug gar zu einem Schachmatt so endet seine Beschreibung mit einem `#`. Bei einer Bauernumwandlung wird der Buchstabe der neuen Figur hinter dem Zug angegeben. Teilweise wird eine Umwandlung noch durch ein zusätzliches Gleichheitszeichen `=` gekennzeichnet.

Einige Beispielzüge die für den Spieler der weißen Figuren auf dem Brett in Abbildung 2.2 möglich wären sind:
`Ng4-e3` - Springer von g4 nach e3
`Ng4xe5` - Springer von g4 schlägt Turm auf e5
`e7-e8Q+` - Bauer von e7 nach e8 und Umwandlung in eine Königin. Damit wird dem schwarzen König Schach geboten

Bei der Verarbeitung durch Computer ist sowohl für einzelne Partien als auch für größere Datenbanken die Portable-Game-Notation (PGN) weit verbreitet. Dieses Format basiert auf der verkürzten algebraischen Notation bei welcher das Ausgangsfeld und bei nicht Schlagzügen auch der Bindestrich

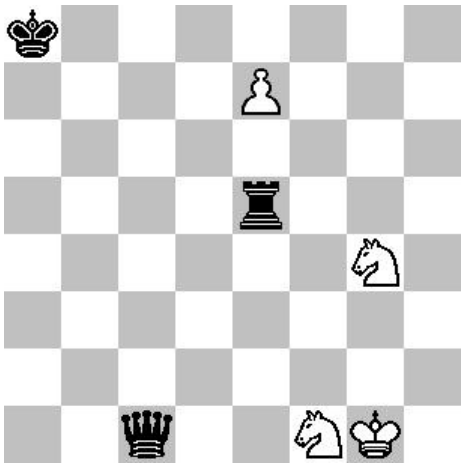


Abbildung 2.2: Beispiele für die Zugnotation

weggelassen wird. Ist ein Zug dadurch nicht mehr eindeutig nachzuvollziehen so wird die Beschreibung durch zusätzliche Angaben zur Reihe oder Linie des Ausgangsfeldes ergänzt. Die Eindeutigkeit eines Zuges kann sich dabei auch aus dem Kontext ergeben, wenn beispielsweise eine der infrage kommenden Figuren echt *gefesselt* ist, das heißt, dass bei Verlassen ihrer aktuellen Position der König geschlagen werden könnte. Dieselben Züge wie im vorherigen Beispiel sehen in dieser verkürzten Notation wie folgt aus: `Ne3`, `Nxe5`, `eQ+`. Der erste Zug ist eindeutig da der Springer auf f1 gefesselt ist und nicht ziehen darf.

## 2.3 ELO-Wertung

Das so genannte ELO-System wurde, aufbauend auf anderen bereits verwendeten Wertungssystemen, 1960 vom Statistiker und Physiker Prof. Arpad Elo im Auftrag des amerikanischen Schachverbandes USCF[2] entwickelt und einige Jahre später auch vom Weltschachverband FIDE[3] übernommen. Das System beruht auf der Annahme, dass die Spielstärke aller Spieler normalverteilt ist. Es ermöglicht statistische Vorhersagen über den Ausgang einer Partei indem die Wahrscheinlichkeit, mit der ein Spieler gegen einen anderen gewinnt, berechnet wird.

[2]United States Chess Federation

[3]Fédération Internationale des Échecs (franz. Internationale Schachföderation)

Zu beachten ist hierbei, dass die ELO-Zahl eines Spielers immer auch von der Spielstärke anderen in seiner Spielergemeinschaft abhängt. Daher sind die Wertungen nicht direkt miteinander zu Vergleichen und es wird zwischen nationalen und internationalen ELO-Zahlen unterschieden. So hat Beispielsweise jemand mit einer FIDE-Wertung von 2500 eine USCF-Zahl von etwa 2600 und kann auf einem Internet Schachserver wiederum eine abweichende Wertung haben. Gerade die Möglichkeit des Online-Schach wird gerne genutzt um Strategien auszuprobieren die man in einem Turnier nicht wagen würde.

Mit Hilfe des ELO-Systems können Schachspieler grob in verschiedene Klassen eingeteilt werden. Die Tabelle 2.1 enthält hierbei eine Mischung aus offiziellen USCF b.z.w. FIDE Einstufungen. Die durchschnittliche Stärke aller Menschen die die Regeln des Schachspieles beherrschen wird auf etwa 800 geschätzt. [5]

| ELO-Zahl | Klassifikation |
|---|---|
| $> 2700$ | „Super-Großmeister“ |
| 2500-2699 | Großmeister |
| 2400-2499 | Internationaler Meister |
| 2200-2399 | Nationale Meister |
| 2000-2199 | Meisteranwärter |
| 1800-1999 | sehr gute Vereinsspieler |
| 1600-1799 | starker Freizeitspieler |
| 1400-1599 | überdurchschnittlicher Spieler |
| 1200-1399 | durchschnittlicher Hobbyspieler |
| 1000-1199 | Gelegenheitsspieler |
| $< 1000$ | Anfänger |

Tabelle 2.1: Spielstärken nach ELO

Das ELO-System funktioniert derart, dass Anhand der Wertungsdifferenz zwischen beiden Spielern ein Erwartungswert für den Ausgang der Partie(n) bestimmt wird. Weicht das Ergebnis von dieser Erwartung ab so werden die Wertungszahlen der beiden Spieler entsprechend modifiziert. Durch diese schrittweisen Veränderung erhält jeder Spieler mit der Zeit eine Wertungszahl welche in etwa seine reale Spielstärke widerspiegelt.

In Formeln ausgedrückt sieht dies folgendermaßen aus: $E_A$ ist die Gewinnerwartung für Spieler A, sie berechnet sich aus $R_A$ und $R_B$, den ELO-Zahlen der beiden Spieler. Werden mehrere Partien gespielt multipliziert man diesen Erwartungswert mit der Anzahl der Spiele um eine Vorhersage für den Punktestand zu erhalten.

$$E_A = \frac{1}{1+10^{(R_B-R_A)/400}}$$

In der Regel umfasst eine Klasse eine Spanne von 200 ELO-Punkten. Eine solche Differenz führt zu einem Erwartungswert von 0,7597 für den Stärkeren und 0,2403 für den Schwächeren der beiden Spieler. Sollten diese beiden Spieler nun beispielsweise einen Wettkampf über 10 Partien austragen so würde das erwartete Ergebnis 7,5/2,5 lauten.

$$\begin{aligned} E_A &= \frac{1}{1+10^{(-200)/400}} = \frac{1}{1+0{,}31623} = 0,7597 \\ E_B &= \frac{1}{1+10^{(200)/400}} = \frac{1}{1+3{,}1623} = 0,2403 \end{aligned}$$

Nach dem Spiel wird die Wertung beider Spieler entsprechend des tatsächlich erzielten Punktestandes $S$ angepasst. Der Wert $k$ bestimmt hierbei wie stark die Veränderung der Wertung ausfällt. Für einen neuen Spieler verwendet man bis zu seinem 30ten gewerteten Spiel üblicherweise einen Wert von k = 25, dies hat den Sinn die ELO-Zahl schnell in die für diesen Spieler passende Region zu bringen. Unterhalb von 2400 ELO Punkten wird anschließend ein Wert von k = 15 verwendet, oberhalb davon k = 10.

$$\begin{aligned} R'_A &= R_A + k \cdot (S - E_A) \\ R'_B &= R_B + k \cdot (S - E_B) \end{aligned}$$

Problematisch ist, dass die offiziellen FIDE ELO-Zahlen lange Zeit eine Untergrenze von 2000 Punkten hatten. Fiel eine Wertung unter diesen Grenzwert wurde sie gelöscht. Erst seit 2002 wurden Zahlen bis 1800 gewertet und aktuell liegt die Untergrenze bei 1400. Für die in dieser Arbeit betrachteten Wertungsbereiche musste daher auf nationale Wertungszahlen zurückgegriffen werden. Diese werden zwar grundsätzlich alle nach dem selben System bestimmt, doch werden unterschiedliche Werte für $k$ verwendet. Auch sind sie dadurch, dass jedes dieser Systeme auf einer anderen Spielermenge arbeitet, nicht 1 zu 1 mit einander zu vergleichen.

## 2.4 Spieltheorie und Komplexität

In der Spieltheorie nach John von Neumann [25] werden Spiele, bei denen nur ein Sieg, ein Unentschieden oder eine Niederlage als mögliche Spielausgänge zulässig sind als Nullsummenspiele bezeichnet. Nullsumme weil eine Niederlage automatisch zum Sieg des Gegners führt und die Summe der auf beide Spieler verteilten Punkte daher konstant ist. Existieren des Weiteren keine zufälligen oder verdeckten Elemente, wie etwa die Verwendung von Würfeln, verdeckte Karten oder gleichzeitige Züge beider Seiten, so spricht man von einem *Nullsummenspiel mit perfekter Information*.

Für diese Klasse von Spielen, zu der neben dem Schach Spiele wie Shogi, Vier Gewinnt, Tic Tac Toe, Dame, Go, Halma oder Mühle gehören, lässt sich zumindest theoretisch immer eine berechenbare Gewinnstrategie ermitteln. Auf das Schachspiel bezogen lässt sich also, unter der Annahme das keiner der beiden Spieler einen Fehler begeht, für jede mögliche Stellung der Ausgang der Partie ermitteln. Ausgehend von der Startaufstellung ist es also möglich zu bestimmen ob Weiß oder Schwarz gewinnt oder aber ob solch eine perfekte Schachpartie immer mit einem Unentschieden enden würde. Hierzu muss lediglich mit dem so genannten MiniMax-Algorithmus (2.5.1) die als Spielbaum bezeichnete Darstellung aller möglichen Abfolgen von Spielzügen ausgewertet werden. Dieser Algorithmus wird in Abschnitt 2.5.1 beschrieben.

Hier jedoch liegt das Problem; der Spielbaum des Schachspieles ist viel zu groß um berechnet zu werden. Sein durchschnittlicher Verzweigungsgrad (*branching factor*) pro Halbzug beträgt etwa 30, wobei der größte bisher bekannte Wert bei 218 möglichen Nachfolgestellungen liegt. Insgesamt wären für das Lösen des Schachspieles etwa $10^{120}$ Knoten notwendig [8]. Allein für das Berechnen der nächsten 5 vollständigen Züge (10 Halbzüge) müssen über 590 Billionen[4] Stellungen ausgewertet werden. Zu beachten ist hierbei zwar, dass sich viele Stellungen wiederholen und einige Abschnitte des Baumes daher mehrfach verwendet werden können. Die Zahl der möglichen legalen Stellungen beläuft

[4] 590 490 000 000 000

sich nach N. Petrovic[5] aber immer noch auf $2,28 * 10^{46}$. Allein das Speichern dieser Menge an Informationen übersteigt die heute technischen Möglichkeiten bei Weitem, und es ist nicht zu erwarten, dass sich dies in näherer Zukunft ändern wird. Daher muss das Problem des schachspielenden Computers mit anderen Mitteln angegangen werden.

## 2.5 Computerschach

Meisterhaftes Schachspiel wird seit jeher als Zeichen großer Intelligenz verstanden. So besagt etwa die so genannte Levitt-Gleichung [18], dass die ELO-Zahl die jemand, einige Jahre schachlicher Anstrengung vorausgesetzt, erreichen kann in etwa dem zehnfachen seines IQ plus 1000 entspricht. Daher wurde das Schachspiel zu einem beliebten Forschungsobjekt bezüglich künstlicher Intelligenz. Zwar haben sich die Hoffnungen aus diesem Fachgebiet tief greifende Erkenntnisse über die Funktion des menschlichen Denkens zu erhalten nicht bewahrheitet, doch ist es im Laufe der Jahre gelungen die Leistungen der Menschen in einigen Bereichen, wie etwa dem Schachspiel, zu übertreffen.

Allerdings heißt dies nicht, dass Computer, wie der Mensch, kreativ nachdenken und Probleme lösen können. Vielmehr beruht ihre Leistungsfähigkeit auf der Verwendung vorgegebener Methoden mit denen intelligentes Verhalten simuliert wird. In der scheinbaren Intelligenz des Computers spiegelt sich also vielmehr die Intelligenz derjenigen welche an der Gestaltung und Verbesserung seiner Algorithmen mitgewirkt haben. Und was dem Computer an Erfahrung und Spielverständniss fehlt, kann er inzwischen bei Weitem durch überlegene Rechenleistung wettmachen.

> Genie ist ein Prozent Inspiration und neunundneunzig Prozent Transpiration.
> - Thomas Alva Edison

[5] 1948 in Sahovski Vjesnik

Die Erfolgsgeschichte des Computerschach begann in den 50er Jahren, als durch die schnell wachsende Leistungsfähigkeit der Digitalcomputer die Menge der notwendigen Berechnungen in den Bereich des Möglichen kam. Die meisten grundlegenden Konzepte waren bereits in den Jahren davor erfunden worden. So stellte Claude Shannon [28] den grundlegenden Aufbau eines Schachprogrammes sowie die Anwendung des MiniMax-Algorithmus für die Bestimmung des besten Zuges vor.

Ein Schachprogramm besteht in der Regel aus drei Komponenten:
Einem Zuggenerator welcher, ausgehend von einer Stellung auf dem Brett, eine Liste aller zulässigen Spielzüge erzeugt. Dies ist der trivialste Teil und daher wird hier im Folgenden nicht weiter darauf eingegangen werden. Die beiden anderen Elemente sind ein Programmteil zur Steuerung der Suche und zur Auswahl des nächsten Zuges sowie die Bewertungsfunktion.

### 2.5.1 MiniMax-Suche

> Es ist im Leben wie im Schachspiel: Wir entwerfen einen Plan; dieser bleibt jedoch bedingt durch das, was im Schachspiel dem Gegner, im Leben dem Schicksal zu tun belieben wird.
> - Arthur Schopenhauer

Der MiniMax-Algorithmus beruht auf dem Prinzip, dass jeder Spieler für sich selbst das bestmögliche Ergebnis erzielen will. Im Folgenden signalisiert eine hohe Bewertung einer Spielstellung, dass die weiße Seite im Vorteil ist, während eine negative Wertung einen Vorteil für Schwarz darstellt. Für den weißen Spieler bedeutet die Auswahl des *besten* Zug also dass er jene Option wählt deren Ergebnis maximal positiv ist. Andererseits wird der Gegenspieler ebenfalls versuchen seine Chancen zu optimieren und eine möglichst niedrig bewertete Stellung zu erreichen.

Entspricht die aktuelle Spielposition der Wurzel eines Baumes und jeder mögliche Spielzug einer Verzweigung so kann man die einzelnen Ebenen jeweils dem Spieler zuordnen der dort am Zug ist. Bei erreichen der maximalen Suchtiefe werden die Stellungen in den Blattknoten durch die Bewertungsfunktion evaluiert. Anschließend werden diese Ergebnisse an die höheren Ebenen weitergereicht und dort aus der Sicht des Spielers beurteilt der am Zug ist. Es wird also abwechselnd Minimiert und Maximiert.

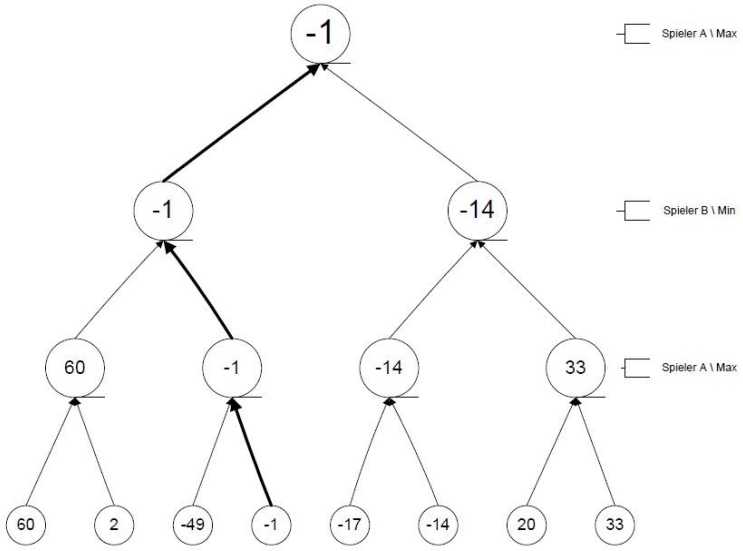



Abbildung 2.3: MiniMax-Suche

Das folgende Beispiel soll diesen Ablauf verdeutlichen: Der maximierende Spieler A würde gerne den Blattknoten mit der Wertung von +60 erreichen. Wenn er diesen Weg einschlägt wird der minimierende Spieler B ihn jedoch mit seinem Gegenzug in eine Position bringen aus welcher er bestenfalls noch eine Wertung von -1 erreichen kann. Folgt er der zweiten Alternative so hat auch hier der Gegenspieler die Möglichkeit ihm eine positive Entwicklung zu verwehren. Das beste Ergebnis das aus Sicht von Spieler A innerhalb der nächsten drei Halbzüge erreicht werden kann ist also die -1.

**Algorithm 1** MiniMax-Algorithmus in der NegaMax-Variante

```
int NegaMax(Stellung s, int tiefe)
{
        IF (tiefe == 0 OR Blatt(s)) RETURN Bewertung(s);

        int best = INT_MIN;

        Finde Nachfolger s_1,...,s_x von s mit Zuggenerator

        for (int i = 1; i <= x; i++)
        {
                best = MAX(best, -NegaMax(s_i, tiefe-1);
        }
        return best;
}
```

Der MiniMax-Algorithmus kann relativ einfach durch eine rekursive Funktion implementiert werden, welche den Baum in einer Tiefensuche durchläuft. Eine weitere Vereinfachung des Codes lässt sich mit der so genannten *NegaMax*-Variante erreichen. Dabei wird nicht mehr zwischen Maximierungs- und Minimierungsebenen unterschieden. Statt dessen wird die Bewertungsfunktion so modifiziert das eine positive Bewertung immer vorteilhaft für den Spieler ist der auf der aktuell ausgewerteten Ebene am Zug ist. So kann die Wertung der Folgestellung negiert übernommen werden und es ist möglich auf jeder Ebene den maximalen Nachfolger zu wählen. [Algorithm 1] beschreibt den NegaMax-Algorithmus in Pseudocode. *Blatt(s)* ist hierbei wahr wenn in der betrachteten Stellung keine nachfolgenden Zugmöglichkeiten bestehen. *Bewertung(s)* evaluiert die Stellung aus Sicht des Spielers der auf der aktuellen Ebene am Zug ist.

Die Idee des MiniMax-Algorithmus ist vom eigentlichen Spiel unabhängig und lässt sich für alle Spiele aus der Klasse der Nullsummenspiele mit perfekter Information verwenden. Es müssen hierzu lediglich der Operator für die Bewertung der Blätter und der Zuggenerator für die Verzweigungen angepasst werden. Der Aufwand der Suche wird bestimmt durch die Suchtiefe $d$ sowie die durchschnittliche Anzahl an Zugmöglichkeiten pro Stellung (engl. *branching factor*) $b$. Es ergibt sich eine Komplexität von $O(b^d)$.

### 2.5.2 Die Bewertungsfunktion

Da es, wie in Abschnitt 2.4 erläutert, in der Regel nicht möglich ist alle aus einer Spielposition möglichen Zugfolgen bis zu ihrem Ende zu verfolgen, wird eine Funktion benötigt die die Stellung auf dem Spielbrett danach bewertet ob sie für eine der beiden Parteien vorteilhaft oder nachteilig ist. Eine solche Bewertungsfunktion setzt sich in der Regel aus einer materiellen und einer positionellen Komponente zusammen.

Für die materielle Komponente werden die Werte der auf dem Brett befindlichen eigenen Spielsteine addiert und die Figurenwerte des Gegners subtrahiert. Dies liefert bereits eine grundlegende Aussage darüber ob sich die eigene Seite (positives Ergebnis) oder die des Gegners (negatives Ergebnis) im Vorteil befindet. Der König wird hierbei durch eine ausreichend hohe Zahl dargestellt da sein Verlust das Spiel beendet. In der Regel wird anhand der materiellen Bewertung auch festgelegt in welcher Phase sich das Spiel befindet. Danach richtet sich dann die weitere Analyse der Spielsituation.

Da die Stärke der einzelnen Spielfiguren auch von ihrer Position auf dem Spielfeld abhängt, wird die materielle Komponente durch eine positionelle ergänzt. Diese berücksichtigt die Positionen auf dem Spielfeld und die Stellung der Figuren zueinander durch Parameter für die Bauernstrukturen oder die Königssicherheit. Diese werden aus der vorherrschenden Stellung extrahiert und fließen entsprechend ihrer Gewichtung in die Bewertung ein.

Details zu der in dieser Arbeit verwendeten Bewertungsfunktion finden sich in Abschnitt 2.7.

### 2.5.3 AlphaBeta-Verfahren

Die Größe des Spielbaumes wächst, wie bereits beschrieben, exponentiell schnell und der größte Teil davon ($> 99\%$) ist für die aktuelle Suche vollkom-

men uninteressant [6]. Daher hat man sich schon früh darauf fokussiert, Wege zu finden um die Anzahl der zu untersuchenden Knoten zu reduzieren. Als dann das AlphaBeta-Pruning [7] entwickelt wurde, brachte dies einen gewaltigen Leistungsschub. Dieses Verfahren ist durchaus vergleichbar mit dem Verhalten von menschlichen Spielern, denn es werden offensichtlich unsinnige Züge nicht weiter verfolgt. Doch wo menschliche Großmeister auf ihre Erfahrung zurückgreifen benötigt der Computer Hilfskonstruktionen um es ihnen gleichzutun.

Das Alpha-Beta Verfahren funktioniert derart, dass ein Suchfenster eingeführt wird welches durch die Werte Alpha und Beta begrenzt wird. Der Alpha-Wert ist hierbei das Ergebnis, das der maximierende Spieler mindestens erreichen kann und der Beta-Wert enthält das Mindestergebnis für den minimierenden Spieler.

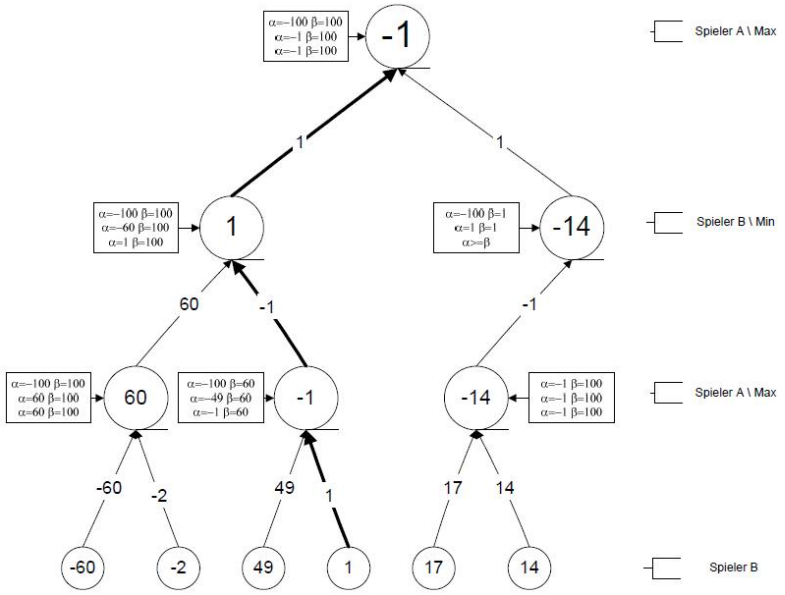



Abbildung 2.4: NegaMax Suche mit AlphaBeta Pruning

[6] http://de.wikipedia.org/wiki/Alpha-Beta-Suche
[7] pruning = engl. Beschneiden von Bäumen und Büschen

Wird nun an einen Knoten auf der maximierenden Ebene ein Ergebnis zurückgegeben, welches größer oder gleich dem Beta-Wert ist, so kann die Suche an diesem Knoten abgebrochen werden, dies bezeichnet man als *cuttoff*. Der auf diese Weise abgeschnittene Unterbaum muss nicht weiter untersucht werden, da er keinen Einfluss auf die Suche haben kann. Denn da das Mindestergebnis des minimierenden Spielers unterschritten wurde wird dieser ohnehin nicht zulassen, dass diese Zugfolge gespielt wird. Dasselbe gilt wenn ein Rückgabewert den Alpha-Wert einstellt oder unterschreitet. In diesem Fall würde der maximierende Spieler diesen Spielverlauf nicht zulassen da er auf einem anderen Weg ein mindestens genauso gutes Ergebnis erzielen kann. Entsprechend werden Alpha und Beta modifiziert wenn auf einer maximierenden Ebene der Alpha-Wert überschritten b.z.w. auf einer minimierenden der Beta-Wert unterschritten wird.

In der NegaMax-Variante [Algorithm 2] werden die Alpha und Beta-Werte bei der Übergabe an die nächste Ebene negiert und vertauscht. Dies ist notwendig da in dieser Variante nicht zwischen Minimierungs- und Maximierungsebene unterschieden wird.

**Algorithm 2** NegaMax-Algorithmus in der AlphaBeta-Variante

```
int NegaMaxAlphaBeta(Stellung s, int tiefe, int alpha, int beta) {
{
        IF (tiefe == 0 OR Blatt(s)) RETURN Bewertung(s);

        Finde Nachfolger s_1,...,s_x von s mit Zuggenerator

        for (int i = 1; i <= x; i++)
        {
                alpha = MAX(alpha, -NegaMaxAlphaBeta(s_i, tiefe-1, -beta, -alpha));

                IF (alpha >= beta) return alpha; //Cutoff
        }
        return alpha;
}
```

Der Aufwand für eine Alpha-Beta-Suche hängt davon ab wie häufig und auf welcher Ebene die cuttoffs auftreten. Wenn auf jeder Ebene der für den aktuelle Spieler beste Zug zuerst untersucht wird (*best case*) so reduziert sich die Komplexität auf $O(b^{d/2})$ [16]. Ist die Reihenfolge der Züge rein zufällig (*average case*) so beläuft sich der Aufwand auf $O(b^{2d/3})$. Im schlechtest

möglichen Fall entspricht die Suche dem kompletten Traversieren des Baumes, wie bei einer unmodifizierten MiniMax-Suche, hat also eine Laufzeit von $O(b^d)$. Die Verwendung der Alpha-Beta Variante ist daher mit keinen Risiken verbunden und liefert in der Regel eine exponentielle Beschleunigung der Suche. Dies erlaubt es in der gleichen Zeit um einiges tiefer zu suchen als zuvor.

### 2.5.4 Ruhesuche

Ein großes Problem bei der Anwendung der MiniMax-Suche war lange Zeit der so genannte *Horizont-Effekt*. Dieser Ausdruck bezeichnet die Eigenart von Schachprogrammen, unerwünschte aber eigentlich unvermeidliche Situationen, wie etwa den Verlust einer wichtigen Figur, durch Zwischenzüge soweit hinauszuzögern, dass sie sich jenseits der maximalen betrachteten Suchtiefe befinden. So werden sie durch das Programm vorerst nicht mehr erkannt da sie hinter dem Horizont liegen bis zu welchem sich die Suche erstreckt. Im schlimmsten Falle ist das Programm sogar bereit eine weniger wertvolle Figur zu opfern, um den Verlust scheinbar abzuwenden. Das wichtigste Hilfsmittel gegen dieses Problem entspringt der Grundlagenforschung von Turing und Shannon [28].

Es handelt sich um die Einführung der Ruhesuche (*Quiescence-Search*). Dabei wird der Spielbaum, von *unruhigen* Blättern aus, über die eigentlich gewünschte Suchtiefe hinaus weiter traversiert. Jedoch werden ab diesem Punkt nur noch mögliche Schlagkombinationen sowie sich daraus ergebende Mattangriffe ausgewertet. Sobald eine Stellung erreicht ist in der es keine Schlagzüge mehr gibt so wird diese als *ruhig* (oder auch tot) bezeichnet und normal bewertet. Diese zusätzliche Expansion des Baumes stellt einen großen Mehraufwand (15-20%) dar, da sie für alle Knoten am Suchhorizont durchgeführt werden muß. Gerade bei geringeren Suchtiefen ist diese Arbeit jedoch unbedingt zu leisten da die Ergebnisse sonst sehr stark fehlerbehaftet sind. Bei den in heutigen Hochleistungsschachprogrammen üblichen Suchtiefen von 13 bis 15 Halbzügen verliert der Horizont-Effekt jedoch zunehmend an Bedeutung, da dort oft schon die durch Ausnutzung positoneller Vorteile erreichten materiellen Gewinne deutlich werden. [8]

[8] http://www.stmintz.com/ccc/index.php?id=198940

## 2.6 XBoard/Winboard und ICS

Schachprogramme, auch *Schach-Engines* genannt, sind in der Regel so ausgelegt, dass man ihnen eine Spielstellung und je nach Programm weitere Parameter übergibt und von ihnen als Rückgabewert einen Spielzug erhält. Für die graphische Darstellung des Spielfeldes und die Interaktion mit dem Benutzer bedient man sich meist eines zusätzlichen Tools welches man als *Schach-Frontend* oder auch *Schach-Interface* bezeichnet. Durch diese Trennung ist es einfach möglich die verwendete Engine auszutauschen und die Schachprogramme an sich können übersichtlicher programmiert werden.

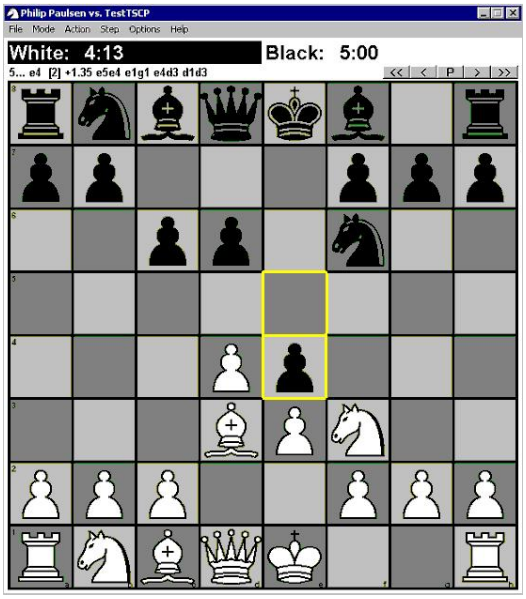



Abbildung 2.5: Grafische Benutzeroberfläche - WinBoard

Als Schnittstelle zwischen Engine und Frontend werden spezielle Protokolle verwendet. Eines der Standardprotokolle nennt sich *XBoard*. Es verbindet eine kompatible Engine mit dem gleichnamigen XBoard Frontend. Zusätzlich steuert das XBoard Protokoll[9] die Übertragung der Daten zum Gegenspieler und erlaubt es über das Internet gegen andere Spieler oder auch gegen Schachprogramme zu spielen. Für Windows existiert unter dem Namen *Winboard* eine äquivalente Portierung.

[9] http://www.tim-mann.org/xboard/engine-intf.html

Vermittelt werden Spiele im Internet in der Regel über einen ICS *(Internet Chess Server)*. Dort kann man Nachrichten austauschen, Gegner suchen oder herausfordern und sich als Beobachter in laufende Partien einklinken.

## 2.7 Das Schachprogramm

Als Grundlage für die Bearbeitung der gestellten Aufgabe war ein Schachprogramm notwendig welches anschließend an einigen Stellen modifiziert und erweitert wurde. Die Wahl fiel auf TSCP 1.81 [10] von Tom Kerrigan welches der Autor freundlicherweise für diese Arbeit zur Verfügung stellte. TSCP zeichnet sich aus durch seine klare und gut kommentierte C-Programmierung, seine Kompatibilität mit dem XBoard/Winboard Protokoll sowie durch die Implementierung der heute in Schachprogrammen üblichen Mechanismen wie MiniMax-Suche mit Alpha-Beta Pruning und Ruhesuche. Das Basisprogramm spielt nach den Angaben zweier Ranking Listen für Schachprogramme[11] [12] mit einer Stärke von etwa 1700 ELO.

| **TSCP Originalwerte** | **Sonderwerte:** |
|---|---|
| | 10 Doppelbauer |
| **Figurengewichte:** | 20 Isolierter Bauer |
| Bauer: 100 | 8 Rückständiger Bauer |
| Pferd: 300 | 20 Freibauer |
| Läufer: 300 | 15 Offene Linie |
| Turm: 500 | 10 Halboffene Linie |
| Dame: 900 | 20 Turm auf 7ter Reihe |

Tabelle 2.2: Original Figuren- und Sonderwerte

Die Bewertung einer Position verläuft bei TSCP nach folgendem Schema. Zuerst werden bei einem Durchlauf über alle Felder des Schachbrettes die Summen der Figurenwerte beider Seiten bestimmt. Des Weiteren wird für

[10] tscp kann man unter http://www.tckerrigan.com/Chess/TSCP/ herunterladen
[11] http://www.computerchess.org.uk/ccrl/404/index.html
[12] http://wbec-ridderkerk.nl/html/BayesianElo_ed15.htm

jede Linie die Reihe des am weitesten vorgedrungenen Bauern festgehalten. Liegt die Summe der Figurengewichte des Gegners bei 1200 oder darunter so wird angenommen das sich das Spiel in der Endphase befindet. In einem zweiten Durchlauf werden nun die Werte der Bauern hinzu addiert und es wird eine Analyse der einzelnen Sonderfälle durchgeführt welche gegebenenfalls einen Bonus oder Malus zur Folge haben. Die in der Grundversion von TSCP verwendeten Werte für Figuren und Sonderwerte finden sich in der Tabelle 2.2. Die einzelnen Sonderfälle werden im folgenden kurz erläutert.

*Doppelbauer:* Befinden sich zwei Bauern einer Seite in derselben Linie behindern sie sich gegenseitig in ihrem Bewegungsspielraum und können einander keine Deckung geben. Daher wird eine solche Situation mit Punktabzug bestraft.

*Isolierter Bauer:* Befindet sich auf keiner der benachbarten Linie ein weiterer Bauer der eigenen Seite, welcher Deckung bieten könnte, so führt dies ebenfalls zu einem Abzug

*Rückständiger Bauer:* Befindet sich ein Bauer hinter den Bauern auf den benachbarten Linien und kann somit nicht von diesen gedeckt werden so führt dies ebenfalls zu einem Malus.

*Freibauer:* Hat ein Bauer weder auf der eigenen noch auf den beiden benachbarten Linien einen Bauern des Gegners vor sich und kann so nur noch durch den Einsatz von Figuren geschlagen werden, so wird dieser Vorteil mit einem Bonus belohnt.

*Offene Linie:* Wird ein Turm durch keinen Bauern auf seiner aktuellen Linie aufgehalten und kann sich frei Bewegen wird dies ebenfalls belohnt.

*Halboffene Linie:* Wird ein Turm nicht mehr durch einen eigene Bauern auf der aktuellen Linie behindert so fällt der Bonus etwas geringer aus. Durch das schlagen des gegnerischen Bauern kann aber schnell eine offene Linie geschaffen werden.

*Turm auf 7ter Reihe:* Hat ein Turm die zweite Reihe des Gegners, die 7te Reihe aus der eigenen Sicht, erreicht wird dies ebenfalls mit Bonuspunkten belohnt, da er von dieser Position aus gegnerische Bauern bedrohen kann die noch nicht gezogen wurden und in der Regel den König in seinem Bewegungsspielraum stark einschränkt.

Zusätzlich werden die Punktzahlen beider Seiten durch die sich aus den *Piece-Square-Tables* ergebenden Modifikationen für den Wert der einzelnen Figuren auf bestimmten Positionen angepasst. Bei den diesen handelt es sich um 2-Dimensionale Arrays welches für jedes Feld auf dem Spielbrett beschreibt ob eine Figur an diesem Platz stärker oder schwächer gewertet werden sollte. Mit den gegnerischen Figuren wird auf genau dieselbe Art verfahren, wobei eine spezielle Matrix die Stellungen zwischen weiß nach schwarz „übersetzt“ so das die selben Piece-Square-Tables für beide Seiten verwendet werden können. Schlussendlich wird die Punktzahl des Gegners von der eigenen abgezogen und so der Wert der betrachteten Stellung bestimmt.

Die graphische Darstellung der Piece-Square-Tables wie sie in der unmodifizierten Version des Schachprogrammes Verwendung finden zeigt die Abbildung 2.6. Die hier verwendete Darstellungsmethode hat sich bereits in [1] und [7] bewährt. Ein helles Feld steht für eine Position auf dem Spielfeld an der die entsprechende Figur einen geringen Wert hat während dunklere Grautöne bessere Positionen auf dem Feld anzeigen.

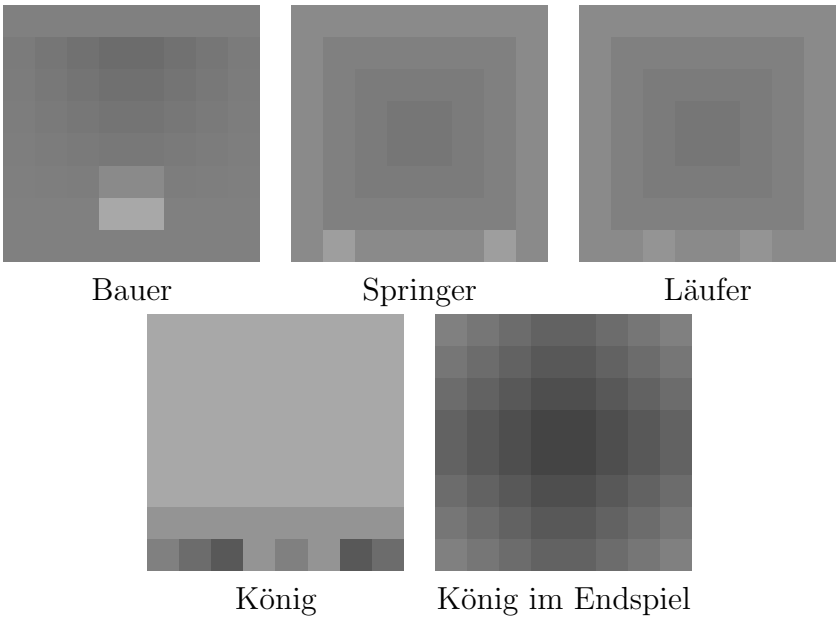



Abbildung 2.6: Original Piece Square Tables

# Kapitel 3

# Maschinelles Lernen

Das Gebiet des maschinellen Lernens (engl. *machine learning*) stellt einen Teilbereich der künstlichen Intelligenz (*KI*) dar und umfasst verschiedene Methoden zur Ableitung von Wissen aus Daten welche durch direkte Beobachtung oder auf anderem Wege gewonnen wurden. Im folgenden Abschnitt werden die grundlegenden Ideen des maschinellen Lernens sowie die konkret für das Lösen der vorliegenden Aufgabe verwendeten Mechanismen erläutert.

## 3.1 Grundlagen des Maschinellen Lernen

Man kennt im maschinellen Lernen drei verschiedene Vorgehensweisen welche sich in der Art der vorliegenden Informationen unterscheiden und durch natürliche Lernvorgänge inspiriert sind.

Das unüberwachte Lernen (engl. *unsupervised learning*) generiert aus einer Menge von Eingaben ein Modell welches diese Eingaben beschreibt. Ein solches Modell hat das Ziel Muster und Zusammenhänge in den Daten zu erkennen. Ein Beispiel hierfür sind die so genannten Clustering Algorithmen welche Datensätze mit ähnlichen Merkmalen zu Gruppen zusammenfassen.

Beim Verstärkungslernen (engl. *reinforcement learning*) werden durch den Algorithmus getroffenen Entscheidungen als positiv oder negativ bewertet. Diese Bewertung kann sowohl durch einen Experten als auch durch andere Faktoren, wie etwa den Ausgang eines Spieles, erfolgen. Aufgrund dieser Rückmeldungen wird der Prozess der Entscheidungsfindung angepasst um so eine Taktik zum Lösen der gestellten Aufgabe zu erhalten. Man könnte auch sagen, dass dieses Verfahren das Lernen aus den eigenen Fehlern beschreibt. Ein Schachprogramm welches nach dem Konzept des reinforcement learning arbeitet lernt in der Regel aus Partien die es gegen sich selbst oder andere Gegner spielt.

Als überwachtes Lernen (engl. *supervised learning*) bezeichnet man das Lernen aus eine Menge von Ziel- $y_i$ und Eingabewerten $x_i$. Der Algorithmus zieht durch Betrachtung der vorhandenen Informationen Rückschlüsse auf Verbindungen zwischen den Ein- und Ausgaben. Auch wenn keine konkrete mathematische Transformation existiert können Muster und Zusammenhänge erkannt und etwa in Form von Entscheidungsbäumen gelernt werden. Dies ist dem Lernprozess bei menschlichen Kindern sehr ähnlich. Diese lernen Objekte zu unterscheiden indem ihnen durch ihre Eltern Beispiele präsentiert werden. Wird also wie in dieser Arbeit auf vorhandene Spieldatenbanken zurückgegriffen, ohne während des Lernprozesses selbst zu Spielen, so fällt dies ebenfalls in diesen Teilbereich.

Die Trainingsdaten liegen in Form von Paaren aus einem Eigenschaftsvektor $x_i \in \mathbb{R}^m$ und einem Label $y_i \in Y$ vor. Gesucht wird eine Funktion $f$ welche für möglichst viele Eigenschaftsvektoren das richtige Label vorhersagt $f(x_i) = y_i$. Also die Beziehung zwischen den Eigenschaften und dem von diesen abhängigen Label beschreibt. Je nach Umfang der Menge möglicher Label unterscheidet man unterschiedliche Klassifikationsaufgaben.

Als binäre Klassifikation $Y = \{-1, 1\}$ bezeichnet man die Zuweisung von Datenpunkten zu einer von zwei Klassen. Von einer Multi-Class Klassifikation spricht man, wenn eine endliche Menge von Klassen $Y = \{1, 2, ..., m\}$ vorliegt. Im Falle einer kontinuierlichen Menge von Klassen $Y \subseteq \mathbb{R}$ bezeichnet man die Aufgabe als Regression.

## 3.2 Support Vector Machines

Bei den so genannten Support Vector Machines (SVMs) handelt es sich um ein überwachtes Lernverfahren. Die grundlegende Idee der SVMs ist es, die durch Vektoren repräsentierten Eingabeobjekte, durch das Einfügen einer Hyperebene, entsprechend ihrer Klassifikation zu trennen. Lassen sich zwei Klassen durch eine Hyperebene eindeutig von einander trennen, so existiert jedoch oft eine Vielzahl von weiteren Hyperebenen, welche dieselbe Aufgabe erfüllen können. Die Suche nach der bestmöglichen formuliert man daher als ein Optimierungsproblem. Dieses wird in den folgenden Abschnitten genauer betrachtet.

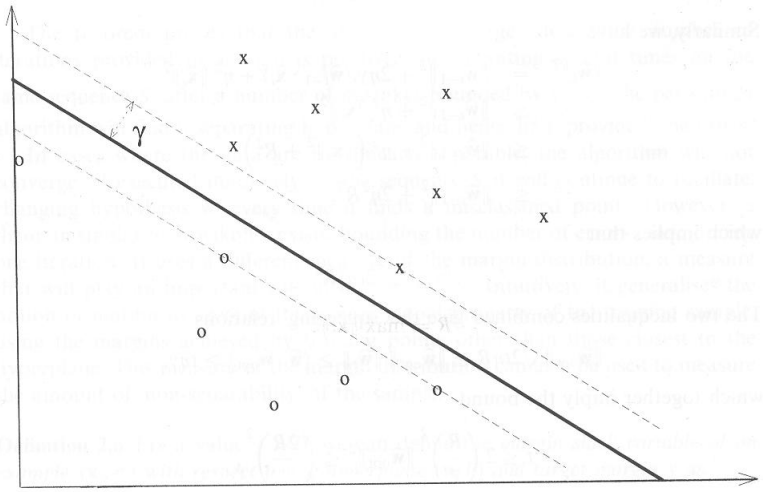



Abbildung 3.1: Optimal Trennende Hyperebene mit Rand $\gamma$, aus [29]

Der Abstand $\gamma_i$ zwischen einem Trainingspunkt $(x_i, y_i)$ und einer Hyperebene $(w, b)$ wird definiert durch die vorzeichenbehaftete Distanz zwischen dem Punkt und der Ebene, multipliziert mit der Klassifikation des Punktes $\{-1, 1\}$. Ein Wert $\gamma_i > 0$ bedeutet hierbei das der entsprechende Punkt $(x_i, y_i)$ sich auf der Seite der Ebene befindet welche seiner Klassifikation entspricht.

$$\gamma_i = y_i(\langle w * x_i \rangle + b)$$

Der Abstand einer Hyperebene $(w, b)$ zu der gesamten Menge an Trainingsdaten $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ wird bestimmt durch das Minimum der

Abstände aller Datenpunkte zur Ebene. Man nennt diesen Abstand $\gamma$ auch den *Rand der Hyperebene*. Abbildung 3.1 verdeutlicht eine solche lineare Trennung und den resultierenden Abstand graphisch.

$$\gamma = \min_{(x_n, y_n) \in S} y_n(\langle w * x_n \rangle + b)$$

Ziel einer SVM ist es diejenige Hyperebene zu finden, welche den Rand $\gamma$ maximiert. Diese nennt man auch *optimal trennende Hyperebene*. Nach der Vapnik-Chervonenkis Theorie *(VC-Theorie)* [31] lassen sich mit derartigen Hyperebenen die besten Klassifikationsergebnisse erzielen. Eine optimal trennende Hyperebene lässt einen möglichst breiten „Sicherheitsabstand“ für die Klassifikation von Punkten, welche nicht genau den Trainingsdaten entsprechen. Die Trainingspunkte welche genau auf dem Rand liegen sind ausreichend um die Hyperebene mathematisch zu beschreiben. Man bezeichnet sie als Stützvektoren (engl. *support vectors*).

Bei der Klassifikation wird das Label $fx_i$ eines Punktes durch das Vorzeichen der Entscheidungsfunktion $f(x) = (\langle w * x_i \rangle + b)$ bestimmt. Wird diese Funktion derart skaliert, dass sie an den Stützvektoren genau den Wert +1 b.z.w -1 annimmt so beträgt die Breite des Randes $\gamma = \frac{1}{\|w\|_2}$ [19]. $\|w\|_2$ ist hierbei die Länge des Normalenvektors $w$. Die Suche nach der optimal trennenden Hyperebene lässt sich nun als Optimierungsproblem der folgenden Form beschreiben:

$$\begin{aligned} &minimize_{w,b} \;\; \tfrac{1}{2} \|w\|_2^2 \\ &subject\ to \;\; y_i(\langle w * x_i \rangle + b) \geq 1, \;\; \forall i = 1, \ldots, n \end{aligned}$$

In der Praxis sind die Trainingsdaten jedoch selten streng linear separierbar. Dies kann sowohl an Messfehlern in den Daten liegen als auch daran, dass die Verteilungen der Klassen überlappen. Durch die Einführung von Schlupfvariablen $\xi_i \geq 0$ in den Nebenbedingungen ergibt sich die Möglichkeit, einzelne Objekte falsch zu klassifizieren. Die Summe dieser Verletzungen wird der Zielfunktion hinzugefügt und ebenfalls minimiert. Die Konstante $C$ regelt dabei den Einfluss der falsch klassifizierten Trainingspunkte. Dies hilft dabei eine Überanpassung zu vermieden und die benötigte Anzahl an Stützvektoren zu senken. Das Optimierungsproblem ändert sich wie folgt:

$$minimize_{w,b} \quad \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i$$
$$subject\ to \quad y_i(\langle w * x_i\rangle + b) \geq 1 - \xi_i, \quad \forall i = 1, \ldots, n$$

Wenn das zu Grunde liegende Klassifikationsproblem jedoch nicht linear ist, wie es bei vielen Anwendungen der Fall ist, so reicht die Verwendung dieser Schlupfvarianten nicht aus um zufrieden stellende Ergebnisse zu erzielen. Daher verwenden Support Vector Machines den so genannten „Kernel-Trick“, welcher sich aus drei Teilen zusammensetzt und im folgenden Abschnitt erläutert wird.

### 3.2.1 Der Kernel-Trick

**1.)** Der erste Schritt besteht in einer weiteren Umformung. Dabei wird der Normalenvektor $w$ als Linearkombination der Trainingsdaten beschrieben $w = \sum_{i=1}^{m} \alpha_i y_i x_i$. Für die Stützvektoren gilt hierbei $\alpha_i \neq 0$. Unter Verwendung der Lagrange-Multiplikatorenregel lautet die sogenannte *duale Form* des Optimierungsproblemes [19]. Das Besondere an dieser Form ist, dass die Trainingsdaten nur noch in Form ihrer Skalarprodukte vorkommen.

$$minimize_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \langle x_i x_j\rangle - \sum_{i=1}^{m}\alpha_i$$
$$subject\ to \quad \sum_{j=1}^{m}\alpha_i y_i = 0 \quad and \quad 0 \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n$$

**2.)** Die zweite Idee ist es die nicht linear trennbaren Trainingsdaten in einen höherdimensionalen Raum abzubilden, in welchem eine solche Trennung möglich ist. $\phi : \mathbb{R}^n \to \mathbb{R}^m, \mathbf{x} \mapsto \phi(\mathbf{x})$ für $n < m$. Nach der Rücktransformation wandelt sich die Hyperebene dann in eine nicht lineare Trennlinie, welche den gewünschten Effekt erzielt. Das folgende Beispiel wird dies noch einmal verdeutlichen.

Die auf der linken Seite der Abbildung 3.2 gezeigten $(x, y)$ Datenpunkte sind im 2-dimensionalen Raum nicht durch eine lineare Hyperebene zu trennen. Wird nun eine dritte Dimension eingeführt, deren Koordinaten durch $z = (x^2 + y^2)$ berechnet werden, welche also den Abstand jedes Punktes vom Ursprung beschreibt, so lässt sich eine optimal trennende Hyperebene bestimmen. Bei der Rücktransformation bildet diese Ebene eine nicht lineare Trennlinie zwischen den beiden Klassen.

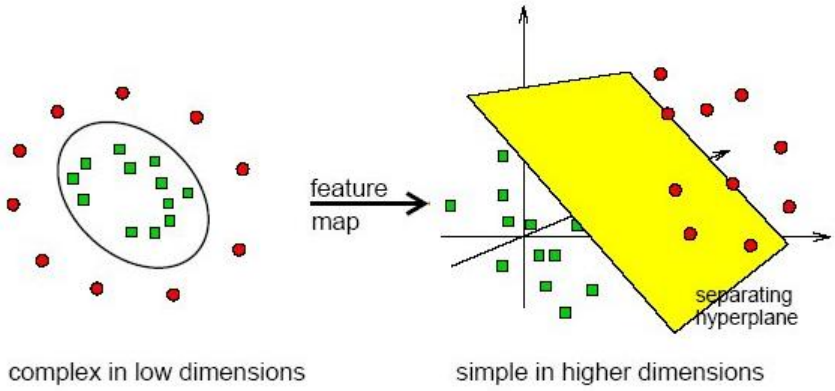



Abbildung 3.2: Lineare Trennung nach Transformation, aus [30]

**3.)** Die Berechnung von Skalarprodukten im höherdimensionalen Raum ist allerdings sehr komplex und rechenlastig. Daher kommen im dritten Schritt die für den Trick namensgebenden *Kernel-Funktionen* zur Anwendung. Diese Funktionen verhalten sich wie das Skalarprodukt der transformierten Vektoren, lassen sich jedoch wesentlich einfacher berechnen. Die eigentliche Transformation wird damit nur implizit durchgeführt. Es gilt:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

Auch dies wird noch einmal anhand eines Beispiels verdeutlicht. Gegeben sind zwei Vektoren $x_i = (i_1, i_2)$ und $x_j = (j_1, j_2)$ sowie die Abbildung $\phi : (v_1, v_2) \mapsto (v_1^2, \sqrt{2v_1v_2}, v_2^2)$. Das Skalarprodukt zweier transformierter Vektoren wird wie folgt umgeformt:

$$\begin{aligned} \langle \phi(x_i), \phi(x_j) \rangle &= (i_1^2, \sqrt{2i_1i_2}, i_2^2)(j_1^2, \sqrt{2j_1j_2}, j_2^2)^T \\ &= (i_1^2 j_1^2 + 2i_1 j_1 i_2 j_2 + i_2^2 j_2^2) \\ &= (i_1 j_1 + i_2 j_2)^2 := K(x_i, x_j) \end{aligned}$$

Die Kernel-Funktion $K(x_i, x_j)$ kann nun verwendet werden um mit wesentlich weniger Rechenaufwand das Skalarprodukt zu berechnen, ohne die Transformation wirklich durchführen zu müssen.

In der Praxis werden Kernels jedoch selten mühsam aus dem Skalarprodukt im hochdimensionalen Raum berechnet. Statt dessen wird eine Kernel-Funktion

mit den gewünschten Eigenschaften konstruiert und die implizit durchgeführte Transformation durch diese Wahl bestimmt. Zulässige Funktionen werden durch den Satz von Mercer [22] charakterisiert. Die verbreitetste Methode zur Erzeugung von komplexen Kernels ist die Konstruktion aus anderen Kernel-Funktionen [29].

## 3.3 Verwendung einer SVM für Ranking Probleme

Die Möglichkeiten der Nutzung von Support-Vector-Machines zur Klassifikation und Regression werden bereits seit einigen Jahren intensiv erforscht. Vergleichsweise neu ist jedoch ihre Verwendung zum Lernen von Reihungen.

Das Problem des überwachten Ordnens (eng. *supervised ordering*) besteht darin für eine (oder mehrere) Menge(n) von Eigenschaftsvektoren $x_i \in \mathbb{R}^m$ eine Funktion $f$ zu finden, dass die Funktionswerte $f(x_i)$ eine vorgegebene Ordnung einhalten. Kamishima, Kazawa und Akaho haben eine Bestandsaufnahme über die zur Lösung solcher Probleme verfügbaren Verfahren zusammengestellt [13]. Dabei zeigte sich das SVM-basierte Verfahren zwar lange Laufzeiten für die Berechnung benötigen im Gegenzug aber gut mit großen Mengen an Trainingsdaten und Eigenschaften klarkommen. Auch sind Verfahren die SVMs verwenden weniger anfällig gegenüber verrauschten Attributen. In dieser Arbeit wurde die sogenannte $SVM^{light}$ [1] verwendet, diese verfügt auch über einen Modus für Ranking-Probleme.

In diesem Ansatz wird die Reihenfolge der Punkte durch ihre Projektion auf einen Gewichtsvektor $w$ bestimmt. Dies ist äquivalent zur Berechnung des vorzeichenbehafteten Abstand zu einer Hyperebene mit $w$ als Normalenvektor [11]. Dies wiederum ist genau dasselbe Problem das auch für die Klassifikation gelöst werden muss.

[1] http://svmlight.joachims.org/

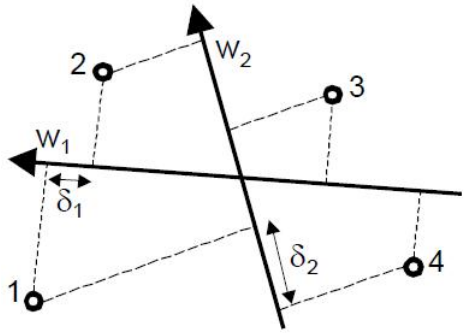



Abbildung 3.3: Reihung von Punkten durch Projektion auf Vektoren [11]

Zum besseren Verständnis zeigt die Abbildung 3.3 wie vier Punkte durch zwei Gewichtsvektoren $w_1$ und $w_2$ unterschiedlich gereiht werden. Die Projektion auf $w_1$ erzeugt die Reihenfolge $(1, 2, 3, 4)$ während $w_2$ die Punkte in der Folge $(2, 3, 1, 4)$ anordnet.

Trainingsgrundlage ist eine Menge von Entscheidungssituationen $1...n$ mit entsprechender Query-ID *qid* $q_k$, einer Reihe von Trainingsdaten $(x_i, y_i)$ und einer partiellen Ordnung $R_k$. Für diese Ordnungen gilt, wenn ein Trainingsvektor $x_i$ in der Entscheidung $q_k$ höher bewertet wurde als ein anderer Vektor $x_j$ dann ist $(y_i > y_j) \in R_k$. Für den zu bestimmenden Gewichtsvektor $w$ soll nun gelten das $w * (x_i, q_k) > w * (x_j, q_k)$ also $w * ((x_i, q_k) - (x_j, q_k)) > 0$. Dies führt zu folgendem Optimierungsproblem:

$$\begin{aligned} &minimize_w \quad \tfrac{1}{2} \|w\|_2^2 + C \textstyle\sum_{i=1}^{n} \xi_{i,j,k} \\ &subject\ to \quad w * ((x_i, q_k) - (x_j, q_k)) \geq 1 - \xi_{i,j,k}, \quad \forall (y_i > y_j) \in R_k \end{aligned}$$

Welches sich mit den selben Mechanismen lösen lässt wie sie auch für die Klassifikation eingesetzt werden.

Inzwischen ist eine $SVM^{rank}$ genannte Weiterentwicklung der $SVM^{light}$ verfügbar welche für die Durchführung von Ranking Aufgaben optimiert worden ist und wesentlich schneller arbeiten soll. Für die vorliegende Arbeit wurde sie allerdings nicht verwendet, da der Lernprozess zum Zeitpunkt ihrer Veröffentlichung bereits abgeschlossen war.

# Kapitel 4

# Versuchsaufbau

Das folgende Kapitel enthält detaillierte Erklärungen zu den einzelnen Schritten mit denen die gestellte Aufgabe bearbeitet wurde. Es werden die verwendeten Vorgehensweisen und Tools vorgestellt und der Informationsfluss zwischen den einzelnen Arbeitsschritten verfolgt.

## 4.1 Die Datenquelle

Die Spielprotokolle, welche die Grundlage der Arbeit bilden, stammen aus einer frei verfügbaren Datenbank[1]. Aus den dort vorhandenen, mehr als 3.5 Millionen, Partien wurden Spielprotokolle extrahiert bei denen der ELO Wert des Spielers auf der weißen Seite um nicht mehr als 25 Punkte von den betrachteten Spielstärken von 1000, 1200, 1400 und 1600 Punkten abweicht. Da für den Lernprozess eine große Zahl von Schachpositionen benötigt wird wurden für die Erstellung des Trainingssets einige kleine Hilfsprogramme geschrieben.

Folgender Auszug aus dem PGN Protokoll einer der betrachteten Partien wird im weiteren Verlauf des Kapitels zur Veranschaulichung einzelner Schritte dienen.

[1] http://chessdb.sourceforge.net/

```
... 16.Nc3xd5 c6xd5 17.Ne3xd5 Nb8-c6 18.Ra1-c1 Kg8-h8 19.Nd5-b4 Nc6-e7
20.Rc1-c7 Ne7-f5 21.Qd2-c3 e4-e3 22.f2xe3 Nf5-h4 23.Rf1xf8+ Ra8xf8
...
```

Da das verwendete Schachprogramm mit der, im PGN-Format verwendeten, verkürzten algebraische Notation nicht kompatibel ist mussten die Protokolle in eine Zwischenschritt mithilfe eines weiteren Datenbankprogrammes[2] noch in die ausführliche algebraische Notation übersetzt werden.

| ELO | Spiele | Entscheidungen | Trainingsdaten |
|---|---|---|---|
| 1000 | 3.000 | 20.732 | 614.246 |
| 1200 | 5.499 | 39.170 | 1.185.524 |
| 1400 | 10.499 | 75.196 | 2.325.861 |
| 1600 | 10.500 | 75.114 | 2.352.536 |

Tabelle 4.1: Datails zur Trainingsgrundlage

Es ergibt sich für die einzelnen ELO Werte die in Tabelle 4.1 gezeigte Anzahl an zur Verfügung stehenden Spielen. Wie sich aus diesen Spielen die Entscheidungssituationen und auszuwertenden Stellungen ergeben wird im nächsten Abschnitt erläutert. Die Menge an Spielen für die letzte Kategorie wurde limitiert da die Verarbeitung durch die SVM bei dieser Menge an Daten bereits über 48 Stunden beanspruchte.

Anschließend werden die Daten ein weiteres mal geparst, in eine für das Schachprogramm direkt lesbare Folge von Zügen umgewandelt und alle 10 Halbzüge mit einem speziellen Bearbeitungsbefehl versehen dessen Funktion im folgenden Abschnitt erläutert wird.

[2] http://jose-chess.sourceforge.net/

## 4.2 Erzeugung der Trainingsdaten

Die einzelnen Spiele liegen nun in folgender Form vor:

```
...c3d5 c6d5 e3d5 b8c6 a1c1 g8h8 d5b4 c6e7 work c1c7 e7f5 d2c3 e4e3
f2e3 f5h4 f1f8 a8f8 ...
```

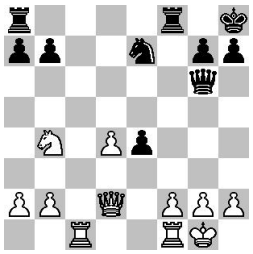
Wobei *work* der Befehl an das Schachprogramm ist den nächsten, nach der Bearbeitungsanweisung eintreffenden, Zug als eine zu lernenden Entscheidungssituation zu betrachten und speziell zu verarbeiten. Dabei wird für diesen und alle anderen aus der aktuellen Position möglichen Züge eine Analyse der Schlagfolge durchgeführt, um die sich schlussendlich ergebende Ruhestellung auf dem Spielfeld zu bestimmen.

Alle 10 Halbzüge liegt eine Entscheidungssituation vor und diese werden mit einer eindeutigen Query-ID identifiziert. Für jede der erreichbaren Stellungen wurden die für die Bewertungsfunktion relevanten Informationen als Trainingsdatensatz für den Lernprozess übernommen. Der vom Spieler ausgeführte Zug wird hierbei als positives Beispiel gewertet während die Positionen welche aus nicht gewählten Zügen resultieren als unerwünscht Klassifiziert werden. Auf diese Weise erhält man im Durchschnitt 29 negative und einen positiven Trainingsfall pro ausgewertetem Zug.

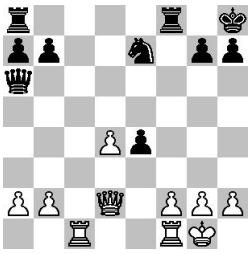
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| b4a6= | -1 | qid:173 | 1:1 2:-1 | 41:1 54:1 55:1 59:1 60:1 61:1 | | 260:1 | 327:-1 329:1 330:1 | |
| f2f3= | -1 | qid:173 | | 41:1 54:1 55:1 60:1 61:1 | 103:1 | 260:1 | 327:-1 329:1 330:1 | |
| b4d5= | -1 | qid:173 | 1:1 2:-1 | 41:1 54:1 55:1 59:1 60:1 61:1 | | 260:1 | 327:-1 329:1 330:1 | |
| c1c7= | 1 | qid:173 | 1:1 | 41:1 54:1 55:1 59:1 60:1 61:1 | 103:1 | 260:1 | 327:-1 329:1 330:1 | 332:1 |

Tabelle 4.2: kodierte Ruhestellungen aus Abbildung 4.1

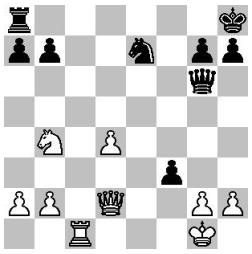
Abbildung 4.1 zeigt beispielhaft 4 der 39 Stellungen die sich aus der Ruhesuche aller möglichen legalen Züge ergeben. Der Zug c1c7 ist hierbei derjenige den der Spieler tatsächlich durchgeführt hat. Es ergeben sich die in 4.2 gezeigten kodierten Datensätze. Die Felder beschreiben die in 2.7 erläuterten Werte und der Aufbau eines Datensatzes wird in 4.3 detaillierter beschrieben.



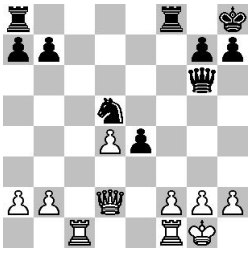
Ausgangsstellung



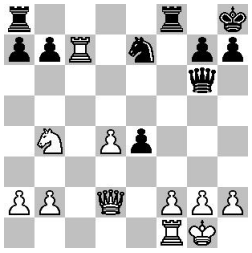
Zugfolge: b4a6, g6a6



Zugfolge: f2f3, f8f3, f1f3, e4f3



Zugfolge: b4d5, c7d5



Zugfolge: c1c7

Abbildung 4.1: Ruhestellungen nach verschiedenen Zugfolgen

Die Gewichte der gelernten Bewertungsfunktionen werden durch die SVM anhand der Unterschiede zwischen den einzelnen Unterschiede festgelegt. So haben für die Entscheidung nicht relevante Figuren, wie etwa der in allen Datensätzen gleich bleibende König auf g1, keinen Einfluss.

## 4.3 Berechnung der Piece Square Tables

Zur Berechnung der Werte für die einzelnen Variablen wurde die $SVM^{light}$ im Ranking Modus verwendet. Ziel dieses Modus ist nicht den Klassifikationswert eines Datensatzes möglichst genau zu bestimmen. Durch die Label $l$ wird

| Variable | | Bedeutung |
|---|---|---|
| label | = | Vom Spieler gemachter Zug (1) oder nicht gewählte Alternative (-1) |
| qid | = | Identifikator für die Entscheidungssituationen (Integer) |
| 1 - 5 | = | Differenz (Bauer, Springer, Läufer, Turm, Dame) aus Sicht von Weiß (Integer) |
| 6 - 69 | = | Bauer auf Position a8-h1 (46 Werte - Boolean) |
| 70 - 133 | = | Springer auf a8-h1 (64 Werte - Boolean) |
| 134 - 197 | = | Läufer auf a8-h1 (64 Werte - Boolean) |
| 198 - 261 | = | König auf a8-h1 (64 Werte - Boolean) |
| 262 - 325 | = | König auf a8-h1 (Endspiel) (64 Werte - Boolean) |
| 326 - 332 | = | Sonderwerte (Integer) (siehe Abschnitt 2.7) |

Tabelle 4.3: Details zu den Datenfeldern eines Trainingsdatensatz

eine Ordnung auf allen Datensätzen mit identischer *qid* vorgegeben und diese Ordnung soll für die gewichteten Summen der einzelnen Datensätze eingehalten werden. Jeder Datensatz wird hierzu als Vektor $\vec{v} \in \mathbb{Z}^{332}$ betrachtet welcher mit dem gelernten Gewichtsvektor $\vec{w} \in \mathbb{R}^{332}$ multipliziert wird. Gilt nun für zwei Vektoren das $l_1 > l_2$ so soll in möglichst vielen Fällen ebenfalls $\vec{v_1} * \vec{w} > \vec{v_2} * \vec{w}$ gelten.

Im aktuellen Beispiel bedeutet dies, dass das Ergebnis des vierten in Tabelle 4.2 verzeichneten Datensatzes größer sein soll als das der anderen Drei.

Auf die genauen Abläufe innerhalb der $SVM^{light}$ soll an dieser Stelle nicht weiter eingegangen werden. Eine kurze Beschreibung findet sich in Abschnitt 3.3 und ausführliche Erklärungen in den entsprechenden Arbeiten [9] [11]. Betrachten wir sie im Folgenden einfach als eine *Black Box* welche aus den Beschreibungen der Spielpositionen und deren Bewertung aus Sicht des Spielers, Gewichte für die einzelnen Variablen berechnet. Die SVM liefert eine Reihe unterschiedlich gewichteter Support-Vektoren, welche eine Untermenge der Trainingsdaten bilden. Die Gewichte der einzelnen Variablen berechnet man anschließend durch das Summieren der Vektorgewichte an allen Positionen in denen sie in den Supportvektoren vorkommen. In Abschnitt 5.1 wird eine genauere Analyse der aus den Trainingsdaten gelernten Gewichte und Piece-Square-Tables durchgeführt.

Abbildung 4.4 zeigt die durch eine $\xi\alpha$-Schätzung ermittelten Fehlerraten. Die Trefferquote (engl. *Recall*) des Verfahrens beschreibt die Wahrscheinlichkeit

| ELO | Laufzeit (cpus) | Anzahl SVs | Fehler | Trefferquote | Genauigkeit |
|---|---|---|---|---|---|
| 1000 | 24.098 | 205.740 | 34.46% | 66.96% | 72.42% |
| 1200 | 17.251 | 383.263 | 33.26% | 67.84% | 73.30% |
| 1400 | 87.603 | 758.598 | 33.55% | 67.28% | 72.74% |
| 1600 | 86.638 | 763.646 | 33.40% | 67.18% | 72.89% |

Tabelle 4.4: Details zum Lernen der Gewichte

das ein Datensatz mit Label $y = 1$ korrekt klassifiziert wird. Die Genauigkeit (engl. *Precision*) gibt die Wahrscheinlichkeit an mit der ein durch $f(x) = 1$ klassifizierter Datensatz auch wirklich das Label $y = 1$ aufweist. Das $\xi\alpha$-Verfahren verwendet die während der Problemlösung ermittelten Vektoren $\xi$ [3.2] und $\alpha$ [3.2.1] um Aussagen über die Güte der Lösung zu treffen. Dies hat den Vorteil, dass kaum zusätzliche Berechnungen notwendig sind und sich diese Abschätzung damit gut für große Datenmengen eignet. Der Nachteil dieses Schätzverfahrens liegt darin, dass es im Vergleich mit einer Leave-One-Out-Kreuzvalidierung (engl. *leave one out cross-validation*) den Fehler überschätzt während Trefferquote und Genauigkeit unterschätzt werden [10].

# Kapitel 5

# Versuchsdurchführung und Ergebnisse

Für die folgenden in dieser Arbeit verwendeten Darstellungen werden die Gewichte derart normiert, dass sich für die Bauern ein Wert von 100 ergibt. Auf diese Weise lassen sich die Gewichte besser mit den Originalwerten vergleichen. Diese Normierung ändert jedoch nichts an der Gültigkeit der Werte, da für die Bewertungsfunktion nur die relativen Wert der Figuren zueinander eine Rolle spielt.

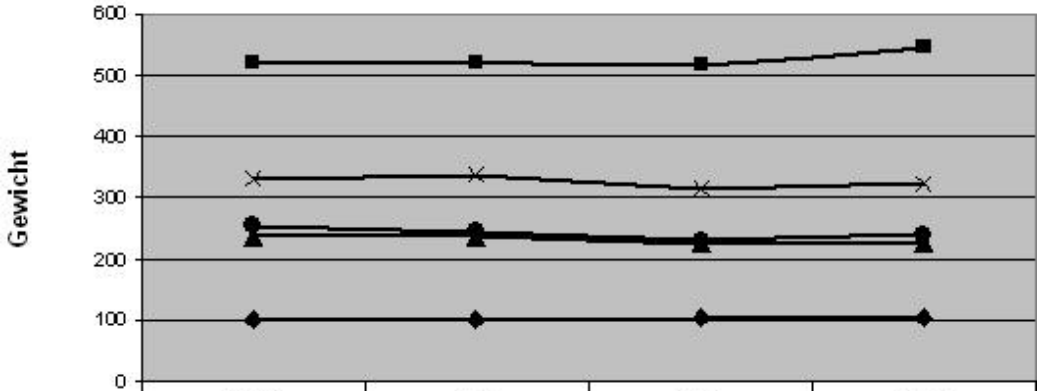
## 5.1 Analyse der berechneten Gewichte

### 5.1.1 Figuren- und Sonderwerte

Die folgenden drei Abbildungen zeigen die Entwicklung der gelernten Gewichten zwischen den einzelnen Spielstärken. Auf der Y-Achse finden sich die gelernten Gewichte während die X-Achse sich in vier Abschnitte für die unterschiedlichen Trainingsmengen teilt.

Die Normierung mit dem Wert des Bauern führt zu dem Problem, dass sich Veränderungen an dessen Wert in den Werten der anderen Figuren niederschlagen. Normiert man die Figurenwerte nicht wie üblich auf den

Bauern so sieht man das dieser, verglichen mit den traditionellen Werten und abhängig von der für die Normierung verwendeten Figur, um 25-75% stärker gewertet wird als üblich. Für die Darstellung der Veränderungen im Wert des Bauern wurde in Abbildung 5.1 der auf der Basis der 1000 ELO Spielprotokolle ermittelte Wert als Referenz festgelegt. Es zeigt sich, dass der Wert des Bauern, im Vergleich der einzelnen gelernten Bewertungsfunktionen, nur wenig schwankt und sich daher als Grundlage der Normierung eignet.

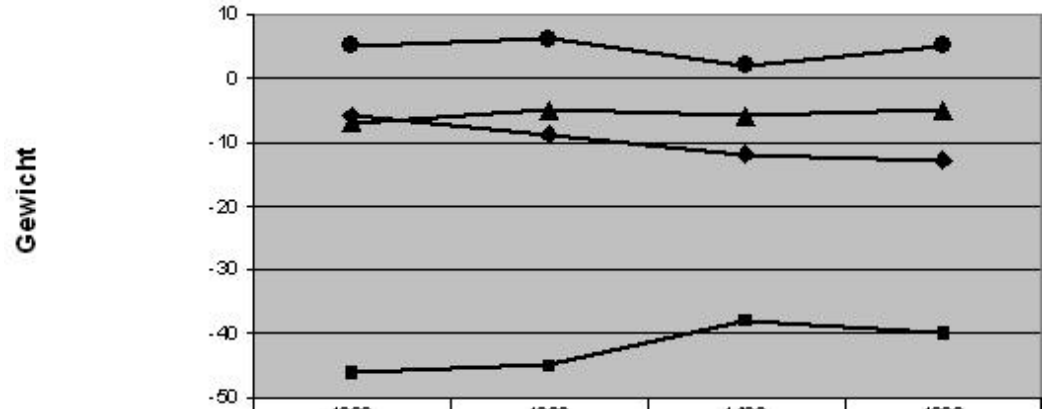



| | 1000 | 1200 | 1400 | 1600 |
|---|---|---|---|---|
| Bauer | 100 | 100 | 105 | 104 |
| Turm | 332 | 338 | 313 | 323 |
| Springer | 253 | 246 | 231 | 238 |
| Läufer | 237 | 236 | 226 | 228 |
| Dame | 520 | 519 | 517 | 546 |

Abbildung 5.1: Entwicklung der Figurengewichte

Im Gegensatz zu den Bauern werden alle anderen Figuren deutlich schwächer gewertet. Untereinander gleicht sich dies zwar wieder aus, doch führt es generell zu einem stärkeren Einfluss der Bauern und der positionellen Komponente der Bewertungsfunktion.

Die Sonderwerte des Bauern liegen in etwa auf dem selben Niveau wie die Ursprungswerte. Da der Wert des Bauern jedoch gestiegen ist haben sie in den gelernten Bewertungsfuntionen einen geringeren Stellenwert. Die Gewichte für die einzelnen Positionen innerhalb der Piece-Square-Tables fallen hingegen stärker aus als in den Originalwerten. So werden Opfer zum Erreichen



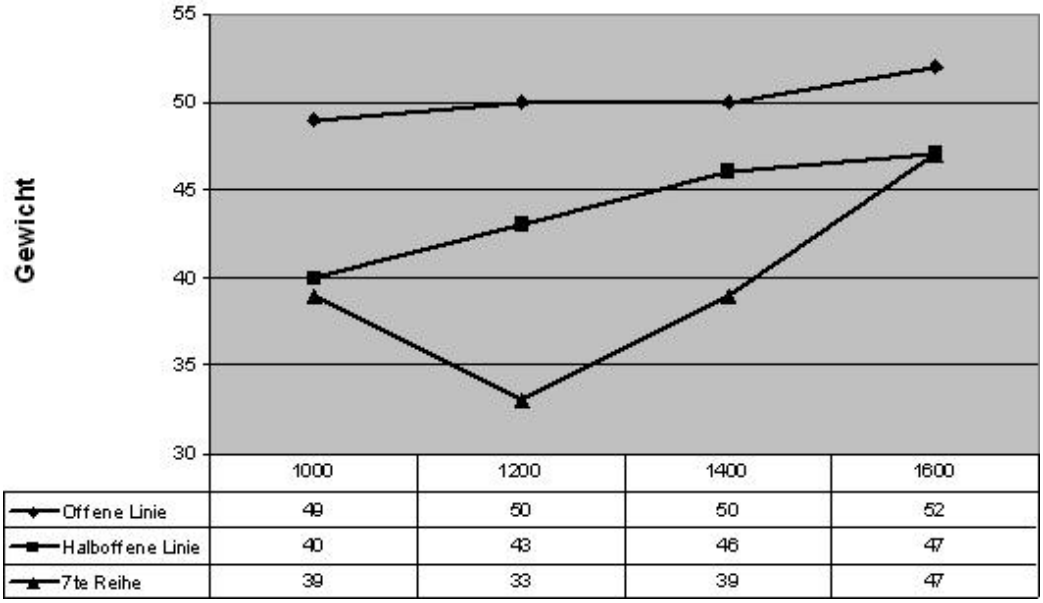

| | 1000 | 1200 | 1400 | 1600 |
|---|---|---|---|---|
| Doppelbauer | -46 | -45 | -38 | -40 |
| Isolierter Bauer | -6 | -9 | -12 | -13 |
| Rückständiger Bauer | -7 | -5 | -6 | -5 |
| Freibauer | 5 | 6 | 2 | 5 |

Abbildung 5.2: Entwicklung der Sonderwerte (Bauer)

einer vorteilhaften Position schneller in Kauf genommen. Auffällig ist der verhältnismäßig hohe Strafterm für einen Doppelbauern.

Sowohl die Dame als auch die beiden Türme verlieren beträchtlich an Wert. Bei den Türmen wird dies durch einen starken Anstieg der ihnen zugeordneten Sonderwerte ausgeglichen. So ist ein gut aufgestellter Turm ähnlich stark wie bei Verwendung der Standardgewichte. Auch kann man bei allen Sonderwerten des Turmes einen Anstieg mit zunehmender Spielstärke der betrachteten Spieler feststellen.

Die ewige Frage ob nun der Läufer oder der Springer höher zu bewerten sei entscheidet im vorliegenden Fall der Springer für sich. Wobei die Stärke beider Figuren stark von der Situation und ihren Positionen auf dem Spielfeld abhängt.




| | 1000 | 1200 | 1400 | 1600 |
|---|---|---|---|---|
| Offene Linie | 49 | 50 | 50 | 52 |
| Halboffene Linie | 40 | 43 | 46 | 47 |
| 7te Reihe | 39 | 33 | 39 | 47 |

Abbildung 5.3: Entwicklung der Sonderwerte (Turm)

### 5.1.2 Die relevanzskalierte Gewichtsdarstellung

Für den Vergleich der gelernten Piece-Square-Tables wird eine relevanzskalierte Gewichtsdarstellung verwendet. Im Folgenden wird kurz erläutert wie diese Darstellung berechnet wird und welche Vorteile sie bietet.

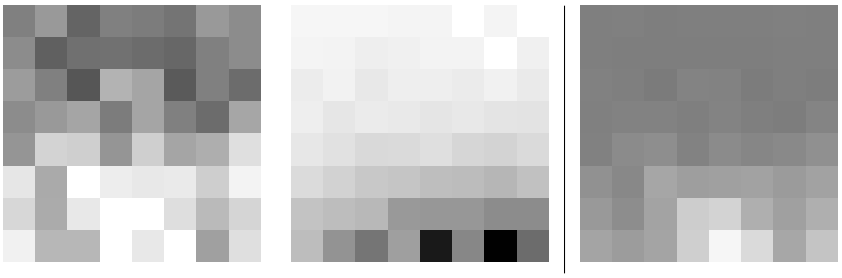


Abbildung 5.4: Erzeugen der relevanzskalierten Gewichtsdarstellung

Auf der linken Seite der Abbildung 5.4 befindet sich eine graphische Darstellung der gelernten Piece Square Table für einen König außerhalb des Endspiels. Helle Farben symbolisieren schlechte Felder während eine dunkle Färbung eine vorteilhafte Position anzeigt. Vor allem in der oberen Hälfte widersprechen

diese Gewichte den Erwartungen da diese Felder aus spieltechnischer Sicht eher schlecht bewertet werden sollten. Betrachtet man nun die Darstellung in der Mitte, welche die Häufigkeit [1] anzeigt mit der sich der König auf den entsprechenden Feldern befunden hat, so zeigt sich, dass es bei den Werten oberhalb der dritten Reihe um das Resultat einiger weniger Trainingsfälle handelt.

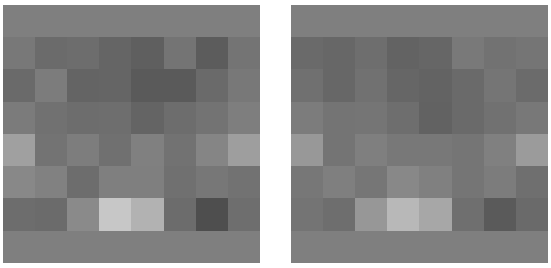
Die Gewichte für diese Positionen sind also unzuverlässig. Allerdings sind sie für die weitere Analyse auch weniger interessant da sie für die jeweilige Figur ohnehin schlecht zu erreichen oder in der Regel durch den Gegner gedeckt sind. Des Weiteren sind die Züge die die betreffende Figur auf diese Felder geführt haben wahrscheinlich nicht durch positionelle Überlegungen erfolgt sondern um dort befindliche Figuren direkt zu bedrohen, zu schlagen oder um sich selbst vor Angriffen zu schützen.

In der weiteren Analyse soll nun den Gewichten von Feldern die durch die betreffende Spielfigur nur selten besetzt wurden weniger Bedeutung beigemessen werden. Dies wird realisiert indem die Gewichte mit der Häufigkeit multipliziert werden. Im betrachteten Beispiel findet sich dieses Resultat auf der rechten Seite. Unveränderte graphische Darstellungen der Piece-Square-Tables finden sich im Anhang der Arbeit. Diese relevanzskalierten Gewichte werden lediglich für die Analyse und nicht für die praktische Verwendung der Bewertungsfunktionen genutzt.

### 5.1.3 Die Bauern

Die negativen Werte der Felder e2 und d2 führen schnell dazu, dass die dort befindlichen Bauern in Richtung des Zentrum gezogen werden. Im Vergleich dazu bekommen die Anfangsfelder auf beiden Flanken hohe positive Gewichte, so, dass die Bauern dort bleiben und einen Schutzwall für den rochierten König bilden. Die Felder auf der Königsflanke sind dunkler da die Rochade auf dieser Seite einfacher durchzuführen und daher auch häufiger ist.

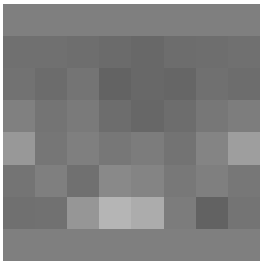
[1] $\sqrt[4]{absoluteHaeufigkeit}$ - normiert auf $[0, 255]$



1000 ELO



1200 ELO



1400 ELO

1600 ELO

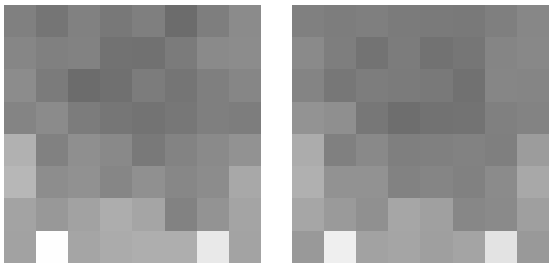
Abbildung 5.5: Relevanzskalierte Gewichtsdarstellung - Bauer

Ergibt sich dazu eine Gelegenheit werden jedoch alle Bauern von den hohen Gewichten der Felder in 7ter Reihe auf die Seite des Gegners gezogen, um dort anschließend in stärkere Figuren umgewandelt zu werden.

### 5.1.4 Die Springer

Während die Ursprungsfunktion für den Springer lediglich die beiden Startfelder deutlich schlechter bewertet so treiben die gelernten Gewichte beide Springer geradezu aus der eigenen Hälfte in das Zentrum oder auch gleich in die Linien des Gegners. Auch wenn die üblichen Positionen c3 und f3 vom Startpunkt der Springer noch am höchsten gewertet werden so sind die mit einem weiteren Zug zu besetzenden Felder im Zentrum des Brettes noch deutlich attraktiver.

Auch fällt die Umsetzung des Sprichwortes „Ein Springer am Rand bringt Kummer und Schand“ in den gelernten Daten relativ deutlich aus. Jedoch beinhaltet dies nicht die Randfelder auf der Seite des Gegners. Ein Blick auf


1000 ELO


1200 ELO


1400 ELO

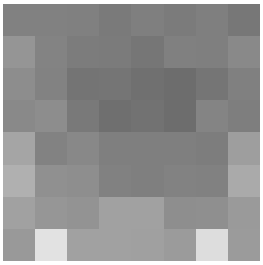

1600 ELO

Abbildung 5.6: Relevanzskalierte Gewichtsdarstellung - Springer

die relevanzskalierten Gewichtsdarstellungen erklärt dies dadurch, dass die Springer nur selten bis auf die gegnerische Grundlinie vordringen und die Gewichte dort daher nur durch wenige Trainingsfälle gestützt werden.

### 5.1.5 Die Läufer

Im Gegensatz zu den Originalgewichten, welche auf eine Positionierung im Zentrum hinwirken, ist es für die Läufer nach dem Verlassen ihrer Anfangsfelder nicht so zwingend sich nach vorne zu orientieren. Stark gewichtet sind auch die Felder neben dem Zentrum, sowie die Felder g2 und b2 auf denen die Läufer, im sogenannten Fianchetto, innerhalb der eigenen Bauernstruktur aufgestellt werden. Von diesen aus kann ein Läufer sowohl das Zentrum als auch die gegenüberliegende Flanke des Gegners bedrohen. Erwähnenswert ist noch die stärkere Gewichtung der vom Königsläufer erreichbaren Felder welcher dadurch leicht bevorzugt wird. Dies kann mit der Notwendigkeit diesen Läufer, für eine möglichst frühe Rochade, zuerst zu bewegen begründet werden.

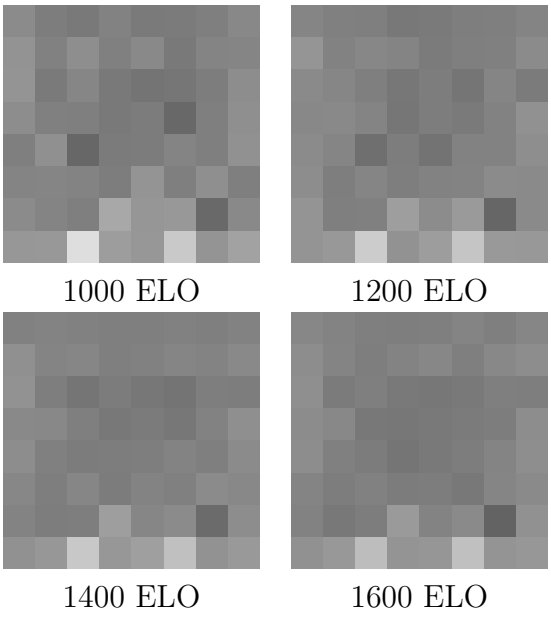
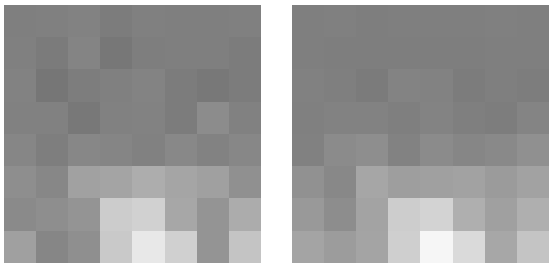



Abbildung 5.7: Relevanzskalierte Gewichtsdarstellung - Läufer

### 5.1.6 Der König

Die relevanzskalierte Gewichtsdarstellung für den König zeigt das die Felder oberhalb der dritten Reihe außerhalb des Endspieles nahezu nie betreten werden. Daher sind die für diese ermittelten Wertungen zwar unzuverlässig aber auch von geringerem Interesse. Einige Felder haben zwar aufgrund einiger weniger Trainingsfälle bei denen der König aufgrund von Drohungen gezwungen wurde diese Felder zu betreten ein vergleichsweise hohes Gewicht. Doch finden sich in der zweiten und dritten Reihe genug niedrig bewertete Felder um das passieren dieser Reihen für den König unattraktiv zu machen. Die Bedrohung durch gegnerische Figuren macht das Vorrücken des Königs, außerhalb des Endspiels, ohnehin uninteressant. Wird der König gezwungen, vor erreichen des Endspieles, seine sichere Deckung zu verlassen hat der betreffende Spieler in der Regel größere Probleme, so dass die unzuverlässigen Werte hier wenig Einfluss haben werden.

Betrachtet man nun die Felder für deren Bewertung ausreichend Informationen zur Verfügung stehen so ist deutlich, dass die durch eine Rochade schnell



1000 ELO



1200 ELO

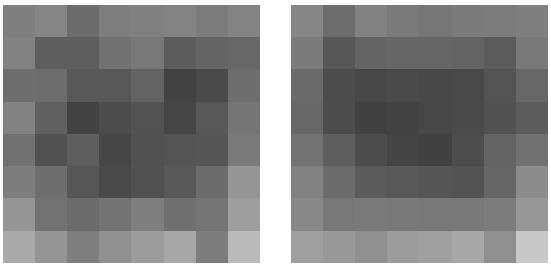
1400 ELO

1600 ELO

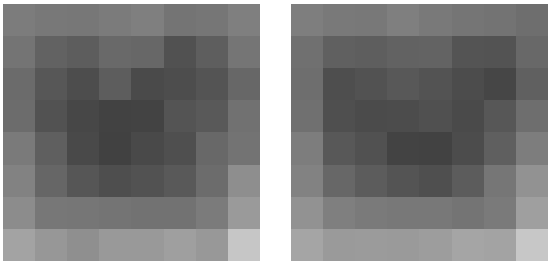
Abbildung 5.8: Relevanzskalierte Gewichtsdarstellung - König

erreichbaren Felder am höchsten gewertet werden und diese so möglichst frühzeitig durchgeführt wird. Im Gegensatz zu den Originalwerten wird jedoch das Feld b1 gegenüber c1 stärker gewichtet. Dieses bietet dem König, auf Kosten des Spieltempos, eine etwas sicherere Stellung welche von Spielern auf diesem Niveau augenscheinlich bevorzugt wird.

Ist das Endspiel erreicht werden die Felder im Zentrum bis hinauf in die 7te Reihe sehr interessant. Der König wird zu diesem Zeitpunkt weit weniger durch gegnerischen Figuren eingeschränkt und sollte zum Schutz der verbliebenen eigenen Figuren oder zur Abwehr einer möglichen Bauernumwandlung selbst aktiv werden.



1000 ELO 1200 ELO



1400 ELO 1600 ELO

Abbildung 5.9: Relevanzskalierte Gewichtsdarstellung - König im Endspiel

## 5.2 Wettstreit der Bewertungsfunktionen

Durchgeführt wurde dieses Experiment mit dem *Galis Winboard Tournament Manager*[2]. Dieses Tool erlaubt es Winboard kompatible Schachprogramme in unterschiedlichen Turnieren gegeneinander antreten zu lassen um ihre Leistungen zu vergleichen. Gespielt wurden 100 Spiele gegen jeden Konkurrenten.

| Basis | 1000 ELO | 1400 ELO | 1600 ELO | 1200 ELO | + | = | - | $\sum$ |
|---|---|---|---|---|---|---|---|---|
| 1000 ELO | - | 49,5 | 55 | 50,5 | 35 | 240 | 25 | 155 |
| 1400 ELO | 50,5 | - | 50 | 54 | 18 | 273 | 9 | 154,4 |
| 1600 ELO | 45 | 50 | - | 53 | 19 | 258 | 23 | 148 |
| 1200 ELO | 49,5 | 46 | 47 | - | 12 | 261 | 27 | 142,5 |

Tabelle 5.1: Turnierergebniss - Suchtiefe: 1 Halbzug

Suchtiefe 1: [Abb. 5.1] Das Schachprogramm kam mit dieser Suchtiefe unerwarteter weise nicht gut klar. Die sehr große Zahl an unentschiedenen Spielen resultiert daraus das Remis durch Stellungswiederholung nicht erkannt b.z.w.

[2]http://wbec-ridderkerk.nl/html/downloada/Galis.html

von beiden Seiten als annehmbarer Ausgang akzeptiert wurden. Es wäre eigentlich zu erwarten gewesen das eine der beiden Seiten sich im Vorteil sehen und das unentschieden vermeiden würde.

| Basis | Orginal | 1600 ELO | 1400 ELO | 1200 ELO | 1000 ELO | + | = | - | $\sum$ |
|---|---|---|---|---|---|---|---|---|---|
| Original | - | 56,5 | 63,5 | 63 | 63,5 | 232 | 29 | 139 | 246,5 |
| 1600 ELO | 43,5 | - | 61 | 57 | 63,5 | 210 | 30 | 160 | 225,0 |
| 1400 ELO | 36,5 | 39 | - | 54 | 56 | 167 | 37 | 196 | 185,5 |
| 1200 ELO | 37 | 43 | 46 | - | 59 | 171 | 28 | 201 | 185,0 |
| 1000 ELO | 36,5 | 36,5 | 44 | 41 | - | 130 | 56 | 214 | 158,0 |

Tabelle 5.2: Turnierergebniss - Suchtiefe: 2 Halbzüge

Suchtiefe 2: [Abb. 5.2] Mit dieser Suchtiefe treten die Probleme der Stellungswiederholung nicht mehr auf und es ergeben sich durchaus aussagekräftige Unterschiede was die Stärken von ELO 1600 und ELO 1000 betrifft. Diese Unterschiede reichen allerdings bei weitem nicht an die 200 ELO-Punkte Differenz zwischen den einzelnen Trainingsgrundlagen heran. Es muss daher davon ausgegangen werden, dass der Einfluss der Bewertungsfunktion nicht so groß ist wie erhofft.

| Basis | Orginal | 1200 ELO | 1600 ELO | 1400 ELO | 1000 ELO | + | = | - | $\sum$ |
|---|---|---|---|---|---|---|---|---|---|
| Original | - | 64,5 | 73,5 | 62 | 78 | 266 | 24 | 110 | 278,0 |
| 1200 ELO | 35,5 | - | 38,5 | 53,5 | 68,5 | 174 | 44 | 182 | 196,0 |
| 1600 ELO | 26,5 | 61,5 | - | 54 | 53 | 173 | 44 | 183 | 195,0 |
| 1400 ELO | 38 | 46,5 | 46 | - | 58 | 171 | 35 | 194 | 188,5 |
| 1000 ELO | 22 | 31,5 | 47 | 42 | - | 121 | 43 | 236 | 142,5 |

Tabelle 5.3: Turnierergebniss - Suchtiefe: 4 Halbzüge

Suchtiefe 4: [Abb. 5.3] Mit einer Suchtiefe von vier Halbzügen bietet sich ein ähnliches Bild. Die unerwartet gute Position der aus 1200 ELO gelernten Bewertungsfunktion ergibt sich durch die vielen Punkte im Spiel gegen ELO 1000 und dadurch das ELO 1600 gegen das Original sehr deutlich verloren hat. ELO 1600 besiegt ELO 1200 im direkten Vergleich. Hohe Suchtiefen führen generell zu einem geringeren Einfluss der Bewertungsfunktion. Auch ist die hier verwendete Tiefe weit größer als diejenige welche als Basis für den Lernprozess verwendet wurde. Dies führt ebenfalls zu Verzerrungen.

## 5.3 Test auf einem Internet Schachserver

Für dieses Experiment spielten die vier unterschiedlichen Versionen auf dem kostenlosem FICS-Schachserver[3]. Computeraccounts sind dort mit Kennzeichnung und Autorisierung der Server Administratoren gestattet. Es wurden 4 Accounts eingerichtet: DATSCPI spielt mit der von den 1000 ELO Spielern gelernten Bewertungsfunktion. DATSCPII nutzt die 1200 ELO, DATSCPIII die 1400 ELO und DATSCPIV die 1600 ELO Evaluationsfunktion.

Getestet wurde in zwei Bedenkzeitkategorien:
Blitz: 3 b.z.w. 5 Minuten Bedenkzeit oder 2 Minuten + 12 Sekunden pro Zug. - etwa 1000 Spiele pro Suchtiefe
Standard: 15 b.z.w 20 Minuten Bedenkzeit. - etwa 500 Spiele pro Suchtiefe

Das Programm ICSDrone[4] wäre für diese Aufgabe eigentlich prädestiniert aber leider wird es seit 1991 nicht mehr gepflegt und der Programmcode ist mit aktuellen C-Compilern nicht mehr Kompatibel. Eine Überarbeitung und Aktualisierung des Programmes wurde im Rahmen dieser Arbeit als zu aufwändig erachtet. Die Durchführung des Experimentes erfolgte daher mit Winboard 2.6. Gesteuert wurde dieses über Kommandozeilenparameter. Im Anhang [A.2] findet sich eine Übersicht über die verwendeten Befehle.

Getestet wurde auch hier mit zwei verschiedenen Suchtiefen. Zuerst mit 1 Halbzug, da nicht gegen eine andere Version von TSCP gespielt wurde gab es auch keine Probleme mit der Stellungswiederholung. Anschließend mit zeitbegrenzter Suchtiefe, das bedeutet, dass lediglich die noch verfügbare Bedenkzeit Einfluss auf den Umfang der Suche hat, die durchschnittliche Suchtiefe bei dieser Version lag etwa bei 6 ply. Es sei hierzu noch angemerkt, dass eine nicht unerhebliche Anzahl von Siegen durch das Ablaufen der gegnerischen Uhr oder durch Verbindungsabbrüche zustande gekommen ist.

[3] Free Internet Chess Server - http://www.freechess.org/
[4] http://sourceforge.net/projects/icsdrone/

Die in den Abbildungen 5.4, 5.5, 5.6 und 5.5 gezeigten ELO-Werte und Standardabweichungen $\sigma$ stammen aus den Berechnungen des Schachservers. Die Unterschiede waren in allen Fällen gering und hielten sich relativ eng im Rahmen der Standardabweichung. Daher lassen sich abgesehen von der relativen Schwäche der 1000 ELO Version aus diesem Experiment keinerlei Aussagen über die Unterschiede in der Spielstärke der einzelnen Evaluationsfunktionen machen.

| Rang | Engine | Rating | $\sigma$ |
|---|---|---|---|
| 1. | DATSCPII | 1143 | $\pm$15.1 |
| 2. | DATSCPIII | 1122 | $\pm$14.6 |
| 3. | DATSCPIV | 1105 | $\pm$14.9 |
| 4. | DATSCPI | 1088 | $\pm$15.1 |

Tabelle 5.4: FICS Wertung - Blitz - Suchtiefe: 1 Halbzug

| Rang | Engine | Rating | $\sigma$ |
|---|---|---|---|
| 1. | DATSCPIII | 1397 | $\pm$20.0 |
| 2. | DATSCPIV | 1391 | $\pm$20.4 |
| 3. | DATSCPII | 1383 | $\pm$19.1 |
| 4. | DATSCPI | 1348 | $\pm$19.7 |

Tabelle 5.5: FICS Wertung - Standard - Suchtiefe: 1 Halbzug

| Rang | Engine | Rating | $\sigma$ |
|---|---|---|---|
| 1. | DATSCPIV | 1898 | $\pm$28.9 |
| 2. | DATSCPIII | 1891 | $\pm$28.7 |
| 3. | DATSCPII | 1874 | $\pm$30.5 |
| 4. | DATSCPI | 1839 | $\pm$28.1 |

Tabelle 5.6: FICS Wertung - Blitz - Suchtiefe: zeitbegrenzt

| Rang | Engine | Rating | $\sigma$ |
|---|---|---|---|
| 1. | DATSCPII | 2075 | $\pm$50.5 |
| 2. | DATSCPIII | 2043 | $\pm$40.2 |
| 3. | DATSCPIV | 2029 | $\pm$44.0 |
| 4. | DATSCPI | 2003 | $\pm$46.7 |

Tabelle 5.7: FICS Wertung - Standard - Suchtiefe: zeitbegrenzt

# Kapitel 6

# Zusammenfassung und Ausblick

Rückblickend auf die vorangegangenen Kapitel lässt sich sagen, dass es möglich ist, das Lernen von Variablen für Schach Bewertungsfunktionen als Ranking Problem zu betrachten und unter Verwendung einer Support-Vector-Machine zu lösen. Das Ziel mit diesen Funktionen verschiedene Spielstärken zu simulieren konnte jedoch nicht erreicht werden. Die Unterschiede in den gelernten Funktionen und den erreichten Spielstärken sind zu gering um hier von einem Erfolg sprechen zu können. Dafür sind unterschiedliche Gründe denkbar.

Zum einen wäre es möglich, dass die für die Bewertungsfunktion verwendeten Parameter zu einfach waren und daher die Unterschiede zwischen den einzelnen Spielstärken nicht ausreichend abgebildet werden konnten. Es ist anzunehmen das komplexere Evaluationsfunktionen nicht nur bei der Verbesserung des Spieles sondern auch bei der Approximation von schlechtem Spiel, durch die höhere Anzahl an lernbaren Variablen, erfolgversprechender sind. Wahrscheinlicher ist aber, dass die möglichen Gründe aus denen jemand „schlecht“ spielt zu vielfältig sind um aus einer Menge von Protokollen gelernt zu werden. Möglicherweise ist es erfolgreicher sich darauf zu beschränken das Spielverhalten einzelner Personen zu modellieren.

Auch sind die gelernten Piece-Square-Tables an einigen Stellen sehr unzuverlässig so das eine mögliche Verbesserung in der Verwendung relevanzskalierter Werte liegen könnte.

Die beobachteten Veränderungen der Spielstärke durch unterschiedliche Bewertungsfunktionen lassen jedoch vermuten, dass die zugrunde liegende Annahme falsch und der Einfluss der Bewertungsfunktionen auf die Spielstärke zu gering ist. Veränderungen an der Such-Komponente, welche es ermöglicht Teile des Spielbaumes zu „übersehen“, wären möglicherweise besser geeignet das Verhalten menschlicher Schachspieler zu simulieren.

Eine ähnliche Idee wäre es die Bewertung mit Zufallszahlen zu modifizieren. Durch Veränderungen des Intervalls, aus welchem diese Werte gezogen werden, ließe sich ein Effekt des „Übersehens“ erzeugen. Problematisch wäre hierbei, dass bei der nächsten Auswertung andere Modifikatoren für die selbe Stellung gewählt und die bisher verfolgte Strategie auf den Kopf gestellt werden könnte.

# Literaturverzeichnis


[1] D.F. Beal and M.C. Smith. Learning piece-square values using temporal differences. 22(4):223–235, December 1999.

[2] Casper Willestofte Berg and Hans Gregers Petersen. A simplex approach for the tuning of a chess evaluation function. January 2006.

[3] M. Buro D. Gomboc, T. A. Marsland. Evaluation function tuning via ordinal correlation. In *The Advances in Computer Games Conference*, volume 10, pages 1–18, 2003.

[4] T. A. Marsland D. Gomboc, M. Buro. Tuning evaluation functions by maximizing concordance. In *Theoretical Computer Science*, volume 349 (2), pages 202–229, 2005.

[5] Monty Newborn David Levy. *How Computers play chess.* Computer Science Press, 1990.

[6] Dr. Joe Otim Dramiga. *Eine kurze Kulturgeschichte des Schachspiels.* Verteilt durch Satranç Club 2000.

[7] Sacha Droste. Lernen von evaluierungsfunktionen für schachvarianten. Master's thesis, TU-Darmstadt, 2008.

[8] Nilsson J. Nils. *Principles of Artificial Intelligence.* Springer Verlag, 1982.

[9] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel*

*Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[10] T. Joachims. Estimating the generalization performance of a SVM efficiently. In *International Conference on Machine Learning*, pages 431–438, San Francisco, 2000. Morgan Kaufman.

[11] T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 133–142, 2002.

[12] Andrew Tridgell Jonathan Baxter and Lex Weaver. Experiments in parameter learning using temporal difference. July 1998.

[13] H. Kamishima, T. Kazawa and S. Akaho. Unsupervised ordering - an empirical survey. In *Proceedings of The 5th IEEE International Conference on Data Mining*, pages 674–676, 2005.

[14] Wolfgang Kantschik. *Genetische Programmierung und Schach*. PhD thesis, Universität Dortmund, 2006.

[15] Graham Kendall and Glenn Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pages 995–1002. IEEE Press, 2001.

[16] D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. In *Artical Intelligence 6*, number 4, pages 293–326, 1975.

[17] R Levinson and G Ellis. Adaptive pattern oriented chess. In *Proceedings of AAAI-91*, pages 601–605. Morgan-Kaufman, 1991.

[18] Jonathan Levitt. *Genius in Chess*. Batsford Ltd, 1997.

[19] J.P. Lewis. A short svm (support vector machine) tutorial. http://scribblethink.org/Work/Notes/svmtutorial.pdf, 2004.

[20] Henk Mannen. Learning to play chess using reinforcement learning with database games. Master's thesis, Utrecht University, 2003.

[21] Ernesto Tapia Marte Ramírez Ketill Gunnarsson Erik Cuevas Daniel Zaldivar Marco Block, Maro Bader and Raúl Rojas. Using reinforcement learning in chess engines. 2008.

[22] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations.

[23] P. Mysliwietz. *Konstruktion und Optimierung von Bewertungstfunktionen beim Schach*. PhD thesis, Universität Paderborn, 1994.

[24] H. Nasreddine, H. S. Poh, and G. Kendall. Using an evolutionary algorithm for the tuning of a chess evaluation function based on a dynamic boundary strategy. In *Proc. IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–6, 7–9 June 2006.

[25] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[26] Joshi Nishant. Using genetic algorithms to learn evaluation coefficients in chess playing programs.

[27] M. Koppel O. David-Tabibi and N. Netanyahu. Genetic algorithms for mentor-assisted evaluation function optimization. In *Annual conference on Genetic and Evolutionary computation*, volume 10, pages 1469–1475. ACM, 2008.

[28] Claude E. Shannon. Programming a computer for playing chess. In *Philosophical Magazine, Ser.7*, volume 41, March 1950.

[29] John Shawe-Taylor and Nello Cristianini. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[30] Phillip H. Sherrod. Introduction to support vector machine (svm) models. http://www.dtreg.com/svm.htm, 2009.

[31] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 1999.

# Anhang A

# Anhang

## A.1 Numerische Werte

### A.1.1 Figuren- und Sonderwerte

| **Figur** | 1000 ELO | 1200 ELO | 1400 ELO | 1600 ELO |
|---|---|---|---|---|
| Bauer | 126 | 127 | 132 | 131 |
| Pferd | 253 | 246 | 231 | 238 |
| Läufer | 237 | 236 | 226 | 231 |
| Turm | 332 | 338 | 313 | 323 |
| Dame | 520 | 519 | 517 | 546 |

Tabelle A.1: Gelernte Figurenwerte

| **Sonderwert** | 1000 ELO | 1200 ELO | 1400 ELO | 1600 ELO |
|---|---|---|---|---|
| Doppelbauer | -46 | -45 | -38 | -40 |
| Isolierter Bauer | -6 | -9 | -12 | -13 |
| Rückständiger Bauer | -7 | -5 | -6 | -5 |
| Freibauer | 5 | 6 | 2 | 5 |
| Offene Linie | 49 | 50 | 50 | 52 |
| Halboffene Linie | 40 | 43 | 46 | 47 |
| Turm auf 7ter Reihe | 39 | 33 | 39 | 47 |

Tabelle A.2: Gelernte Sonderwerte

### A.1.2 Piece Square Tables - 1000 ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | 78 | 68 | 111 | 132 | 48 | 167 | 50 |
| 66 | 12 | 81 | 71 | 106 | 103 | 74 | 22 |
| 11 | 29 | 34 | 25 | 38 | 36 | 25 | 4 |
| -45 | 17 | 3 | 16 | -1 | 19 | -10 | -46 |
| -11 | -3 | 21 | -1 | -1 | 19 | 10 | 14 |
| 15 | 17 | -10 | -96 | -83 | 15 | 39 | 14 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

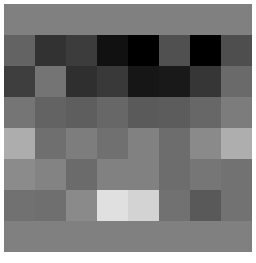


Tabelle A.3: Bauer - 1000ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -4 | 46 | -11 | 35 | 51 | 11 | 0 6 | -35 |
| -24 | -4 | -8 | 44 | 42 | 12 | -47 | -46 |
| -51 | 15 | 46 | 36 | 11 | 30 | 2 | -22 |
| -16 | -21 | 8 | 15 | 18 | 18 | 0 | 7 |
| -83 | -6 | -32 | -14 | 11 | -8 | -25 | -31 |
| -105 | -24 | -15 | -17 | -34 | -7 | -22 | -84 |
| -88 | -83 | -81 | -55 | -49 | -11 | -66 | -74 |
| -156 | -115 | -100 | -86 | -90 | -101 | -116 | -129 |

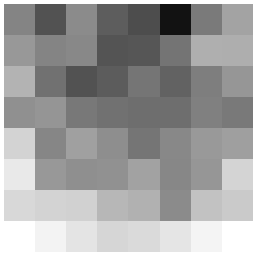


Tabelle A.4: Springer - 1000ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -41 | 12 | 44 | -16 | 27 | 30 | 27 | -37 |
| -54 | -7 | -40 | -7 | -32 | 15 | -19 | -19 |
| -52 | 16 | -19 | 16 | 27 | 24 | 9 | -27 |
| -41 | -2 | -1 | 12 | 7 | 59 | 0 | -43 |
| 0 | -44 | 24 | 8 | 6 | -7 | 0 | -33 |
| -11 | -8 | -7 | 3 | -24 | -1 | -27 | -2 |
| -28 | -7 | 1 | -51 | -28 | -57 | 28 | -26 |
| -75 | -60 | -75 | -68 | -58 | -70 | -78 | -102 |

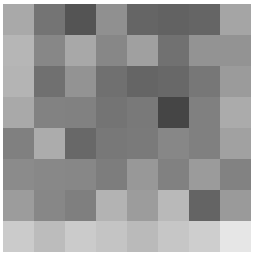


Tabelle A.5: Läufer - 1000ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -14 | -33 | -44 | 33 | -14 | -9 | -9 | -29 |
| -10 | 44 | -55 | 78 | 11 | 0 | -9 | 29 |
| -29 | 54 | 18 | -14 | -44 | 29 | 58 | 29 |
| -24 | -25 | 59 | -29 | -29 | 23 | -84 | -14 |
| -48 | 9 | -65 | -44 | -16 | -47 | -16 | -44 |
| -92 | -39 | -134 | -131 | -136 | -112 | -107 | -57 |
| -49 | -47 | -63 | -151 | -158 | -84 | -41 | -80 |
| -116 | -15 | -23 | -155 | -91 | -133 | -17 | -95 |

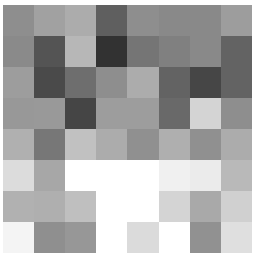


Tabelle A.6: König - 1000ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -9 | -21 | 56 | 6 | -11 | -30 | 36 | -30 |
| -14 | 88 | 113 | 45 | 22 | 128 | 92 | 68 |
| 57 | 52 | 103 | 99 | 79 | 146 | 133 | 50 |
| -9 | 80 | 138 | 118 | 89 | 118 | 83 | 24 |
| 37 | 104 | 70 | 104 | 85 | 76 | 78 | 11 |
| 7 | 36 | 75 | 85 | 67 | 54 | 27 | -37 |
| -59 | 25 | 34 | 17 | 0 | 18 | 12 | -40 |
| -103 | -36 | 2 | -29 | -44 | -49 | 2 | -72 |

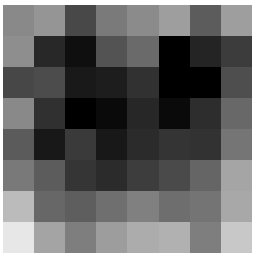


Tabelle A.7: König im Endspiel - 1000ELO

### A.1.3 Piece Square Tables - 1200 ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 81 | 104 | 68 | 113 | 106 | 31 | 65 | 51 |
| 49 | 75 | 42 | 70 | 81 | 62 | 38 | 66 |
| 7 | 22 | 17 | 25 | 39 | 41 | 28 | 17 |
| -34 | 15 | 0 | 7 | 7 | 13 | -2 | -40 |
| 9 | 0 | 10 | -11 | -2 | 13 | 4 | 17 |
| 9 | 14 | -22 | -80 | -60 | 13 | 30 | 17 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

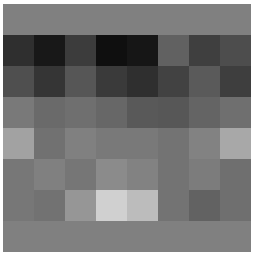


Tabelle A.8: Bauer - 1200ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -5 | 12 | 4 | 21 | 28 | 41 | 27 | -30 |
| -37 | 4 | 34 | 11 | 41 | 27 | -24 | -30 |
| -18 | 25 | 5 | 11 | 16 | 40 | -21 | -21 |
| -57 | -27 | 19 | 29 | 20 | 26 | -2 | -12 |
| -74 | -4 | -18 | 0 | -2 | -7 | -2 | -47 |
| -94 | -30 | -16 | -7 | -12 | -2 | -20 | -88 |
| -88 | -75 | -37 | -44 | -41 | -20 | -31 | -60 |
| -95 | -102 | -83 | -74 | -60 | -78 | -109 | -105 |

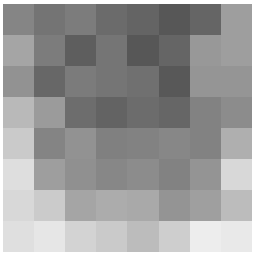


Tabelle A.9: Springer - 1200ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -22 | -4 | -3 | 27 | 27 | 7 | -3 | -20 |
| -62 | -8 | -22 | -19 | 13 | -1 | -4 | -33 |
| -27 | -20 | -1 | 20 | 6 | 28 | -18 | 8 |
| -28 | -15 | -12 | 17 | 5 | 15 | -4 | -45 |
| -25 | -15 | 17 | 4 | 23 | -4 | -7 | -29 |
| -29 | 2 | -10 | 2 | -4 | -6 | -20 | -26 |
| -40 | -1 | -3 | -38 | -15 | -58 | 28 | -27 |
| -59 | -54 | -61 | -42 | -72 | -64 | -95 | -65 |

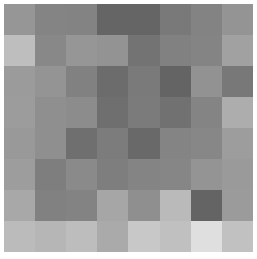


Tabelle A.10: Läufer - 1200ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | -25 | 28 | 0 | 4 | 12 | -25 | -12 |
| -12 | 32 | 16 | 15 | 20 | 25 | 0 | -12 |
| -28 | 0 | 42 | -51 | -37 | 38 | 0 | 20 |
| -12 | -25 | -37 | 4 | -37 | 0 | 20 | -38 |
| -21 | -83 | -79 | -21 | -79 | -37 | -46 | -95 |
| -102 | -42 | -145 | -109 | -104 | -106 | -78 | -115 |
| -87 | -43 | -104 | -154 | -162 | -94 | -58 | -85 |
| -113 | -55 | -55 | -167 | -104 | -152 | -32 | -95 |

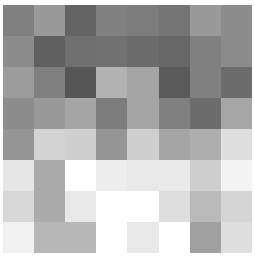


Tabelle A.11: König - 1200ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -37 | 74 | -12 | 23 | 49 | 17 | 16 | -47 |
| 15 | 107 | 81 | 75 | 75 | 81 | 106 | 21 |
| 65 | 133 | 139 | 133 | 139 | 133 | 107 | 64 |
| 74 | 116 | 132 | 134 | 116 | 111 | 93 | 77 |
| 36 | 74 | 96 | 102 | 103 | 85 | 53 | 30 |
| -8 | 39 | 62 | 58 | 58 | 61 | 35 | -22 |
| -21 | 14 | 9 | 9 | 9 | 9 | 4 | -29 |
| -72 | -43 | -23 | -48 | -48 | -48 | -14 | -87 |

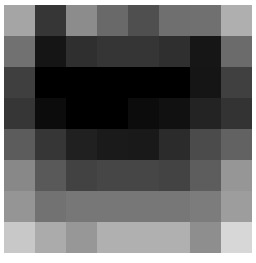


Tabelle A.12: König im Endspiel - 1200ELO

### A.1.4 Piece Square Tables - 1400 ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 63 | 61 | 67 | 80 | 92 | 81 | 82 | 65 |
| 43 | 58 | 32 | 74 | 66 | 70 | 50 | 53 |
| -3 | 21 | 9 | 25 | 31 | 33 | 16 | 5 |
| -34 | 12 | 0 | 8 | 3 | 16 | -7 | -43 |
| 12 | 0 | 17 | -13 | -6 | 9 | 0 | 9 |
| 12 | 12 | -22 | -80 | -65 | 3 | 23 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

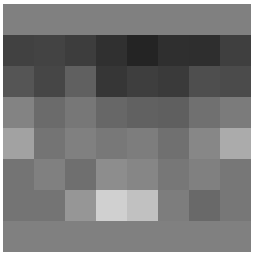


Tabelle A.13: Bauer - 1400ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -2 | -8 | -3 | 32 | 39 | 11 | 36 | 0 |
| -33 | -7 | 26 | 29 | 32 | 31 | -23 | -38 |
| -28 | 1 | 27 | 25 | 29 | 48 | 24 | -2 |
| -18 | -31 | 22 | 33 | 15 | 30 | -1 | -7 |
| -60 | -2 | -16 | 8 | -1 | 0 | 3 | -44 |
| -94 | -24 | -7 | -6 | -7 | 2 | -7 | -85 |
| -60 | -55 | -41 | -41 | -41 | -35 | -48 | -36 |
| -111 | -94 | -71 | -60 | -51 | -60 | -94 | -84 |

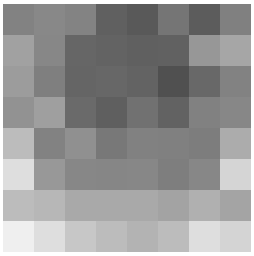


Tabelle A.14: Springer - 1400ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -12 | -21 | -15 | 0 | -18 | -11 | -7 | -17 |
| -49 | -15 | -21 | 0 | -9 | -19 | -15 | -31 |
| -46 | 3 | 27 | 8 | 22 | 31 | 4 | 5 |
| -29 | -15 | -1 | 15 | 8 | 21 | -5 | -42 |
| -31 | -4 | 4 | 4 | 3 | -5 | -2 | -24 |
| -15 | 2 | -12 | 4 | -5 | -2 | -21 | -20 |
| -8 | 3 | 2 | -38 | -8 | -25 | 22 | -34 |
| -50 | -49 | -58 | -50 | -74 | -58 | -66 | -67 |

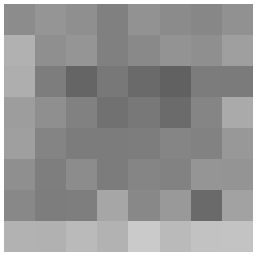


Tabelle A.15: Läufer - 1400ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -11 | 0 | -13 | -11 | 0 | -13 | -11 | -11 |
| 0 | -13 | 26 | -13 | -26 | 15 | 0 | 11 |
| -47 | 39 | -23 | 39 | -23 | -47 | 19 | -67 |
| -9 | -14 | 141 | -11 | -11 | -94 | -24 | -71 |
| -67 | -84 | -36 | -62 | -98 | -30 | -34 | -36 |
| -90 | -35 | -136 | -115 | -94 | -101 | -76 | -88 |
| -86 | -37 | -83 | -136 | -145 | -82 | -48 | -82 |
| -90 | -37 | -46 | -158 | -92 | -129 | -29 | -82 |

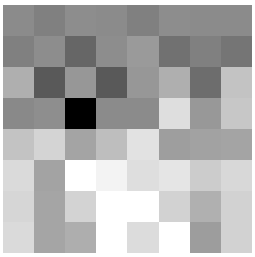


Tabelle A.16: König - 1400ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 23 | 32 | 34 | 19 | 0 | 47 | 44 | -14 |
| 47 | 85 | 99 | 68 | 75 | 128 | 93 | 38 |
| 62 | 105 | 128 | 88 | 129 | 121 | 104 | 68 |
| 52 | 105 | 118 | 126 | 127 | 90 | 80 | 35 |
| 15 | 68 | 102 | 108 | 91 | 80 | 41 | 24 |
| -8 | 48 | 65 | 68 | 61 | 51 | 27 | -25 |
| -31 | 15 | 15 | 15 | 17 | 15 | 5 | -34 |
| -84 | -37 | -22 | -44 | -38 | -37 | -20 | -82 |

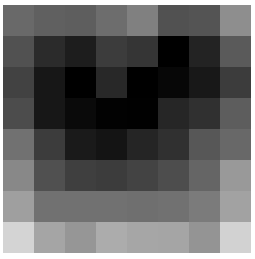


Tabelle A.17: König im Endspiel - 1400ELO

### A.1.5 Piece Square Tables - 1600 ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 61 | 63 | 59 | 94 | 94 | 97 | 71 | 54 |
| 30 | 43 | 54 | 71 | 87 | 77 | 75 | 49 |
| 8 | 21 | 10 | 26 | 40 | 35 | 23 | 7 |
| -31 | 16 | -4 | 12 | 8 | 15 | -7 | -39 |
| 11 | 1 | 19 | 0 | 0 | 7 | 7 | 9 |
| 11 | 16 | -27 | -63 | -58 | 0 | 25 | 9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

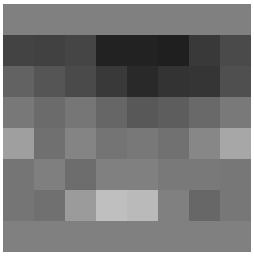


Tabelle A.18: Bauer - 1600ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -10 | -13 | -7 | 20 | 0 | 26 | 21 | 32 |
| -72 | -14 | 11 | 13 | 28 | -3 | 2 | -39 |
| -50 | -10 | 27 | 22 | 40 | 53 | 32 | -4 |
| -26 | -24 | 14 | 27 | 20 | 38 | -8 | 5 |
| -62 | -8 | -17 | 0 | 0 | -1 | 5 | -50 |
| -90 | -26 | -13 | -5 | 0 | -3 | -4 | -87 |
| -75 | -66 | -41 | -37 | -40 | -35 | -40 | -47 |
| -94 | -90 | -76 | -66 | -56 | -53 | -98 | -93 |

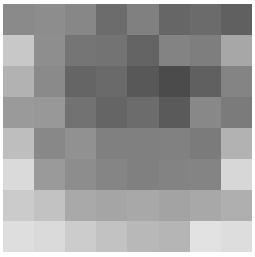


Tabelle A.19: Springer - 1600ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -41 | -21 | 2 | 10 | -1 | -25 | -13 | -50 |
| -40 | -16 | 3 | -17 | -27 | 2 | -33 | -40 |
| -42 | 11 | -2 | 14 | 23 | 18 | -1 | 3 |
| -37 | -15 | 16 | 17 | 12 | 11 | 3 | -37 |
| -31 | -3 | 4 | 19 | 12 | 3 | -12 | -28 |
| -11 | 4 | -4 | 4 | 3 | 10 | -10 | -22 |
| -6 | 9 | 4 | -32 | -5 | -22 | 30 | -40 |
| -48 | -47 | -49 | -42 | -50 | -58 | -69 | -60 |

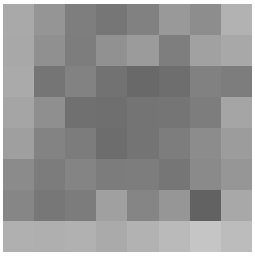


Tabelle A.20: Läufer - 1600ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | -23 | -24 | -35 | -11 | 0 | 23 | 0 |
| -12 | 0 | 95 | 20 | -11 | 11 | 0 | 0 |
| -33 | 34 | -13 | -11 | -32 | -9 | 11 | -23 |
| -74 | 19 | -11 | 11 | -35 | -4 | 0 | -57 |
| -21 | -41 | -16 | -98 | -61 | -127 | -109 | -127 |
| -94 | -104 | -119 | -95 | -100 | -114 | -73 | -89 |
| -70 | -83 | -87 | -140 | -141 | -88 | -42 | -81 |
| -114 | -35 | -61 | -173 | -95 | -136 | -35 | -88 |

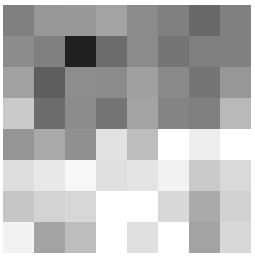


Tabelle A.21: König - 1600ELO

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 35 | 35 | -1 | 36 | 77 | 82 | 82 |
| 62 | 106 | 105 | 88 | 88 | 130 | 141 | 82 |
| 58 | 133 | 116 | 95 | 110 | 125 | 138 | 86 |
| 47 | 114 | 109 | 109 | 98 | 111 | 88 | 42 |
| 7 | 82 | 87 | 98 | 98 | 83 | 57 | 5 |
| -9 | 46 | 57 | 58 | 61 | 45 | 13 | -31 |
| -43 | 1 | 9 | 9 | 9 | 13 | 6 | -38 |
| -90 | -43 | -38 | -43 | -43 | -43 | -29 | -82 |

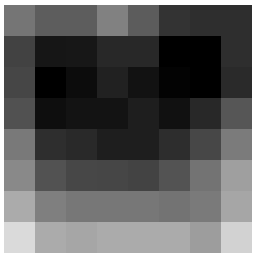


Tabelle A.22: König im Endspiel - 1600ELO

## A.2 Winboard Kommandozeilenparameter

### A.2.1 Beispiel Batchdatei

Die Schleife sorgt für eine sofortige Wiederherstellung nach Verbindungsabbrüchen.
**:loop**
**winboard.exe /zp -ics -icshost freechess.org -icshelper timeseal -fcp TSCP1400.exe -fd TSCP -icslogon fics1400.ini -zab -zippyMaxGames 3 -zippyGameEnd='resume\n seek 2 12 m\n seek 5 0 m\n seek 20 0 m\n'**
**goto loop**

### A.2.2 Details zu den Paramatern

*/zp -ics* = Versetzt Winboard in den ICS Modus und liest weitere Befehle für das Unterprogramm „Zippy“
-*icshost* = Adresse des ICS (freechess.org)
-*icshelper timeseal* = Aktiviert Timeseal (Hilfsprogramm um Spielzeitverlust durch Latenzen bei der Datenübertragung zu vermeiden)
-*fcp* = Name der zu verwendeten Schach-Engine (TSCP1400.exe)
-*fd* = Pfad der zu verwendeten Schach-Engine (TSCP)
-*icslogon* = Name der Initialisierungsdatei mit den Zugangsdaten des Computeraccounts (fics1400.ini)
-*zab* = Gestattet dem Gegner Partien abzubrechen
-*zippyMaxGames* = Maximale Anzahl an Spielen die nacheinander gegen den selben Gegner gespielt werden (3)
-*zippyGameEnd 'X'* = Zeichenfolge *X* wird nach jedem Spiel direkt an den Schachserver gesendet
*resume* = Fordert Gegner aus noch nicht abgeschlossenen Partien zur Fortsetzung auf
*seek X Y m* = Sucht neue Partie (X = Bedenkzeit [min.], Y = zusätzliche Zeit pro Zug [sec.], m = unerwünschte Gegner können abgewiesen werden [für -zippyMaxGames notwendig])

### A.2.3 Aufbau der Initialisierungsdatei

Die einzelnen Zeilen werden direkt an den Schachserver gesendet
Kontoname
Kennwort
Set 1 - Set 5 = Text der in der Beschreibung des Kontonamen stehen soll
+ Befehle an den ICS Server (*resume seek 2 12 seek 5 0 seek 20 0*)