Technische Universität Darmstadt
Knowledge Engineering Group
Hochschulstrasse 10, D-64289 Darmstadt, Germany

http://www.ke.informatik.tu-darmstadt.de

Proceedings of the 1st Poker Challenge

Johannes Fürnkranz JUFFI@KE.INFORMATIK.TU-DARMSTADT.DE
Ulf Lorenz LORENZ@MATHEMATIK.TU-DARMSTADT.DE
Frederik Janssen JANSSEN@KE.INFORMATIK.TU-DARMSTADT.DE
Sang-Hyeun Park PARK@KE.INFORMATIK.TU-DARMSTADT.DE
Eneldo Loza ENELDO@KE.INFORMATIK.TU-DARMSTADT.DE
Jan-Nikolas Sulzmann SULZMANN@KE.INFORMATIK.TU-DARMSTADT.DE

(Editors)

Preliminary

Es gibt verschiedene Aspekte, die das Spiel Poker so attraktiv machen:

- Das Spiel ist theoretisch ungelöst. Es gibt ca. 10^{18} verschiedene Spielsituationen. Um das Spiel zu lösen müsste man bestimmen, was in jeder dieser Situationen die beste Aktion ist (also Fold, Check/Call oder Bet/Raise). Das wäre theoretisch möglich (durch den Minimax-Algorithmus), ist aber praktisch nicht durchführbar.

- Beim Poker hat man zwar nur drei mögliche "Züge" in jeder Stellung (Fold, Check/Call oder Bet/Raise), aber man kennt die Spielsituation nicht komplett, da man weder alle Karten der Gegner kennt, noch weiß, welche Karten man als nächstes erhalten wird. D.h. man müßte nicht nur alle möglichen Züge, sondern auch alle möglichen Kartenverteilungen in der Suche berücksichtigen. Mittlerweile gibt es Methoden, mit denen man die Anzahl der Zustände von 10^{18} auf 10^7 reduzieren kann, wodurch die Suche schon etwas durchführbarer wird, da viele verschiedenen Spielsituationen als äquivalent behandelt werden. Da diese nicht beweisbar äquivalent sind, sind die Lösungen, die auf diesem reduzierten Raum entwickelt werden nicht beweisbar korrekt, können aber als eine gute Annäherung an die spieltheoretisch optimale Strategie betrachtet werden.

- Letztendlich ist der möglicherweise interessanteste Aspekt des Spiels, daß es mit einer theoretischen Lösung des Spiels nicht getan ist. Selbst wenn man in jeder Situation genau weiß, welche Aktion im Allgemeinen am erfolgversprechendsten ist, reicht das nicht unbedingt aus, um ein Poker-Turnier zu gewinnen. Die optimale Spiel-Strategie garantiert nur, daß man nicht schlechter spielt, als es in der momentanen Situation im Durchschnitt über alle möglichen Verteilungen der Karten möglich ist. Sie garantiert aber nicht, daß man seinen Gewinn maximiert. Dazu ist es notwendig, daß man seine Spielstrategie an die Eigenheiten der Strategien der Gegner anpasst.

- Ein weiterer Aspekt ist das "Opponent Modeling". Hier ist es sogar so, daß Spieler oft bewußt von der theoretisch richtigen Aktion abweichen, um Ihre Gegner zu bluffen. Ein wesentlicher Teil eines guten Poker-Spielers, ist einerseits, daß er nicht erkennen läßt, ob er blufft oder nicht (das berühmte Poker-Face), aber andererseits in der Lage ist, einzuschätzen, ob sein Gegner blufft oder nicht. D.h., er muß ein Modell seiner Gegenspieler erstellen, das im helfen kann, die Aktionen der Gegner besser einzuschätzen und diese Information in die eigenen Entscheidungen einfliessen zu lassen.

Die vorliegenden Proceedings sind eine Zusammenfassung der Seminarausarbeitungen der teilnehmenden Studenten. Ziel des Seminars war zu ergründen, was der aktuelle Stand der Forschung ist, sowie die vorliegenden Arbeiten auf ihre Tauglichkeit zum Programmieren eines Computerspielers zu beleuchten. Letzendlich soll im anschließenden Praktikum ein Computerspieler von den Studenten implementiert werden.

Contents

Limit Texas Hold'em - Einführung in Konzepte menschlicher Strategie

Hendrik Schaffer HENDRIK.SCHAFFER@STUD.TU-DARMSTADT.DE

Oliver Uwira OLIVER.UWIRA@STUD.TU-DARMSTADT.DE

Abstract

In folgendem Artikel wollen wir versuchen einen einleitenden Überblick über menschliche Strategien zu geben.

Wir definieren zunächst fundamentale Grundlagen, wie bspw. Odds und Outs, die für das Spielverständnis von großer Wichtigkeit sind. Anschließend konzentrieren wir uns auf die Preflop-Phase und stellen dort mit dem Prinzip der Pot Equity ein Konzept vor, mit dem man seine Starthände nahezu optimal bewerten und spielen kann, was für die späteren Straßen eine solide Basis bietet. Danach betrachten wir ansatzweise die verschiedenen Straßen im Texas Hold'em, nämlich den Flop, den Turn und den River und geben beim Riverplay Denkanstöße, wie man hier zu einer Entscheidung kommen kann.

Im abschließenden Teil stellen wir dann noch Deception (Täuschung) als wichtige Komponente korrekten Spiels vor. Hier werden Bluffing, Slow Play und Semi-Bluffing betrachtet.

Im Anhang findet sich eine kurze Einleitung zu Techniken des Hand Reading sowie ein Glossar der in dieser Arbeit verwendeten Begriffe des Pokerjargons.

1. Allgemeine Grundlagen

1.1 Expected Value

Der **Erwartungswert** (Expected Value) dient uns auch im Pokern als Grundlage für unsere Entscheidungen. Wir wählen dementsprechend immer die Strategie, die unseren Erwartungswert maximiert.

Als Beispiel, wie der Erwartungswert unsere Entscheidung bestimmt, nehmen wir folgendes Spiel an. Spieler A setzt \$1 und würfelt einmal mit einem fairen Würfel. Würfelt er ein 5 oder 6, so gewinnt er \$5, ansonsten verliert er seinen Einsatz. Sollte er dieses Spiel spielen?

Der Erwartungswert des Spielers ist $EV = (-\$1) * \frac{4}{6} + \$5 * \frac{2}{6} = 1$.

Das Gesetz der großen Zahlen garantiert Spieler A, dass unser Gewinn gegen \$1 pro Spiel konvergiert, je öfter wir dieses Spiel spielen. Spieler A sollte dieses Spiel also spielen, auch wenn er aufgrund der Varianz zunächst sogar in Minus kommen könnte. *Gewinn* oder *Verlust* eines einzelnen Spiels haben keinen Einfluss auf den langfristen Vorteil, den Spieler A durch Spielen dieses Spiels erhält.

1.2 The Fundamental Theory of Poker

Poker ist, wie alle Kartenspiele, ein Spiel mit unvollkommener Information. Solche Spiele unterscheiden sich dadurch grundlegend von Spielen mit vollkommener Information wie bspw. Schach oder Dame, in denen die Aktionen der Gegner sichtbar sind, wodurch sich eine optimale Strategie bestimmen lässt.

Würden in einer Pokerhand nun die Karten der Spieler offenliegen, so ließe sich eine mathematisch optimale Strategie für jeden der Spieler bestimmen. Wiche ein Spieler von dieser Strategie ab, würde sich sein Erwartungswert reduzieren und gleichzeitig der Erwartungswert der Gegner steigen.

Auf Grundlage dieser Beobachtung formulierte David Sklansky in [Sklansky99] einen Fundamentalsatz des Pokers:

The Fundamental Theorem of Poker *Wann immer ein Spieler eine Hand unterschiedlich von der Art und Weise spielt, wie er sie bei Ansicht der Karten aller Spieler spielen würde,* ***gewinnen die Gegner****; und wann immer ein Spieler eine Hand auf die gleiche Art und Weise spielt, wie er sie bei Ansicht der Karten aller Spieler spielen würde,* ***verlieren die Gegner****.*

Umgekehrt, wann immer die Gegner ihre Hände unterschiedlich von der Art und Weise spielen, wie sie sie bei Ansicht der Karten des Spielers spielen würden, ***gewinnt der Spieler****; und wann immer die Gegner ihre Hände auf die gleich Art und Weise spielen, wie sie sie bei Ansicht der Karten des Spielers spielen würden,* ***verliert der Spieler****.*

Unter *Gewinnen* und *Verlieren* ist hierbei eine Veränderung des Erwartungswertes zu verstehen. Das Gewinnen oder Verlieren einzelner Hände spielt, wie im Abschnitt Expected Value erläutert,

für die Beurteilung einer Strategie keine Rolle. Die Maximierung des Erwartungswertes ist das einzig relevante Kriterium.

1.3 Odds und Outs

Draws oder Drawing Hands sind unvollständige Hände und benötigen "Hilfe", um die Hand gewinnen zu können. Die zentrale Fragestellung ist also, wann man seine Draws profitabel spielen kann.

Diese Frage lässt sich durch das mathematische Konzept der Odds & Outs beantworten.

- Als **Outs** bezeichnen wir alle Karten, die unsere Hand verbessern
- Die **Odds** geben an, wie hoch die Wahrscheinlichkeit ist, dass man eines seiner Outs trifft

  Sie lassen sich nach folgender Formel berechnen: $Odds = \frac{nichthilfreicheKarten}{hilfreicheKarten}$
- **Pot Odds** beschreiben das Verhältnis von möglichen Gewinn zu zu zahlendem Einsatz.

  Sie berechnen sich als: $Pot\ Odds = \frac{moeglicherGewinn}{zuzahlenderEinsatz}$
- Vergleich von Pot Odds und Odds führt zu einer Entscheidung:
  - wenn **Pot Odds < Odds**, dann Fold
  - wenn **Pot Odds >= Odds**, dann ist ein Call profitabel(in speziellen Situationen kann man auch raisen)

Beispiel: Wir haben 3♣ 2♡ (32 "offsuit") auf einem Board von K♠ 4♣ 5♢ (ein **"Rainbow-Board"**). Jedes der vier Asse und jede der vier Sechsen gibt uns eine Straße, so dass wir hier insgesamt **acht Outs** haben.

Unsere Odds sind $\frac{39}{8} = 4,87$

Angenommen der Pot würde am Flop 5 SB betragen und unser Gegner bettet 1 SB. Dann befinden sich im Pot insgesamt 6 SB und wir müssten einen SB zahlen um zu callen. Wir bekommen für unseren Call also Pot Odds von 6:1. Da wir aber Odds von ca. 1:5 haben, ist hier ein Call für uns profitabel: in fünf Fällen verlieren wir $1 und in einem Fall gewinnen wir $6.
Im Anhang findet ihr eine Tabelle, in der die Odds und Outs zusammengefasst sind.

1.3.1 DISCOUNTED OUTS

Nicht jedes unserer Outs kann immer als volles Out gezählt werden. Oft müssen wir einzelne Outs abziehen(d̈iscounten)̈, da manche unserer Outs dem Gegner eine Hand geben, die unsere schlägt.

Hierzu wieder ein Beispiel: Wir haben wieder 3♣ 2♡ , aber diesmal auf einem **Two-Flush-Board** von K♠ 4♣ 5♠ . Jetzt können wir uns nicht die ganzen 8 Outs geben, da es die Möglichkeit gibt, dass einer unser Gegner ein Flush-Draw hält und ihm zwei unserer Outs einen Flush geben, der unsere Straße schlägt. Wir können uns hier also effektiv nur sechs Outs geben!

Hieran erkennt man, dass es fundamental wichtig ist, seine Outs enstsprechend zu discounten, da wir sonst Draws spielen, die auf lange Sicht einen negativen Erwartungswert haben.

1.3.2 IMPLIED ODDS

Implied Odds berücksichtigen, dass man auf späteren Straßen oft noch zusätzliche Bets gewinnt, die dazu führen können, dass man auch ohne die benötigten Pot Odds noch einen profitablen Call hat.

Nehmen wir beispielsweise an, dass wir mit einem Flush-Draw am Flop sind und unser Gegner eine starke Hand hat, die aber gegen unseren Flush verlieren würde. Da der Gegner sich nicht hundertprozentig sicher sein kann, dass wir auch wirklich einen Flush-Draw haben, wird er auch am River oft noch eine Bet callen, selbst wenn der Flush angekommen ist, so dass wir auf hier noch eine zusätzliche Bet gewinnen.

1.4 Position

Die Position (in der Betreihenfolge) ist ein weiterer wichtiger Aspekt, den wir beim Treffen einer Entscheidung berücksichtigen müssen. Zunächst sei gesagt, dass es immer von Vorteil ist, wenn wir Postition auf einen Gegner haben, also nach ihm agieren können, da wir diesem Gegner gegenüber einen Informationsvorsprung haben. Das spiegelt sich beispielsweise auch darin wieder, dass wir preflop in späten Posittionen wesentlich mehr Hände spielen können, als in einer frühen Position.

Andererseits sind manche Spielweise nur in bestimmten Situationen überhaupt möglich, wie zum Beispiel das Check-Raise, welches nur möglich ist, wenn wir out of Position sind.

Folgende Abbildung zeigt die verschiedenen Positionen einem Zehnertisch sowie deren Kurzbezeichnungen, die wir im folgenden häufiger verwenden werden: SB = Small Blind, BB = Big Blind, EP = Early Position, MP = Middle Position, LP = Late Position, BU = Button.

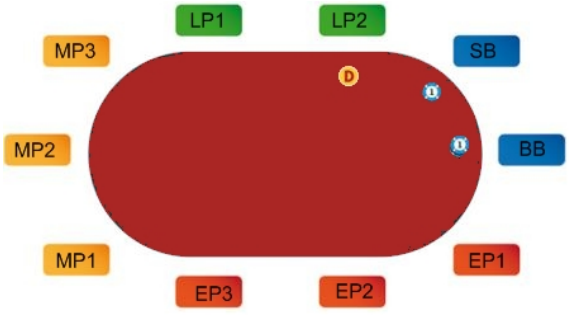

Abbildung 1: Chart: Position an einem Zehnertisch

2. Spielertypen

Wir unterscheiden beim Pokern grob vier Spielertypen:

- **Die Calling Station:**

  Die Calling Station stellt unseren liebsten Gegner dar. Sie spielt preflop in der Regel sehr **loose** (d.h. sie geht mit zu vielen Händen mit) und callt oft gegen ihre Odds, also auf minderwertigen Draws. Außerdem spielt sie sehr **passiv** und versäumt es, aus ihren Händen das Maximum an Value herauszuholen.

  Gegen eine Calling Station erhöhen wir unseren Gewinn, wenn wir mit mehr Händen für value-betten. Stoßen wir auf Widerstand, so können wir uns weiterhin oft sicher sein, gegen eine sehr starke Hand zu spielen und haben für unsere schlechteren Hände einen einfachen Fold.

- **Der Loose-Aggressive Spieler (LAG):**

  Der LAG spielt wie die Calling Station auch preflop sehr viele Hände. Allerdings ist ein LAG sehr viel aggressiver und er versucht oft durch Raises und Bets den Pot zu stehlen.

  Auch hier haben wir die Möglichkeit, mit einer etwas größeren Range unserer Hände für Value zu betten, wenngleich der LAG oft versucht trickreich zu spielen. Der Extremfall das LAGs ist der **Maniac**, der praktisch mit jeder Hand raist und dadurch versucht, seine Gegner aus dem Pot zu treiben.

- **Der Rock:**

  Der Rock ist das genaue Gegenstück zum Maniac. Er spielt preflop extrem tight, also nur die besten Starthände. Wenn der Rock im Spiel ist, dann können wir also von einer sehr starken Hand ausgehen. Ein Schwachpunkt des Rocks ist es jedoch, dass er teilweise ängstlich und mit seinen starken Händen ziemlich gradlinig spielt, so dass wir einerseits unsere schwächeren Hände wieder relativ leicht folden können und andererseits versuchen können Pots zu stehlen, in denen wir beide nichts getroffen haben, da er solche Hände in den allermeisten fällen foldet. Weiterhin sollten wir die Möglichkeit nutzen und versuchen häufig seine Blinds zu stehlen.

- **Der Tight-Aggressive Spieler (TAG):**

  Der TAG repräsentiert den Gegnertyp, dem wir selbst am nächsten kommen und kann deshalb in gewisser Weise als unser Angstgegner angesehen werden. Er spielt preflop relativ tight, also nur ausgewählte Starthände. Ist er im Pot, müssen wir immer aufpassen, da er ziemilch aggressiv und trickreich spielt und wir oft nicht mit Gewissheit sagen können, welche Hand unser Gegner hält.

3. Spielphasen

3.1 Pre-Flop

3.1.1 FUNDAMENTALES PRINZIP: POT EQUITY

Die **Pot Equity** ist ein prozentualer Wert, der die Gewinnchance unser Hand relativ zu den Händen unserer Gegner angibt (vorausgesetzt, dass alle bis zum Showdown gehen).

Hierzu ein Beispiel:

- mit **AQo** haben wir **gegen zwei zufällige Hände** eine Equity von ca. **46**%.
  Angenommen, wir raisen vor dem Flop Small Blind sowie Big Blind callen unser Raise. Dann sind insgesamt 6 Small Bets (SB) im Pot. Laut unserer Equity gehören uns davon 46% oder 2,76 SB. Da wir selbst aber nur 2 SB bezahlt haben, haben wir durch das Raise somit 0,76 SB "gewonnen".

Zunächst einmal ist das nur ein theoretischer Wert, da die Hand hier ja noch nicht zu Ende ist. Durch schlechtes Spiel, können wir von unserer Equity wieder viel verlieren. Jedoch zeigen Simulationen, dass sich dieser Equityvorteil bei angemessenem Spiel letztendlich in einem höheren Erwartungswert, die Hand auch zu gewinnen, niederschlägt!

⇒ Equityvorteil Preflop durch Raise maximal nutzen!

Wir haben zunächst immer dann einen Equityvorteil, wenn die Equity unserer Hand höher ist als die Durchschnittsequity, die sich aus $\frac{1}{AnzahlderSpieler} * 100\%$ ergibt.

- Bei einem Spieler der vor uns eingestiegen ist, liegt die durchschnittliche Equity also bei $\frac{1}{2} * 100\% = 50\%$, bei zwei vor uns eingestiegenen Gegnern bei 33%, usw.

Um dieses Wissen nun für unser Preflopspiel nutzen zu können, müssen wir zunächst zwei Fragen klären:

1. Was passiert im Durchschnitt nach mir? (Wieviele Gegner sind noch dran? Wie groß ist die Wahrscheinlichkeit für eine besser Hand?)
2. Was ist vor mir passiert? (Wieviele Spieler sind schon in den Pot eingestiegen? Was sind das für Gegner? Welche Hände spielen sie normalerweise?)

Um die erste Frage beantworten zu können schauen wir, wieviele Gegner nach uns noch dran sind und berechnen die Equity unserer Hand gegen diese Anzahl an zufälligen Händen. Daraus abgeleitet ergibt sich eine Hand Range abhängig von der Gegneranzahl hinter uns, mit der wir raisen sollten, wenn vor uns noch kein Gegner in den Pot eingestiegen ist.

In der Tabelle „Open Raising Chart", die ihr im Anhang findet haben wir diese Ranges zusammengefasst. In Abhängigkeit von unserer Position open-raisen wir mit jeder dort eingetragenen (oder besseren) Hand!

Um die zweite Frage beantworten zu können, müssen wir uns auch wieder zwei Fragen stellen:

1. Hat unsere Hand einen Equityvorteil gegen die Gegner hinter mir?
   ⇒ wir haben immer dann einen Equityvorteil gegen die Gegner hinter uns, wenn unsere Hand im ORC aufgelistet ist.

2. Hat meine Hand einen Equityvorteil gegen die Gegner vor mir?
   ⇒ sehr komplexe Frage, da sie von unseren Gegner und deren Hand Ranges abhängt. Wir müssen also versuchen die Hand Ranges der Gegner abzuschätzen und berechnen dann unsere Equity gegen diese Range

Dazu zwei Beispiele:

- Ein Gegner mit einem PFR von 10% raist in MP2. Wir sind in MP3.
  Mit welchen Händen sollten wir hier raisen, callen oder folden?

  Dazu schätzen wir jetzt die Hand Range dieses Gegners ab: in MP1 wird dieser Gegner ungefähr 8% (die -2% wegen der "schlechten" Position) seiner Hände raisen
  ⇒ das ergibt eine Range von: 88+, ATs+,KTs+,OJs+,AJo+

  Haben wir seine Range bestimmt, schauen wir, mit welchen Händen wir eine Equity von mindestens 50% haben.

  - TT (53%)
  - AKo (57%)
  - AQs (50%)

  Das heißt also, dass wir mit **allen Händen TT+,AKo+ und AQs+** gegen diesen Gegner 3-betten und die restlichen Hände folden.
  Wir callen mit KEINER Hand!

- Ein Gegner der im Schnit 30% seiner Hände callt und mit 10% dieser Hände raist, callt vor uns in MP2. Wir sitzen wieder in MP3 und fragen, mit welchen Händen wir raisen, callen oder folden sollen?

  Wir ermitteln seine Range für die 30%, ziehen davon aber die oberen 8% der Hände ab, da er mit diesen geraist hätte.
  ⇒ daraus ergibt sich seine Range hier mit: 88-55,A9s-A2s,K9s-K5s,Q7s+,J8s+,T8s+,98s,ATo-A7o,A5o,K9o+,Q9o+,J9o+,T9o

Schauen wir jetzt wieder, mit welchen Händen wir eine Equity von mindestens 50% haben:

- 55 (51%)
- A7o (52%)
- KTo (53%)
- QJo (50%)
- A2s (51%)
- K9s (51%)
- QTs (50%)

Überprüfen wir jetzt noch, welche von diesen Händen auch im Open Raising Chart eingetragen sind, stellen wir fest, dass wir mit folgenden Händen gegen diesen Gegner raisen sollten: **55+,A9o+,KJo+,QJo+,A6s+,K9s+,QTs+**

3.1.2 FEINTUNING DES EQUITY PRINZIPS

Laut dem Prinzip der Pot Equity und den oben angestellten Überlegungen sollten wir gegen einen einzelnen Raiser vor uns nur mit Händen 3-betten, die im Open Raising Chart aufgelistet sind und die eine Equity von mindestens 50% gegen seine Hand-Range haben.

Allerdings können wir in der Regel diese 50%-Anforderung etwas nach unten korrigieren. Dafür gibt es zwei Gründe: Erstens haben wir Position auf den Gegner und zweitens bringen wir durch eine 3-Bet oft beide Blinds zum Folden und haben somit "Dead Money" im Pot, das sich auf uns und den Gegner verteilt. Berücksichtigt man diese Faktoren so ergibt sich, dass man gegen einen Loose-Aggressive-Gegner (nur) mindestens 48% Equity und gegen typischen Gegner sogar nur 46% Equity benötigt um profitabel raisen zu können.

3.1.3 3-BETTING GEGEN OPEN RAISES

Stellen wir uns folgende typische Situation vor: Vor uns raist ein Gegner und wir müssen uns entscheiden, ob wir 3-betten oder folden sollen. Wir haben noch keine bzw. nur sehr wenig Informationen über diesen Gegner. Wie sollen wir uns jetzt verhalten?

Wir nehmen an, der Gegner würde genauso wie wir nach obigen Überlegungen spielen. Dann können wir wieder ausrechnen, mit welchen Händen wir gegen seinen Raise 3-betten können. In der Tabelle: „ORC 3-betting", die ihr im Anhang findet, sind alle Hände eingetragen, mit denen wir nach einem Raise 3-betten sollten. Hier bedeutend die Spalten MP2, MP3 usw. die Position aus der das Raise kam. Es ist also egal wo wir sitzen, es kommt nur darauf an, aus welcher Position der Raise kam.

3.1.4 CALLEN NACH GEGNERAKTIONEN VOR UNS

Wenn vor uns exakt ein Gegner gecallt oder geraist hat, so ist es fast nie korrekt, nur zu callen. Entweder wir raisen oder folden (so wie wir es nach obigem Equityprinzip berechnet haben)

Haben vor uns zwei oder mehr Gegner nur gecallt, raisen wir wieder alle Hände, die nach dem Equityprinzip einen Vorteil haben. Zusätzlich können wir aber hier noch mit bestimmten Händen mit guten Implied Odds nur callen. Hierzu zählen vor allem mittlere und kleine Paare und die stärkeren Suited Connectors wie T9s und 98s. Allerdings kommt es an Sechsertischen nicht oft zu solchen Situationen!

In extrem seltenen Fällen sollte man auch gegen ein Raise und mehrere Caller vor uns nur callen. Dies kommt gelegentlich mit einem Pocket Pair in Betracht.

Z.B. haben wir 66 in Late Position und vor uns sind 3 Caller und ein Raiser. Hier können wir die 2 Bets callen. (Für ein Reraise ist unsere Hand definitiv zu schwach, aber aufgrund der vielen Gegner und der guten Implied Odds ist ein Call profitabel)

3.1.5 KORREKTES SPIELEN DER BLINDS

Das Equityprinzip ist fundamental und gilt somit auch für die Blinds. Jede Hand mit einem Equityvorteil wird nach wie vor geraist oder gereraist.

Es gibt allerdings Fälle, in denen wir Hände spielen können, die wir auf einer Non-Blind-Position nicht hätten spielen können, da wir bereits einen gewissen Grundeinsatz (die Blinds) investiert haben. Damit werden natürlich sowohl Calls als auch Raises für uns günstiger, als sie es eigentlich wären. Dies hat allgemein zur Folge, dass wir in den Blinds häufig auch Hände spielen können, die weniger als die durchschnittliche Equity haben.

Wie tief können wir dann mit unseren Equityanforderungen nach unten gehen, damit eine Hand noch profitabel ist?

- **Callen im BB gegen Raises**
  - Wir raisen wieder alle Hände, mit denen wir einen Equityvorteil gegen die Ranges der Gegner haben
  - Mit den Händen, die für einen Raise zu schwach sind, deren Equity aber mindestens 70% der durchschnittlichen Equity beträgt callen wir nur.

    Im Heads-Up ist die durchschnittlichen Equity zum Beispiel 50%. 70% davon sind also 35%. Das bedeutet, wenn wir im BB sitzen, der Button raist und der Small Blind foldet, eine Hand mit mindestens 35% Equity gegen die Range des Gegners braucht.

- **Loose Isolation 3-Bets aus dem SB** (Small Blind Defense)

  Hier raisen wir wieder mit allen Händen aus dem Open Raising Chart

- **Completen im Small Blind nach Callern vor uns**
  - Wir completen im Small Blind mit allen Händen, mit denen wir auch am Button nur gecallt hätten. Zudem completen wir mit den Händen, deren Equity mindestens 70% der durchschnittlichen Equity beträgt (z. B. mit T6s nach 2 tighten Callern vor uns).

  - Falls der Small Blind $\frac{2}{3}$ einer Small Bet beträgt, completen wir mit allen Händen, die wir auch am Button gecallt hätten und zudem mit solchen Hände, deren Equity mindestens 50% der Durchschnitts-Equity beträgt.
  - Falls der Small Blind $\frac{1}{3}$ einer Small Bet beträgt, completen wir mit allen Händen, die wir auch am Button gecallt hätten.

- **Small Blind vs. Big Blind**

  - Gegen einen typischen, aggressiven Gegner gilt hier nach wie vor: Raise oder Fold. Das heißt wir raisen mit allen Händen aus dem Open Raising Chart.
  - Gegen einen passiven Gegner ist das Completen im Small Blind oft eine sehr gute Option. Ein passiver Gegner raist uns hier selten. Falls wir geraist werden, dann können wir sogar oft folden (immer dann, wenn unsere Equity gegen seine Raise Range kleiner als 35% ist). Wenn er nur checkt und es zum Flop kommt, dann betten wir JEDEN Flop!

3.2 Flop

Das Aufdecken des Flops ist grundsätzlich der prägende Moment in einer Hold'em-Hand. Durch die drei weiteren Karten, die den Spielern jetzt zusätzlich zu ihren zwei eigenen Karten zur Verfügung stehen, sind nun auch stärkerer Hände als Paare möglich.

Dies impliziert, dass sich Verhältnis im Vergleich zum Beginn der Hand preflop grundlegend geändert haben können. Während A♡ A♢ preflop gegen 10♡ 9♣ ein solider Favorit ist, verhält sich die Situation auf einem Flop 8♠ 7♣ 6♡ genau umgekehrt. Für erfolgreiches Spiel ist es daher sehr wichtig, sich adäquat an die geänderten Verhältnisse auf dem Flop anzupassen.

Insofern man auf dem Flop nicht "überhaupt nichts"hat, so kann man zwei Arten von Händen halten. Made Hands (fertige Hände) sind Hände, die mindestens ein Paar oder besser darstellen. Drawing Hands sind Hände, die noch weiter Hilfe auf dem Turn oder River benötigen, um zu einer Made Hand zu werden, dafür aber meistens eine sehr starke Made Hand wie etwas ein Flush oder eine Straße werden.

Wir beschreiben im folgenden einige typische Flopsituationen aus der Sicht eines Spielers und geben allgemein einige Handlungsrichtlinien an, ohne aus Platzgründen hierbei zu sehr ins Detail zu gehen. Dies trägt auch der Tatsache Rechnung, dass das Spiel auf dem Flop sehr komplex ist, und stark von den Begleitfaktoren Gegner und Position abhängt.

Die folgenden Abschnitte basieren auf [Ciaffone06].

3.2.1 OVERCARDS

Unter Overcards verstehen wir zwei ungepaarte Karten, die höher als die höchste Karte des Flops sind. Im Allgemeinen sind sie als Drawing Hand zu betrachten, wobei starke Overcard-Hände wie AK manchmal auch alleine gewinnen können, wenn die Gegner keine Hilfe auf Flop, Turn und River erhalten und bspw. nach Bet/Call auf dem Flop der Turn und der River gecheckt werden.

Allgemein sollte die Entscheidung, ob mit Overcards weitergespielt oder gefoldet wird, von den Pot Odds abhängig gemacht werden, da man in den meisten Fällen auf Hilfe angewiesen ist. Auf einem stark koordinierten Board, das starke Draws unter den Händen der Gegner möglich macht, ist ein Weiterspielen mit Overcards im Allgemeinen nicht empfehlenswert.

Hat man selber mit einer starken Hand wie AK preflop Stärke gezeigt, so kann man mit einer so genannten Continuation Bet versuchen, die Hand auf dem Flop zu gewinnen, falls die anderen Spieler checken.

Während es also Fälle gibt, in den man mit zwei Overcards in der Hand verbleiben kann, ist lediglich eine Overcard im Allgemeinen nicht zum weiterspielen ausreichend.

3.2.2 Top Pair und Overpair

Unter Top Pair verstehen wir, dass eine der eigenen Karten mit der höchsten Karte des Flops paart. Größte Bedeutung kommt in einer Top-Pair-Situation die Höhe der Beikarte zu, dem sogenannten Kicker. Je höher der Kicker, desto stärker die Top-Pair-Hand. Einer Top-Pair-Hand droht zudem nicht nur Gefahr durch eine andere Top-Pair-Hand mit höherem Kicker sondern auch durch Overcard-Hände, die auf dem Turn Hilfe bekommen könnte. Je niedriger also das Top Pair ist, desto gefährlicher sind Overcards, die auf dem Turn fallen. In den meisten Situationen sollte man seine Top-Pair-Hand durch Betten und damit durch Verschlechterung der Pot Odds beschützen.

Ein Overpair ist hingegen ein Pocket Pair, das höher als die höchste Karte des Flops ist. Es ist stärker als eine Top-Pair-Hand, weil sie Top Pair, Top Kicker schlägt, Mit Ausnahme von AA besteht für ein Overpair aber ebenfalls Gefahr durch eine Overcard (d.h. höher als das Overpair) auf dem Turn und sollte ähnlich einer Top-Pair-Hand gespielt werden.

3.2.3 Two Pair

Two-Pair-Hände, bei denen beide eigenen Karten mit den Flopkarten gepaart sind, sind zumeist sehr starke Hände, die sich aber selten für ein Slow Play eignen, da diese Handkategorie gegen ein Overpair in Gefahr ist, das im Falle eines offenen Paares auf dem Board die Two-Pair-Hand schlagen würde.

3.2.4 Straight Draw

Ein Straight Draw besteht aus vier Karten einer Straße, so dass das Fallen der fünften Straßenkarte auf dem Turn eine sehr starken Made Hand ergeben würde. Die Entscheidung, ob mit einem Straight Draw weitergespielt wird, hängt im Allgemeinen streng von den Pot Odds ab.

Einem sogenannten *Inside-Straight-Draw* fehlt eine Karte in der Mitte der Straße. Dieses Draw hat somit nur 4 Outs, wodurch im Allgemeinen die Pot Odds das Weiterspielen nicht rechtfertigen. Weitaus stärker ist das sogenannte *Open-End-Straight-Draw*, bei dem man schon vier Karten in Reihenfolge hält und die Straße sowohl am oberen als auch am unteren Ende vervollständigt werden kann. In diesem Fall hat das Draw 8 Outs und ist damit annähernd so mächtig wie ein Flush Draw.

3.2.5 FLUSH DRAW

Ein Flush Draw besteht aus vier Karten einer Farbe, so dass auf dem Turn die fünfte Karte in dieser Farbe den Flush vervollständigen würde. Hält man das As der Flushfarbe, so hat man meistens ein Draw auf die Nuts und kann das Draw eventuell sogar aggressiv spielen.

Gefahr droht einem Flush Draw im Falle eines offenen Paares auf dem Flop, was Full-House-Draws oder ein fertiges Full House unter den Händen der Gegner ermöglicht.

3.2.6 STARKE MADE HANDS

Made Hands, die besser als Two Pair sind, sind im Allgemeinen so stark, dass die Hauptaufgabe eines Spielers darin besteht,so viel wie möglich an Value herauszuholen, d.h. so viel Geld wie möglich in den Pot zu bekommen, an dem man die meiste Equity hält.

Dies beschützt gleichzeitig die eigene Hand gegen eventuelle Straight- oder Flush-Draws, denen durch das Value-Betting schlechtere Pot Odds gegeben werden.

Ist der Pot sehr klein, bietet sich ein Slow Play an, um auf späteren Straßen die Chance auf den Gewinn weiterer Bets zu haben.

3.3 Turn

Da das Spiel auf dem Turn sehr kompliziert ist und im allgemeinen schon stark von spieltheoretischen Überlegungen geprägt ist, wollen wir uns in diesem Abschnitt auf einige allgemeine Aussagen beschränken.

Grundsätzlich gilt der Turn als die ”Straße der Wahrheit”. Das bedeutet, dass während Spieler auf dem Flop die geringere Höhe der Bets zu Zwecken der Deception nutzen, auf dem Turn das Zeigen von Stärke zum Preis einer Big Bet zumeist auch echte Stärke bedeutet. Sind die Aktionen des Gegners auf dem Flop und auf dem Turn inkonsistent, so ist der Aktion auf dem Turn im Allgemeinen größere Aussagekraft zuzugestehen.

Wird man auf dem Turn mit Stärke konfrontiert, so sind im Allgemeinen die Pot Odds für die Entscheidung weiter zu spielen in Betracht zu ziehen.

Scheinen alle Spieler auf dem Turn nicht stark zu sein, so kommen Bluffmanöver in Betracht, da die Konfrontation mit einer Big Bet schwache Gegner zum häufig zum Folden bringen kann.

3.4 River

Da an Sechsertischen sehr selten mehr als zwei Spieler bis zum River gehen, betrachten wir in diesem Abschnitt lediglich das Riverplay Heads-Up.

3.4.1 RIVERPLAY HEADS UP

Sind wir Heads-up am River und wollen eine Entscheidung treffen, so müssen wir uns zunächst folgende Fragen stellen:

1. **Wie stark ist meine eigene Hand in Relation zum Board?**

   Wir teilen unsere Hände grob in folgende Kategorien ein:

   - Sehr schwach bis schwach: Underpair, Bottom Pair, Ace high
   - Marginal: Middle Pair, Pocket Pair
   - Stark: Top Pair, Overpair, Two Pair
   - Sehr stark: Straight, Flush, Set, Full House, Quads

2. **Wie stark ist meine eigene Hand in Relation zum Gegner?**

   Unsere Handstärke ist abhängig von unseren Gegnern bzw. deren Hand Ranges und dem Handverlauf. Hierzu wieder ein Beispiel (aus Sicht von "Hero":

   Wir sind am BU mit A♡ 4♢
   3 Folds, Hero raist, 1 Fold, BB calls

   Flop: A♣ 10♡ 3♢
   BB checkt, Hero bettet, BB callt

   Turn: 6♠
   BB checkt, Hero bettet, BB callt

   River: 8♣ [5.25BB im Pot]
   BB checkt, Hero ???

   Abhängig vom Gegner hat unsere Hand hier wahrscheinlich folgenden Wert:

   - Gegen (guten) *TAG*:

     Der Handverlauf deutet oft auf eine "Way Ahead - Way Behind"-Situation hin. Unsere Hand sollten wir daher hier als marginal bewerten.

   - Gegen *Rock*:

     Der Rock spielt sehr tight und hat einen niedrigen WTS-Wert. Es gab hier keine Draws, auf die er gecallt haben könnte, so dass wir hier in den meisten Fällen auf ein höheres Ass treffen werden. Daher ist unsere Hand gegen diesen Gegner als schwach bis sehr schwach einzustufen.

   - gegen *Calling Station*

     Die Calling Station ist passiv und hat oft einen hohen WTS. Der Gegner hat hier oft auf Gutshots oder mit Bottom Pair oder Middle Pair gecallt. Gegen diese Range des Gegners haben wir eine starke Hand.

3. **Wie hoch ist die Chance, dass unser Gegner mit einer schlechteren Hand callt?**

   Diese Frage ist schwer zu beantworten. Man muss versuchen abhängig vom Handverlauf und seinen Reads über den Gegner ein Gefühl dafür zu bekommen, wie oft ein konkreter Gegner(typ) hier callen wird.

4. **Wie hoch ist die Chance, dass unser Gegner mit einer besseren Hand callt?**

   Spielen wir gegen einen passiven Gegner, so ist die Wahrscheinlichkeit höher, dass er unsere Bet nur callen wird. Ein weiteres Entscheidungsmerkmal könnte ein sehr gefährliches Board sein, das den Gegner davon abhält uns hier noch zu raisen oder dass wir im Handverlauf klar gezeigt haben, dass wir eine Hand halten, die besser ist, als die des Gegners.

5. **Wie hoch ist die Chance, dass unser Gegner mit einer schlechteren Hand auf dem River blufft (Bluff Inducing)?**

   Auch diese Frage ist wieder schwer zu beantworten und man sollte versuchen abhängig vom Gegner(typ) ein Gefühl für solche Situationen zu bekommen. Man kann sich hier beispielsweise fragen, ob der Gegner fähig ist, seine Busted Draws als Bluff zu betten. Oft kann man aber auch durch seine eigenen Betsequenzen einen Bluff provozieren (engl. ”to induce a bluff”), indem man zum Beispiel am Turn nur einen Check Behind spielt oder am River out of Position zu meinem Gegner hin checkt.

6. **Wie hoch ist die Chance, dass unser Gegner mit einer besseren Hand raist?**

   Auch diese Frage sollte für jeden Gegner konkret betrachtet werden. Gegen passive Gegner kann man aber beispielsweise nach einem Raise am River seine schwachen Hände in der Regel guten Gewissens folden.

7. **Wie hoch ist die Chance, dass unser Gegner mit einer schlechteren Hand raist?**

   Hängt wiederum vom Gegner ab. Die Wahrscheinlichkeit ist bei aggressiven Spielern aber deutlich höher, als bei passiven Spielern.

8. **Wie hoch ist die Chance, dass unser Gegner eine schlechtere Hand foldet?**

   Falls wir in Position sind, ist diese Frage irrelevant. Ansonsten kann man sich beispielsweise überlegen, dass wenn wir out of Position sind und wir uns relativ sicher sind, dass der Gegner nach einer Bet von uns folden wird, es keinen Sinn macht in einer solchen Situation selbst zu betten, da wir dadurch nichts gewinnen können. Besser wäre es hier nur zu checken und zu hoffen, dass der Gegner selbst zu bluffen versucht und dass wir dadurch noch eine Bet gewinnen können.

9. **Wie hoch ist die Chance, dass unser Gegner eine bessere Hand foldet?**

   Diese Frage brauch man sich nur stellen, wenn der Gegner auch mitdenkt und fähig dazu ist, seine Hände auch zu folden. Wir müssen hier überlegen, welche Hände der Gegner halten

könnte und welche wir davon zum Folden bringen können. In der Regel sind das oft nicht angekommene Draws, Ace high, Overcards, eventuell auch Middle Pair, Bottom Pair und kleine Pocket Pairs

Berücksichtigt man diese Fragen, so kann man anschließend eine optimale Entscheidung treffen. Dabei haben wir folgende Spielmöglichkeiten:

- **Bet/Fold**

  Wir spielen Bet/Fold am River, um knappe Value Bets gegen passive Gegner rauszuholen, denn gegen passive Gegner können wir nach einem Raise relativ leicht folden und zahlen so maximal auch nur eine Bet, falls wir hinten liegen. Wir werden aber maximal ausbezahlt, falls wir vorne liegen. Der Nachteil dieser Strategie ist, dass wir selbst auch sehr angreifbar werden. Erkennt der Gegner nämlich, dass wir mit schwachen Händen Bet/Fold spielen, so wird er mit vielen Händen einfach bluffen und wir verlieren den ganzen Pot. Ein weiterer Nachteil von Bet/Fold ist der, dass wir dem Gegner nicht selbst die Möglichkeit zu bluffen geben und wir von einem Gegner, der auf eine Bet von uns sowieso folden wird, nichts mit unserer Bet gewinnen können.

- **Bet/Reraise**

  Wir spielen Bet/Reraise mit unseren sehr starken Händen, um das Maximum an Value herauszuholen.

- **Bet/Call**

  Bet/Call spielen wir oft, wenn am River ein Draw angekommen ist, wir selbst aber auch eine starke Hand halten (Set, Two Pair) und dem Gegner keinen kostenloses Showdown geben wollen.

- **Check/Call**

  Diese Strategie ist eigentlich gegen jeden Gegnertyp anwendbar. Die Vorteile hier sind, dass wir nicht mehr mit einem Bluff Raise konfrontiert werden können und dass wir eventuell einen Bluff unserer Gegner provozieren können, die auf eine Bet gefoldet hätten.

- **Check/Fold**

  Check/Fold spielen wir in der Regel immer dann, wenn der Pot sehr klein ist, unsere Hand keinen Showdown Value besitzt und wir uns sicher sein können, dass der Gegner eine bessere Hand hält.

- **Check/Raise**

  Check-Raise-Versuche spielen wir eigentlich nur gegen TAGs und LAGs. Allerdings ist es schwierig den richtigen Zeitpunkt für ein Check-Raise zu finden. Einerseits riskieren wir nämlich eine Bet zu verlieren, wenn der Gegner ebenfalls checkt oder wenn er auf unseren

Bet selbst geraist hätte, so dass wir 3-betten hätten können. Damit ein Check-Raise also profitabel sein kann, muss der Gegner den River oft genug betten und nach einem Check-raise von uns noch oft genug callen.

4. Deception

Unter Deception verstehen wir Aktionen eines Spielers, die der wirklichen Stärke seiner Hand diametral entgegenstehen. Dies kann sich als Betten und Raisen mit einer schwachen Hand oder Checken und Callen mit einer starken Hand äußern.

Den ersteren Fall bezeichnen wir als Bluffing, den letzteren als Slow Play. Diese beiden Fälle werden in den zwei folgenden Abschnitten behandelt. Im dritten Unterabschnitt betrachten wir eine Spezialvariante des Bluffings, den Semi-Bluff.

4.1 Bluffing

Ein erfolgreicher Bluff, durch den der Gegner eine stärkerere Hand foldet, ist eine der profitabelsten Szenarios des Pokers. Dies können wir direkt aus dem Fundamentalsatz entnehmen, denn hätte der Gegner die Karten des Bluffers gesehen, wäre anstatt eines Folds ein Raise gefolgt und der Bluffer wäre zum Folden gezwungen gewesen.

Um die Wichtigkeit von Bluffs zu unterstreichen, untersuchen wir ein Beispiel aus der Sicht des Gegners. Angenommen man ist heads-up auf dem River und der Gegner bettet \$20 in einen Pot von \$100. Für einen Call hat man nun Odds von 6:1. Weiterhin nehmen wir an, dass wir, wenn wir callen, nur dann gewinnen können, wenn der Gegner blufft.

Wissen wir von unserem Gegner, dass er niemals blufft, so ist die Entscheidung einfach: Fold. Wissen wir von unserem Gegner, dass er sehr oft blufft, so ist die Entscheidung ebenfalls einfach: Call. Variiert der Gegner aber zwischen Bluffs und Value Bets im Verhältnis 1:6, so haben wir keine Möglichkeit, die Strategie des Gegners auszunutzen. Dies würde sogar dann gelten, wenn der Gegner vorher ankündigen würde, dass er mit einer Wahrscheinlichkeit von 1:6 blufft.

Unser Erwartungswert in dieser Situation berechnet sich folgendermaßen: $EV = \$120 * 1 - \$20 * 6 = \$0$. Unser Erwartungswert für einen Call ist 0, so wie auch im dem Fall, dass wir folden. Der Spieler, der sich einem solchen Bluff gegenübersieht, ist indifferent zwischen Callen und Folden.

Ein optimaler Bluff erfüllt also folgende Bedingungen:

- Die Hand kann nur durch einen Bluff gewonnen werden.
- Die Odds für einen Bluffs decken sich mit den Pot Odds

Damit der mit einem Bluff konfrontierte Gegner nicht anhand von Mustern Informationen darüber gewinnen kann, ob man blufft oder nicht, ist es notwendig, die Entscheidung ob Bluff oder nicht zufällig zu treffen. Angenommen die optimale Bluffwahrscheinlichkeit in einer bestimmten Situation läge bei 1:3, so könnte man den Bluff nur dann ansetzen, wenn bspw. die linke Karte in der Hand

ein Herz ist - eine Situation, für die die Odds ebenfalls bei 1:3 liegen. Diese Randomisierung zusammen mit einer den Pot Odds angepassten Frequenz ist die spieltheoretisch optimale Umsetzung eines Bluffs.

4.2 Slow Play

Ein Slow Play ist eine bei weitem weniger mächtige Waffe als ein Bluff aus dem Grund, dass man bei einem Bluff nichts zu verlieren hat. Hat man aber eine starke Hand, so riskiert man mit einem Slow Play, auf der folgenden Straße durch eine Karte geschlagen zu werden, die der Gegner korrekt aufgrund der günstigeren Pot Odds gesehen hat.
Mit einem Slow Play gibt man dem Gegner also günstige Pot Odds auf einen Call. Checkt man und der Gegner checkt ebenfalls, so bekommt er eine Free Card, mit der er seine Hand möglicherweise zur besten Hand verbessern kann. Callt man eine Bet des Gegners, anstatt zu raisen, obwohl die Pot Equity ein Raise gefordert hätte, so gibt man Value auf und riskiert gleichzeitig, durch die nächste Karte geschlagen zu werden.

Aus den genannten Gründe sind Gelegenheiten für ein korrektes Slow Play weit seltener als für einen korrekten Bluff. Wir möchten an dieser Stelle lediglich auf zwei typische Slow-Play-Situationen hinweisen.

- **Das Big-Blind-Special:** In dieser Situation sieht man aus dem Big Blind heraus den Flop in einem ungeraisten Pot. Hat man nun auf dem Flop eine starke, aber gut verborgene Hand wie zum Beispiel Two Pair mit zwei kleinen Karten, so bietet sich ein Check-Raise an. Da nämlich bisher niemand Stärke gezeigt hat, besteht eine hohe Wahrscheinlichkeit dafür, dass einer der anderen Spieler entweder etwas getroffen hat und bettet oder in Late Position blufft.
- **Die Monster-Hand:** Hat man eine sehr starke Hand, wie etwa ein Full House oder Quads, so bestehen meist wenig Chancen darauf, dass eine Bet vom Gegner gecallt würde. In einem solchen Fall kann man ein Slow Play versuchen, um es dem Gegner zu ermöglichen, seine Hand ebenfalls zu einer starken Hand, in etwas einem Flush, zu verbessern um so die Chancen zu verbessern, vom Gegner noch eine Bet zu gewinnen.

4.3 Der Semi-Bluff

4.3.1 DEFINITION

Unter einem Semi-Bluff verstehen wir das Betten oder Raisen mit einer Hand, die mit hoher Wahrscheinlich zu diesem Zeitpunkt nicht die beste Hand ist, dafür aber eine angemessene Chance besitzt, sich auf den folgenden Straßen so zu verbessern, dass sie die momentan beste Hand schlagen kann.

4.3.2 MOTIVATION

Die Absicht hinter einem Semi-Bluff ist es, ähnlich eines Bluffs, die Hand nach Möglichkeit sofort dadurch zu gewinnen, dass der Gegner foldet. Für den Fall, dass der Gegner nicht foldet, besteht bei einem Semi-Bluff aber zusätzlich die Chance, die eigene Hand noch zu verbessern und dadurch die

Hand zu gewinnen.

Ein weitere Gewinnmöglichkeit auf einer späteren Straße besteht darin, dass auf eine Scare Card ein echter Bluff bessere Erfolgschancen aufgrund der vorher gezeigten Stärke hat.

4.3.3 EIN BEISPIEL

Im folgenden ist ein Beispiel für eine Semi-Bluff-Situation gegeben, dass [Sklansky99] entnommen ist. Angenommen wir halten A♣ 4♣ auf einem Flop von J♢ 3♣ 8♠. Der Flop wird von allen verbliebenen Spielern sowie von uns gecheckt und der Turn bringt die 5♣.

Diese Karte bringt uns ein Draw auf den Nut-Flush mit jedem weiteren Kreuz sowie ein Inside-Straight-Draw mit jeder weiteren 2. Werden wir nun auf dem Turn mit einer Bet konfrontiert, so können wir mit einem Semi-Bluff-Raise antworten. Daraufhin können wir die Hand folgendermaßen gewinnen.

- Alle Gegner folden
- Wir treffen auf dem River eines unserer Draws und gewinnen mit der besten Hand
- Wir halten mit einem As schon die beste Hand und gewinnen den Showdown
- Wir bluffen auf dem River, wenn eine Scare Card, bspw. eine Overcard (K, D), fällt.

Die Verbindung dieser Gewinnmöglichkeiten macht den Semi-Bluff zu einer profitablen Spielweise. Nach Sklansky ist dies in dem Beispiel schon der Fall, wenn der Gegner nur in 20% der Fälle foldet. Es ist allerdings zu beachten, dass auch wirklich alle die Gewinnbedingungen realistisch sind, da der positive Erwartungswert des Semi-Bluff sich aus allen diesen Gewinnmöglichkeiten zusammensetzt.

4.3.4 VERTEIDIGUNG GEGEN EINEN SEMI-BLUFF

Die Verteidigung gegen einen möglichen Semi-Bluff stellt sich schwierig dar, falls der Gegner auch mit starken Händen Stärke zeigt, also das Verhältnis zwischen Bluffs und legitimem Spiel spieltheoretisch angemessen ist.

Die Gefahr, dass man auf einen einer Scare Card folgenden Bluff des Gegners zum Folden gezwungen ist, bedeutet in den meisten Fällen, dass man entweder sofort folden muss oder raist bzw. reraist. Das Raise kann dabei sowohl mit einer legitimen Hand als auch als Re-Semi-Bluff erfolgen. Sklansky listet folgende Einflussfaktoren für die zu treffende Entscheidung auf:

- Fold, falls man eine sehr schwache Hand hält.
- Raise, falls man eine sehr starke Hand hält, die aber nicht so stark ist, dass ein Slow Play gerechtfertigt wäre.
- Betrachtung folgender Faktoren für marginale Hände

1. Die Wahrscheinlichkeit eines Bluffs oder Semi-Bluffs des Gegners
2. Die Wahrscheinlichkeit, dass sich die Hand des Gegners zur besten Hand verbessert.
3. Die Chance, dass man die eigene Hand ausreichend verbessert.

Das Callen eines möglichen Semi-Bluffs wird von Sklansky nur in Ausnahmefällen als korrekt erachtet. Diese sind im folgenden kurz aufgelistet.

- Der Pot ist schon sehr groß, wobei die eigene Hand nicht stark genug für ein Raise ist. Hier ist ein Call aufgrund der Pot Odds angebracht, um den Showdown zu erreichen (siehe Calldown)
- Man hält eine gute, aber nicht sehr gute Hand und die Vermutung liegt nahe, dass der aggressive Gegner ein starkes Draw angespielt hat. In diesem Fall hätte der Gegner auf das gegen den Semi-Bluff gerichtete Raise einen einfachen Call aufgrund der Pot Odds. Hatte er kein Draw sondern eine fertige Hand, so ist diese mit hoher Wahrscheinlichkeit stärker als die eigene Hand und man wird gereraist.
- Der Call als *verspäteter Semi-Bluff*: Falls der Gegner weiß, dass man auf einen Semi-Bluff mit einem Semi-Bluff-Raise zu antworten bereit ist, bietet sich ein Call an, um den Gegner auf der nächsten Straße zu raisen.

5. Anhang A: Hand Reading

Da nach dem Fundamentalsatz der Erwartungswert des eigenen Spiels dann maximal ist, wenn man den Karten der Gegner entsprechend spielt, kommen Techniken zur Bestimmung der gegnerischen Karten, bzw. der Hand Range des Gegners grundlegende Bedeutung zu.

David Sklansky nennt in [Sklansky99] zwei essentielle Techniken, namentlich die Auswertung des Spiels der Gegner und deren offenliegenden Karten, sowie eine mathematische Methode, die Bayes'sche Inferenz.

5.1 Ein Beispiel

Bevor wir diese Methoden erläutern, soll ein [Sklansky99] entnommenes Beispiel einen Einblick in die Denkweise eines Pokerexperten geben. Die Hand wurde bei der World Series of Poker 1977 zwischen Doyle Brunson und Gary Berland. Es handelte sich hierbei um No-Limit Hold'em. Brunson mit einem Stack von ca. $20000 hielt QQ und wurde mit einem hohen Raise Berlands in EP konfrontiert. Berland hatte einen Stack von ca. $50000 Nach Brunsons Call waren die Spieler auf dem Flop heads-up.

Auf einem Flop J52 Rainbow folgte wieder eine substantielle Bet Berlands, die von Brunson ebenfalls gecallt wurde. Als auf dem Turn eine weitere niedrige Karte fiel, setzte Berland so viel, dass Brunson bei einem Call all-in gewesen wäre.

An dieser Stelle, einen Bluff, der unwahrscheinlich erschien, ausgenommen, wäre die einzige legitime Hand, die Brunsons QQ an dieser Stelle schlagen konnte, AJ für Top Pair, Top Kicker gewesen. Brunson callte schließlich auf der Grundlage einer Beobachtung, dass Berland mit hoher Wahrscheinlichkeit nicht AA oder KK hielt, da er mit diesen Händen preflop in EP nur zu callen pflegte, um dann einen eventuellen Raiser zu reraisen.

Diese Einschätzung erwies sich als korrekt. Berland hielt genau AJ und Brunson gewann letztendlich die Hand. Bemerkt sei hierzu, dass die Gewohnheit Berlands, bestimmte Hände auf immer dieselbe Art und Weise zu spielen, einen Fehler darstellte, den Brunson in diesem Beispiel ausnutzen konnte.

5.2 Auswertung der Aktionen des Gegners und seiner offenliegenden Karten

Um diese Technik anzuwenden, ist eine genaue Beobachtung der Gegner notwendig. Jeglicher beobachtbarer Aspekt des gegnerischen Spiels ist hierfür von Bedeutung. Sklansky nennt als Beispiele folgende Fragen.

- Raist der Gegner mit starken Händen in EP? Oder versucht er ein Slow Play?
- Wie oft blufft der Gegner?
- Raist der Gegner mit einer Drawing Hand?
- Spielt der Gegner seine starken Hände immer auf dieselbe Art und Weise oder variiert er?

Diese Liste ließe sich leicht fortsetzen, aber der entscheidende Punkt wurde deutlich gemacht. Eine akkurate Einschätzung des gegnerischen Spiels ist unabdingbar.

5.3 Statistische Ansätze

Hat man keine weiteren Informationen über das Spiel des Gegners, so kann man statistische Überlegungen anstellen, um aufgrund von ermittelten Wahrscheinlichkeiten für die Hände der gegnerischen Range den Erwartungswert einer Entscheidung zu berechnen. Wir verweisen hierzu auf [Chen06], Kapitel 5, und bringen hier lediglich ein einfaches Rechenbeispiel aus [Sklansky99].

Angenommen ein Spieler raist aus EP und wir schätzen ihn derart ein, dass er dies nur mit AA, KK oder AK tun würde. Wir selbst halten QQ und möchten die Wahrscheinlichkeit bestimmen, mit der wir in dieser Situation nicht mit einem höheren Paar konfrontiert sind. Die hier zu Grunde liegenden Wahrscheinlichkeiten sind wie folgt.

- $p(AA) = 0,45\%$
- $p(KK) = 0,45\%$
- $p(AA, KK) = 0,9\%$
- $p(AK) = 1,2\%$

Hieraus können wir Odds von $1,2 : 0,9$ bzw. $4 : 3$ dafür ableiten, dass der Gegner AK und nicht AA oder KK hält.

6. Anhang B: Glossar

3-Betting ***(n.)*** Beantworten eines Raise mit einem erneuten Raise (auch: Reraise)

Board ***(n.)*** Die offen liegenden Karten (Flop, Turn, River)

Big Bet ***(n.)*** Die Höhe der Bets auf Turn und River

Busted Draw ***n.*** Ein Draw, das nach der Riverkarte immer noch nicht getroffen wurde.

Check Behind ***m.*** Heads-up in Positon nicht betten.

Completion ***(f.)*** Das Zahlen einer halben Small Bet im Small Blind

Connector, Suited Connector ***(m.)*** Zwei Karten zu einer Straße ohne Abstand (Bsp. 87, AK). ”Suited” bedeutet Karten in nur einer Farbe.

Dead Money ***(f.)*** Geld im Pot, das von Spielern stammt, die schon gefoldet haben.

In Position ***adv., adj.*** Position auf einen Gegner haben, d.h. nach ihm in der Betreihenfolge zu sitzen.

Multiway Pot ***(m.)*** Ein Pot, in dem mehr als zwei Spieler noch aktiv sind.

Value ***(m.)*** Equity * Größe des Pots

Value Bet ***(f.)*** Eine Bet zur Erhöhung des Values, da der Value mit der Größe des Pots steigt.

Open Raise ***(n.)*** Ein Raise, zu dessen Zeitpunkt noch kein anderer Spieler im Pot ist.

Pocket Pair ***(n.)*** Ein Paar als Starthand

PFR ***(m.)*** Preflop Raise: Eine Kennzahl die prozentual angibt, wie oft ein Gegner preflop raist.

Read ***m.*** Zusatzinformation über einen Gegner, die das Hand Reading erleichtert.

Small Bet ***(n.)*** Die Höhe der Bets preflop und auf dem Flop

VPIP ***(m.)*** Voluntarily Put Into Pot: Eine Kennzahl, die prozentual angibt, wie oft ein Gegner freiwillig bettet (Das Setzen des Small Blinds und des Big Blinds ist also ausgenommen).

WTS ***n.*** Went To Showdown: Eine Kennzahl, die prozentual angibt, wie oft ein Gegner bis zum Showdown in der Hand verbleibt.

Out of Position ***adv., adj.*** Vor einem Gegner in der Betreihenfolge zu sitzen.

7. Anhang C: Tabellen

In der unten stehenden Tabelle sind alle Hände eingetragen, mit denen man raisen sollte, sofern vor einem noch kein Gegner in die Hand eingstiegen ist. (Selbstverständlich gilt dies auch für alle Hände die besser sind als die angegebenen)

	MP2	MP3	CO	BU	SB
Pairs	66	55	44	33	22
A	AT	A9	A7	A2	A2
K	KJ	KJ	KT	K7	K3
Q		QJ	QT	Q9	Q4
J			JT	J9	J5
T					T6
9					97
8					87
As	A8s	A6s	A2s	A2s	A2s
Ks	KTs	K9s	K8s	K2s	K2s
Qs	QTs	Q9s	Q8s	Q6s	Q2s
Js	JTs	J9s	J8s	J7s	J2s
Ts		T9s	T9s	T8s	T3s
9s			98s	98s	94s
8s				87s	85s
7s				76s	76s

Abbildung 2: Open Raising Chart

In dieser Tabelle findet ihr die Hände ab denen ihr nach einem Raise von einem unbekannten Gegner vor euch 3-betten solltet. Habt ihr weitere Informationen über den Gegner bestimmt ihr wieder eine Hand-Range des Gegners und spielt dann wieder nur mit den Händen weiter, die einen Equity Vorteil gegen diese Hände haben.

Die Spalten bedeuten dabei jeweils die Position aus der die Erhöhung kam. Hat der Spieler aus MP2 erhöht, so schauen wir in der Spalte für MP2 nach und sehen, dass wir dort mit allen Händen 88+ erhöhen. Einträge mit einem Slash bedeuten, dass wenn wir im SB oder BB sitzen auch schon mit diesen Händen 3-betten können, da wir hier keine ganz so hohe Equity brauchen, wie in einer Nicht-Blind Position, da wir ja schon einen Teil bezahlt haben und der Raise dadurch günstiger wird.

	MP2	MP3	CO	BU
Pairs	88 / 66	66 / 44	55 / 22	22 / 22
A	AJo / ATo	AJo / ATo	A8o / A7o	A2o
K		KQo	KQo / KTo	K9o
As	ATs	A9s / A7s	A2s / A2s	A2s
Ks	KQs	KQs / KJs	KTs / K9s	K7s
Qs			QTs / QTs	Q9s
Js				JTs

Abbildung 3: Chart: ORC 3-Betting

	Chance mindestens eines unserer Outs am Turn oder am River zu		Chance eines unserer Outs am Turn zu treffen		Chance eines unserer Outs am River zu treffen		Beispiele
Outs	**%**	**Odds X:1**	**%**	**Odds X:1**	**%**	**Odds X:1**	
21	69,94	2,326 favor	44,68	1,238	45,65	1,190	Flushdraw und OESD + 2 live Overcards
20	67,53	2,080 favor	42,55	1,350	43,48	1,300	
19	65,03	1,860 favor	40,43	1,474	41,30	1,421	
18	62,44	1,663 favor	38,30	1,611	39,13	1,556	
17	59,76	1,485 favor	36,17	1,765	36,96	1,706	
16	56,98	1,325 favor	34,04	1,938	34,78	1,875	
15	54,12	1,179 favor	31,91	2,133	32,61	2,067	Flushdraw und OESD
14	51,16	1,047 favor	29,79	2,357	30,43	2,286	Flushdraw + One Pair
13	48,10	1,079	27,66	2,615	28,26	2,538	OESD + One Pair
12	44,96	1,224	25,53	2,917	26,09	2,833	Flushdraw + Gutshot
11	41,72	1,397	23,40	3,273	23,91	3,182	
10	38,39	1,605	21,28	3,700	21,74	3,600	
9	34,97	1,860	19,15	4,222	19,57	4,111	Flushdraw
8	31,45	2,179	17,02	4,875	17,39	4,750	OESD
7	27,85	2,591	14,89	5,714	15,22	5,571	
6	24,14	3,142	12,77	6,833	13,04	6,667	zwei Overcards
5	20,35	3,914	10,64	8,400	10,87	8,200	Ein Paar zu Drilling oder two pair verbessern
4	16,47	5,073	8,51	10,750	8,70	10,500	Gutshot
3	12,49	7,007	6,38	14,667	6,52	14,333	
2	8,42	10,879	4,26	22,500	4,35	22,000	sein Set treffen
1	4,26	22,500	2,13	46,000	2,17	45,000	Runner-Runner Flush

Abbildung 4: Chart: Odds und Outs

8. Bibliographie

[Ciaffone01] Ciaffone, B./Brier, J. (2001), Middle Limit Hold'em Poker, USA

[Chen06] Chen, B./Ankenman, J. (2006), The Mathematics of Poker, ConJelCo LCC, USA

[Sklansky99] Sklansky, D./Malmuth, M. (1999), The Theory of Poker, 4. Auflage, Two Plus Two Publishing LCC, USA

Artikel aus dem Strategieforum von Pokerstrategy.de, www.pokerstrategy.de

The Challenge of Poker

Björn Heidenreich B.Heidenreich@ocean2.de

1. The Challenge of Poker

Diese Arbeit stellt eine Zusammenfasssung des Artikels "The challenge of poker"[1] dar. Sie beschränkt sich auf die wesentlichen Aspekte, welche für die Computer Poker Challenge im Sommersemester 2008 an der Technischen Universität Darmstadt von Interesse sind und soll eine Einordnung der anderen Seminarthemen ermöglichen.

Poker stellt ein interessantes Forschungsfeld der Künstlichen Intelligenz dar. Im Gegensatz zu anderen Spielen, wie z.B. Schach - wo Künstliche Intelligenzen bereits auf Weltklasse-Niveau spielen - existieren beim Poker nur unvollständige Informationen: Karten anderer Spieler sind nicht sichtbar. Deshalb reichen Methoden wie reine Tiefensuche nicht aus, um Poker auf einem ähnlichen Niveau spielen zu können. Aber noch andere Eigenschaften machen Poker für die Forschung interessant. So gibt es mehrere konkurrierende Agenten (Spieler). Desweiteren zeichnen Risikomanagement, Modellierung der Gegner und der Umgang mit nur unvollständigen oder nicht vorhandenen Informationen Poker aus.

Es gibt viele Ansätze, eine Künstliche Poker-Intelligenz zu entwickeln. In dieser Arbeit soll auf Poki, ein Programm der University of Alberta, näher eingegangen werden. Daneben gibt es viele weitere Ansätze um der Komplexität und Schwierigkeit des Poker Spielens Herr zu werden: Es kann z.B. mit vereinfachten Poker-Varianten gearbeitet werden, aber auch Subsets des Spiels können unabhängig voneinander betrachtet werden. Die Gefahr dieser Vereinfachungen liegt jedoch darin, dass sie das eigentliche Poker-Problem zerstören können. Ein anderer Ansatz wäre deshalb der, ein Programm zu entwickeln, welches gegen die besten realen Pokerspieler bestehen kann. Dieser Ansatz wird von Poki verfolgt.

2. Anforderungen an einen guten Poker-Spieler

Es können fünf verschiedene Anforderungen an einen guten Poker-Spieler gestellt werden. Diese müssen nicht explizit in der KI designed, sollten jedoch aber zumindest zufriedenstellend gelöst werden, um eine starke KI entwickeln zu können.

2.1 Hand Strength

Hand Strength misst die Stärke der Hand in Relation zu anderen. Eine einfache Lösung könnte aus einer Funktion mit den eigenen Karten und den community cards als Input bestehen. Eine bessere Funktion berücksichtigt unter anderem zusätzlich die verbleibende Anzahl an Spieler, die Position am Tisch sowie die Wahrscheinlichkeitsverteilung der gegnerischen Karten.

1. Billings, Davidson, Schaeffer, Szafron: The Challenge of Poker, Department of Computer Science, University of Alberta

2.2 Hand Potential

Mit Hand Potential wird die Wahrscheinlichkeit bezeichnet, dass eine Hand verbessert oder verschlechtert wird, nachdem weitere community cards gegeben werden. Beispielsweise kann eine Hand mit vier gleichen Farben eine geringe Hand Strength besitzen. Die Wahrscheinlichkeit mit einem Flush zu gewinnen, ist jedoch entsprechend hoch.

2.3 Bluffing

Bluffing ermöglicht es, auch mit einer schwachen Hand zu gewinnen. Um die optimale Bluffing-Frequenz in einer bestimmten Situation herauszufinden, kann die Spieltheorie verwendet werden. Ein minimaler Ansatz besteht darin, mit einer der Bluffing Frequenz entsprechenden Anzahl von Händen zu bluffen. Ausgefeiltere Systeme versuchen hingegen profitable Situationen herauszufinden, indem sie die gegnerische Hand Strength und ihre Folding-Wahrscheinlichkeit vorherzusagen versuchen.

2.4 Unberechenbarkeit

Die Unberechenbarkeit macht es den Poker-Gegnern schwer, ein akkurates Modell des eigenen Spielers zu formen. Indem der Stil des eigenen Spiels über die Zeit variiert wird, wird die Wahrscheinlichkeit erhöht, dass der Gegner aufgrund von falschen Annahmen Fehler macht.

2.5 Gegner-Modellierung

Opponent Modeling versucht eine Wahrscheinlichkeitsverteilung der Kartenpaare der gegnerischen Hand zu erhalten. Der einfachste Ansatz besteht darin, für jeden Gegner das gleiche Modell zu verwenden. Dieses kann im Laufe des Spiels durch die Modifikation der Wahrscheinlichkeitsverteilung aufgrund des Verlaufs der investierten Einsätze und gesammelten Statistiken der Gegner verbessert werden.

3. Pokis Architektur

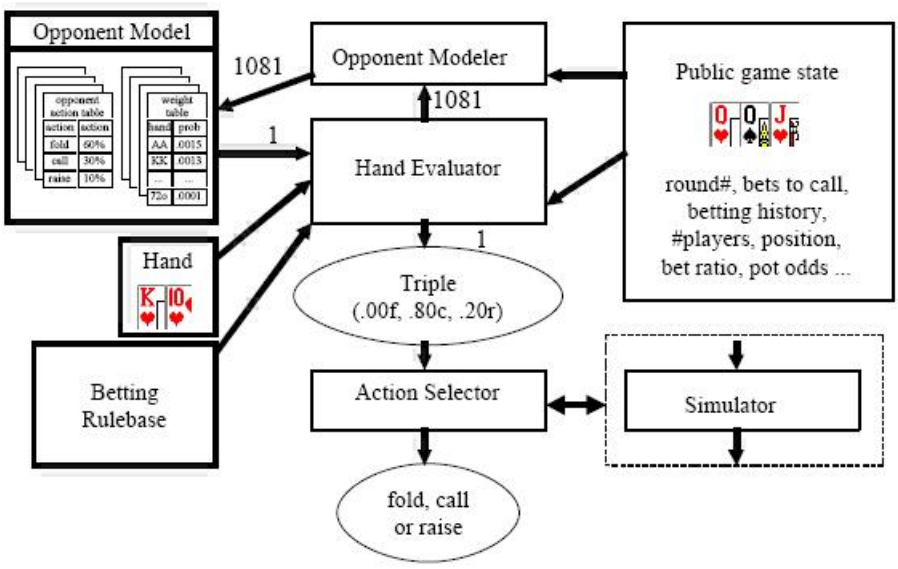

Abbildung 1: Pokis Architektur

Hand Evaluator dient dazu, die Güte der eigenen Hand zu bestimmen. Hierzu wird neben dem Kontext des Spiels (public game state) auch die Gegner-Modellierung mit einbezogen. Letztere versucht möglichst genau die Karten der Gegner und deren Strategie vorherzusagen. Der Action-Selector bzw. der Simulator schließlich wählt die eigene Aktion (fold, call oder raise) aus.

4. Spiel-Strategien

Spiel-Strategien vor und nach dem Flop sind sehr unterschiedlich: Vor dem Flop sind nur sehr wenige Informationen, welche zur Bestimmung des Einsatzes herangezogen werden können, vorhanden. Entsprechend wird nur ein einfaches System zur Bestimmung des Einsatzes benötigt. Post-Flop-Einsätze werden durch wesentlich mehr Faktoren determiniert. Eine entsprechende Strategie verwendet den vollständigen Kontext des Spiels, die eigene Hand und Gegner-Modelle zur Entscheidungsbildung.

4.1 Pre-Flop-Strategien

Es existieren 1326 (52 über 2) verschiedene mögliche Kartenpaare für die private hand. Der Wert solch einer Hand wird income rate genannt und mittels einer roll-out Simulation, einer Offline-Berechnung, welche aus mehreren Millionen Händen besteht, bestimmt. Hierbei callen alle Spieler den ersten Einsatz. Im Anschluss werden alle weiteren Karten ohne einen Einsatz der Spieler verteilt. Diese sehr unrealisitsche allways call assumption liefert nicht zwangsweise ein akkurates Ergebnis für den Wert eines Kartenpaares. Jedoch reicht das Ergebnis für eine relative Gewichtung der Karten in der Pre-Flop-Phase.

4.1.1 Vergleich von Pre-Flop-Strategien

David Slansky beschreibt in seinem Buch "Texas Hold'em for the Advanced Player" ein Klassifikationsschema für die private hand mit einer hohen Korrelation zwischen seinem Ranking und den roll-out Simulationen. Es gibt jedoch keine Bewertung von Karten, welche in allen Situationen zutrifft. Ein fortgeschrittener Spieler wird seine Strategie den vorangegangenen Spielen anpassen und damit die Bewertung entsprechend beeinflussen. Desweiteren hängt der Kartenwert auch von dem Einsatz der vorherigen Spieler ab. Eine Hand wird beispielsweise anders gewertet, nachdem alle vorherigen Spieler gefolded haben, als wenn alle gecallt haben.

Group 1		Group 2		Group 3		Group 4		
+2112	AA*	+714	TT*	+553	99*	+481	T9s	
+1615	KK*	+915	AQs	+657	JTs	+515	KQo	
+1224	QQ*	+813	AJs	+720	QJs	+450	88*	
+935	JJ*	+858	KQs	+767	KJs	+655	QTs	
+1071	AKs	+718	AKo	+736	ATs	+338	98s	
				+555	AQo	+449	J9s	
						+430	AJo	
						+694	KTs	
Group 5		**Group 6**		**Group 7**		**Group 8**		
+364	77*	+304	66*	+214	44*	-75	87o	
+270	87s	+335	ATo	+92	J9o	+87	53s	(> 43s)
+452	Q9s	+238	55*	+41	43s	+119	A9o	
+353	T8s	+185	86s	+141	75s	+65	Q9o	
+391	KJo	+306	KTo	+127	T9o	-129	76o	
+359	QJo	+287	QTo	+199	33*	-42	42s	(< 52s)
+305	JTo	+167	54s	-15	98o	-83	32s	(< 52s)
+222	76s	+485	K9s	+106	64s	+144	96s	
+245	97s	+327	J8s	+196	22*	+85	85s	
+538	A9s			+356	K8s	-51	J8o	
+469	A8s			+309	K7s	+206	J7s	
+427	A7s			+278	K6s	-158	65o	
+386	A6s			+245	K5s	-181	54o	
+448	A5s			+227	K4s	+41	74s	
+422	A4s			+211	K3s	+85	K9o	
+392	A3s			+192	K2s	-10	T8o	
+356	A2s			+317	Q8s			
+191	65s							

Abbildung 2: Vergleich von Slankys Rating und roll-out Simulation

Die Tabelle vergleicht Slankys Bewertung mit einer roll-out Simulation. Der Buchstabe s steht hierbei für "suited hand", also ein Kartenpaar gleicher Farbe, o steht für "offsuit hand", zwei Karten unterschiedlicher Farbe, und * markiert zwei Karten mit dem gleichen Wert. Die Tabelle ist nach Slanky in acht Gruppen unterteilt, wobei Gruppe 1 die beste und Gruppe 8 die schlechteste Hand, welche nur unter bestimmten Umständen gespielt werden sollte, darstellt. Im Allgemeinen lässt sich eine starke Korrelation zwischen Slankys Ranking und den income rates der roll-out Simulationen feststellen.

Allerdings zeigt die Simulation eine Verzerrung zu Gunsten von Karten, die gut gegen mehrere Spieler gespielt werden können. Dies sind in der Regel Karten welche leicht zu

entsprechend starken ausgebaut werden können, wie z.B. flush, straight, three of a kind, usw. Da bei einer roll-out Simulation alle Spieler im Spiel bleiben, muss die durchschnittliche Gewinner-Hand deutlich stärker sein, als in einem realen Spiel. Aus dem gleichen Grund sind hohe Paare entsprechend unterbewertet. Umgekehrt liefert Slankys Ranking eine Verzerrung zu Gunsten von sogenannten Connectors unterschiedlicher Farben, also Karten, die sich im Wert nur um 1 unterscheiden (bspw. Herz 7 und Kreuz 8). Desweiteren existieren in Slankys Wertung auch logische Fehler. So befindet sich zum Beispiel 43s in Gruppe 7 vor 53s in Gruppe 8. Es kann jedoch gezeigt werden, dass letzteres Paar 43s dominiert.

Da der Unterschied zwischen roll-out Simulation und Slankys Rating nicht allzu groß ist, lässt sich eine roll-out Simulation gut nutzen um den Pre-Flop-Kartenwert zu bestimmen. Von der Benutzung Slankys Ratings sei abgeraten, da einerseits menschliches Wissen ausgeschlossen werden sollte und andererseits eine roll-out Simulation spezifisch für verschiedene Situationen durchgeführt werden kann. Slankys Klassifikation wiederum gilt nur für bestimmte Spielsituationen.

4.1.2 Iterierte roll-out Simulationen

Eine Verbesserung der roll-out Simulation stellt die iterierte roll-out Simulation dar. Hierbei wird die Simulation mehrmals ausgeführt, wobei die vorherigen Ergebnisse die nachfolgenden Simulationen beeinflussen. Erhält man beispielsweise für einen Spieler einen negativen Wert, so wird davon ausgegangen, dass er die Hand folded. Dies führt zu einer realistischeren Verteilung, da im Gegensatz zur einfachen roll-out Simulation Spieler aus dem Spiel ausscheiden können. Es werden verschiedene Simulationsrunden durchgeführt, solange, bis das Ergebnis gegen ein Gleichgewicht konvergiert. Man erhält so eine Menge von gewinnträchtigen Karten, also eine Entscheidungshilfe ob gespielt werden sollte oder nicht. Dieses Verfahren ist noch lange nicht perfekt, aber doch besser wie die roll-out Simulation.

Eine weitere Verbesserung besteht darin, einen Noise-Faktor einzuführen. Damit erhalten auch Karten mit negativer Bewertung eine Chance gespielt zu werden.

Hand	IR-10	Iterated	Hand	IR-10	Iterated	Hand	IR-10	Iterated
AA*	+2112	+2920	ATs	+736	+640	KQo	+515	+310
KK*	+1615	+2180	99*	+553	+630	QTs	+655	+280
QQ*	+1224	+1700	KQs	+858	+620	QJs	+720	+270
JJ*	+935	+1270	AQo	+555	+560	A9s	+538	+220
TT*	+714	+920	KJs	+767	+480	ATo	+335	+200
AKs	+1071	+860	88*	+450	+450	KTs	+694	+190
AKo	+718	+850	77*	+364	+390	KJo	+391	+160
AQs	+915	+780	AJo	+430	+380	A8s	+469	+110
AJs	+813	+680	JTs	+657	+360	66*	+304	+40

Abbildung 3: Vergleich roll-out Simulation und iterierte roll-out Simulation

Die Ergebnisse solch eines Versuches sind in der obigen Tabelle zusammengefasst. IR-10 bezeichnet die income-rates für eine roll-out Simulation mit zehn Spielern, iterated die Ergebnisse einer iterierten roll-out Simulation. Hierbei sind die Werte in Milli-Bets angegeben: Karten mit einem Wert von +1000 gewinnen im Durchschnitt jedes Mal 1 Small Bet. Im Gegensatz dazu stellen die Werte der simplen roll-out Simulation nur ein relatives Maß dar.

4.2 Post-Flop-Strategien

Die Entscheidung für den Einsatz nach dem Flop kann beispielsweise wie folgt getroffen werden:

1. Effective had strength (EHS) von Pokis Hand berechnen (Hand Evaluator).
2. Mittels des Spiel-Kontexts, Regeln und Funktionen lässt sich die EHS in ein Wahrscheinlichkeits-Tripel übersetzen: {P(fold), P(call), P(raise)}
3. Generiere eine Zufallszahl zwischen 0 und 1 und wähle mit ihr eine Entscheidung entsprechend der Wahrscheinlichkeitsverteilung (Action Selector).

EHS ist ein Maß, wie gut Pokis Karten in Relation zu den verbleibenden Spielern und den community cards sind.

4.2.1 Hand Strength (HS)

HS ist die Wahrscheinlichkeit mit der eine gegebene Hand besser ist, als die eines Gegners. Angenommen die Wahrscheinlichkeit möglicher Kartenpaare des Gegners ist gleichverteilt, so lassen sich diese aufzählen und wie folgt gewichten: Wenn Pokis Hand besser ist +1, bei unentschieden +0.5 und wenn sie schlechter ist 0. Werden diese Werte nun aufsummiert und schließlich durch die Zahl aller möglichen Hände des Gegners geteilt, so erhält man die ungewichtete Hand Strength.

Angenommen Pokis Hand ist Karo Ass und Kreuz Dame, der Flop ist Herz Bube, Kreuz 4 und Herz 3 und es gibt nur einen aktiven Gegner. Es verbleiben also 47 unbekannte Karten und damit 1081 (47 über 2) mögliche Kartenpaare des Gegners. In unserem Beispiel ist jedes three of a kind, two pair, one pair oder Ass König besser (444 Fälle), andere Ass Dame Kombinationen sind gleichwertig (9 Fälle) und die restlichen Kombinationen (628 Fälle) sind schlechter. Die Hand Strength beträgt folglich $(0 \cdot 444 + 0,5 \cdot 9 + 1 \cdot 628)/1081 = 0,585$. Mit anderen Worten: Es besteht eine 58,5 prozentige Wahrscheinlichkeit, dass das Kartenpaar Karo Ass und Kreuz Dame besser ist als eine zufällige Hand. Sind mehrere Gegner vorhanden, so muss die Hand Strength einfach mit der entsprechenden Gegner-Anzahl potenziert werden. Bei fünf Gegnern beträgt die HS in unserem Beispiel $0{,}585^5 = 0,069$.

4.2.2 Hand Potential (HP)

Nach dem Flop verbleiben immer noch zwei aufzudeckende Karten, nach dem Turn noch eine. Diese sollen nun vorhergesagt werden. Positive Potential (PPot) ist hierbei die Wahrscheinlichkeit, dass eine Hand, welche momentan nicht die beste ist, im Showdown gewinnt. Negative Potential (NPot) bezeichnet analog die Wahrscheinlichkeit, dass eine momentan führende Hand im Showdown verliert.

PPot und NPot werden berechnet, indem alle möglichen Karten der Gegner und des Boards aufgezählt werden. Für alle Kombinationen zählt man die Fälle, in denen die eigene Hand nicht die beste ist, aber am Ende gewinnt (PPot) und die Fälle in denen sie die beste ist, aber am Ende verliert (NPot).

A♢-Q♣ hole cards			J♡-4♣-3♡ board cards		
5 Cards	7 Cards				
	Ahead	Tied	Behind	Sum	
Ahead	449,005	3,211	169,504	628x990 =	621,720
Tied	0	8,370	540	9x990 =	8,910
Behind	91,981	1,036	346,543	444x990 =	439,560
Sum	540,986	12,617	516,587	1,081x990 =	1,070,190

Abbildung 4: Beispiel für Hand Potential

Anhand der Tabelle lässt sich die Berechnung leicht erläutern: Benutzen wir die Form T(Zeile, Spalte) und B für Behind, T für Tied, A für Ahead und S für Sum, so ergeben sich die folgenden Formeln:[2]

$$PPot = \frac{T(B,A)+T(B,T)/2+T(T,A)/2}{T(B,S)+T(T,S)/2}$$

$$NPot = \frac{T(A,B)+T(A,T)/2+T(T,B)/2}{T(A,S)+T(T,S)/2}$$

Wenn die in der Tabelle gegebenen Karten besser als die des Gegners nach 5 gegebenen Karten sind, dann beträgt die Wahrscheinlichkeit 449005/621720 = 72%, dass sie es auch noch nach 7 Karten sind.

Die entsprechenden Potentiale nach dem Pot zu berechnen kann sehr aufwendig sein, weshalb meistens eine schnellere Berechnung benutzt wird, z.B. wird nur die nächste aufzudeckende Karte betrachtet.

4.2.3 Effective Hand Strength (EHS)

EHS kombiniert Hand Strength und Hand Potential in einem Wert um eine einzelne Metrik für die relative Stärke einer Hand zu bekommen. Ein möglicher Ansatz für die Gewinn-Wahrscheinlichkeit ist der folgende:

$$\begin{aligned} P(\text{Gewinn}) &= P(\text{ gute Hand}) \cdot P(\text{Gegner verbessert sich nicht}) \\ &\quad + P(\text{keine gute Hand}) \cdot P(\text{Hand verbessert sich}) \\ &= HS \cdot (1 - NPot) + (1 - HS) \cdot PPot \end{aligned}$$

Nun interessiert uns aber lediglich die Wahrscheinlichkeit, dass unsere Hand die beste ist, oder sich zur besten verbessert. Wir können also NPot = 0 setzen:

$$EHS = HS + (1 - HS) \cdot PPot$$

Für n verbleibende Gegner lässt sich diese Formel verallgemeinern zu:

$$EHS = HS^n + (1 - HS^n) \cdot PPot$$

Diese Formel nimmt an, dass die entsprechenden Variablen für alle Gegner gleich sind. Dieser Ansatz ist jedoch nicht gut geeignet, da verschiedene Spieler verschiedene Stile besitzen. Wir führen also für jeden Spieler eine eigene HS und PPot ein:

$$EHS_i = HS_i + (1 - HS_i) \cdot PPot_i$$

2. vgl. http://www.cs.ualberta.ca/ jonathan/Grad/Papers/ai98.poker.html, 27. März 2008

4.2.4 Weight Tables

Die bisherige Berechnung von HS und HP geht davon aus, dass alle Kombinationen von Kartenpaaren gleich wahrscheinlich sind. Im Poker-Spiel ist dies jedoch in der Regel nicht der Fall. Zum Beispiel ist die Wahrscheinlichkeit, dass ein Gegner nach dem Flop Ass und König auf der Hand hat größer, als dass er eine sieben und eine zwei besitzt; da die meisten Spieler mit sieben und zwei vor dem Flop folden würden.

Um dies zu berücksichtigen, besitzt Poki eine weight table für jeden Gegner, welche die Wahrscheinlichkeit für jedes Kartenpaar zu einem Zeitpunkt des Spiels beinhaltet. Entsprechend wird bei der Berechnung von HS und HP jedes mögliche Kartenpaar mit seinem Gewicht multipliziert. Uns interessieren nur relative Werte, so dass es sich bei den Gewichten nicht um eine Wahrscheinlichkeitsverteilung handeln muss. Ihre Summe muss also nicht zwingend 1 ergeben. Deshalb verwendet Poki eine Zahl zwischen 0 und 1 als Gewicht. Am Anfang einer neuen Runde wird jedes Kartenpaar-Gewicht mit 1 initialisiert. Wenn mehr Karten sichtbar werden, werden entsprechende Paare unmöglich und deren Gewichte werden auf 0 gesetzt.

Nach jedem Einsatz werden die Gewichte des entsprechenden Gegners neu berechnet, dies wird re-weighting genannt. Das entsprechende Gewicht wird anhand der Aktion des Spielers gesenkt oder erhöht. Die mögliche Hand jedes Gegners wird mittels dieser Gewichte neu berechnet und eine Wahrscheinlichkeitsverteilung der gegnerischen Aktionen erstellt. Mittels dieser werden die Gewichte nach jeder Aktion entsprechend berichtigt.

4.2.5 Wahrscheinlichkeits-Tripel

Gesucht ist ein Wahrscheinlichkeits-Tripel $P = \{P(fold), P(call), P(raise)\}$, so dass P(fold) + P(call) + P(raise) = 1. Es repräsentiert die Wahrscheinlichkeiten der entsprechenden Aktionen. Hand Strength, Hand Potential und Effective Hand Strength stellen hierbei nur einige der Informationen, welche zur Berechnung der Wahrscheinlichkeiten nötig sind, zur Verfügung. Weitere beeinflussende Faktoren sind z.B. pot odds, relative Position am Tisch, Verlauf der bisherigen Einsätze.

Ein professioneller Spieler bewertet seine Optionen beispielsweise nach dem erwarteten Return on Investment. Hierzu kann eine Regel- oder Formel-basierte Strategie verwendet werden. Ein großer Vorteil der Verwendung dieses Wahrscheinlichkeits-Trippels besteht darin, dass verschiedenartiges Wissen in einer Information zusammengefasst wird.

4.3 simulationsbasierte Strategien

Es ist zeitintensiv und (fast) unmöglich, einen Experten alle nötigen Regeln zur Platzierung des Einsatzes identifizieren zu lassen. Das Spiel ist sehr komplex und Entscheidungen müssen im jeweiligen Kontext getroffen werden. Aus diesem Grunde müssen dynamische, adaptive Techniken eingesetzt werden um eine gute Poker-Software generieren zu können.

Eine wissensbasierte Strategie ist sinngemäß das gleiche wie eine evaluation function in deterministischen Spielen mit vollständigen Informationen: Vom aktuellen Kontext des Spiels würde sie die Aktionen wählen, welche zu dem besten Ergebnis führen. Es findet also eine Suche statt. Dies ist bei Poker jedoch nicht möglich, da grundlegende Unterschiede in Game-Trees mit unvollständigen Informationen und der Anzahl der zu betrachtenden Möglichkeiten existieren.

Poki benutzt deshalb eine simulations-basierte Strategie: Es werden viele Szenarien simuliert und berechnet, wie viel Geld verloren oder gewonnen wird. Jedesmal wenn Poki eine Entscheidung treffen muss, wird dieser Simulator verwendet, um einen Erwartungswert (expected value, EV) für den Einsatz zu gewinnen. Hierbei werden von dem aktuellen Spiel-Kontext ausgehend verschiedene Aktionen bis zum Ende durchgespielt. In Abbildung 1 ersetzt der Simulator hierbei den Action Selecter. Jede Runde wird zwei Mal simuliert um die Konsequenzen von check oder call und bett oder raise zu simulieren. Die gegnerischen Karten werden hierbei entsprechend der weight table angenommen. Der durchschnittliche Gewinn bei call oder check wird call EV, der durchschnittliche Gewinn bei bet oder raise wird raise EV genannt. Fold EV lässt sich einfach berechnen, da für die Hand keine zukünftigen Gewinne oder Verluste existieren. Die aktuelle Implementierung von Poki wählt die Aktion mit der größten Gewinn-Erwartung aus. Wenn die Erwartung zweier Aktionen gleich ist, wird die aggressivere gewählt. Um die Unvorhersehbarkeit des Programms zu steigern, könnte man bei ähnlich hohen Erwartungen die Aktion zufällig wählen. Jedoch birgt das Rauschen in den Simulationsdaten schon eine entsprechende Zufälligkeit.

Ein Problem besteht darin, dass sich aufgrund der Komplexität nicht alle Kartenverteilungen und Aktionen der Gegner simulieren lassen. Poki verwendet Opponent Modeling um dieses Problem zu umgehen indem die Aktionen der Gegner vorhergesagt werden. Die Simulation erlaubt komplexe Strategien ohne entsprechendes Expertenwissen zu benötigen.

5. Gegner-Modellierung

Jede Poker-Strategie benötigt eine gute Modellierung des Gegners, um dessen Schwächen identifizieren und ausnutzen zu können. Da jeder Gegner andere Strategien benutzt und diese gegebenenfalls im Laufe des Spiels wechselt, muss diese Modellierung adaptiv erfolgen.

5.1 Statistisch-basierte Gegner-Modellierung

Opponent Modeling wird beim Poker in mindestens zwei Weisen genutzt: Einerseits soll auf die Stärke der gegnerischen Karten geschlossen, andererseits sollen ihre Aktionen vorhergesagt werden. Im Zentrum steht hierbei wieder das Wahrscheinlichkeits-Tripel {P(fold), P(call), P(raise)}, welches den Spiel-Kontext in eine Wahrscheinlichkeit der gegnerischen Aktionen abbildet.

Ein Weg besteht darin, die eigene Strategie zu unterstellen. Dieses Verfahren heißt generic opponent modeling (GOM). Ein anderes Verfahren basiert auf dem vergangenen Verhalten des einzelnen Gegners: specific opponent modeling (SOM). Beispielsweise wird beobachtet, dass ein Gegner in 40% der Fälle direkt nach dem Flop seinen Einsatz macht. Es lässt sich annehmen, dass er in dieser Spielsituation mit den Top 40% der Kartenpaare einen Bet ausführt. Soll das Verhalten der Gegner in bestimmten Situationen gelernt werden, so tritt die folgende Problematik auf: Wird der Kontext zu weit definiert, so lernt das System vielleicht nicht alle wichtigen Aspekte des Gegners. Wird er zu eng definiert, so lernt das System nur sehr langsam und manches vielleicht überhaupt nicht, da die zu lernenden Situationen zu selten auftreten.

Opponent Modeling beim Poker besitzt viele Eigenschaften, welches es für maschinelles Lernen so schwierig macht, gute Algorithmen zu entwickeln: Rauschen, Unberechenbarkeit

der Gegner, Forderung nach schnellem Lernen und den Anspruch von relativ wenigen Trainings-Beispielen gute Vorhersagen zu bilden.

5.2 Gegner-Modellierung basierend auf neuronalen Netzen

Um ein allgemeineres System zur Modellierung des Gegners zu erhalten, implementiert Poki ein neuronales Netz für die Vorhersage der gegnerischen Aktionen in jedem Kontext. Als Input dienen Eigenschaften des Spiel-Kontexts, welche sich auf die Spieler-Entscheidungen auswirken können oder mit ihnen korreliert sind. Der Output-Layer besteht aus drei Knoten, welche für fold, call und raise stehen. Legt man die entsprechenden Werte auf die Input-Knoten, so erhält man durch Normalisierung der Werte der Output-Knoten das bereits bekannte Wahrscheinlichkeits-Tripel.

Nachdem das Netz mittels verschiedener Gegner trainiert wurde, sieht man, dass manche Input-Knoten für die Vorhersage dominant sind, während andere fast keine Rolle spielen. Die Vorhersagegenauigkeit kann mittels Cross-Validation der realen Spielzüge gemessen werden.[3] Man erhält so eine relativ kleine Klasse an wichtigen Faktoren, welche die statistische Gegner-Modellierung signifikant verbessert.

6. Performance Evaluation

Auf die Performance Evaluation soll an dieser Stelle nur kurz eingegangen werden. Die Performanz einer Poker-KI ist nur schwer messbar. Poki ist ein komplexes System, wo kleine Änderungen der Software zu großen, unvorhergesehenen Auswirkungen führen können.

Der Faktor Glück hat bei Poker einen großen Einfluss, was es schwer macht, entsprechende Software miteinander zu vergleichen. Ebenso handelt es sich um adaptive Systeme, welche sich auf die Gegner einstellen.

Eine Methode neue Features zu testen besteht darin, die Software gegen eine alte Version ihrer selbst spielen zu lassen. Für die Evaluation der Spielstärke ist dieser Ansatz aber nur bedingt geeignet, da verschiedene Spielstile nicht abgedeckt werden. Die Software kann beispielsweise gute Ergebnisse gegen sich selbst erzielen, aber gegen einen bestimmten Spieltyp versagen. Die Streuung dieser Strategien ist bei menschlichen Spielern größer als bei künstlichen, weshalb es sinnvoll ist die Software gegen menschliche Spieler antreten zu lassen. Dies kann zum Beispiel mittels eines Java Web Applets erfolgen. Hier herrscht allerdings keine kontrollierte Umgebung: Ein gutes Resultat kann darin begründet liegen, dass lediglich gegen schwache Gegner gespielt wurde. Selbst leichte Verbesserungen lassen sich erst nach dem Spielen von mehreren tausend Händen gegen verschiedene Gegner erkennen. Ein Maß für die Performanz besteht zum Beispiel in den durchschnittlich gewonnen small bets pro Hand (sb/hand): Bei \$10/\$20 Hold'em mit 40 Händen pro Stunde und +0,05 sb/hand gewinnt der Spieler durchschnittlich \$20 die Stunde. Je nach Gegnergruppen (z.B. sehr gute, mittlere, schwache Spieler) kann sich sb/hand stark unterscheiden. Beispielsweise kann es sein, dass ein Programm gegen starke Gegner gut spielen kann, aber gegen mittlere Gegner schlechter abschneidet, da es deren Aktionen schlechter voraussagen kann. Bei mindestens 20.000 gespielten Händen gegen frühere Versionen der Software kann eine Verbesserung um +0,05 sb/hand als signifikant angenommen werden.

3. vgl. Davidson, Billings, Schaeffer, Szafron: Improved opponent modeling in poker, in International Conference on Artificial Intelligence, Seiten 1467 - 1473, 2000

7. Fazit

Poker ist ein komplexes Spiel mit vielen Anforderungen. Um erfolgreich zu sein, müssen die Aktionen des Spielers einerseits unvorhersagbar sein, andererseits müssen die gegnerischen Aktionen jedoch bestmöglich vorhergesagt werden.

Bei der Entwicklung von Poki wurde eine zyklische Vorgehensweise benutzt. Eine Komponente wurde so lange verbessert, bis eine andere den Performanzengpass darstellte. Aktuell sind manche Funktionen in Poki noch extrem einfach und bestehen beispielsweise aus einer einfachen Konstante, was die Performanz jedoch nicht negativ beeinflusst. Trotz den Fortschritten, gerade auch im Bereich der Gegner-Modellierung, existieren noch zahlreiche möglichen Verbesserungen in der Funktionsweise von Poki.

Es ist möglich umfangreiche Daten über seine Gegner zu sammeln. Das Problem besteht darin, diese auszuwerten und die nützlichen Features zu finden. Gegebenfalls sagt eine einfachere Metrik das Verhalten der Gegner besser voraus, als eine komplexere, welche mehr Features mit einbezieht.

Momentan besteht in diesem Bereich noch viel Forschungsbedarf.

Strategien bei der Entwicklung von Poker-Agenten

Andreas Eismann andreas.eismann@gmx.net

Abstract

Die Masterarbeit von Michael Johanson war Grundlage dieser Zusammenfassung verschiedener Strategien bei der Implementierung von Pokerprogrammen. Die fundamentalen statistischen und spieltheoretischen Hintergründe waren ebenso Bestandteil der Thesis, wie auch die tatsächliche, praktische Anwendung des zusammengetragenen Wissens bei der Erstellung eines Poker-Bots sowie der Teilnahme mit diesem an der AAAI Computer Poker Challenge 2007.

1. Einleitung

Poker ist aus spieltheoretischer Sicht ein Spiel mit unvollständiger Information. Kennt ein Spieler selbst nur seinen eigenen Typ, während andere Spieler dessen Situation lediglich wahrscheinlichkeitstheoretisch schätzen können, so spricht man von unvollständiger, speziell asymmetrischer Information . In diesem Zusammenhang können Reputationseffekte auftreten, kurz zusammengefasst bedeutet dies, dass es sich für einen Spieler lohnen kann, einige Spielrunden mit Absicht zu seinem eigenen Nachteil zu spielen, um aus seinem daraus entstandenen Ruf bei seinen Mitspielern in den folgenden Spielrunden mehr als diesen entstandenen Nachteil zu Erwirtschaften.

Pokerterminologie+ Spielregeln Poker besteht aus vier Spielrunden (Preflop, Flop, Turn, River); nach Austeilen der Karten erfolgt in jeder Spielrunde eine Setzrunde mit den unten genannten drei Aktionsmöglichkeiten für jeden Spieler. Preflop: Austeilen von zwei (privaten) Karten pro Spieler, danach Setzrunde mit den Aktionsmöglichkeiten Flop: Drei Karten werden offen in der Mitte ausgeteilt. Alle Spieler können Ihre privaten Karten mit den offenen kombinieren um eine der Kombinationen zu erhalten. Turn: Eine weitere Karte wird offen aufgedeckt. River: Die letzte der insgesamt nun fünf Karten wird offen aufgedeckt.

Nach der Spielrunde River kommt es zum sogenannten Showdown. Alle noch im Spiel befindlichen Spieler zeigen offen ihre zwei privaten Karten. Derjenige Spieler mit der höchsten Karten-Kombination gewinnt den gesamten Pot. Jeder Spieler hat die Auswahl aus drei Aktionsmöglichkeiten in jeder Spielrunde (Fold, Call, Raise) bedacht werden sollten etwas deutlicher.

Fold: Spielt ein Spieler fold, so verlässt er die aktuelle Spielrunde und verliert damit gleichzeitig die Chance den Pot dieser Runde zu gewinnen. Im Gegenzug verliert er kein (weiteres) Geld.

Call: Beim call erhöht der Spieler seinen eigenen Einsatz bis zu der Gesamtsumme, die derjenige Spieler mit dem aktuell höchsten Einsatz in den Pot eingezahlt hat. Hat in der aktuellen Spielrunde noch niemand einen Betrag gesetzt, so wird diese Aktion check genannt. Bet/Raise: Ein Spieler erhöht den aktuellen Einsatz über das Ausmaß des bis

dahin höchsten Einzeleinsatzes hinaus. Hat in der aktuellen Spielrunde schon ein weiterer Spieler den Einsatz erhöht, so wird diese Aktion raise genannt.

Varianten von Texas Holdem Limit/No-Limit In der Limit-Variante ist der Betrag der Wettgröße vorher fix festgelegt. Bei jeder bet/raise-Aktion kann jeder Spieler immer nur den gleichen, festgelegten Betrag setzen. In Turn und River wird dabei der Betrag verdoppelt. Der Betrag aus Preflop und Flop nennt sich small bet, derjenige in Turn und River big bet. Small Blind und Big Blind, den die beiden Spieler links vom Kartengeber zu setzen haben sind hierbei 0,5 small bet bzw. 1 small bet. In der No-Limit-Variante werden lediglich small blind und big blind festgelegt, anschließend ist jeder Spieler frei, Beträge zu setzen wie es für seine Strategie am günstigsten ist.

Heads-Up / Ring In der Heads-Up-Variante spielen lediglich zwei Spieler bzw. Poker-Agenten gegeneinander. Jedes Spiel mit mehr als zwei Mitspielern wird als Ring-Game bezeichnet.

Weitere Terminologie: Bluff Bei einem Bluff hat man selbst ein schwaches Blatt und versucht dem Gegner durch geschicktes Setzen zu suggerieren, man hätte ein starkes Blatt.

Semi-Bluff - Im Unterschied zum Bluff hat man beim Semi-Bluff mit dem eigenen Blatt noch eine Chance, ein gutes Blatt durch die öffentlichen Karten zu bekommen.

Trapping Hierbei versucht ein Spieler mit einem starken Blatt durch Auslassen einer Möglichkeit des bet/raise den Mitspielern zu suggerieren er habe ein schwaches Blatt, wodurch im Optimalfall für diesen Spieler die übrigen Gegenspieler motiviert werden trotz schwacher Blätter weiter zu setzen und somit mehr Geld in den Pot einbringen.

Value Bet Ein Spieler erhöht den Einsatz per bet und damit die potentielle Pot-Größe, die er mit seinem starken Blatt höchstwahrscheinlich gewinnt.

2. Hintergründe

Grundlegend handelt es sich bei der Entwicklung eines guten Poker-Agenten um die Lösung bzw. Annäherung der Lösung der Spieltheorie. Es wird dabei versucht, ein Nash-Gleichgewicht zu seinem Gegner zu finden, also eine Situation, in welcher keiner der beiden Spieler einen Vorteil daraus ziehen kann, wenn er von seiner Strategie abweicht.

2.1 Typen von Pokerstrategien

Bei einer gegenspielerabhängigen Spielstrategie ist eine Abwägung zu treffen zwischen dem Durchschauen des Gegners und Ausnutzen dieses Wissens und der eigenen Durchschaubarkeit. Gegen schwächere Gegner wird hierbei von einer anderen, mitunter offensiveren Spielstrategie ausgegangen, gegen einen stärkeren Gegner oder Gegner mit unbekannter Stärke wird eher auf Sicherheit gespielt, um nicht zu verlieren oder den Versuch zu unternehmen zunächst den Gegner zu durchschauen. Dabei gibt es zwei verschiedene, strategische Ansätze. Im ersten Ansatz werden verschiedene Strategien mit Wahrscheinlichkeiten gewichtet und dann zufällig (aber gewichtet) ausgewählt. Beim zweiten Ansatz wird mit einer (Nash-)Gleichgewichtsstrategie begonnen und fortgefahren, bis die Schwächen des Gegners entdeckt sind und darauf angemessen reagiert werden kann.

2.2 Pokerprogramme

Poker-Agenten spielen einige tausend Spiele pro Sekunde, so dass die statistische Auswertung mit hoher Genauigkeit erfolgen kann. In der vorliegenden Arbeit kommen zwei Techniken zur Anwendung, um die Varianz der auszuwertenden Strategie / Situation zu reduzieren und die Ergebnisse damit signifikanter zu machen: duplicate games und DIVAT Analysis. Die Performanz eines Poker-Agenten wird in small-bets/game (sb/g) gemessen. Diese Maßeinheit vergleicht den Einsatz eines Poker Agents mit dem Gewinn in der Serie von Spielen. Bei der Duplicate-games-Technik werden Spiele mit gleichen Kartenkonstellationen wiederholt um die Varianz zu reduzieren. Die DIVAT-Technik vergleicht bei jeder Kartenkonstellation die Stärken der Blätter der Spieler und entscheidet, wie eine normale (Basis-)Strategie das Spiel für jeden Spieler durchführen würde. Beide Techniken können kombiniert werden, um noch signifikantere Ergebnisse zu erzielen.

2.3 Erfassung von Spielsituationen

Man kann sich die Serien von Entscheidungen, die ein Pokerspieler während eines Spiels treffen muss als Baum vorstellen. Ein solcher Poker-Baum ist ein gerichteter Graph mit dem Startzustand des Spiels als Wurzel. Jeder Knoten im Baum stellt eine Entscheidung des Spielers dar. Es gibt dabei zwei verschiedene Strategieausprägungen, die sich auch im Detailierungsgrad des Baumes niederschlagen. Bei der einfachen Strategie wird bei einer bestimmten Kartenkonstellation immer dieselbe Aktion vollzogen (fold, call oder raise). Beispielsweise wird bei 2 Assen im Preflop immer die Aktion raise gewählt. Im Rahmen der verhaltensabhängigen Strategie wird mit einer festgelegten Wahrscheinlichkeit eine der drei Aktionen bei gleicher Kartenkonstellation ausgewählt. Beispielsweise wird mit (70% raise; 20 % call; 10 % fold) bei 2 Assen im Preflop vollzogen. Zur Herleitung der perfekten Pokerstrategie für den Poker-Agenten ist es notwendig, alle möglichen Spielsituationen sowie alle möglichen Spielzüge eines Spielers zu erfassen. Beim Texas Holdem (Heads up / Limit) ergeben sich sehr große Poker-Bäume, die sehr unübersichtlich erscheinen und dementsprechend schwer auszuwerten sind, um den besten Poker-Agenten zu entwickeln. Die Anzahl der Spielzustände in einem Pokerbaum hat eine Dimension von ca. $3 \cdot 10^{17}$, die der möglichen Spielzüge ca. $3 \cdot 10^{14}$.

2.4 Abstraktionsebenen

Die Größe des Pokerbaums muss also verringert werden, so dass sich handhabbare Größen ergeben. Da sich durch Ausnutzen des Kartenisomorphismus der Baum nicht hinreichend verkleinern lässt, verwendet man bei der Erfassung von Pokerspielsituationen die Bucketing-(Eimer-) Technik. Diese überführt die Spielgröße effizient in besser auszuwertende Dimensionen. Es wird eine definierte Anzahl von Eimern bereitgehalten, in welche diejenigen Spielkartenkonstellationen eingefüllt werden, die ähnliche oder gleiche Eigenschaften besitzen. Alle Konstellationen in einem Eimer werden anschließend mit der gleichen Strategie behandelt. Die Anzahl der bereitgehaltenen Eimer ist variabel. Werden beispielsweise 5-10 Eimer für die Kartenkonstallationen in der Preflop-Phase verwendet, so entsteht eine handhabbare Größe von Spielsituationen und Spielstrategien. Die Grundform Bucketing-Technik wird in weiteren Varianten verfeinert. Von der Computer Poker Research Group (CPRG) wurden abstrakte Spiele entwickelt, die unter Einsatz von 5-10 Eimern Strategien für Poker-Agenten entwickelt haben. Beim Vergleich dieser Strategien sowie der daraus resultierenden

Poker-Agenten ergab sich ein abnehmender Grenznutzen hinsichtlich einer größer werdenden Anzahl an Eimern und der Güte der Poker-Agenten. Die Poker-Agenten basierend auf sehr vielen Eimern werden also ab einer gewissen Anzahl nicht mehr signifikant besser beim Hinzufügen von Eimern. Es ist demzufolge hinreichend eine kleine Anzahl an Eimern zu wählen, da hieraus schon ein starker Poker-Agent entwickelt werden kann. Die Anzahl der Eimer steht in diesem Zusammenhang für die Abstraktionsebene.

2.5 Gegenzugstrategie

Generell ist es sehr schwierig und in Rechenschritten gesehen teuer auf die Strategie eines Gegners den bestmöglichsten Gegenzug bzw. eine Gegenzugstrategie zu entwickeln. Eine Annäherung dieses Ansatzes liefert die Gegenstrategie best response. In jeder Entscheidungssituation (der Spieler befindet sich gerade am Zug) wählt best response diejenige Aktion, die den Nutzen maximiert. Da dieser Algorithmus die Strategie bei jedem Spielzug sowie ebenso Informationen über die des Gegners als Eingabevariablen besitzt ist er bei der Entwicklung von Poker-Agenten nur insofern nützlich, als entwickelte Poker-Agenten mit ihrer Gegenstrategie hinsichtlich ihrer Stärke getestet werden können.

2.6 Auswahl aus mehreren Agenten

Ein weiterer Ansatz bei der Auswahl bzw. Entwicklung eines guten Poker-Agenten ist es, mehrere Poker-Agenten mit unterschiedlichen Strategien zu entwickeln und anhand der Performanz der einzelnen Agenten in vergangenen Spielsituationen denjenigen in einer ähnlichen oder gleichen Spielsituation erneut auszuwählen, der am besten in der damaligen Situation abgeschnitten hat. Doch auch hier ergeben sich Schwierigkeiten, da das Auswahlverfahren unglückliche Umstände bei der Entscheidung einzelner Agenten in der Vergangenheit nicht bzw. kaum berücksichtigen kann und es damit zur Situation kommen kann, dass der beste Agent zu Beginn der Spielserie einige unglückliche Situationen und Entscheidungen getroffen hat und somit unter Umständen nie wieder ausgewählt wird, obwohl dessen Strategie mitunter die beste wäre.

3. Modellierungsstrategien: Nicht-Verlieren vs. Gewinnen

Bei einer kleinen Anzahl von Eimern in der Bucketing-Technik wird im Allgemeinen derjenige Agent, der auf der kleineren Anzahl von Eimern basiert gegen denjenigen mit einer größeren Anzahl von Eimern als Basis verlieren. Demnach wird versucht, die größtmögliche Anzahl von Eimern in der Abstraktion zu verwenden und mathematisch zu lösen. Beschränkende Faktoren sind dabei u.a. die Größe des Speichers des verwendeten Rechners sowie die für einen Zug zur Verfügung stehende Zeit.

3.1 Nicht-Verlieren

Zunächst wird versucht, lediglich nicht zu verlieren. Der dabei behandelte Ansatz Counterfactual Regret Minimization (CFR) benötigt dabei Speicher in linearer Abhängigkeit zu den möglichen Spielzügen (nicht zu den Spielständen). Durch diesen Zusammenhang ist es möglich, bessere Nash-Gleichgewichtsstrategien zu entwickeln als bei den vorigen Ansätzen. Die Regret-Regel ist eine der strategischen Grundlagen der Entscheidungstheorie aus dem Bereich Operations Research . Es werden Opportunitätskosten für vergangene

Entscheidungen ermittelt. Beim CFR wurden zwei Spieler (dealer, opponent) konstruiert, die immer wieder (wiederholte) Spiele gegeneinander spielen und dabei ihre Strategie nach jedem Spiel so anpassen, dass die Opportunitätskosten (regret) minimiert werden. Kennt der Algorithmus in diesem Kontext beide Strategien, so ist er, ähnlich wie best response lediglich zur Untersuchung der angewandten Strategien zu verwenden. Die Errechnung des Regrets wird in folgendem Beispiel verdeutlicht: Gegeben sind die Wahrscheinlichkeiten für die drei Aktionen fold (0.5) , call (0.25) und raise (0.25). Die Veränderung des Vermögens des Spielers sei bei fold (-5), call (+10) und raise (+20). Der Erwartungswert errechnet sich in diesem Beispiel zu E(x) = 0,5*(-5)+0.25*10+0.25*20 = 5. Der Regret ist nun die Differenz aus der Veränderungen des Vermögens jeder einzelnen Aktion und dem Erwartungswert. Für fold ergibt sich -5 - E(x) = -10. Für die anderen beiden Aktionen ergibt sich Regret(call)= 5 sowie Regret(raise)=15. Anschließend wird jeder einzelne regret-Wert mit der Wahrscheinlichkeit des Gegners für die jeweilige Aktion gewichtet. Hat dieser für (fold/call/raise) jeweils 1/3 als Wahrscheinlichkeit, so ergeben sich als akkumulierte CFR nun (-1.33, 1.33, 5). Anschließend werden die eigenen Wahrscheinlichkeiten neu gewichtet. Der Aktion fold wird die Wahrscheinlichkeit 0 zugewiesen, da sie einen negativen Regret-Wert besitzt. Der Akti-on call wird 1.33 / 5 = 0.2666 und der Aktion raise 1-0.2666= 0.73333 als neue Wahrscheinlichkeit zugewiesen. So verfährt die Regret-Strategie weiter, bis ein Grenzwert des Regrets, das Nash-Gleichgewicht erreicht ist. Um den Algorithmus so effizient wie möglich zum Nash-Gleichgewicht zu bringen, wird in Implementierungen versucht, so viele Zweige abzuschneiden, wie es die Situation zulässt. In unserem Beispiel wäre dies der Fall bei der Aktion fold, da hier die Wahrscheinlichkeit aus Sicht beider Gegner auf 0 gesetzt wird. So kann die aufwendige Neuberechnung des Regrets für diese Aktion eingespart werden, was wie-derum wichtig für die Einhaltung des Zeitlimits ist bzw. zur Ausnutzung besserer Berechnungen des Nash-Gleichgewichts innerhalb des Zeitlimits dient. Durch das exakte und intelligente Lösen großer Nash-Gleichgewichtsprobleme war es nun möglich neue Gleichgewichtsstrategien zu entwickeln, die in der Lage waren, alle bisher bekannten Poker-Agenten in dieser Sektion zu besiegen. Allerdings befasste sich dieser Abschnitt mit der Strategie bzw. Absicht, nicht zu verlieren (not-to-lose), was in der Realität nicht ausreichend ist. In der Realität möchte man beim Pokern Geld gewinnen und nicht nur keines bzw. möglichst wenig verlieren. Denn es könnte einen Poker-Agenten geben, der allen anderen Spielern wesentlich mehr Geld als der eigene Poker-Agent entzieht und so in der Endabrechnung, trotz keinem einzigen verlorenen Spiel vor dem eigenen Agenten platziert ist. Um einen Wettbewerb zu gewinnen muss der eigene Poker-Agent also auch Spielen um zu gewinnen (playing-to-win), womit sich der folgende Abschnitt befasst.

3.2 Gewinn-Strategie

Die Strategie best response nutzt die Strategie des Gegners am besten aus und maximiert den eigenen Gewinn. Wie schon im vorigen Abschnitt beschrieben, ist es sehr rechenaufwendig und schwierig diese Strategie exakt zu implementieren. In der vorliegenden Master-Thesis wurde deshalb nach einer good response bzw. abstract game best response-Strategie (AGBR) verfahren. Das Ergebnis dieser Strategie ist die Untere Schranke für die wirkliche best response-Strategie, welche ein mindestens genauso gutes Ergebnis beim Pokern erzielen kann. Allerdings beinhaltet die AGBR drei entscheidende Nachteile: - Die Abstraktion des Gegners muss bekannt sein - Die Strategie des Gegners muss ebenfalls

eingegeben werden - Gegenstrategien reagieren ebenfalls auf eigene Anpassungen Daher kommt diese Strategie erst in einem zweiten Schritt zur Anwendung, nämlich dann, wenn genügend Informationen über den Gegner gesammelt sind. Im ersten Schritt wird über diese AGBR-Strategie die Frequentist Best Response-Strategie (FBR) aufgesetzt. Diese beobachtet den Gegner in einigen (Trainings-)Spielrunden, anschließend kommt AGBR zur Anwendung, um eine good response auf die Strategie des Gegners zu erwirken. Auch bei FBR gibt es zu Beginn eine Einschränkung: Zu dessen Anwendung müssen zu Beginn tausende Spiele unter vollständiger Information gespielt werden. Es müssen also alle Karten, sowohl die öffentlichen, als auch die privaten bekannt sein, und zwar auch dann wenn der Gegner die Karten per fold verwirft. Die Größenordnung der Anzahl der Spiele wird in folgendem Beispiel deutlich: Für die 5-Eimer-Variante müssen zwischen 100.000 und 1 Million Spiele gespielt werden, bis genügend Informationen gesammelt sind. Erst dann kann die AGBR angewendet werden.

3.2.1 AGBR und FBR in der Praxis

Je nachdem, wie durchschaubar die Strategie des Gegners ist, benötigt FBR weniger bzw. mehr Trainingsspiele um auf die Gegnerstrategie zu reagieren. Gegen verschiedene Poker-Agenten als Gegner mussten zwischen 10.000 und 600.000 Trainingsspiele absolviert werden um den Break-Even-Punkt zu überschreiten.

3.2.2 Wahl der Abstraktionsebene

Die Wahl der Abstraktionsebene, in diesem Falle der Anzahl der Eimer hat großen Einfluss auf das Ergebnis (millibets/game). Gegenüber einer 5-Eimer-Abstraktion mit 125 millibets/game hat die 8-Eimer-Abstraktion mit ca. 175 millibets/game einen klaren Vorteil. Des Weiteren wurde dabei deutlich, dass sich in der Größenordnung von 10 Millionen Trainingsspielen bei keiner der Abstraktionen eine Veränderung dieser Zahlen mehr ergibt. Es scheint demnach einen Grenzwert zu geben, gegen den die Poker-Agenten bei der Eruierung der FBR konvergieren.

3.2.3 Zusammenfassung der Merkmale von FBR

FBR erlaubt es uns, die eigene Strategie zu entwickeln und gleichzeitig die des Gegners auszuspionieren. Die Restriktionen, die FBR mit sich bringt sind bei weitem nicht so gravierend, wie die der ursprünglichen best response-Strategie. Dennoch gibt es zwei Situationen, in denen FBR keine optimalen Ergebnisse liefert. Ist der Gegner falsch modelliert, so zieht FBR mitunter die falschen Schlüsse, was im Wettbewerb eventuell mit großen Verlusten bezahlt werden muss. Außerdem setzt FBR voraus, dass der Gegner seine Strategie nicht ändert, während wir mit diesem trainieren. Ändert der Gegner auch seine Strategie während der Trainingsspiele, so zieht FBR ebenfalls die falschen Schlüsse hinsichtlich dessen Strategie, was zum selben Ergebnis führt, wie ein falsch modellierter Gegner-Agent.

4. Modellierungsstrategie: Restricted Nash Response

Bislang wurden Strategien von Poker-Agenten betrachtet, die gegen eine bekannte Gegnerstrategie gute Ergebnisse erzielten, gegen unbekannte Gegner jedoch teilweise sehr schwache Performanz boten. Nun wird mit der Restricted Nash Response (RNR) eine Möglichkeit

betrachtet, auch gegen unbekannte Gegner und bekannte Gegner mit teilweise dynamisch veränderbarer Strategie gute Gewinnergebnisse zu erzielen (stabile Gegenstrategie). Die Aufgabe von RNR ist es, die (mitunter neue bzw. geänderte) Strategie des Gegners herauszufinden und eine stabile Gegenstrategie in der Art zu bilden, dass der Gegner diese Strategie nicht gegen den eigenen Agenten nutzen kann bzw. nicht durchschauen kann. Im gebildeten Modell wird davon ausgegangen dass der Poker-Agent des Gegners mit Wahrscheinlichkeit p die Strategie unverändert verfolgt und mit Wahrscheinlichkeit (1-p) die Strategie in den Grenzen der üblichen Spieltheorie dahingehend verändert, dass für ihn das Optimum erreicht wird, bzw. der Versuch dessen gestartet wird. Es muß dabei ein Kompromiss eingegangen werden zwischen der eigenen Durchschaubarkeit (für den Gegner) und dem Durchschauen der Strategie des Gegners und deren (teilweise für den Gegner offensichtlichen) Ausnutzung. Für den Startzustand bedient sich RNR ebenfalls der FBR-Strategie. Diese wird anschließend anhand der Wahrscheinlichkeiten mit dem RNR-Algorithmus zu einer stabilen Gegenstrategie weiterentwickelt. Die Anwendung von RNR ist sinnvoll, wenn wie bei der AAAIPoker Callenge 2008 der Kontostand jedes Spielers nach Ende des Turniers eine Rolle spielt. Einfache best response-Strategien sind hierfür nicht ausreichend, da sie sehr leicht vom Gegner umgangen werden können. Zur weiteren Verbesserung von RNR ist es, wie in Abschnitt 2.6 erwähnt nun sinnvoll anhand eines Teams von Agenten mit einem intelligenten Auswahlverfahren den für die Spielsituation besten Agenten zu wählen und nach dessen Strategie die Pokerspielsituation für sich zu entscheiden. Auch an dieser Stelle muss erneut der Kompromiss zwischen Durchschaubarkeit und Durchschauen+Ausnutzen eingegangen werden. Das Team kann aus FBR-Agenten oder RNR-Agenten bestehen. Auch das Hinzufügen eines Agenten mit reiner Nash-Gleichgewichtsstrategie ist möglich. Zur Auswahl des Agenten mit der besten Chance das einzelne Pokerspiel zu gewinnen wird ein eigener Algorithmus (UCB1) eingesetzt. Bei Tests im Rahmen der behandelten Master-Thesis hatten die Teams aus RNR-Agenten eine weitaus bessere Performanz, so dass diese in Wettbewerben mit oftmals neuen Gegnern besser abschneiden werden, als FBR-Agenten. Teilweise waren diese auch besser als die Nash-Gleichgewichtsstrategie, dies macht sich in Wettbewerben, in denen die Gewinner anhand der Gesamtergewinne (Kontostände) ausgewählt werden. An dieser Stelle ist die Nash-Gleichgewichtsstrategie die sicherere Strategie. Unter einigen RNR-Agenten im Team kann aber unter Umständen ein Teammitglied gefunden werden, welches gegen einen neuen Gegner die Nash-Gleichgewichtsstrategie übertreffen kann.

5. Fazit und Ausblick

Aus Sicht von Michael Johannson gibt es einige Arbeitsfelder, die in Zukunft, auch in Bezug auf weitere AAAI Poker Challenges bearbeitet werden sollten. Zum einen ist dies die Ausnutzung von paralleler Bearbeitung der Probleme (CRM, RNR) auf mehreren Rechnern. Dadurch verspricht sich der Autor, größere Abstraktionslevel effizient errechnen zu können und somit stärkere Poker-Agenten zu implementieren. Ebenso sieht er Möglichkeiten die Modellierung von Poker-Agenten-Teams weiter zu verbessern und so die Strategieauswahl des Gegners vorherzusehen und darauf zu reagieren. Er nennt des Weiteren eine Möglichkeit die Anzahl der Abstraktionsebenen von bisher 12 Eimern im Preflop auf einige tausend zu erhöhen, indem einige Aktionen der Vergangenheit verworfen werden und somit mehr Spe-

icher im System für die weiteren Abstraktionsebenen frei wird. In der Master-Thesis wurden verschiedene Ansätze der Modellierung von Poker-Agenten aufgezeigt, sowie Anstöße für weitere Forschungen auf dem Gebiet der Computer Poker Challen-ges gegeben. Ebenso wurde das Ausmaß der Rechenoperationen und Ausprägungen die im Rahmen der Entwicklung eines Poker-Agenten bedacht werden sollten etwas deutlicher. Auch für die Arbeit an den Poker-Agenten im Rahmen dieses Seminars/Praktikums wurden einige Ideen dargelegt, deren Befolgen ein besseres Abschneiden bei der AAAI Poker Challenge 2008 versprechen kann.

Knowledge Engineering in Spielen - Klassische Artikel

Kamill Panitzek kamill.panitzek@gmx.de

Immanuel Schweizer immanuel.schweizer@gmx.de

Abstract

Die klassischen Artikel beschäftigen sich mit der effizienten Implementierung von Computergestützten Spielern für das Spiel Draw Poker. Dabei werden Methoden beschrieben wie Heuristiken genutzt werden können, um selbstständig lernende Spieler zu erschaffen. Diese können durch menschliches Einwirken, durch ein anderes Programm oder aber durch Deduktion eigenständig Heuristiken erstellen. Neben einigen weiteren dynamisch lernenden Strategien werden noch statische Strategien beschrieben, sowie eine Spielstrategie, die auf einem genetischen Algorithmus beruht. Außerdem wird beschrieben, wie die unterschiedlichen Spieler in Spielen gegeneinander oder gegen einen menschlichen Kontrahenten abschneiden.

1. Einleitung

Die vier Artikel im Themengebiet "Klassische Artikel" beschäftigen sich mit der Anwendung verschiedener Lerntechniken im Zusammenhang mit dem Spiel "Draw Poker". Dabei dient der Artikel (Waterman, 1970) mit seiner Analyse der Problemstellung als Ausgangspunkt für die Betrachtungen der weiteren Artikel. In den Artikeln werden unterschiedlichste Ansätze besprochen und so ein Überblick über die Möglichkeiten in diesem Bereich der Künstlichen Intelligenz geschaffen.

Das in den Artikeln verwendete Draw Poker ist deshalb besonders interessant, da es im Gegensatz zu Spielen wie Schach oder Dame ein Spiel mit imperfekter Information ist. Spiele mit perfekter Information wie z.B. Schach sind dadurch charakterisiert, dass jeder Spieler zu jeder Zeit alle Informationen über das Spiel besitzt und sich das Problem damit auf das Durchsuchen eines endlichen Suchraumes beschränkt. Bei Poker jedoch sind viele Informationen für den Spieler nicht ersichtlich. Die Karten der Gegenspieler z.B. sind immer unbekannt. Aus diesem Grund muss der Spieler die fehlenden Informationen schätzen oder durch Analyse des Verhaltens der Gegenspieler approximieren. Dieses Ziel wird durch Lernen, adaptive Suche oder statische Heuristiken erreicht. Doch um die verschiedenen Lernmethoden der klassischen Artikel erläutern zu können, müssen zunächst kurz die Regeln des Draw Poker beschrieben werden.

Draw Poker verwendet die gleichen Handkombinationen wie alle übrigen Pokerverianten auch, daher werden diese nicht weiter dokumentiert.Jede Spielrunde des Draw Poker besteht aus den folgenden fünf Phasen:

1. *Deal.* Jeder Spieler erhält fünf Karten (eine Hand) und gibt einen Chip in den Pot. Jeder Spieler sieht nur seine eigene Hand.

2. *Betting.* Jeder Spieler hat die Möglichkeit Chips zu setzen (bet), den geforderten Einsatz mitzugehen (call), oder die Spielrunde zu verlassen (drop oder auch fold). Ein call beendet die Bettingphase.

3. *Replace.* Jeder Spieler hat nun die Möglichkeit bis zu drei Karten seiner Hand gegen eine entsprechende Anzahl an Karten vom Deck zu tauschen.

4. *Betting.* Es folgt eine weitere Bettingphase.

5. *Showdown.* Die Spieler, welche die Spielrunde nicht verlassen haben, zeigen in dieser Phase ihre Karten. Der Spieler mit der höchsten Hand gewinnt die Spielrunde und erhält die Chips des Potts, die ihm zustehen.

Sidepots oder ähnliche Zusatzregeln werden hier nicht angeführt, da sie als bekannt vorausgesetzt werden. Genauere Informationen zu Draw Poker können dem Anhang des Artikels (Waterman, 1970) oder den Texten (Jacoby, 1947) und (Yardley, 1961) entnommen werden.

Im folgenden soll nun zuerst der (Waterman, 1970) Artikel vorgestellt werden, da er einen fundierten Einstieg in die Problemstellung und verschiedene Lösungsansätze bietet und teilweise den anderen Artikeln zu Grunde liegt. Anschließend werden die Artikel (Findler, 1977) und (Findler, 1978) besprochen, da hier viele verschiedene Strategien, sowohl statischer als auch lernfähiger Natur miteinander verglichen werden. Zum Schluss soll mit dem Artikel (Smith, 1983) noch eine weitere Herangehensweise an die Problemstellung beleuchtet werden.

2. Generalization Learning Techniques for Automating the Learning of Heuristics

2.1 Problembeschreibung

Der Artikel (Waterman, 1970) stammt aus dem Jahre 1970. Zu dieser Zeit hatten viele heuristische Programme ihre Heuristiken fest im Programm implementiert. Dieses Problem spricht dieser Artikel an. Ein Programm, dass Heuristiken nutzt um Probleme zu lösen, sollte in der Lage sein, die eigenen Heuristiken zu überwachen, ihre Qualität zu messen, sie zu ändern oder sich selbst mit neuen Heuristiken zu erweitern. Dieser Prozess wird als maschinelles Lernen von Heuristiken bezeichnet.

Das Problem des maschinellen Lernens kann in zwei Teilprobleme untergliedert werden:

1. Die Repräsentation der Heuristiken, um diese dann dynamisch verändern zu können.
2. Es werden Techniken benötigt, mit denen das Programm seine Heuristiken erzeugen, evaluieren und verändern kann.

Für die Repräsentation der Heuristiken werden sogenannte *production rules* verwendet. Um sie zu verändern bzw. zu verbessern, so dass sie das Problem effizienter lösen (in Bezug auf Draw Poker gewinnen), werden diese Heuristiken trainiert. Durch die Informationen, die durch das Training gewonnen werden, führt das Programm Generalisierungen durch, die entweder neue Heuristiken erstellen oder bestehende Heuristiken verändern. Hierbei gibt es zwei Arten des Trainings: Explizites Training wird von einem Menschen oder andere Programme durchgeführt, und implizites Training bei dem das Programm die Trainingsinformationen durch logische Deduktion im normalen Problemlösungsbetrieb gewinnt.

2.2 Repräsentation von Heuristiken

Eine gute Repräsentation von Heuristiken sollte vom eigentlichen Programm separiert und mit den Generalisierungsschemata kompatibel sein. Dabei sollte jede Heuristik eindeutig identifizierbar sein. Production rules erfüllen diese Anforderungen. Sie werden dennoch untergliedert in:

- *Heuristic Rule*: Eine Heuristik, die direkt eine Aktion beschreibt, die durchgeführt werden soll.
- *Heuristic Definition*: Eine Heuristik, die einen Term beschreibt.

Programme werden in dem Artikel als eine Menge von Berechnungsblöcken beschrieben. Jeder Block verändert dabei bestimmte Variablen des Programms. Also kann jeder Block als Funktion interpretiert werden. Dabei wird die Menge der sich ändernden Variablen als *state vector* ξ bezeichnet. Für die Funktion eines Berechnungsblocks gilt also die Gleichung $\xi' = f(\xi)$, wobei ξ' der resultierende state vector ist.
Für drei Variablen A, B und C mit den Werten a_1, b_1 und c_1 ergibt sich also

$$\xi' = (a'_1, b'_1, c'_1) = f(\xi) = f(A, B, C) = (f_1(\xi), f_2(\xi), f_3(\xi)).$$

Dementsprechend haben alle Regeln die Form $(a_1, b_1, c_1) \rightarrow (f_1(\xi), f_2(\xi), f_3(\xi))$.

Da auf diese Weise jedoch sehr viele Regeln nötig wären, um jede mögliche Kombination von Variablenwerten abzudecken, werden Mengen verwendet, um ganze Regelblöcke zusammenzufassen. Dabei werden die Prädikate $A1$, $B1$ und $C1$ verwendet. Aus der oben genannten Regel ergibt sich dann die Regel

$$(A1, B1, C1) \rightarrow (f_1(\xi), f_2(\xi), f_3(\xi))$$

mit $A1 = a_1$, $B1 = b_1$ und $C1 = c_1$. Dies sind Heuristic Rules und werden auch *action rules* (ac rules) genannt.

Heuristic Definitions werden in zwei Formen unterteilt. Die *backward form* oder bf rule ist ein Term und weißt einer Variablen ein Prädikat zu falls der Wert der Variablen der Bedingung entspricht: $A1 \rightarrow A, A > 20$.
Die *forward form* oder ff rule ist ein Berechnungsterm, der Variablen des state vectors kombiniert: $X \rightarrow K1 * D$.

Um in der aktuellen Situation den besten Spielzug zu treffen, muss eine Entscheidung vom Programm getroffen werden. Dies geschieht in zwei Schritten:

Schritt 1: Jedes Element des aktuellen state vectors wird mit die rechten Seiten aller bf rules verglichen. Wenn ein Prädikat der Bedingung genügt, wird die linke Seite dieser Regel mit die rechten Seiten aller bf rules verglichen usw. bis keine Übereinstimmung mehr gefunden wird. Die resultierende Menge an Symbolen definiert den neuen symbolischen Subvektor.

Schritt 2: Der symbolische Subvektor, der in Schritt 1 abgeleitet wurde, wird mit die linken Seiten aller action rules verglichen. Dabei werden die Regeln von oben nach unten der Reihe nach geprüft. Wenn die erste Übereinstimmung gefunden wird, werden die Variablen entsprechend der rechten Seite der action rule verändert.

2.3 Programmablauf

Bei Draw Poker betrifft die kritische Entscheidung die Bettingphase. Hier muss entschieden werden, wieviel gesetzt werden soll. Hierbei sind die gegnerische Hand, sowie der gegnerische Spielstil entscheidend. Gleichzeitig muss aber ein Spielstil adaptiert werden, der für den Gegner nicht leicht analysierbar ist. Das bedeutet insbesondere, dass das Programm einen Bluff des Gegners ermitteln können sollte, und ebenso selbst in der Lage sein muss zu bluffen.

Für diese Aufgabe wurden fünf verschiedene Programmvarianten erstellt:

- P[built-in]: Hat fest einprogrammierte Heuristiken.
- P[manual]: Besitzt Heuristiken, die durch Training mit einen Menschen erlernt wurden.
- P[automatic]: Hat Heuristiken, die durch Training mit einem anderen Programm erlernt wurden.
- P[implicit]: Besitzt Heuristiken, die durch implizites Training erlernt wurden.
- P[]: Hat nur eine action rule, die eine zufällige Entscheidung fällt.

In einem *Proficiency-Test* wurden die einzelnen Programmvarianten getestet. Dabei trat jede Variante gegen einen menschlichen Spieler an. Heraus kam dabei, dass das Programm P[built-in] um 5% besser, die anderen Varianten jedoch, meist sehr knapp, schlechter als der menschliche Spieler abschnitten. Gleichzeitig wurde auch gemessen wie schnell und effektiv die lernenden Varianten ihre Heuristiken aufbauten und wie redundant die daraus resultierenden Regeln waren. Dabei stellte sich heraus, dass explizites Training besser abschnitt als Implizites.

Um implizites Training zu realisieren, wurden dem Programm grundlegend die Regeln des Draw Pokers sowie einige Grundverhaltensregeln eingegeben. Aus diesen Pokeraxiomen deduzierte das Programm anschließend seine Heuristiken für das Pokerspiel. Insgesamt konnte ein Programm mit implizit trainierter Heuristik gewonnen werden, das in etwa so gut Poker spielt, wie ein erfahrener unprofessioneller menschlicher Spieler. Dabei blufft das Programm hauptsächlich in der ersten Bettingphase einer Spielrunde, also bevor die Karten der Spieler getauscht werden.

2.4 Erzeugung neuer Heuristiken

Um zu lernen, muss das Programm in der Lage sein neue Heuristiken zu erzeugen und diese seinem Satz Heuristiken hinzufügen. Aber gleichzeitig ist es auch wichtig, dass Heuristiken ausgetauscht werden können, die zuvor zu einer schlechten Entscheidung führten. Dafür wird eine System benötigt, welches die Heuristiken bewertet. Dabei werden die Spielzüge einer kompletten Runde betrachtet und jede Entscheidung wird bewertet. Jede production rule wird entweder positiv oder negativ bewertet, um somit zu ermitteln, in wie weit diese Regel zu einer schlechten oder guten Entscheidung beigetragen hat. Schlechte Regeln können dann durch Neue ausgetauscht oder verändert werden.

Um nun neue Heuristiken zu erzeugen oder bestehende zu verändern, wird die *Trainingsinformation* benötigt. Diese besteht aus drei Teilen. Die *Akzeptanzinformation* beschreibt,

wie gut oder akzeptabel eine Entscheidung für eine gegebene Situation ist. Die *Relevanzinformation* gibt an welche Elemente (Subvektor Variablen) für diese Entscheidung relevant sind. Der Grund, aus dem diese Entscheidung ausgeführt wurde, wird durch die Evaluierung der betroffenen Elemente ausgedrückt und wird durch die *Rechtfertigungsinformation* beschrieben. All diese Informationen werden im expliziten Training durch einen Menschen oder durch ein Programm festgelegt. Beim impliziten Training werden diese Informationen durch Beobachtung des eigenen Spielverlaufs ermittelt.

Das Lernen von Heuristic Rules wird erreicht, indem aus der Trainingsinformation eine action rule extrahiert wird, die *training rule* genannt wird. Dabei bedient die Akzeptanzinformation die rechte und die Relevanz- und Rechtfertigungsinformation die linke Seite der training rule. Nun wird eine Regel, die zur schlechten Entscheidung führte modifiziert, so dass sie dem symbolischen Subvektor der training rule entspricht. Diese Methode wird als *Generalisierung* bezeichnet. Falls das nicht möglich ist, wird die training rule direkt nach dieser schwachen Regel eingefügt.

Das Lernen von Heuristic Definitions besteht daraus, dass der Wertebereich der Variablen partioniert wird. Dies geschieht entweder in Form von mutual exclusion, was bedeutet, das der Wertebereich in genau zwei sich nicht überlappende Mengen geteilt wird, oder aber in Form von überlappender Partitionierung.

Beim impliziten Training kann die Rechtfertigungsinformation aus einer sogenannten *Decision Matrix* entnommen werden. Diese wird dem Programm übergeben, wenn das Training beginnt. Jede Zeile der Matrix steht dabei für eine Spielentscheidung bzw. eine Klasse von Spielentscheidungen und jede Spalte steht für eine Subvektor Variable. Jeder Eintrag in der Matrix beschreibt dann wie wichtig eine Variable des Vektors für eine gegebene Spielsituation ist.

Die Relevanzinformation kann aus der Generierung und dem Testen von Hypothesen über die Relevanz der einzelnen Subvektor Variablen gewonnen werden. Relevant für eine Regel sind Subvektor Variablen dann, wenn in der Regel auf der linken Seite ein anderer Wert als * steht. * auf der linken Seite einer Regel besagt, dass der Wert dieser Variablen irrelevant ist für das Zutreffen dieser Regel. Auf der rechten Seite einer Regel besagt *, dass der Wert der Variablen nicht verändert wird durch diese Regel.

Des weiteren muss darauf geachtet werden, dass Redundanzen in dem Satz Heuristiken möglichst gering gehalten werden. Das soll Speicher und auch Rechenzeit sparen. Es gilt also insbesondere, neue Heuristiken zu verwerfen, falls diese bereits existieren.

2.5 Erweiterungen

In dem Artikel wird außerdem beschrieben, wie die Decision Matrix erlernt werden könnte. Um die Matrix zu erlernen wird dem Programm zu Beginn eine leere Matrix übergeben. Erst durch die Erfahrungen, die das Programm während des Spieles macht, wird die Matrix gefüllt. Wenn das Programm dann also während dem regulären Betrieb auf die Decision Matrix zugreifen will und diese an der nötigen Stelle einen leeren Eintrag besitzt, so wird eine zufällige Entscheidung getroffen.

3. Computer Poker & Studies in Machine Cognition Using the Game of Poker

Dieser Abschnitt beschäftigt sich mit der Arbeit von Nicholas V. Findler (Findler, 1977) (Findler, 1978). In diesen Artikeln werden eine ganze Reihe interessanter und sehr unterschiedlicher Strategien vorgestellt. Ingesamt 6 statische und 12 lernfähige Bots wurden im Forschungsteam an der SUNY Buffalo entwickelt und verglichen.

Das Programmumfeld für diese Bots ist dabei unterteilt in drei Teile, das "Executive Program", das für den Informationsfluss und die Ausführung zuständig ist. Dann ein "Untility Program", das alle wichtigen Informationen über das Spiel sammelt oder berechnet. Sei es die Geschichte der bisherigen Spiele, statistische Informationen oder die Berechnung verschiedener Wahrscheinlichkeiten. Und als letztes die verschiedenen Bots, die in dieses Umfeld geladen werden konnten. Dazu kommt ein graphische Schnittstelle, über die ein Mensch mit den Pokerbots interagieren kann. Dies ist aber nicht nur für die Evaluation verschiedener Bots wichtig, sondern auch für einige Algorithmen, die einen menschlichen Mentor benötigen.

Das "Utility Program" enthält dabei auch eine sogenannte Monte Carlo Verteilung, in der festgehalten wird, wie wahrscheinlich es ist, mit einer Hand zu gewinnen. Dabei wird hier unterschieden zwischen der Wahrscheinlichkeit vor dem Draw, der Wahrscheinlichkeit nach dem Draw mit Bezug auf die Hände vor dem Draw und der Wahrscheinlichkeit nach dem Draw mit Bezug auf die Hände nach dem Draw. Diese Wahrscheinlichkeit wird von vielen Bots verwendet, die Stärke der eigenen Hand abzuschätzen, manche Algorithmen verlassen sich fast ausschließlich darauf.

Wenden wir uns nun aber der Beschreibung der verschiedenen Bots zu, dabei werden wir uns den beiden erfolgreichsten statischen Bots widmen und dann, zumindest kurz, alle lernfähigen Bots ansprechen.

3.1 Statische Bots

Da die statischen Bots ihr Verhalten über die Zeit nicht ändern, sind sie im Zusammenhang mit intelligenten Bots eher uninteressant. Zwei von Ihnen zeigen aber zumindest eine einigermaßen gute Leistung und dienen in der späteren Betrachtung als Benchmark für die lernfähigen Spieler.

Der erste dieser Bots verhält sich wie ein mathematisch perfekter Spieler (MFS). Dazu berechnet er einen Einsatz der sich aus dem Erwartungswert für den Gewinn und den Verlust berechnet.

$$p_j * B_0(j,k) = (1-p_j) * [\sum_{m=1}^{k-1}(B_a(j,m) + B_f(j,k))]$$

Dabei ignoriert dieser Bot, alle Informationen des Spiels außer die Stärke der eigenen Hand. Außerdem blufft er nicht und könnte von einen intelligenten Programm oder Mensch ständig ausgeblufft werden. Trotzdem ist er in den Experimenten als Sieger aus der Menge der statischen Bots hervorgegangen.

Der zweite statische Bot der beschrieben werden soll, benutzt zusätzlich Informationen zweiter Ordnung, wie z.B. wieviele Spieler ausgestiegen sind, Erhöhungen, die Sitzanordnung etc. Dazu wird ein Faktor berechnet, der die Motivation beschreibt, zum jetzigen

Zeitpunkt im Spiel zu bleiben. Der Faktor RH wird dabei berechnet, in dem die positive Motivation (die aktuelle Größe des Potts) durch die negative Motivation (Anzahl der aktiven Spieler, Anzahl der Erhöhungen, nötiger Einsatz, und die Spieler, die danach noch eine Entscheidung treffen können) geteilt und berechnet sich damit wie folgt.

$$RH = \frac{TABLEPOT}{LIVE*(RAISECOUNT+1)*FOLLOWERS*RAISE}$$

Auch dieser Bot verhielt sich sehr stark im Vergleich zu den anderen statischen Spielern und zieht zumindest einige Sekundäreffekte in Betracht. Im den nächsten Abschnitten werden mit den lernfähigen Bots nun die interessanten Teile der Artikel beschrieben.

3.2 Adaptive Evaluator of Opponents (AEO)

Dieser Bot versucht die aktuelle Situtation seiner Gegner abzuschätzen. Dazu beginnt er mit einem Wissensstand, der es ihm erlaubt, dies sehr grob zu tun. Mit steigender Erfahrung werden die sehr einfachen Abschätzungen weiterentwickelt zu einer detaillierten Einschätzen der "Persönlichkeit" jedes einzelnen Spielers.

Mit dieser Einschätzung schätzt der Bot ein, wie Stark die aktuelle Hand des Spielers ist, in dem er sie in Kategorien einsortiert. Die grobe Einschätzung, in welchem Bereich die aktuelle Hand ist, wird dann mit Hilfe der Erfahrung in drei Dimensionen verbessert.

Dabei ist die erste Dimension eine Sammlung von statistischen Werten, die im Spielverlauf nach jedem Showdown angepasst werden. Zum Beispiel der Pott oder der letzte Einsatz etc. In der zweiten Dimension wird die Gewichtung der einzelnen Faktoren für die verschiedenen Bereiche angepasst. Also wie stark sich ein statistischer Wert auf die Auswahl dieses Bereichs auswirkt. Die dritte Dimension versucht verschiedenen Persönlichkeiten, die optimale Auswahl statistischer Werte zuzuordnen. Denn obwohl jeder auf lange Sicht sein Geld maximieren will, können die Unterziele komplett unterschiedlich sein. So dass verschiedene statistische Faktoren bessere Ergebnisse liefern.

3.3 Adaptive Aspiration Level (AAP)

Dieser Bot basiert auf der Beobachtung, dass es bei Menschen zwei verschiedene Zustände gibt, die gleichzeitig, meist aber getrennt auftreten können. Diese Zustände sind, zum einen Maximierung des Gewinns und zum anderen Schutz des Eigenkapitals. Der Übergang zwischen diesen beiden Zuständen geschieht meist als Antwort auf ein einschneidendes Ereignis. Zum Beispiel, wenn man ein Spiel mit einer sehr starken Hand verliert oder eine Glückssträhne hat und mehrere Spiele hintereinander gewinnt.

Der AAP Bot benutzt nun diese Zustände, um die Höhe der eigenen Einsätze anzupassen. Er berechnet z.B. den Einsatz des MFS oder eines AEO und modifiziert diesen nach oben oder unten je nach aktuellem Zustand.

3.4 Selling and Buying Players' Images (SBI)

Dieser Spieler setzt einen gewissen Teil seines Geldes dazu ein, den anderen Spielern ein gewisses Bild von der eigenen Spielweise zu verkaufen. Dazu nutzt er unter anderem den Bluff, um sich als Risikofreudiger oder Konservativer darzustellen.

Außerdem setzt er einen Teil seines Geldes ein, um Informationen ber das Verhalten der anderen Spieler zu erkaufen. Um das Verhalten zu kategorisieren verwendet er zwei verschiedene Maßstäbe. Einmal wird der Grad der Konsistenz und zum anderen der Grad der Zurückhaltung eines Spieler in die Berechnung mit einbezogen. Diese Informationen werden dazu benutzt, um herauszufinden, wieviel Falschinformationen gestreut werden müssen, um ein gewisses Bild zu verkaufen.

Danach wird die MFS Strategie so angepasst, dass das gewünschte Selbstbild verkauft werden kann. Im Gegenzug hilft dieses falsche Bild, in kritischen Situation, den Gegner zu falschen Entscheidungen zu veranlassen.

3.5 Bayesean Strategies (BS)

Der "Bayesean" Spieler versucht zu jedem Zeitpunkt, seine Regeln anzupassen, in dem er Vorraussagen über den Spielausgang, mit dem tatsächlichen Spielausgang vergleicht und so Wahrscheinlichkeiten anpasst. Dabei geht der Artikel auf vier leicht unterschiedliche "Bayesean" Spieler ein. Diese wurden jeweils gegen die statischen Spieler getestet und nur der besten wurde zu weiteren Tests mit den dynamischen Spielern herangezogen.

Das Prinzip hinter allen ist aber das gleiche. Die Spieler haben vorkonfigurierte Heuristiken die auf drei Arten verändert werden können. Einmal können vordefinierte Parameter verändert werden, zum zweiten können die Heuristiken neu angeordnet werden, je nachdem wie erfolgreich sie sind und zum dritten können neue Heuristiken generiert, getestet und, falls erfolgreich, hinzugefügt werden.

Dazu wird ein kontinuirlich steigender Erfahrungspool verwendet, aus dem die Algorithmen Rückschlüsse in Bezug auf ihre Heuristiken ziehen. Die verschiedenen Spieler unterscheiden sich nur in der Menge und Granularität der Informationen, die sie mitspeichern. Der erste Spieler speichert dabei nur die Stärke der eigenen Hand und die dazugehörige Aktion. Er berechnet dann bei einer Entscheidung noch die durchschnittlichen Gewinne mit dieser Hand, der nächst Schlechteren und der nächst Besseren, um dann unter den Aktionen *bet*, *call* und *fold*, die Aktion mit dem maximalen Gewinn oder minimalen Verlust auszuwählen.

Der beste der "Bayesean" Spieler (3) dagegen, speichert sich außer diesen Informationen noch in welcher Phase des Spiels, welche Entscheidungen gefallen sind. Er erhält somit ingesamt bessere Strategien und kann damit auch schlechtere Hände erfolgreich spielen.

Die beiden anderen Spieler versuchen entweder, die Informationen noch nach einzelnen Spielern aufzuteilen (2) . Dabei entsteht aber das Problem, dass es dann selbst nach 3000 Spielen noch zu wenig Erfahrungswerte gibt. Die letzte Spieler (4) speichert sich noch mehr Informationen, in dem er nach jedem Spiel eine 20 x 5 Matrix verändert.

3.6 EPAM-like Player (EP)

Die Motivation hinter diesem Bot ist die Tatsache, dass ein menschlicher Spieler nicht linear besser wird, sondern manchmal Quantensprünge in der eigenen Entwickliung erlebt. Um diese Eigenschaft abbilden zu können basiert dieser Spieler auf einem EPAM Model (Feigenbaum, 1963). Dazu speichert er sich für jeden Gegner einen Entscheidungsbaum. Dieser wird für alle Gegner gleich initialisiert und dann dem Spiel entsprechend angepasst. Je

nachdem wie sich der gegnerische Spieler verhält, wird aus dem Baum so eine Abschätzung des Verhaltens dieses Spielers.

Dazu speichert sich der Spieler noch einen Baum über alle Spieler, der die optimalen eigenen Aktionen enthält. Auch dieser wird im Spielverlauf dem Verhalten aller Gegner angepasst. So entsteht am Ende ein normativer Entscheidungsbaum, der das Verhalten einer festen Menge an Gegnern quasi-optimal beschreibt.

Da zur Verfeinerung und zum Wachstum aller Bäume eine große Menge an Informationen nötig sind, wird bei diesem Spieler eine gewisse Menge vordefinierter Spiele gespielt, bis alle Möglichkeiten ausgetestet und mit Informationen gefüllt sind.

3.7 Quasi-Optimizer (QO)

Dieser Spieler überwacht eine große Anzahl von Spielen, zwischen den verschiedenen anderen Bots und beobachtet die Komponenten, die für eine Strategie zuständig sind. Daraus erstellt er einen "super player" (SP). Das bedeutet, dass seine Strategie innerhalb aller Verfügbaren Strategien zu einem Optimum konvergieren und so eine normative Theorie innerhalb des Strategieraums schaffen würde.

Dazu müssten die Entscheidungen, Aktionen sowie alle Komponenten in Kategorien einteilbar sein, um eine Analyse durch den QO zu ermöglichen. Zum Beispiel in welchem Teil des Spielablaufs Entscheidungen getroffen wurden, welcher Art die Entscheidungen waren und welcher Form die Entscheidung war.

Dies wurde bei den anderen Bots aber nicht bedacht und es ist auch fragwürdig, ob sich alle Entscheidungen in Kategorien pressen lassen. Dieser Bot befand sich zum Zeitpunkt der Artikel auch noch in der Entwicklung und es wurde an diesem Problem gearbeitet.

3.8 Pattern Recognizing Strategy (PRS)

Diese Erweiterung macht sich die Eigenschaft des Menschen zu nutzen, nach gewissen Mustern zu handeln. Ein menschlicher Spieler hat es auf Grund seiner Natur schwer zufällig zu handeln. Seinem Handeln liegt immer ein gewisses Muster zu Grunde.

Sie verfeinert so die Abschätzung des gegnerischen Verhaltens, in dem sie die Zusammenhänge zwischen beobachtbaren und nicht-beobachtbaren Spielelementen abstrakt darzustellen versucht. Dabei sind beobachtbare Elemente z.B. Erhöhungen, Größe des Potts etc. und nicht-beobachtbare Elemente z.B. die Hand des Gegners etc.

Die Zusammenhänge zwischen Elementen werden dann in die Kategorien "landmark", "trend", "periodicity" und "randomness" unterteilt. Diese Erweiterung zielt hauptsächlich darauf ab, die Bots bei der Einschätzung menschlicher Gegner zu unterstützen.

3.9 Statistically Fair Player (SFP)

Von allen statischen Bots zeigte der MFS die beste Performanz, obwohl er durch intelligente Bots einfach auszubluffen wäre. Es liegt also Nahe diesen Bot als Grundlage für einen lernfähigen Bot heranzuziehen und ihn selbst um den Bluff zu erweitern. Dann wird im Laufe des Spiels ähnlich wie beim SBI der Grad der Konsistenz und der Grad der Zurückhaltung berechnet.

Auf Grund dieser Einschätzung seines Gegenüber wird die Häufigkeit und die Höhe der Bluffs dynamisch angepasst, da bei einem konsistenten Spieler weniger geblufft und bei einem zurückhaltendem Spielverlauf niedriger geblufft wird.

3.10 Fazit

In den beiden Artikel wird eine Vielzahl verschiedener Ansätze zur Erstellung lernfähiger Bots gegeben. Soweit sie schon fertiggestellt waren liegen ebenfalls Ergebnisse bezüglich ihres Abschneidens gegeneinandern fest. Diese Ergebnisse werden von den Autoren zur Einschätzung der Stärke der Bots und ihrer Lernfähigkeit verwendet. Diese Ergebnisse sind vorallem im Artikel (Findler, 1977) umfassend beschrieben und um Abschätzungen zu den noch nicht implementierten oder getesteten Algorithmen erweitert.

Zusätzlich wird noch eine Umgebung zur Interaktion zwischen Mensch und Bots beschrieben, die aber im Zusammenhang mit dem Praktikum keine Relevanz besitzt. Im nächsten Abschnitt wird hier nun ein weiterer lernfähiger Bot beschrieben, der auf einer komplett anderen Idee beruht und sich durch seine Unabhängigkeit von der Problemstellung auszeichnet.

4. Flexible Learning of Problem Solving Heuristics through Adaptive Search

Wir haben uns im letzten Abschnitt mit einer ganzen Reihe verschiedener Lerntechniken beschäftigt. Eine Gemeinsamkeit all dieser Techniken ist, dass die Algorithmen und die Regeln sehr stark auf das Problemfeld zugeschnitten werden müssen um zu funtkionieren.

Bei dem letzten Ansatz, den wir uns nun genauer ansehen, ist dies nur eingeschränkt nötig. Im Gegenteil wird ein Framework vorgestellt, dass nur in sehr wenigen Teilbereichen überhaupt auf spezifisches Wissen zurückgreifen muss.

Die Idee, die hinter diesem Ansatz steckt, ist es, Lernen im Allgemeinen als Suche im Problemlösungsraum zu betrachten. Zur Suche wird hier ein Genetischer Algorithmus verwendet. Genetische Algortihmen sind mächtige Suchheuristiken, die nach dem Vorbild der Evolution arbeiten. Ihre Anwendunggebiete sind vorallem NP-schwere Optimierungsprobleme, wie z.B. das Dilemma des Handlungsreisenden.

Nun werden sie hier im Kontext von Künstlicher Intelligenz eingesetzt, um Heuristiken zur Problemlösung zu optimieren. Der Algortihmus besitzt dabei eine vorgegebene Menge an Lösungsheuristiken und geht dann in zwei, sich wiederholenden, Phasen vor. In Phase 1 werden die Heuristiken nach Performanz bewertet und ausgesucht und in Phase 2 wird die Menge durch genetische Operationen auf den Heuristiken erweitert. So wird sichergestellt, dass Teilräume durchsucht werden, die eine gute Performanz versprechen und sich diese Teile durch den kompletten Vorgang propangieren.

Jede Heuristik, oder besser Regel, besteht dabei aus einer Sequenz von Komponenten und die gentischen Operatoren arbeiten auf diesen Komponenten. Die Variante des Algorithmus, die in dem Artikel vorgestellt wird, arbeitet dabei auf verschiedenen Granularitätsebenen. Als oberste Ebene dient eine Sequenz von kompletten Aktionen und als unterstes Level einzelne Symbole.

Die genetischen Operationen, die dabei von den Autoren verwendet werden, sollen nun kurz vorgestellt werden.

Zuerst gibt es die *crossover* Operation. Dieser Operator nimmt zwei Regeln, auf der selben Ebene, und wählt jeweils einen Breakpoint aus. Die Komponenten rechts von diesem Breakpunkt werden dann ausgetauscht. Nimmt man also die Regeln c1c2‖c3c4 und c5c6‖c7c8, dann erhält man c1c2c7c8 und c5c6c3c4.

Als zweiten Operator gibt es die *inversion*. Diese Operation arbeitet auf einer Regel und definiert dort zwei Breakpoints. Die Komponenten zwischen diesen Breakpoints werden dann vertauscht. Als Beispiel wird so aus c1‖c2c3‖c4, c1c3c2c4. Dies funktioniert nur, wenn die Komponenten unabhängig voneinander sind, also nur auf der obersten Ebene.

Der letzte Operator ist die *mutation*. Sie tritt mit sehr geringer Wahrscheinlichkeit auf und verändert einzelne Komponenten zufällig. Sie stellt damit sicher, das alle Punkte des Suchraums erreicht werden können.

Das Programm selbst ist relativ einfach aufgebaut. Es gibt eine Menge von Regeln, die "knowledge base", zur Lösung des Problems. Diese Menge wird von der "problem solving component" auf k Probleme angewendet und dann von der "critic" analysiert. Diese bewertet die Menge und gibt sie danach an den genetische Algorithmus. Hier werden die besten ausgewählt und durch die oben besprochenen Operationen verändert. Ausserdem gibt die "critic" die aktuelle Problemlösung heraus. Dies ist in jedem Durchgang die Regel mit der jeweils besten Bewertung. Dabei ist die "problem solving component" problemunabhängig gestaltet. In dieses Framwork muss dann eine Menge an problemspezifischen Variablen und Operationen eingespielt werden. Auf dieser Menge arbeitet dann jede der Regeln und Aktionen und schafft so die Vorraussetzung, problemspezifische Lösungen zu finden.

Der wichtigste Teil ist aber eigentlich die "critic", denn sie misst die Performanz, der einzelnen Regeln und sorgt so für die Auswahl durch den genetische Algorithmus. Dazu vergleicht sie die Ausgabe des "problem solvers" mit problemspezifischen Benchmarks und Bestimmungen, aber vergleicht auch allgemeine Kriterien, wie z.B. die Komplexität oder Effizienz der Regeln, um eine feinere Abstimmung zu erreichen.

Die "critic" ist damit der Teil, der nicht Problemunabhängig gestaltet werden konnte. Um ihr System zu testen, griff die Gruppe nun auf das Pokersystem von (Waterman, 1970) zurück und übernahm auch die Pokeraxiome in die "critic" um die Performanz der Programme messen zu können. Ausserdem wurde die "problem solving component" mit 7 Zustandsvariablen gefüttert, auf denen seine Entscheidungen beruhen können. Diese sind: *Wert der eigenen Hand*, *Geldbetrag im Pott*, *Betrag des letzten Einsatzes*, *Verhältnis des Geldbetrags im Pott zum letzten Einsatz*, *Anzahl der Karten die der Gegner ausgetauscht hat*, *Wahrscheinlichkeit das der Gegner erfolgreich geblufft werden kann*, *Mass für die Zurückhaltung des Gegners*.

Dazu gibt es vier Grundoperationen, namentlich: *call*, *drop*, *bet low*, *bet high*.

Ausser der Übereinstimmung der Entscheidungen mit den Pokeraxiomen, wurde auch die Anzahl der mit Erfolg gespielten Runden in die Performanz mit eingerechnet.

Dieser Bot trat dann in einem Pokerspiel gegen den im Artikel von (Waterman, 1970) vorgestellten P[built-in] an, der in etwa die Stärke eines durchschnittlichen Pokerspielers hat. Bei den anschliessenden Test gegen P[built-in], sowie eine verbesserte Version von P[built-in] zeigte sich einmal, dass der Bot mit der Anzahl der gespielten Spiele immer besser wurde, als auch, dass er P[built-in] überlegen war oder gleiche Performanz zeigte, obwohl viel wenigerproblemspezifisches Wissen nötig war.

Abschliessend lässt sich damit sagen, dass der in diesem Artikel vorgestellte Ansatz, sehr vielversprechend ist. Die problemunabhängige Ausrichtung des Ansatzes und die trotzdem sehr guten Ergebnisse in einem definierten Problemumfeld zeigen hier die Stärken gut auf. Dabei ist zu beachten, dass die Ergebnisse nur so gut sind, wie die "critic". Denn nur wenn die richtigen Anforderungen an die Performanz gestellt werden, wird der Algortihmus dementsprechend konvergieren. So verschiebt sich das Problem aber von der Erstellung problemspezifischer Algorithmen und Lerntechniken, hin zu der Erstellung problemspezifischer Benchmarks und wird somit ungleich weniger komplex.

5. Fazit

In den letzten Abschnitten wurde deutlich, dass sich die Problemstellungen in der Künstlichen Intelligenz wenig verändert haben. Auch wenn die Spielvariante heute Texas Holding heißt, haben sich die Probleme nicht verändert.

Wie trifft man Entscheidungen in einem Umfeld imperfekter Information und Ungewissheit? Eine der spannenden Fragen in vielen verschiedenen Anwendungsgebieten.

Die hier vorgestellten Strategien sollten dabei als Grundlage für weiterführende Strategien verstanden werden und sind auch oft noch Bestandteil heutiger Strategien. Denn keine Strategie deckt für sich gesehen genug Möglichkeiten ab, um tatsächlich intelligent zu spielen. Aber sie zeigen die Richtung, in der verschiedene Ansätze der Künstlichen Intelligenz sich entwickeln können und haben.

References

Feigenbaum, E. A. (1963). The simulation of verbal learning behavior. *In Computers and Thought.*

Findler, N. V. (1977). Studies in machine cognition using the game of poker. *Communications of the ACM*, *20*, 230–245.

Findler, N. V. (1978). Computer poker. *Scientific American*, 112–119.

Jacoby, O. (1947). *Oswald Jacoby and Poker.* Double Day, Garden City N.Y.

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. *In Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, 421–425.

Waterman, D. A. (1970). Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, *1*, 121–170.

Yardley, H. O. (1961). *The Education of a Poker Player.* Pocket Books, New York.

Eine spieltheoretische Betrachtung des Pokerspiels

Stefan Lück STEFAN.LUECK@STUD.TU-DARMSTADT.DE
Fachbereich 01: Wirtschaftsinformatik
Technische Universität Darmstadt

Claudio Weck CLAUDIO.WECK@STUD.TU-DARMSTADT.DE
Fachbereich 20: Informatik
Technische Universität Darmstadt

Abstract

Die Entwicklung eines automatisierten Agenten, der beim Pokerspiel mit menschlichen Gegnern mithalten kann, beschäftigt schon seit längerer Zeit Wissenschaftler im Bereich der Erfoschung von künstlicher Intelligenz. Nachdem das Schachspiel mittlerweile fast optimal durch einen Computer gespielt werden kann, stellt das Pokerspiel eine ganz neue Herausforderung dar. Diese Arbeit beschreibt, wie man mit den Konzepten der Spieltheorie ein Pokerspiel analysieren und auch (theoretisch) lösen kann. Nach einer allgemeinen Einführung in die Spieltheorie und einer Einordnung des Pokerspiels in jener, werden Methoden für die exakte Darstellung und Lösung des Spiels aufgezeigt. Ähnlich wie beim Schachspiel ist allerdings aufgrund der Spielgröße eine vollständige Berechnung nicht möglich. Wir zeigen daher, welche Ansätze für Optimierungen und Approximationen es gibt, und wie man diese auf eine konkrete Problemstellung in Form der AAAI-PokerChallenge anwenden kann.

1. Spieltheoritsche Grundlagen

Die Spieltheorie beschäftigt sich mit der mathematischen Analyse von Entscheidungssituationen mit mehreren Akteuren. Die Akteure (*Spieler*) interagieren dabei miteinander, sodass die Entscheidungen eines Einzelnen von denen der Anderen beeinflußt werden. Erstmals formal beschrieben wurde die Spieltheorie in den 1920er und 1930er Jahren durch John von Neumann[1], der damit als Begründer dieser Forschungsrichtung gilt und zusammen mit Oskar Morgenstern 1944 das erste Standardwerk der Spieltheorie veröffentlichte[2]. Besonders im volkswirtschaftlichen Bereich hat sich die Spieltheorie für die Modellierung komplexer Interaktionen durchgesetzt[3].

Das Ziel der Spieltheorie besteht darin, Methoden für die Analyse von Spielen zur Verfügung zu stellen. Man unterscheidet dabei einerseits zwischen Werkzeugen zur Modellierung und Darstellung von Spielen sowie Verfahren zur Vorhersage von Spielstrategien andererseits. Die Vorhersage von Spielstrategien bezeichnet man auch als das Lösen eines Spiels (siehe dazu Abschnitt 1.3).

1.1 Definition eines Spiels

Formal betrachtet besteht ein Spiel in der Spieltheorie aus einer Menge an *Spielern*, die *Entscheidungen* treffen müssen. Jeder Spieler kann dazu aus einer vorgegebenen Menge

1. Deutsch-ungarischer Mathematiker (*1903 - †1957).
2. Vgl. (von Neumann & Morgenstern, 1944).
3. Z.B bei der Modellierung der Auktion zur Versteigerung von UMTS-Lizenzen.

an *Handlungsalternativen* wählen. Außerdem gibt es für jeden Spieler eine *Nutzenfunktion*, die in Abhängigkeit der getroffenen Entscheidungen *aller* Spieler einen Auszahlungsbetrag, also seinen persönlichen Nutzen, festlegt. Eine *Strategie* für einen Spieler beschreibt nun, für welche der Alternativen sich ein Spieler unter welchen Umständen entscheidet. Beispiel 1 beschreibt den Aufbau eines einfachen Interaktionsspiels.

Definition 1 Mathematische Formulierung eines Spiels

$\mathcal{P}$		*Menge an Spielern*
$\mathcal{A}_p$	$.\forall p \in \mathcal{P}$	*Handlungsalternativen pro Spieler*
$u_p : \mathcal{A}_1 \times \cdots \times \mathcal{A}_{\lvert\mathcal{P}\rvert} \to \mathbb{R}$	$.\forall p \in \mathcal{P}$	*Auszahlungsfunktion pro Spieler*

Beispiel 1 *Bei dem Spiel Schere-Stein-Papier gibt es zwei Spieler. Jeder Spieler muss sich entscheiden, welche Handlungsalternative in Form von drei möglichen Symbolen (Schere, Stein oder Papier) er wählt. Dabei gilt, dass Schere gegen Papier gewinnt, Papier gegen Stein und der Stein gegen die Schere. Abhängig von der Entscheidung des Gegners gewinnt also entweder ein Spieler selbst (Auszahlung $+1$), sein Gegner (Auszahlung -1) oder es gibt ein Unentschieden, wenn beide Spieler das gleiche Symbol wählen (Auszahlung 0). Eine mögliche Strategie eines Spielers wäre, immer das Symbol Schere zu wählen. Eine andere Strategie könnte sein, dass er sich zufällig für eines der Symbole entschiedet.*

Für die Darstellung eines Spiel wird häufig die sogenannte *Normalform* eines Spiels gewählt. Diese besteht aus einer $|\mathcal{P}|$-dimensionalen Matrix, wobei jede Dimension der Matrix durch die Handlungsalternativen genau eines Spielers aufgespannt wird. Die Zellen der Matrix geben die Werte der Auszahlungsfunktion für die einzelnen Spieler an. Diese sind voneinander durch Semikola getrennt. Im Fall von zwei Spielern ist die Normalform eine einfache Tabelle. Tabelle 1 zeigt eine Normalformdarstellung des Schere-Stein-Papier-Spiels aus Beispiel 1.

		Spieler 2		
		Schere	Stein	Papier
Spieler 1	Schere	0 ; 0	-1 ; 1	1 ; -1
	Stein	1 ; -1	0 ; 0	-1 ; 1
	Papier	-1 ; 1	1 ; -1	0 ; 0

Tabelle 1: Spiel Schere-Stein-Papier in Normalform

1.2 Klassifikation von Spielen

Spiele können nach ihrer Art und ihren Eigenschaften klassifiziert werden. Dies ist notwendig, um die richtigen Methoden und Instrumente für die Analyse wählen zu können. Eine zentrale Unterscheidung erfolgt in statische und dynamische Spiele. Bei *statischen (oder strategischen) Spielen* erfolgt die Entscheidung der Spieler für eine Handlungsalternative simultan. Kein Spieler kennt die Entscheidungen der anderen zu dem Zeitpunkt, an dem er seine eigene Strategie auswählen muss. Statische Spiele sind gut zur Modellierung von Situatuationen geeignet, wie sie häufig in der Wirtschaft und Politik auftreten. Dabei müssen die

Handelnden Entscheidungen über die Zukunft treffen, wobei sich die gewählten Strategien der Mitspieler erst im Laufe der Zeit offenbaren. Typische Beispiele für solche Situationen sind Entscheidungen über Markteintritte, einen Kapazitätenausbau oder das Rüstungsverhalten. Auch das Schere-Stein-Papier-Spiel ist ein statisches Spiel. Für die Darstellung von statischen Spielen kann direkt die Normalform verwendet werden.

Für die Modellierung von Gesellschaftsspielen hingegen sind statische Spiele i.S.d. Spieltheorie häufig nicht geeignet, da bei jenen die Spieler zumeist abwechselnd Entscheidungen treffen. Solche rundenbasierten Situationen werden mit *dynamischen Spielen* beschrieben. Die Spieler sind dabei nacheinander am Zug und müssen jeweils eine Entscheidung treffen. Jede dieser Entscheidungen beeinflusst den weiteren Verlauf des Spiels und beschränkt häufig den Alternativenraum der nachfolgenden Spieler (z.B. wenn ein Spieler beim Schachspiel den König seinenes Gegners bedroht). Die Normalform kann daher nur schwer zur intuitiven Darstellung dynamischer Spiele verwendet werden. Ein besser geeignetes Instrument ist die extensive Darstellung, die in Abschnitt 2.1 beschrieben wird.

Häufig wird der Spielverlauf auch durch (bewusst herbeigeführte) zufällige Ereignisse beeinflusst. Dies ist insbesondere bei Würfelspielen (durch Würfeln) und Kartenspielen (durch Mischen) der Fall. Durch den Einfluss des Zufalls kann es passieren, dass eine Strategie bei ansonsten identischen Spielverlauf entweder eine sehr hohe oder sehr niedrige Auszahlung bewirken kann. Dies ist z.B. offensichtlich bei Würfelwettspielen: Man setzt einen bestimmten Betrag auf eine gerade oder ungerage Augenzahl. Abhängig von der gefallenen Zahl bekommt man nun entweder den doppelten Einsatz zurück oder man verliert seinen Einsatz. Die Existenz von zufälligen Momenten in einem Spiel ist also eine zentrale Eigenschaft, die bei einer Analyse des Spiels berücksichtigt werden muss.

Eine weitere wichtige Eigenschaft von Spielen ist die Existenz von privaten Informationen. Unter privaten Informationen sind spielrelevante (also entscheidungsbeeinflussende) Daten zu verstehen, die nicht allen Spielern offen vorliegen. Spieler sind dadurch in der Lage Entscheidungen zu treffen, die andere nicht vorhersagen können. Bei einem Kartenspiel stellen zum Beispiel die Karten, die ein Spieler auf der Hand hat, eine private Information dar. Auch die eigenen Entscheidungen im Spielverlauf stellen häufig eine private Information dar. Beim Spiel Schiffe-Versenken z.B. liegt die erste Entscheidung für jeden Spieler in der Platzierung der Schiffe. Diese Information behält jeder Spieler allerdings für sich und hat dadurch einen Informationsvorsprung. Spiele, in denen keine privaten Informationen vorliegen, bezeichnet man als *vollkommene* oder *perfekte* Spiele. Ein Beispiel für ein Solches Spiel ist das Schachspiel, bei dem alle Informationen über das Spiel (man spricht vom *Spielstatus*) jedem Spieler offen zugänglich sind.

Tabelle 2 gibt eine kurze Übersicht und klassifiziert beispielhaft Spiele nach ihren strukturellen Eigenschaften. Bei den aufgeführten Spielen handelt es sich um dynamische Spiele.

	ohne Zufall	mit Zufall
vollkommen	Schach Go	Backgammon Kniffel
nicht vollkommen	Schiffe versenken	Poker Rommé

Tabelle 2: Klassifikation von Spielen (Beispiele)

Neben der Klassifikation von Spielen nach strukturellen Kriterien kann man eine weitere Untergliederung nach inhaltlichen Aspekten vornehmen. Diese Eigenschaften können dann häufig zur Vereinfachung der Lösungssuche verwendet werden. Eine mögliche Eigenschaft eines Spiels ist z.B. die *Symmetrie*. Ein Spiel ist genau dann symmetrisch, wenn alle Spieler über identische Alternativenräume und Nutzenfunktionen verfügen ($\mathcal{A}_p = \mathcal{A}_q \wedge u_p(a_1, \ldots, a_{|\mathcal{P}|}) = u_q(a_1, \ldots, a_{|\mathcal{P}|}) \; . \forall p, q \in \mathcal{P}$). Für einen Spieler ist es also unerheblich, ob er an erster oder zweiter Position spielt. Ein Beispiel für ein symmetrisches Spiel ist das Schere-Stein-Papier-Spiel.

Außerdem kann man zwischen nicht-kompetitiven und kompetitiven Spielen unterscheiden. Bei nicht-kompetitiven Spielen erreichen die Spieler alle einen höheren Nutzen, wenn sie zusammenarbeiten. Ein Beispiel für ein solches Spiel ist die Mammutjagd, wo Jäger in der Gruppe jagen müssen, um ein großes Mammut zu erbeuten und so einen hohen Nutzen zu erzielen. Bei (strikt) kompetitiven Spielen hingegen Verhalten sich die Auszahlungen der Spieler reziprok zueinander, d.h. damit ein Spieler eine höhere Auszahlung bekommen kann, muss sich gleichzeitg die Auszahlung einer seiner Mitspieler (Gegner) verringern. Eine typische Klasse von Spielen, die diese Eigenschaft besitzen, sind die Nullsummenspiele. Dabei ergibt die Summe aller Auszahlungen immer Null (oder allgemein eine Konstante c, was formal äquivalent ist: $\sum_{p \in \mathcal{P}} u_p(a_1, \ldots, a_{|\mathcal{P}|}) = c$), woraus das reziproke Verhalten der Nutzenfunktionen direkt folgt.

Wie bereits aus Tabelle 2 zu entnehmen ist, handelt es sich bei Poker um ein dynamisches Spiel mit unvollkommener Informationen und Zufallsereignissen. Poker ist dabei ein strikt kompetatives Nullsummenspiel (einer gewinnt, alle anderen verlieren) und nicht symmetrisch, da die zu wählenden Handlungsalternativen von der Position am Tisch abhängen. Diese Klassifikation trifft auf die meisten Kartenspiele zu. Das Zufallsmoment resultiert dabei aus dem Mischen der Karten und der anschließenden zufälligen Verteilung. Durch das Halten der Karten auf der Hand schützt jeder Spieler seine privaten Informationen, dadurch ist das Spiel unvollkommen. Ein Beispiel für ein Kartenspiel mit perfekten Informationen hingegen ist z.B. Offizierskat, wo alle Karten offen auf dem Tisch liegen.

1.3 Vorhersage des Spielverlaufs

Im Rahmen eines Spiels trifft jeder Spieler Entscheidungen, durch die die Auszahlungen am Ende des Spiels beeinflusst werden. Die Art und Weise, wie ein Spieler seine Entscheidungen trifft, nennt man seine *Strategie*. Die Gesamtheit der Strategien aller Spieler kann man zu einem Strategienvektor zusammenfassen. Dieser Strategienvektor bestimmt den Verlauf des Spiels eindeutig, da für jede mögliche Situation des Spiels die Handlungsweise des jeweils am Zug befindlichen Spielers definiert wird. Fasst man die Strategien von allen Spielern bis auf einen zusammen, erhält man einen Fremdstrategienvektor, der den Verlauf des Spiels in Abhängigkeit der Entscheidungen eines einzelnen Spielers wiederspiegelt. Eine Betrachtung dieses Vektors ist sinnvol, wenn man ein Spiel aus der Sicht eines einzelnen Spielers betrachtet und die Strategien der Mitspieler als gegeben ansieht.

Definition 2 Strategien
Eine Strategie $s_p \in \mathcal{S}_p$ eines Spielers $p \in \mathcal{P}$ beschreibt für jede denkbare Entscheidungssituation eines Spielers, für welche Handlungsalternative er sich entscheiden wird. Sie definiert also genau das Verhalten eines Spielers während des Spiels.

Ein Strategienvektor S ist eine Sammlung, die jeweils genau eine Strategie für jeden Spieler enthält. Dadurch wird das Verhalten aller Spieler und damit der gesamte Spielablauf definiert:

$$S = \{s_p | \forall p \in \mathcal{P}\}$$

Ein Fremdstrategienvektor S_{-p} enthält eine Sammlung aller Strategien außer der eines einzelnen Spielers $p \in \mathcal{P}$:

$$S_{-p} = \{s_q | \forall q \in \mathcal{P} \wedge q \neq p\}$$

Jeder Spieler kann seine Strategie für ein Spiel frei wählen. Dabei geht man davon aus, dass sich ein Spieler aber nicht für eine beliebige Strategie entscheiden wird, sondern versucht, durch die Wahl einer geeigneten Strategie seinen eigenen Nutzen am Spielausgang zu maximieren. Dieses grundlegende Verhalten eines Individuums, dass man in der Spieltheorie für alle Spieler annimmt, nennt man das Rationalitätsprinzip.

Definition 3 Rationalitätsprinzip
Das Rationalitätsprinzip besagt, dass sich jeder Spieler für genau die Strategie s_p^* entscheiden wird, die ihm die höchste Auszahlung am Spielende sichert. Nimmt man die Strategien der Mitspieler als gegeben an, ist eine optimale Strategie *der Spieler also:*

$$s_p^* = \arg\max_{s_p \in \mathcal{S}_p} u_p(s_p | S_{-p}) \quad . \forall p \in \mathcal{P}$$

Damit ein Spieler eine optimale Strategie wählen kann, müsste er aber bereits die Strategie des Gegners kennen. Dies ist normalerweise aber nicht der Fall, daher muss ein Spieler Annahmen über die Strategien seiner Mitspieler treffen. Diese Annahmen können aus Erfahrungen von früheren Teilnahmen an einem Spiel resultieren. Es ist dabei davon auszugehen, dass jeder Spieler seine Strategie solange ändert, bis er seinen Nutzen nicht mehr verbessern kann[4]. Wenn jeder Spieler eine solche Strategie gefunden hat, befindet sich das Spiel in einem Gleichgewicht, d.h. keiner der Spieler wird seine Strategie ändern, solange auch alle anderen Spieler ihre Strategie beibehalten. Ein Strategienvektor, der diese Bedingung erfüllt, nannt man *Nash-Gleichgewicht*[5].

Definition 4 Gleichgewichtsstrategien
Ein Strategienvektor stellt genau dann eine Gleichgewichtsstrategie[6] *dar, wenn die Strategien der Spieler ihre jeweils* gegenseitig besten Antworten *auf die Strategien ihrer Mitspieler darstellen. Es kann also kein Spieler seine Auszahlung durch das Ändern seiner Strategie erhöhen, das Rationalitätsprinzip ist für alle Spieler gleichzeitig erfüllt:*

$$S^* \rightarrow \forall p \in \mathcal{P}. \quad s_p^* = \arg\max_{s_p \in \mathcal{S}_p} u_p(s_p | S_{-p}^*)$$

4. Für weitere Erklärungen über das Zustandekommen eines Gleichgewichts siehe (Osborne, 2004, S. 134ff).
5. Benannt nach dem Mathematiker und Nobelpreisträger John Forbes Nash Jr., der diese Eigenschaft zuerst 1950 beschrieben hat.
6. Vgl. zum Nash-Gleichgewicht insbesondere (Nash, 1951).

In der Spieltheorie bezeichnet man eine Gleichgewichtsstrategie, also die Menge der einzelnen optimalen Spielerstrategien, als die *Lösung* eines Spiels. Ein Spiel kann mehrere Lösungen haben oder auch gar keine. Bei einer Betrachtung des Spiels Schere-Stein-Papier aus Beispiel 1 fällt auf, dass es offensichtlich keine Kombination von Symbolen gibt, die dieses Kriterium erfüllt. Unabhängig von dem Symbol, das ein Spieler wählt, gibt es immer ein Symbol, gegen das er damit verlieren wird. Der Gegner würde also seine Strategie anpassen wollen um ein Symbol zu wählen, mit dem er gewinnt. Es gibt kein Gleichgewicht, und daher auch keine Lösung.

In dem vorangegangenen Beispiel wurde unterstellt, dass sich ein Spieler immer für genau ein Symbol entscheiden muss und seine Strategie daher in der Wahl genau einer einzigen Handlungsalternative besteht. Man spricht in diesem Fall von einer *reinen Strategie*. Eine wichtige Erweiterung der Spieltheorie besteht in dem Konzept, dass man auch gemischte Strategien zulässt. Bei einer *gemischten Strategie* spielt ein Spieler jede Handlungsalternative mit einer bestimmten (positiven) Wahrscheinlichkeit. Die Auszahlung, die ein Spieler mit dieser Strategie erreicht, ist eine nach den Wahrscheinlichkeiten gewichtete Kombination der Auszahlungen, also der Erwartungswert der Auszahlung.

Definition 5 Gemischte Strategie und erwartete Auszahlung
Eine gemischte Strategie ordnet jeder möglichen Handlungsalternative $a_p \in \mathcal{A}_p$ eines Spieler eine bestimmte Wahrscheinlichkeit π zu, mit der er diese Alternative wählen wird.

$$s_p \in \mathcal{S}_p = \left\{ \begin{pmatrix} \pi_1 \\ \vdots \\ \pi_{|\mathcal{A}_p|} \end{pmatrix} \mid \sum_{i=1}^{|\mathcal{A}_p|} \pi_i = 1 \ \wedge \ \forall \pi_i : \pi_i \geq 0, \ i \in \{1, \ldots, |\mathcal{A}_p|\} \right\}$$

Die erwartete Auszahlung ergibt sich aus der Gewichtung der Einzelauszahlungen einer Handlungsalternative mit der Wahrscheinlichkeit, dass diese gespielt wird.

$$u_p(s_p|S_{-p}) = E\left(u_p|S_{-p}\right) = \sum_{i=1}^{|\mathcal{A}_p|} \pi_i \cdot u_p(a_{p,i}|S_{-p})$$

Das Konzept der Gleichgewichtsstrategien kann man nun analog auf gemischte Strategien ausweiten. Lässt man z.B. gemischte Strategien für das Spiel Schere-Stein-Papier zu, exisitert ein Gleichgewicht in der Strategiekombination $s_1 = s_2 = \left(\frac{1}{3} \text{ Schere}, \frac{1}{3} \text{ Stein}, \frac{1}{3} \text{ Papier}\right)$[7]. Allgemein kann man beweisen, dass es für jedes Spiel mit einer endlichen Menge an Handlungsalternativen eine Gleichgewichtsstrategie in gemischten Strategien gibt[8]. Bei strikt kompetitativen Spielen ist dieses Gleichgewicht obendrein eindeutig. Für eine mögliche Berechung des Gleichgewichts siehe Abschnitt 2.2.2.

2. Poker: Ein unvollkommenes dynamisches Spiel mit Zufall

Wie bereits in Abschnitt 1.2 dargestellt, handelt es sich beim Pokerspiel um ein unvollkommenes dynamisches Nullsummenspiel mit Zufallseinfluß. Für diese Klasse von Spielen

7. Die Strategien im Gleichgewicht müssen für alle Spieler gleich sein, da das Spiel symmetrisch ist.
8. Vgl. dazu z.B. (Nash, 1951).

stellt die Spieltheorie spezielle Werkzeug und Methoden zur Verfügung, die für eine exakte Darstellung und genaue Lösung verwendet werden können. Im folgenden Abschnitt 2.1 wird eine Darstellungsform für dynamische Spiele vorgestellt, die eine einfache und intuitive Interpretation ermöglicht, die extensive Darstellung. Im Anschluss daran werden in Abschnitt 2.2 klassische Lösungsverfahren beschrieben, mit denen eine exakte Lösung für die Klasse der unvollkommenen dynamischen Nullsummenspiel berechnet werden kann.

2.1 Extensive Darstellung

Die in Abschnitt 1.1 vorgestellte Normalform eines Spiels eignet sich nur sehr schlecht zur Darstellung eines dynamischen Spiels, da die Modellierung von zugbasierten Entscheidungen nicht vorgesehen ist. Eine intuitivere Darstellung kann durch einen Spielbaum erreicht werden, was auch als die *extensive Darstellung* eines Spiel bezeichnet wird. Dabei werden alle zu treffenden Entscheidungen durch *Knoten* in einem Baum repräsentiert, denen jeweils genau ein Spieler, nämlich der am Zug befindliche, zugeordnet wird. Die einzelnen Handlungsalternativen, die einem Spieler für jede seiner Entscheidungen zur Verfügung stehen, werden durch die ausgehenden *Kanten* im Baum repräsentiert. Die Auszahlungen an die Spieler notiert man in den *Blättern*, die das Spielende symbolisieren. Abbildung 1 zeigt einen einfachen generischen Spielbaum.

Ein Spiel beginnt immer an der Wurzel, wo der erste Spieler eine Entscheidung treffen muss und so die erste Verzweigung wählt. Im Anschluß trifft der nächste Spieler eine Entscheidung und verzweigt so tiefer im Baum. Dieser Prozess wiederholt sich, bis das Spiel schließlich beim Erreichen eines Blattes endet und die Spieler die entsprechenden Auszahlungen erhalten. Ein vollständig gespieltes Spiel ist also ein Pfad von der Wurzel des Baumes zu einem seiner Blätter. Die Folge von Entscheidungen, die zu diesem Verlauf geführt haben, nennt man eine *Spielinstanz*. Sie ist ein Ausschnitt aus den jeweiligen Strategien der Spieler.

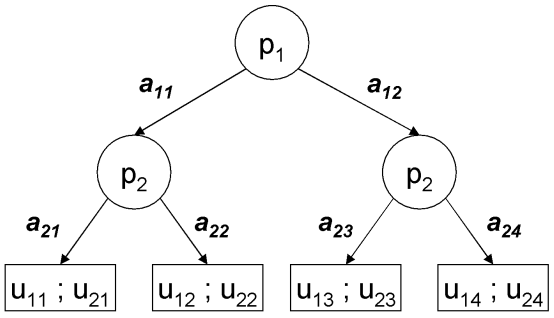

Abbildung 1: Ein einfacher generischer Spielbaum

Ein einfacher Spielbaum ist allerdings noch nicht ausreichend, um auch zufällige Ereignisse sowie unvollkommene Information zu modellieren. Dazu benötigt man weitere Konzepte, die durch einen *erweiterten Spielbaum* dargestellt werden. Für die Modellierung des Zufalls fügt man einen weiteren imaginären Spieler hinzu, gewissermaßen die Natur. Das Verhalten der Natur ist dabei in Form einer gemischten Strategie mit einer vorgegebenen Wahrscheinlichkeitsverteilung gegeben. Die jeweiligen Wahrscheinlichkeiten notiert man an den einzelnen Kanten, die die alternativen Zufallsausgänge repräsentieren.

Für die Modellierung von unvollkommenen Informationen muss zunächst geklärt werden, wodurch diese zustande kommen. Unvollkommene Informationen bedeuten, dass einem Spieler nicht alle Informationen über den aktuellen Status des Spiels zur Verfügung stehen. Der aktuelle Status wird dabei durch die bisherigen im Spiel getroffenen Entscheidungen

repräsentiert und ist somit ein Pfad von der Wurzel des Baumes zum aktuellen Knoten. Liegt unvollkommene Information vor, so kennt ein Spieler nicht den aktuellen Status des Spiels und weiß daher nicht, an welchem Knoten er sich befindet um eine Entscheidung zu treffen. In einem Spielbaum modelliert man diese Situation durch *Informationsbezirke*, die eine Menge von Knoten des Spielbaums definieren, zwischen denen ein Spieler nicht unterscheiden kann. Ein Informationsbezirk ist dabei durch die Tatsache gekennzeichnet, dass er immer nur Knoten genau eines Spielers enthält und alle Knoten eine identische Kantenmenge haben, ein Spieler also dieselben Handlungsalternativen bei allen Entscheidungen eines Informationsbezirks hat. Für den Spieler selbst stellt ein Informationsbezirk logisch auch nur eine einzige Entscheidung dar. Er kann zwischen den verschiedenen Spielstati nicht differenzieren und trifft daher genau eine Entscheidung die von allen Knoten im Bezirk weiterverzweigt. Grafisch symbolisiert man Informationsbezirke in einem Spielbaum durch eine umschließende Linie[9].

Beispiel 2 *Wir definieren ein einfaches Pokerspiel für zwei Spieler S_1 und S_2. Der Grundeinsatz für jeden Spieler beträgt dabei eine Geldeinheit. Gespielt wird mit drei Karten, nämlich einem Buben(B), einer Dame(D) und einem König(K). Jeder der Spieler erhält eine Karte, die er seinem Gegner nicht zeigt. Nun entscheidet Spieler 1, ob er den Einsatz auf 2 Geldeinheiten erhöhen möchte. Wenn sich Spieler 1 für eine Erhöhung des Einsatzes entscheidet, hat Spieler 2 die Möglichkeit, entweder mitzugehen oder auszusteigen. Wenn er aussteigt, verliert er das Spiel und Spieler 1 gewinnt seinen Einsatz. Geht er jedoch mit, kommt es zur Aufdeckung und die Spieler legen ihre Karten offen auf den Tisch. Der Spieler mit der höheren Karte gewinnt. Wenn Spieler 1 nicht erhöht hat, kommt es direkt zur Aufdeckung.*

Beispiel 2 beschreibt eine einfache Variante des Pokerspiels[10]. In Abbildung 2 wird genau dieses Spiel durch einen Spielbaum dargestellt. Man erkennt, dass für Spieler 2 drei Informationsbezirke definiert sind (symbolisiert durch die grauen Kasten), an denen jeweils zwei Entscheidungsknoten zusammengefasst werden. Die Informationsbezirke entstehen dadurch, dass Spieler 2 nicht weiß, welche Karte Spieler 1 durch den Zufall im ersten Spielschritt zugeteilt worden ist. Er kann einzig die Karte, die er selber auf der Hand hält, als Ansatzpunkt nehmen, welche Karte Spieler 1 nicht haben kann.

Bemerkenswert ist, dass es für dasselbe Spiel mehrere strukturgleiche Spielbäume geben kann. Der Spielbaum in Abbildung 2 induziert eigentlich, dass Spieler 2 seine Karte erst bekommt, nachdem Spieler 1 bereits seine Entscheidung getroffen hat. Dies ist natürlich nicht der Fall. Dennoch ist der Spielbaum äquivalent zu dem des beschriebenen Spiels, da Spieler 1 über die Information, welche Karte Spieler 2 auf der Hand hat, kein Wissen erlangt. Durch den Trick, die Kartenvergabe für Spieler 2 erst als dritten Spielzug zu modellieren, spart man die Definition der Informationsbezirke für Spieler 1. Diese wären eigentlich notwendig, da ja auch Spieler 1 über unvollkommene Informationen verfügt.

9. Vgl. zur extensiven Form z.B. (Koller & Pfeffer, 1997, S. 6f).
10. Die vereinfachte Pokervariante basiert auf (Kuhn, 1950). Zu Darstellungszwecken wurde sie für diese Arbeit allerdings weiter vereinfacht.

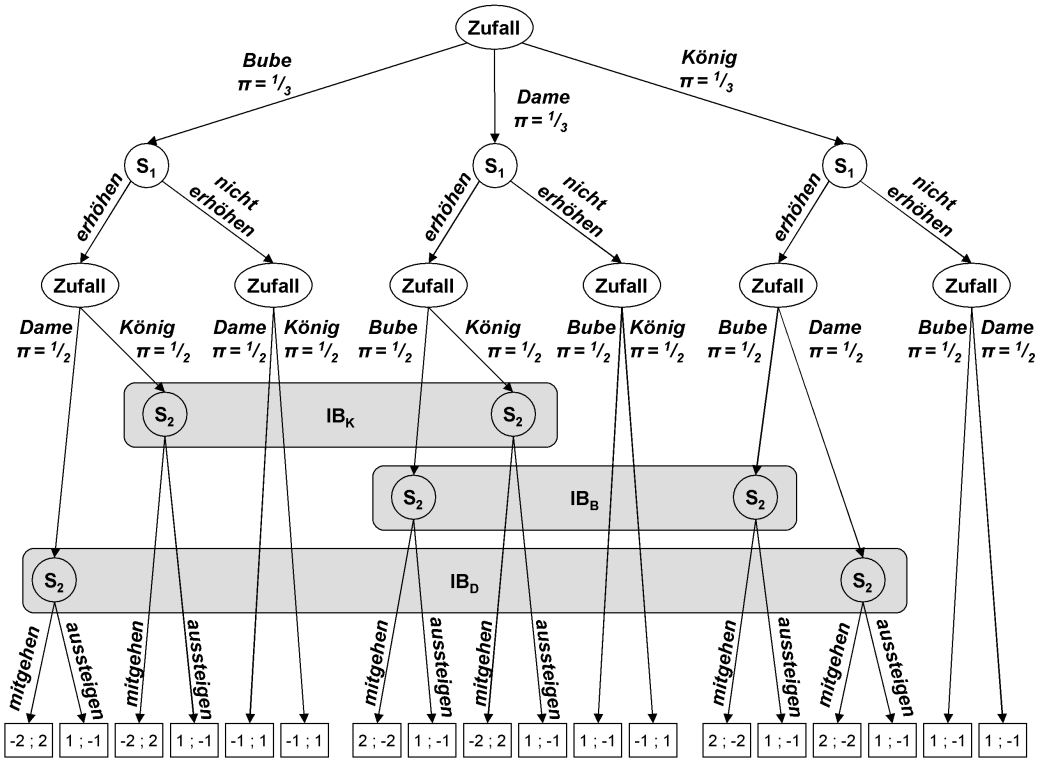

Abbildung 2: Vollständiger Spielbaum eines stark vereinfachten Pokerspiels

2.2 Klassische Lösungsverfahren

Für die Lösung eines unvollkommenen dynamischen Spiels mit Zufallsereignissen gibt es mehrere Lösungsmöglichkeiten[11]. Zunächst wird in Abschnitt 2.2.1 ein einfaches Verfahren vorgestellt, dass darauf basiert, Unsicherheiten aufgrund unvollkommener Informationen aus dem Spiel zu entfernen. Dadurch ist es möglich, ein baumbasiertes Lösungsverfahren für vollkommene Spiele anzuwenden, daß in sehr kurzer Zeit zu einer Lösung führt. Obwohl dieses Verfahren im allgemeinen nicht zu einer Lösung für alle unvollkommenen Spiele führt, kann es dennoch eingesetzt werden, um den zu durchsuchenden Lösungsraum von vorneherein einzuschränken. Ein deutlich aufwändigeres Verfahren, das dafür immer für unvollkommene Nullsummenspiele (und damit auch für Poker) eine exakte Lösung findet, wird im Abschnitt 2.2.2 vorgestellt.

2.2.1 Dominante Strategien und Rückwärtsinduktion

Ein Verfahren zur Ermittlung der Lösung eines vollkommenen dynamischen Spiels ist die *Rückwärtsinduktion*. Dabei analysiert man einen Spielbaum von den Blättern in Richtung Wurzel und antizipiert die Entscheidungen jedes Spielers. Man interpretiert dazu jeden Subbaum des Spielbaums als Teilspiel, in dem man separat nach einer Lösung sucht. Jedes Teilspiel, dass nur eine einzige Entscheidung enthält, ist direkt lösbar. Dies trifft also auf alle Knoten der untersten Ebene zu, wo auf die Entscheidung eines Spielers die Auszahlungen

11. Zur Definition einer Lösung siehe Abschnitt 1.3

folgen. Ein Spieler würde sich in so einer Situation aufgrund des Rationalitätsprinzip für genau die Handlungsalternative entscheiden, die ihm den größten Nutzen liefert.

Da der Ausgang eines Teilspiels auf der untersten Ebene also bekannt ist, kann man im Spielbaum das Teilspiel direkt durch die resultierenden Auszahlungen in Form eines Blattes ersetzen. Die Tiefe des Spielbaums verringert sich dadurch um eine Ebene und man kann dasselbe Vorgehen nun auf die darüberliegende Ebene anwenden. Am Ende des Verfahrens hat man den Spielbaum auf ein einziges Blatt reduziert, der die finale Auszahlung angibt. Die Entscheidungen, die man durch das Ersetzungsverfahren antizipiert hat, bestimmen dabei eindeutig den Spielverlauf[12]. Jeder Spieler hat dadurch eine reine Strategie, die sich im Gleichgewicht mit den Strategien seiner Mitspieler befindet. Dieses Ergebnis wurde erstmalig durch Ernst Zermelo bewiesen, der damit die eindeutige Lösung eines Schachspiel beschrieben hat[13].

In Spielen mit unvollkommener Information ist dieses Verfahren vom Prinzip her nicht anwendbar, da man bei der Auswertung eines Knotens, der sich in einem Informationsbezirk befindet, alle weiteren Entscheidungssituationen innerhalb des Bezirks mitberücksichtigen muss. Unter Umständen ist es aber dennoch möglich, die Entscheidung eines Spielers exakt zu antizipieren. Dazu untersucht man alle möglichen Auszahlungen, zu denen eine bestimmte Entscheidung führen kann. Gibt es eine Entscheidung, die in jedem Ausgang eine niedrigere Auszahlung liefert, als eine alternative Handlung, spricht man von einer *strikt dominierten* Handlung. Mathematisch formuliert wird eine Handlungsalternative $a_i \in \mathcal{A}_p$ innerhalb eines Informationsbezirk B strikt dominiert, wenn gilt: $\exists a_j \in \mathcal{A}_p . u_p(a_i|w_b) < u_p(a_j|w_b) \forall w_b \in B$, wobei w_b die Pfade zu den einzelnen Knoten des Informationsbezirks darstellt. Gibt es eine solche strikt dominierte Handlung a_i, sagt man auch a_j dominiert a_i und streicht die Alternative a_i aus dem Spielbaum, da ein Spieler aufgrund des Rationalitätsprinzips nie eine Handlung wählen wird, die ihm eine niedrigere Auszahlung als eine alternative liefert[14].

Man betrachte nun den Informationsbezirk IB_B von Spieler 2 aus dem vereinfachten Pokerspiel in Abbildung 2, der die Spielknoten gruppiert, bei denen der Spieler einen Buben auf die Hand bekommen hat. Vergleicht man die durch die Strategie Mitgehen möglichen Auszahlungen mit denen der Strategie Aussteigen, fällt sofort ins Auge, dass die Strategie Austieg in jedem Fall eine höhere Auszahlung liefert, als man durch ein Mitgehen erreichen könnte. In beiden Fällen verliert Spieler 2 zwei GE, wenn er die Erhöhung mitmacht, aber nur eine GE, wenn er das Spiel vorzeitig beendet. Dies ist intuitiv klar, da Spieler 2 genau weiß, dass er die niedirgste Karte des Spiels auf der Hand hat, und bei der anschließenden Auszahlung gar nicht gewinnen kann[15].

Analog kann man mit dem Informationsbezirk IB_K verfahren, bei dem Spieler 2 einen König auf die Hand bekommt. In diesem Fall dominiert die Strategie Mitgehen. Nachdem man die beiden Handlungsalternativen aus dem Spielbaum eliminiert hat, können die Informationsbezirke aufgelöst werden, da die Entscheidung von Spieler 2 durch die letzte verbliebene Alternative feststeht. Der Informationsbezirk IB_D kann auf diese Weise jedoch nicht eliminiert werden, daher ist eine vollständige Lösung durch Rückwärtsinduktion wei-

12. Vgl. zu Rückwärtsinduktion z.B. (Helm, 2007, Vl. 6, S. 3ff).
13. Für einen Beweis vgl. (Zermelo, 1929).
14. Zu dominierten Strategien vgl. z.B. (Osborne, 2004, S. 120f).
15. Man beachte, dass dieser Fall nur eintritt, da die Aufdeckung unmittelbar bevorsteht. (Koller & Pfeffer, 1997, S. 5) zeigen, dass auch Bluffen eine optimale Spielstrategie darstellen kann.

terhin nicht möglich. Jedoch kann man die Entscheidung, die Spieler 1 treffen wird, wenn er eine Dame auf die Hand bekommt, nun bestimmen. Da die erwartete Auszahlung für die Strategie Erhöhen bei $-\frac{1}{2}$ liegt, die bei der Strategie Nicht-Erhöhen jedoch bei 0, wird Spieler 1 grundsätzlich den Einsatz nicht erhöhen.

Abbildung 3 zeigt den Spielbaum nach der Eliminiation der dominierten Strategien. Außerdem wurden in dem Spielbaum bereits alle Zufallszüge, die direkt auf Auszahlungen verzweigen, durch ihre Erwartungswerte ersetzt.

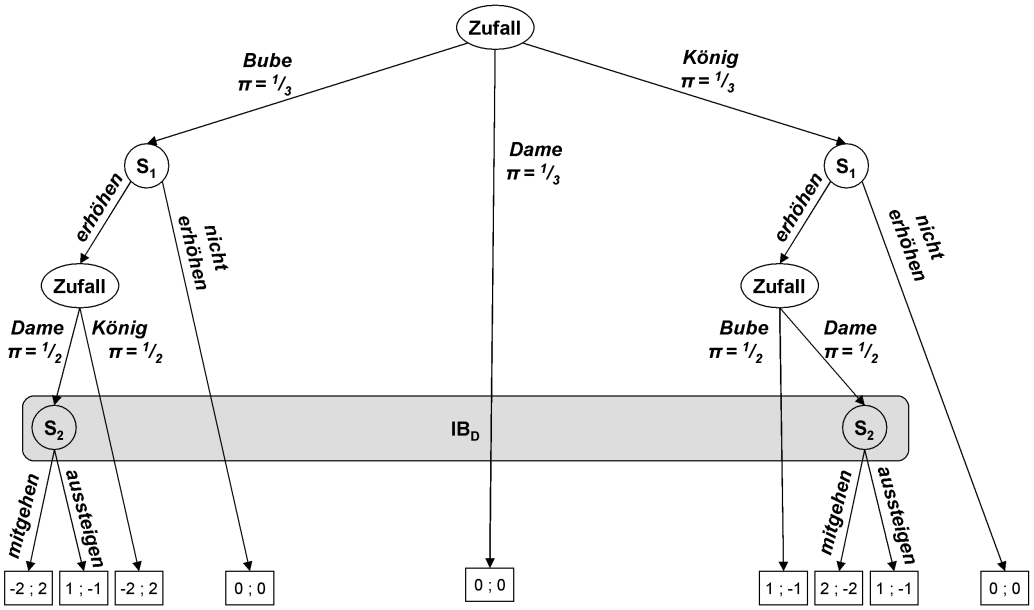

Abbildung 3: Spielbaum des vereinfachten Pokerspiels vollständig reduziert

2.2.2 Lösung durch den MaxiMin-Algorithmus

Ein Lösungsverfahren, das für unvollkommene dynamische Nullsummenspiele immer eine korrekte Lösung liefert, ist der MaxiMin-Algorithmus. Dabei versucht ein Spieler diejenige Strategie zu wählen, bei der er im schlechtesten Fall noch die höchste Auszahlung erhält. Um dies zu erreichen analysiert ein Spieler für alle eigenen Strategien, welche die minimale Auszahlung ist, die er mit dieser Strategie auf jeden Fall erreichen wird. Er wählt im Anschluss diejenige Strategie aus, für die er die höchste minimale Auszahlung erwartet, er *maximiert also seine minimale Auszahlung.*

Bei einem Nullsummenspiel, das ein strikt kompetitives Spiel ist, entspricht die Minimierung der Auszahlung des gegnerischen Spielers gleichzeitig der Maximierung der eigenen Auszahlung. Daraus folgt, dass jede Stragie, die den erwarteten Nutzen eines Spielers durch die minimal erwartete Auszahlung maximiert, auch gleichzeitig den Nutzen des Gegners bei Einhaltung der eigenen Strategie maximiert. Jede Strategie, die durch den MaxiMin-

Algorithmus in einem Nullsummenspiel ermittelt wird, stellt also eine Gleichgewichtsstrategie und damit eine Lösung des Spiels dar[16].

Betrachtet man nun auch gemischte Strategien[17] in einem Nullsummenspiel mit den Spielern p_1 und p_2, kann man das MaxiMin-Problem wie folgt mathematisch formulieren:

Definition 6 Lösung des MaxiMin-Algorithmus
Gesucht werden zwei gemischte Strategien s_1 und s_2 in einem Nullsummenspiel, die die Gleichgewichtsbedingung erfüllen. $\mathcal{A}_1$ und $\mathcal{A}_2$ seien dabei die Handlungsalternativen (reinen Strategien) der beiden Spieler, U_1 die Auszahlungsmatrix[18] für Spieler 1. Eine Lösung erfolgt durch das Optimierungsproblem:

$$\begin{aligned} \textit{Löse:} \quad & \max_{s_1} \min_{s_2} (s_1^T U_1 s_2) \\ \textit{unter den Bedingungen:} \quad & \sum_{i=1}^{|\mathcal{A}_1|} s_{1,i} = 1, \\ & \sum_{i=1}^{|\mathcal{A}_2|} s_{2,i} = 1, \\ & x \geq 0, y \geq 0. \end{aligned}$$

Bei dem in Definition 6 formulierten Optimierungsproblem handelt es sich um ein Problem der linearen Programmierung (LP), dass z.B. mit einem Simplex-Verfahren in polynomieller Zeit gelöst werden kann[19].

Damit man den MiniMax-Algorithmus auf ein Spiel in extensiver Form anwenden kann, muss man jedoch zunächst das Spiel in die Normalform umwandeln. Dafür trägt man die Kombination aller möglichen Handlungsoptionen, die ein Spieler im Spielverlauf hat, als seine Menge an reinen Strategien in der Normalformtabelle ab. Als Auszahlungen notiert man den jeweiligen Erwartungswert der Einzelauszahlungen, die durch die eine Zelle definierende Strategiekombination erreichbar sind. Diesen Erwartungswert erhält man also, wenn man alle durch eine Strategie erreichbaren Blätter das Baumes mit den Wahrscheinlichkeiten entlang des Pfades zu dem Blatt multipliziert[20].

Man betrachte dazu noch einmal das vereinfachte Pokerspiel aus Beispiel 2. Für die Umwandlung in die Normalform eignet sich der in Abschnitt 2.2.1 vereinfachte Spielbaum aus Abbildung 3 am besten. Zunächst werden alle Entscheidungenskombinationen gebildet, die die Spieler treffen können. Für Spieler 2 gibt es nur noch eine einzige Entscheidung zu treffen, für die er zwei mögliche Alternativen hat: Wenn er eine Dame auf der Hand hält, kann er entweder aussteigen oder mitgehen. Alle anderen möglichen Entscheidungen wurden bereits durch die Auswertung dominanter Strategien eliminiert. Wenn er einen König auf der Hand hält, wird er immer mitgehen, hat er hingegen einen Buben, wird er immer aussteigen. Spieler 1 hat hingegen noch zwei Entscheidungen zu treffen: Einmal, wie er sich mit einem Buben auf der Hand verhält und einmal, wie er bei einem König handelt. Er hat jeweils zwei Alternativen, also insgesamt vier (2×2) mögliche Strategien.

16. Für einen ausführlichen Beweis, dass eine MaxiMin-Strategie eine Gleichgewichtsstrategie darstellt, siehe (Osborne, 2004, S. 367f).
17. Zur Erläuterung gemischter Strategien siehe Abschnitt 1.3.
18. Die Auszahlungsmatrix entspicht der Normalformdarstellung eines Spiels, in der nur die Auszahlungen eines Spielers enthalten sind. Bei Nullsummenspielen gilt: $U_1 = -(U_1)^T$.
19. Zum Simplex-Algorithmus siehe z.B. (Domschke & Drexl, 2002, S. 20ff).
20. Für eine Beschreibung der Umwandlung von der extensiven Form in die Normalform vgl. (Koller & Pfeffer, 1997, S. 24f).

Die Berechnung der Erwartungswerte der Auszahlungsfunktion sei beispielhaft demonstriert an der Strategiekombination, an der Spieler 1 sowohl mit einem Buben, als auch mit einem König auf der Hand erhöht und Spieler 2 mit einer Dame auf der Hand mitgeht. Man betrachtet nun für alle erreichbaren Knoten, bzw. für alle möglichen Kartenkombinationen (BD, BK, D*, KB, KD) die Auszahlungen und gewichtet diese mit ihrer Wahrscheinlichkeit:

$$E(u_1(B_e, K_e; D_m)) = \frac{1}{6} \cdot -2 + \frac{1}{6} \cdot -2 + \frac{1}{3} \cdot 0 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 = -\frac{1}{6}$$

Tabelle 3 zeigt die Normalform des Spiels, auf der man nun direkt den MaxiMin-Algorithmus anwenden kann(angegeben sind nur die Auszahlungen für Spieler 1). Den eben berechnete Erwartungswert findet man links oben in der ersten Zeile und ersten Spalte wieder. Bemerkenswert ist, dass durch die Vereinfachungen des Spielbaumes in Abschnitt 2.2.1 erhebliche Einsparungen beim Bilden der Normalform erzielt worden sind. Für den nicht vereinfachten Spielbaum hätte man 36 (6×6) mögliche Strategiekombinationen untersuchen müssen, also die vierfache Menge.

Die Möglichkeiten der Reduktion sollten wenn möglich immer genutzt werden, da die Größe der Normalformmatrix exponentiell mit der Größe des Spiels (Runden × Handlungsalternativen) ansteigt. Für das vereinfachte Pokerspiel von Kuhn, das noch eine weitere Setzrunde zur Erhöhung durch Spieler 2 enthält, besteht die nicht reduzierte Matrix bereits aus 1728 Einträgen (27×64)[21].

		Spieler 2	
		D→mitgehen	D→aussteigen
Spieler 1	B→erhöhen; K→erhöhen	$-\frac{1}{6}$	$\frac{1}{6}$
	B→erhöhen; K→lassen	$-\frac{2}{3}$	$-\frac{1}{6}$
	B→lassen; K→erhöhen	$\frac{1}{2}$	$\frac{1}{3}$
	B→lassen; K→lassen	0	0

Tabelle 3: Normalform des reduzierten Spielbaums für Beispiel 2

Als Lösung für das modellierte vereinfachte Pokerspiel erhält man nach der Berechnung, dass Spieler 1 bei einem Buben nie und bei einem König immer erhöhen wird. Spieler 2 wird mit einer Dame auf der Hand immer aussteigen. Es handelt sich also um ein Gleichgewicht mit reinen Strategien. Diese Lösung erhält man auch direkt, wenn man die Normalform-Tabelle des Spiels untersucht. Es fällt ins Auge, das die dritte Strategie von Spieler 1 alle anderen Strategien dominiert, da sie für jede mögliche Strategie des Gegners immer die höchste Auszahlung liefert. Die erwarteten Auszahlungen für die Spieler liegen im Gleichgewicht also bei $\left(\frac{1}{3}; -\frac{1}{3}\right)$.

3. Verbesserte Lösungsstrategien

Nachdem wir im vorherigen Kapitel allgemeinere Konzepte einer spieltheoretischen Berechnung beschrieben haben, gehen wir nun auf weiterentwickelte aber auch speziellere Konzepte ein.

21. Vgl. (Kuhn, 1950).

3.1 Optimierung

Berechnungen für Spiele sind meistens sehr komplex. Der Rechenaufwand für reale Spiele ist enorm und oft gar nicht nur mit den bisher vorgestellten Methoden zu lösen. Es gibt viele Möglichkeiten die Berechnung an sich zu optimieren während das Ergebnis weiterhin eine exakte Lösung bleibt. Zwei weitreichende Methoden werden hier vorgestellt.

D. Koller und A. Pfeffer[22] haben bis 1997 die Sprache „Gala“ entwickelt um realistische und große Spiele unter einer spieltheoretischen Betrachtung analysieren zu können. In diesem Kapitel werden wir es häufiger für Beispiele verwenden.

3.1.1 Kompakte regelbasierte Darstellung

Wird einem Menschen ein Spiel beigebracht, so werden ihm in den meisten Fällen die Regeln erklärt. Bei Computern ist das häufig anders, hier werden Spielmodelle oft anders dargestellt, etwa mit Spielbäumen. Bei der Regelbasierten Darstellung wird ein System geschaffen, das normale Spielregeln versteht. Diese können oft schneller in linear zu lösende Probleme umgewandelt werden.

Wird das Spiel verändert, dann gewöhnlich durch neue oder veränderte Spielregeln. Diese lassen sich in vielen Fällen schneller in einer regelbasierten Darstellung umsetzen, weil der Entwickler keine ganzen Daten- und Baumkonstrukte per Hand abändern muss.

Man kann die Regeln in zwei Kategorien einteilen. In der ersten werden alle Objekte des Spieles wie etwa die Karten, die Spieler deklariert. In der zweiten Kategorie werden Handlungsabfolgen der Spieler und des Spielbetreibers (bei Poker der Dealer) beschrieben. Dies kann beispielsweise die Kartenausgabe und das Setzen eines Betrages sein.

Konkret wurde in unserer Beispiel-Programmiersprache Gala wie auch in vielen anderen Programmiersprachen eine Datenflusskontrolle *(flow control)* implementiert. Die drei wichtigsten Elemente davon heißen *choose*, *reveal* und *outcome*. Choose definiert jeweils einen Entscheidungspunkt, reveal ändert den Informationszustand der Spieler und outcome bestimmt den tatsächlichen Gewinn oder Verlust jedes Spielers. Die Syntax für Entscheidungspunkte bei Gala für den choose Ausdruck ist *chosse(Player, Template, Constraint)*, wobei ein Anwendungsbeispiel *choose(peter, InitialBet, between(0, $money(peter), Bet))* wäre. Man kann gut erkennen, dass der InitalBet zwei Voraussetzungen erfüllen muss: zum Einen muss der Betrag positiv und maximal soviel, wie der Spieler peter zu Verfügung hat, sein. Zum Anderen unterliegt das erste Setzen den allgemeinen Regeln eines Setzens *(Bet)*. Die Erklärung weiterer Beispiele die in Gala genutzt werden würde hier zu weit führen.

Den Vorteil von dieser regelbasierten Darstellung ist, dass in dem choose Ausdruck schon die Möglichkeiten als Antwort enthalten sind. Das Gleiche gilt analog für reveal und outcome. Schließlich kann man also sagen, dass diese Darstellung angenehm für die Softwareentwicklung ist und Rechenaufwand spart.

3.1.2 Sequentieller Lösungsalgorithmus

Das exponentielle Wachstum, das mit der Normalform verbunden ist, macht Standardlösungsalgorithmen für viele Spiele unrealistisch. 1994 wurde daher der sequentielle Lösungsansatz

22. Der Artikel (Koller & Pfeffer, 1997) beschreibt Gala ausführlich.

von Koller, Megiddo und von Stengel (Koller, Megiddo, & von Stengel., 1994, S. 750-759) entwickelt.

Der Algorithmus, der vermeidet den Rechenaufwand exponentiell ansteigen zu lassen, basiert auf strategischen Variablen. Statt Wahrscheinlichkeiten von Einzelaktionen wie in der extensiven Form, oder Wahrscheinlichkeiten von deterministischen Strategien - repräsentieren einzelne Realisationswerte ganze Abfolgen bzw. Sequenzen von Aktionen. Eine solche Sequenz eines Spielers kann anschaulich in einem Spielbaum als Pfad von der Wurzel vertikal bis in ein Blatt des Baumes angesehen werden.

Sei k ein Spieler und p ein Knoten des Spielbaumes, so gibt es genau einen Weg von der Wurzel zu p . Auf diesem Pfad kann es einige Entscheidungspunkte die der Spieler k zu entscheiden hat und wird als $\delta^k(p)$ definiert. In manchen Fällen ist eine solche Sequenz eine leere Menge, in allen anderen Fällen gibt $\delta^k(p)$ an, wie sich der Spieler zu entscheiden hat um zu p zu gelangen.

Wir beschreiben nun eine zufällige Spielsstrategie, die aus einer Menge von deterministischen Strategien besteht mit μ_k . Natürlich hängt das Erreichen des Knotens p auch von den Entscheidungen der anderen Spieler ab, daher berechnen wir die Wahrscheinlichkeit mit μ_k δ_k spielen zu können. Dies ist dann der Realisationswert und wir bezeichnen ihn mit $\mu_k(\delta_k)$.

Definition 7 Realisationswert
Wird eine Sequenz δ_k nach μ_k gespielt, so bezeichnet $\mu_k(\delta_k)$ den Realisationswert dieser. Hierbei ist k der Spieler und der Realisationswert gibt die Wahrscheinlichkeit an, mit der er den entsprechenden Informationsbezirk von δ_k erreicht.

Der Realisationsplan ist die Menge aller möglichen Realisationswerte $\mu_k(\delta_k^1)$, ..., $\mu_k(\delta_k^m)$ wenn δ_k^1 ... δ_k^m alle möglichen Sequenzen sind.

Wir suchen nun unter den Realisationswerten einen, der unsere Optimierungsbedingungen, etwa minimax, erfüllt. Koller und Megiddo führen weiter aus, dass mit der Matrix E , einem Vektor e und dem positiven Vektor x durch Lineare Optimierung $Ex = e$ die optimal Aktion berechnet werden kann.

Der Vektor x repräsentiert dabei eine zufällige Strategie die den Bedingungen der Matrix E und des Vektors e entsprechen. Die Bedingungen sind dabei aufaddierte Wahrscheinlichkeiten von einzelnen Entscheidungen. $Ex = e$ läßt sich als lineares Problem lösen.

Abschließend kann dies mit der intuitiven Spielweise eines Menschen verglichen werden. Etwa wenn der Spieler in der ersten Biet-Runde passen möchte und in der Letzten Bieten. Dann überlegt er sich, wie es möglich ist, dass dieser Fall eintreten kann, ohne dass die Gegner etwa durch ein Wegwerfen oder Erhöhen bis zum All-in. Der Spieler wählt dann die erste Strategie, die ihm einfällt und seinen Ansprüchen entspricht.

3.1.3 Resümee GaLa-System

Das in den vorhergehenden Kapiteln behandelte System „Gala“ hat den Anspruch Spiele, die in der Welt auch echt gespielt werden, unter spieltheoretischen Gesichtspunkten berechenbar zu machen. Dieser Anspruch wird auch weitgehend erfüllt. Die Entwickler D. Koller und A. Pfeffer haben bis 1997 unter Ausnutzung verschiedener Optimierungen Spielstrategien von verschienden Spielen berechnet. Gala baut auf der Logik Programmiersprache auf, dies wurde vorallem für die Regelbasierte Darstellung sehr passend gewählt.

Leider wird Gala nicht weiterentwickelt und die Autoren geben auf der ehemaligen Projekt-Homepage[23] an, dass das System vermutlich nie ein komplettes Poker-Programm (mit allen Bietrunden) berechnen werden kann.

Zur Entscheidung zwischen Handlungsalternativen greift jedoch Gala auf das noch weiterhin exsitierende Programm Gambit zurück.[24]

3.2 Approximation

In diesem Abschnitt werden Annäherung an gewünschte Optimums behandelt. Wie schon beschrieben ist es nötig Wege zu finden, die enorme Komplexität von spieltheoretischen Aufgaben in berechenbare Probleme umwandeln. Approximation bedeutet das wir die in geringem Maße von den gesuchten Werten abweichen, diese aber dafür in akzeptabler Zeit finden können.

Die Autoren Peter B. Miltersen und Troels B. Sorensen haben im Artikel „*A Near-Optimal Strategy for a Heads-Up No-Limit Texas Hold'em Poker Tournament*“ (Miltersen & Sorensen, 2007) eine fast optimale Strategie für eine Poker Variante berechnet. Im Folgenden werden wir anhand dieser Variante drei Approximations Möglichkeiten erläutern.

Das realistische Beispiel gilt für *No-Limit Texas Hold'em Poker* mit zwei Spielern und wird auch bei Partypoker.com angeboten. Der Small Blind ist fest 300, der Big Blind 600. Insgesamt ist die Summe beider Spieler 8000, der jeweilige Stack muss jedoch nicht zu Beginn gleich groß sein. Wir betrachten den *Payoff* als 1 oder 0, je nachdem ob der betrachtete Spieler gewinnt oder verliert.

3.2.1 Grundsatz

Die zugrunde liegende Approximation besteht daraus, dass wir die Berechnung vereinfachen. Das Spiel an sich soll weiterhin die reale Spielsituation darstellen, wir schränken jedoch die Möglichkeiten unseres Spielers ein. Gegner können weiterhin nach den normalen Turnierregeln gegen diesen Spieler agieren.

Indem wir die Alternativenmenge verkleinern, verkleinert sich gleichzeitig die zu berechnende Aufgabe, ebenso der Spielbaum.

In unserem Turnier-Beispiel, auf das wir noch eingehen werden, wird dem Spieler nur noch erlaubt entweder die Hand weg zu werfen oder all-in zu gehen. Wir nennen diese Strategie wie auch im Artikel von Miltersen und Sorensen „*jam/fold*“-Strategie.

3.2.2 Knotengruppierung

Durch die eben angekündigte Regeleinschränkung muss nur noch die Pre-Flop Situation ausgewertet werden. Da der Spieler entweder die Hand weg wirft und damit die Runde beendet oder all-in geht und keiner weiteren Aktionsmöglichkeiten mehr in dieser Runde hat.

Dadurch müssen nur noch die Handkarten berechnet werden. Die Wertigkeit der Zahlen sowohl und ob sie von der gleichen Farbe sind, spielt eine Rolle. Dadurch erhält man 169 verschiedene Möglichkeiten. Da der eigentliche Spielbaum nicht diese Einschränkung hat,

23. Die Projekt-Homepage befindet sich unter: http://robotics.stanford.edu/ koller/gala.html
24. Eine freie Sammlung von Bibliotheken für Anwedungen der Spieltheorie http://gambit.sourceforge.net/

sondern nur der Spieler, haben wir alle möglichen Spielstrategien zu 169 so genannten Informationsbezirken zusammengefasst.

Eine zweite Approximation um die Berechnung noch mehr zu vereinfachen entsteht indem wir die Aktionen des Gegners nur in Kategorien einteilen. Der Gegner hat an sich die Möglichkeiten die Hand weg zu werfen, mit zugehen und um einen beliebigen Betrag zu erhöhen.

Da jeder unterschiedliche Betrag einen anderen Zustand generiert, und das für jeden einzelnen Informationsbezirk, ergeben sich über hunderttausend weitere Zustände. *In unserem Beispiel* wird die Aktion des Gegners daher nur in die Kategorie Fold und Check/Raise geteilt. Im Folgenden wird illustriert warum wir dadurch zwar die Knotenmenge stark verkleinern aber nicht schwächer gegen gute Spieler spielen:

Haben wir eine fast-optimale Strategie für den Spieler gefunden der auf das Mitgehen des Gegners reagiert, so befinden wir uns im Nash-Gleichgewicht. Diese Strategie liefert bei unserem jam/fold-Beispiel als Ergebnis, das der Spieler die Hand wegwerfen oder ihr all-in gehen soll.

Der Gegner muss nun ebenfalls mitgehen oder die Hand wegwerfen. Im ersten Fall macht es nun keinen Unterschied für unsere Auszahlung (Gewinn oder Verlust) mehr um wieviel der Gegner schon vor dem all-in erhöht hat. Im zweiten Fall wirft der Gegner die Karten weg – er verliert seinen Einsatz. Ein optimaler Gegner würde daher gegen unseren Spieler nur mitgehen aber nie zuerst erhöhen. Ein suboptimaler Gegner verliert einfach einen höheren Betrag gegen uns. Mit dieser Approximation wird unser Spieler nicht schlechter gegen (spieltheoretisch) optimale Gegner. Gegen suboptimale Gegner ist er überlegen aber reizt seine Möglichkeiten nicht ganz aus.

Diese Einschränkung führt dazu, dass der binäre Spielbaum, wenn der Spieler an der Position mit dem Big Blind sitzt, nur eine Tiefe von 3 besitzt, beim Small Blind eine Tiefe von 4.

Die letzte Approximation, die wir in diesem Artikel behandeln, fasst ebenfalls Knoten in Gruppen zusammen. Hierbei wird die Tatsache, dass die Spielstrategie von der Größe des Stacks eines Spielers abhängt betrachtet und Intervalle gebildet.

Als Grundvorraussetzung ist es wichtig zu verstehen, dass bei einem Turnier die Größe des stacks eine große Auswirkung auf die optimale Spielstrategie hat. Es erscheint intuitiv, dass es für einen Spieler mit der überwiegenden Mehrheit an Chips gut ist, etwas mehr zu riskieren um das Spiel zu beenden.

Die Spielstrategien ändern sich also auch bei gleichen Händen aber unterschiedlicher Chip Verteilung. Der Spielbaum hat daher für jede Hand und jeden stack eine optimale Strategie. Wir nähern uns an die optimalen Strategien an, indem wir den stack eines Spielers in Intervalle einteilen. Je mehr Intervalle es gibt, um so geringer ist die Abweichung der berechneten Strategie von der Optimalen.

In unserem Beispiel liegt die Summe der Chips, die beide Spieler zusammen haben, 8000, wir reduzieren den Berechnungsaufwand, indem wir Intervalle von 50 bilden. Daraus ergeben sich noch 158 nicht triviale Fälle. Zu jedem Fall wird eine Tabelle mit allen verschiedenen Starthänden gebildet, jeweils wird angegeben, ob der Spieler die Hand wegwerfen, all-in setzen oder eine zufällige Aktion durchführen soll.

Für die meisten Hände ist festgelegt, in welcher Situation sie gespielt werden. Wenige werden jedoch nur zu einem bestimmten Prozentsatz gespielt, als Beispiel dient die Starthand [sechs, acht] von verschiedenen Farben.

Die Autoren Koller und Pfeffer stellen dazu folgendes Theorem auf: Bei einer optimalen jam/fold Spielweise, definiert durch minimax, muss nur genau mit [6,8] verschiedener Farben mixed gespielt werden, wenn beide Spieler einen stack von 4000 haben. Dies läßt sich einfach beweisen, indem man die Strategie gegen die beiden Teilstrategien jam/fold als deterministische Strategie spielen lässt.

Eine Auffälligkeit dabei ist, dass ein Spieler der an der Position des Small Blind mit einer stack-Größe von 1800 die Starthand [3,4] von der gleichen Farbe wegwirft, [Bube, 2] jedoch spielt. Ist der stack größer und zwar bei 3600 ist dies genau umgekehrt.

Die rationale Erklärung ist leicht nach zu vollziehen, ein Gegner mit einer schlechten Hand (Trash-hand) würde bei einem größeren stack eher bei all-in mitgehen um das Spiel für sich zu entscheiden. Gegen eine Trash-hand ist besser mit dem Jack auf eine Highcard hoffen, als auf seltenere Strasse oder Flush.

3.2.3 Ober- und Unterschranken

Um eine Gewinnwahrscheinlichkeit zu berechnen - gehen wir wie folgt vor: Everett (Everett, 1957) hat in gezeigt, dass im rekursiven Spiel für alle Spielelemente „critical values“ berechnet werden können. Diese Werte sind analog zu den Werten einer minimax Berechnung von garantiert terminierenden Spielen und geben die Wahrscheinlichkeit an, zu der ein Spieler in seiner aktuellen Position mit einer optimalen Spielweise das gesamte Spiel gewinnt.

Häufig können jedoch keine optimalen Spielweisen berechnet werden, sondern nur Approximationen wie die weiter oben beschriebenen. Um nun die Gewinnwahrscheinlichkeit eines Spielers zu berechnen der nur eine fast-optimale Strategie spielt werden Ober- und Untergrenzen (*upper* bzw. *lower bound*) eingeführt. Die Obergrenze gibt an, um bis zu wie viel Prozent die eingesetzte Strategie verbessert werden kann, bis sie die Optimale Strategie wäre.

Damit haben wir das entscheidende Werkzeug an der Hand um zu vergleichen, wie gut verschiedene Spielweisen sind. Je kleiner die Prozentzahl der Abweichung ist, um so besser ist eine Strategie.

Um zurück zu Poker zu kommen: Ein Turnier mit festen Blinds findet unter Umständen kein Ende. Etwa wenn beide Spieler fortgehend ihre Hände wegwerfen. Daher sind die „critical values“ für uns von Bedeutung. Je nachdem wie groß der stack eines Spielers ist, um so größer ist die Wahrscheinlichkeit, dass er das Turnier gewinnen kann. Die „critical values“ geben diese Werte in Prozent an, wenn ein Spieler optimal spielen würde. Aber da es in diesem Artikel ja genau darum geht, dass die optimale Strategie nicht berechnet werden kann sind für Ober- und Untergrenzen für uns interessant.

3.2.4 Genauigkeitsverlust

Durch die Approximationen werden einige Probleme erst in angemessener Zeit berechenbar, dafür nähern sich die gefundenen Ergebnisse lediglich den exakt gesuchten Werten an.

Bei Poker sind Strategien mit *Starting Hand Charts* weit verbreitet. David Sklansky hat ein einfaches System „*The System*“ entwickelt, das ähnlich zu unserem Beispiel vorgibt,

ob ein Spieler nichts oder all-in setzen sollte. Es werden jedoch nur die Starthände und Erhöhungen der Gegner betrachtet.

Sklansky meinte 2003 „I am extremly curious as to how strong it [the system] really might be. And if it is strong, how much stronger yet a more complex, move all-in system would be?“ (Sklansky, 2003, S. 14)

Eine verbesserte Version des Systems, nammte er dann *revised system*, dieses hat zusätzlich noch die Größe des Blinds und die Anzahl der Gegner in der Kalkulation. Berechnet man hierfür den Genauigkeitsverlust nach wie im vorherigen Kapitel beschrieben kommt man immer noch auf einen Wert von 5,9%.

Das ist deutlich deutlich schlechter als jam/fold aus unseren Beispielen mit 1,4% und zieht gerade einmal mit der Strategie bei jedem Zug all-in zu setzen bei unserem Turnier gleich.

Eine noch geringere Abweichung vom Optimum konnte mit dem System von Giplin und Sandholm (Gilpin & Sandhold, 2004, S. 160-169) erreicht werden. Hier konnte man die maximale Abweichung in Voraus angeben, der Rechenaufwand stieg dementsprechend bei sehr kleinen Werten. Außerdem waren die fast optimalen Strategien mittels Gleichgewichten nur für die einfache Rhode Island Poker Variante und auch nur für die ersten drei Biet-Runden berechenbar.

Daher stellen die Autoren Koller und Pfeffer die Frage, ob es nicht sinnvoller ist Systeme wie in unseren Beispielen zu entwickeln. Die Abweichungen sind zwar höher, etwa bei jam/fold 1,4%, aber taugen wenigstens für eine echtes und populär Spiel.

3.2.5 Resümee Jam/Fold

Das System von Koller und Avi nutzt viele Möglichkeiten aus, durch Approximation ein Partypoker.com Turnier nahezu optimal spielen zu können. Erstaunlich erscheint, das man selbst wenn man nur in der Pre-Flop Situation agiert nur mit maximal 1,4% von der Gewinnwahrscheinlichkeit der optimalen Strategien abweicht, die diese Einschränkung nicht haben. Das Verhältnis zwischen den Blinds und der Gesamtsumme an Chips macht die jedoch wieder greifbarer: alle Chips, also 8000, ergeben weniger als 14 Big Blinds.

Ein ähnlich optimales System für ein No-Limit Cash-Game ist wiederum um ein vielfaches komplexer und daher zur Zeit nicht berechenbar. Das Gleiche gilt für Limit-Poker, welches eine noch höhere Berechnungskomplexität hat.

4. Umsetzung bei der Poker Challenge

Die Variante Limit Texas Hold'em Tunier mit bis zu 6000 Händen werden wir nicht im spieltheoretischen Sinne optimal lösen können. Das Spiel ist zu komplex für eine vollständige Berechnung selbst mit allen möglichen Optimierungen werden wir noch zusätzlich auf Approximationen angewiesen sein.

Die bisher behandelten Beispiele und Poker Varianten weisen alle entscheidende Unterschiede zu der Poker Variante bei unserem Poker Challenge auf, in diesem Kapitel werden jedoch Lösungen und Hilfen abstrahiert und auf das AAAI Modell abgebildet.

4.1 Knoten des AAAI-Modells

Bei dem AAAI Multi-Table Limit-Turnier gibt es eine viele Faktoren welche die Knotenanzahl eines Poker Spielbaumes in die Höhe treiben.

Dazu gehören die Hand- sowie Tischkarten, die Anzahl der Spieler und sämtliche Handlungsmöglichkeiten, die Verteilung der Chips, und wie viele Hände noch gespielt werden.

Würde man nicht optimieren und approximieren so gäbe es für die verschiedenen Kartenkombination des Spielers mit Hand und Tischkarten alleine $169! - 162! \approx 3,4 \cdot 10^{15}$ Knoten. Die anderen genannten Faktoren würden ohne Optimierungen diese Zahl in eine Höhe katapultieren, jenseits der $3,4 \cdot 10^{26}$ Knoten. In heutigen Zeiten nicht berechenbar.

Im Gegensatz dazu haben Darse Billings et al. (Billings, Burch, A., Holte, Schaeffer, Schauenberg, & Szafron, 2003) gefolgert, dass man zur optimierten Berechnung von Texas Hold'em Limit Poker einen Spielbaum mit $\sim 10^{18}$ Knoten braucht um es ganz analysieren zu können. Billings ist unter anderem Entwickler der derzeit führenden Poker-Bot Software Polaris die verschiedene Strategien der aktuellen AAAI Poker Challenge Gewinnern implementiert.

4.2 Operationalisierungsansätze

Wir haben viele Optimierungen und Approximationen besprochen die auch für unsere Poker Challenge anwendbar sind. Der Schwerpunkt liegt eindeutig bei der Knotenmengenreduzierung. Aber auch bevor wir die Knoten festlegen und wenn wir sie auswerten gibt es gut Verbesserungsmöglichkeiten.

4.2.1 Sequentielle Form

Die besprochene sequentielle Form hilft uns, dass exponentielle Wachstum der Normal Form zu umgehen. Der Ansatz ist recht allgemein und kann vollständig auf unser Poker Turnier angewendet werden. Hiermit ist es möglich die Berechnung einfach in Lineare Gleichungssysteme umzuwandeln, was sicher das Ziel von den meisten unserer Poker-Bots sein wird.

4.2.2 Regelbasierte Darstellung

Daneben ist die regelbasierte Darstellung sehr interessant. Auch sie ist so weit offen gestaltet, dass sich die meisten Spiele mit ihr Modellieren lassen. Hat man ein Framework wie Gala, das die Regeln versteht und interpretieren kann, ist die Umsetzung unseres Limit Texas Hold'em Turnieres ein einfaches. Wählt man diese Darstellung für die eigene Software wird man vermutlich auf schon vorhandene Framework zurückgreifen. Ansonsten ist eine Implementierung in Java sinnvoll, die Teilaspekte davon umsetzt. Die Spielelemente lassen sich wegen Javas Objektorientierung gut als Klassen erstellen. Auch der Einsatz von choose, reveal und payout sollten abgewogen werden.

4.2.3 Evaluation der Strategien

Neben diesen Verbesserungsmöglichkeiten haben wir in diesem Artikel auch die Evaluation von Spielstrategien behandelt. Mit den angegebenen Schranken, der Abweichung vom Optimum, kann der Genauigkeitsverlust berechnet werden. So erhält man einen Überblick, wie gut oder schlecht die eigenen Strategien sind. Der Vergleich mit anderen Poker-Bots unserer

Gruppe oder des AAAI Wettbewerbs lässt sich hier auch einfach mit einer signifikanten Zahl anstellen.

4.2.4 Zusammenfassung von Knoten

Die Verdichtung der Information ist anschaulich und sehr effektiv. Die Spiele und Poker Varianten in den beschrieben Beispielen waren zwar alle sehr unterschiedlich zu unserem Multi-Table Limit Texas Hold'em Poker Turnier, dennoch lassen sich viele Aspekte auf unsere Challenge übertragen.

Die Ausnutzung von Symmetrien erscheint fast schon selbst verständlich. Die Farben haben beispielsweise bei Poker keine unterschiedlichen Bedeutungen und es gibt auch keine Rangfolge. Etwa bei der jam/fold Variante wurde wurden die Starthände in Informationsbezirke geteilt. Die Information ob die zwei Karten von der gleichen Farbe sind und ihre Wertigkeit reichten aus. Dies gilt auch für unsere Spielvariante für den Pre-Flop.

Die Einteilung von Aktionen in Kategorien sollten wir ebenso berücksichtigen. Für uns ist es möglich häufig wiederholte reraises von Gegnern in Bereiche einzuteilen.

Genauso können wir auch bei der Betrachtung der Größe der gegnerischen Stacks Intervalle bilden, wie es am Beispiel beschrieben wurde. Da neben gibt es natürlich noch eine Reihe andere Möglichkeiten Knoten zusammenzufassen.

Allgemein läßt sich sagen, dass wir zur Berechnung von komplexen Spielen wie unserem Turnier eine gute Kontenverdichtung durch Optimierung und Approximation genauso, wie ein effizienter Berechnungsalgorithmus brauchen.

References

Billings, D., Burch, N., A., D., Holte, R., Schaeffer, J., Schauenberg, T., & Szafron, D. (2003). Approximating game theoretic optimal strategies for full-scale poker. In *18th International Joint Conference on AI*, Vol. 18.

Domschke, W., & Drexl, A. (2002). *Einführung in Operations Research* (5. edition). Springer (Berlin Heidelberg New York).

Everett, H. (1957). Recursive games. In *Contributions to the Theory of Games Vol. III*, Vol. 39.

Gilpin, A., & Sandhold, T. (2004). Finding equilibria in large sequential games of incomplete information. *Electronic Commerce*, *94*, 160–169.

Helm, C. (2007). Skript zur vorlesung: Angewandte spieltheorie..

Koller, D., Megiddo, N., & von Stengel., B. (1994). Fast algorithms for finding randomized strategies in game trees. In *26th Annual ACM Symposium on the Theory of Computing*, Vol. 26.

Koller, D., & Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *Artificial Intelligence*, *94*, 167–215.

Kuhn, H. W. (1950). A simplified two-person poker. In Kuhn, H. W., & Tucker, A. W. (Eds.), *Contributions to the Theory of Games I.* Princeton University Press (Princeton).

Miltersen, P. B., & Sorensen, T. B. (2007). A near-optimal strategy for a heads-up no-limit texas hold'em poker tournament. In *6th international joint conference on Autonomous agents and multiagent systems*, Vol. 6.

Nash, J. F. J. (1951). Non-cooperative games. *Annals of Mathematics*, *54*, 286–295.

Osborne, M. J. (2004). *An introduction to Game Theory*. Oxford University Press (New York).

Sklansky, D. (2003). The system. *Card Plaxer Magazine*, *94*.

von Neumann, J., & Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press (Princeton).

Zermelo, E. F. F. (1929). Die berechnung der turnier-ergebnisse als ein maximumproblem der wahrscheinlichkeitsrechnung. *Mathematische Zeitschrift*, *29*, 436–460.

Fiktives Spiel und Verlustminimierung zur Berechung optimaler Lösungen in der Pokervariante Texas Hold'em

Alexander Marinc alexander.marinc@gmx.de

Abstract

Diese Ausarbeitung gibt zentral zwei verschiedene Ausarbeitungen wieder, die versuchen möglichst genaue Nash Gleichgewichte in einem Poker Spiel (Texas Hold'em) zu berechnen. Der erste Ansatz trägt den Titel "Using Fictitious Play to Find Pseudo-Optimal Solutions for Full-Scale Poker" (Duziak, 2006). Die Autoren versuchen pseudooptimale Lösungen durch eine geschickte Art der Abstraktion des Spieles und anschließendem "Training" allgemeiner Konvertierungsmatrizen zu finden. Der zweite der beiden Ansätze mit dem Titel "Regret Minimization in Games with Incomplete Information" (M. Zinkevich & Piccione, 2007) beschreibt einen auf Verlustminimierung basierenden Gedanken, welcher von den Autoren als "Kontrafaktischer Verlust" bezeichnet wird. Es werden die Grundlagen beider Ausarbeitungen dargestellt und ihre Verwendung im Rahmen des Spiels Poker beschrieben.

1. Einleitung

Poker ist ein Spiel, welches in der letzten Zeit immer mehr Beachtung gefunden hat. Parallen zu "realen" Konkurenzsituationen, wie zum Beispiel in der Wirtschaft zwischen zwei Firmen, und eine dennoch annähernd berechenbare Komplexität von 10^{18} Spielzuständen, machen das Spiel auch für theoretische Betrachtungen im Bereich der künstlichen Intelligenz interessant. Zwei Ansätze spieltheoretische Optima für das Spiel Poker zu berechnen werden im folgenden dargestellt. Der Ansatz von (Duziak, 2006) reduziert die Komplexität des Spiels auf 10^7 und liefert laut den Autoren dennoch gute Ergebnisse auf dieser "pseudooptimalen Lösung". Der im Original als "Fictitious Play" bezeichnet englische Ausdruck wird im Rahmen dieser Arbeit als "Fiktives Spiel" übersetzt. Allgemein läßt sich feststellen, dass die Gewinnchancen eines Algorithmus verbessert werden, umso weniger er das originale Spiel abstrahiert und umso höher die berechenbare Komplexität wird. Der zweite in dieser Arbeit beschriebene Ansatz von (M. Zinkevich & Piccione, 2007), macht in seiner Gesamtheit einen wesentlich komplexeren Eindruck und verwendet eine viel formalere und mathematischere Ausdrucksweise. Entsprechend den Angaben im Dokument ist mit den hier verwendeten Techniken zur Verlustminimierung (im englischen als "Regret Minimization" bezeichnet) und Abstaktion lediglich eine Reduktion auf 10^{12} Zustände nötig. Zu beiden Ansätzen liegen Ergebnisse im Vergleich zu bekannten Pokeralgorithmen vor, jedoch leider kein Vergleich untereinander. Der Vergleich zwischen beiden Algorithmen wird sich demnach im Wesendlichen auf Inhaltliche Aspekte beziehen. Der Fokus liegt aber ohne hin auf der Beschreibung der beiden Algorithmen. Hierzu gehört die Erörterung der verwendeten Begriffe im Zusammenhang des jeweiligen Ansatzes, sowie die theoretischen Hintergründe (sowei angegeben) und zuletzt die konkrete Umsetzung für das Spiel Poker. Allgemeine Kenntnisse der Regeln und Abläufe im Texas Hold'em Poker werden

vorausgesetzt, sind aber zum Beispiel auch in der Einleitung des Textes von (Duziak, 2006) beschrieben.

2. Fiktives Spiel

2.1 Grundlagen

Um den Gedanken des Fiktiven Spiels erklären zu können, werden einige allgemeine Einführungen der Spieltheorie, welche von den Autoren aufgeführt werden, benötigt. Manche hier erörterten Begriffe werden in der Beschreibung des zweiten Algorithmus erneut dargestellt. Da sich jedoch die Art und die Komplexität der Darstellung in dem Paper über das Fiktive Spiel leichter verständlich und weniger präzise darstellt, ist der nachfolgende Teil als Einführung in die behandelten Problematiken besser geeignet und eine gewissen Redundanz durchaus hilfreich für das Verständnis.

2.1.1 Nash Gleichgewicht und dominierende/nicht dominierende Fehler

Eine optimale Lösung, oder auch Nash Gleichgewicht, eines gegebenen Spiels ist nach den Autoren des Textes ein aus intelligentem Verhalten abgeleitete Strategie, welche den Verlust eines Teilnehmers minimiert. Ein Spieler welcher strategisch dominante Fehler begeht, wird langfristig gegen eine optimale Strategie verlieren. Der einzige Nachteil hierbei ist die Voraussetzung, dass der Gegner Fehler machen muss, damit die optimale Strategie zum Erfolg führt. Anhand des Beispiels "Schere, Stein, Papier" werden die drei Begriffe Nash Gleichgewicht, dominierender und nicht dominierender Fehler verdeutlicht. Die optimale Strategie in diesem Spiel ist es (über die Summe der Spiele) jede der drei Möglichkeiten zu einem Drittel zu spielen. Immer nur zum Beispiel Stein zu spielen, wäre keine optimale Lösung, aber es besteht immer noch zu je einem Drittel die Möglichkeit zu verlieren, gewinnen und Gleichstand zu spielen. Der Fehler in der Strategie ist daher ein "nicht dominierender". Wenn man jetzt noch eine vierte Auswahlmöglichkeit hinzunehmen würde die nur einmal gewinnen und zweimal verlieren kann, wäre es ein dominierender Fehler diese zu wählen, da in der optimalen Strategie diese in null Prozent der Fälle zu wählen ist. In komplizierteren Spielen treten solche Fehler ausreichend oft auf, um zu deutlichen Schwächen in der Strategie führen zu können.

2.1.2 Definition eines Fiktives Spiels

Frei übersetzt aus Quelle (Duziak, 2006) ist ein "Fiktives Spiel" eine Menge von Lernregeln, entworfen damit Teilnehmer (eines Spiels) befähigt werden ihr Handeln einem Optimum anzunähern. In einem fiktiven Spiel gelten folgende vier Regeln:

1. Jeder Spieler analysiert die Strategie des Gegners und erfindet eine beste Antwort
2. Wurde eine beste Antwort berechnet, wird sie in die aktuelle Strategie eingesetzt (oder ersetzt diese)
3. Jeder gegnerische Spieler führt ebenfalls Schritt 1 und 2 durch
4. Die vorhergehenden Schritte werden wiederholt, bis eine stabile Lösung erreicht wird

Laut dem Autor des Artikels kann nicht immer eine stabile Lösung berechnet werden.

2.2 Abstraktionen für das Fiktive Spiel im Poker

Wie bereits einleitend erwähnt, wird die Komplexität von Texas Hold'em in diesem Ansatz von 10^{18} auf 10^7 reduziert. Jedoch werden die Schlüsseleigenschaften des Spiels durch eine angemessene Abstraktion erhalten. Lösungen auf dieser Abstraktion werden pseudooptimal genannt. Zwei grundlegenden Techniken sind hierbei das *position isomorph* und das *suit equivalenz isomorph*. Erstere besagt, dass die Reihenfolge der Karten in der Hand und im Flop keine Rolle spielt, und die Zweite ignoriert die Farben der Karten. Beide Techniken haben die Eigenschaft die Optimalität einer Lösung auf der Abstraktion nicht zu beeinflussen. Um jedoch das Problem für eine Berechnung genügend zu reduzieren, sind weitere Schritte notwendig. Eine Möglichkeit hierfür ist das so genannte *Bucketing*. Hierbei wird versucht Kartensätze mit der gleichen Gewinnwahrscheinlichkeit zu gruppieren. In dem vorliegenden Ansatz werden 169 solcher Buckets im Preflop und 256 in den folgenden Runden verwendet. Beispielsweise kommen die Hände "2h,4d,3c,5s,6s" und "2d,5c,4h,6d,3h" (d=diamonds,h=heart,s=spades,c=cross) in die gleichen Buckets, da sie sich nur in Farbe und Reihenfolge unterscheiden.
Weiterhin wird eine Technik namens *Chance Node Elimination* verwendet. Für FS benötigt man einen genau definierten und (nach guten Lösungen) durchsuchbaren Spielbaum. Allerdings hat jede der vier Runden im Poker (Preflop, Flop, Turn und River) zwischen zwei Spielern zehn Zustände, welche bedingt durch die Vielzahl der möglichen Karten der Spieler und im Flop, jeweils zu einer vollkommen anderen Nachfolgestruktur führen. Der zugehörige Baum zu jeder Runde wird als Domäne bezeichnet. Der Übergang zwischen diesen Domänen durch Check oder Call kann durch so genannte Zufallsknoten (engl.: Chance Node) beschrieben werden. Ein Zufallsknoten ist eine Struktur aus den Bayesschen Netzwerken (Stockhammer, 2006), welche Verwendung findet in Einflussdiagrammen. Bayesschen Netzwerke kombinieren Graphentheorie und Wahrscheinlichkeitsrechnung. Ein Zufallsknoten repräsentiert hierbei eine Zufallsvariable, welche eine bestimmte, sich gegenseitig ausschließende Anzahl von Zuständen annehmen kann, wobei jeder Zustand eine gewisse Eintrittswahrscheinlichkeit hat. Die hohe Menge der möglichen Zustände sorgt für ein schnelles, exponentielles Wachstum. Die Lösung in diesem Ansatz besteht daher darin diese Zustandsknoten zu eliminieren und durch Konvertierungsmatrizen zu ersetzen, welche die gleiche Funktion erfüllen, aber das exponentielle Wachstum einschränken. Jeder Knoten einer Domäne hat auf diese Weise nur noch einen Domänenunterbaum. Als Möglichkeit eine Konvertierungsmatrix darzustellen, wir im Text folgendes Beispiel angegeben:

$$\begin{bmatrix} P(a) \\ P(b) \\ P(c) \\ P(d) \end{bmatrix} := \begin{bmatrix} P(a|A) & P(a|B) & P(a|C) & P(a|D) \\ P(b|A) & P(b|B) & P(b|C) & P(b|D) \\ P(c|A) & P(c|B) & P(c|C) & P(c|D) \\ P(d|A) & P(d|B) & P(d|C) & P(a|D) \end{bmatrix} \begin{bmatrix} P(A) \\ P(B) \\ P(C) \\ P(D) \end{bmatrix}$$

Ausgedrückt wird eine *Übergangswahrscheinlichkeit* (engl.: Transition Probabilitie) von einer Domäne 1 mit den möglichen Zuständen $\{A, B, C, D\}$ zu einer Domäne 2 mit den möglichen Zuständen $\{a, b, c, d\}$. Die Wahrscheinlichkeit, dass das Spiel im Zustand a in Domäne 2 angelangt, entspricht demnach der Summe der konjugierten Wahrscheinlichkeiten

zwischen a und allen Zuständen der Domäne 1 (da P(a|b)*P(b)=P(a,b)). Im Grunde wird der Übergang eines Bucket in Domäne 1 zu dem passenden in Domäne 2 beschrieben, oder anders gesagt von Kartenblättern gleicher Gewinnwahrscheinlichkeit einer Runde zu denen der nächsten. Für das (aufwendige) Berechnen dieser Matrizen werden zwei Beispiele angegeben. Beim *Abdeckenden Übergang* (engl.: Masking Transition) werden zunächst allgemeine Übergangswahrscheinlichkeiten von Domäne zu Domäne erstellt und dann mit speziellen Informationen über den Zustand der Zieldomäne überdeckt. Der nächste Ansatz des *Perfekten Übergangs* (engl.: Perfect Transition) hingegen verwendet Vorberechnungen zu jedem möglichen Spielzustand, welche während der Laufzeit verwendet werden können um perfekt angepasste Umwandlungsmatrizen, welche den Zustand von Domäne 1 und 2 mit einbeziehen, zu generieren. Auf Grund der vielen Spielzustände und der einfacheren Berechnung und Speicherung der Buckets, fiel die Auswahl auf die erste Möglichkeit.

2.3 Verwendung des "Fiktiven Spiel" Ansatzes

Die als *Adam* bezeichnete Implementation des Spiels, wurde nach den Prinzipien des FS trainiert (Anpassung einer allgemeinen Lösung). Dies erfolgte durch zwei Spieler, welche alles übereinander wissen. Für zufällige Spielsituationen wird mit dem Wissen beider Spieler die "korrekte" Handlungsweise bestimmt und eine allgemeine Lösung entsprechende angepasst. Für jeden Knoten des Entscheidungsbaumes wurde dieser Vorgang hunderttausende Mal angewendet, bis eine stabile (optimale) Lösung gefunden wurde in der dominante Fehler weitgehend Beseitigt sind und die ein annährendes Nashgleichgewicht darstellt. Im "realen" Spiel muss Adam auf verschiede Wege Ansätze zur Lösungsbestimmung finden. Im Preflop (welcher durch die Abstraktion unverändert bleibt) kann sich Adam noch vollkommen auf die vorberechnete Lösungen verlassen. Erst danach können diese mehr und mehr nur als Referenz verwendet werden um den Spielbaum effizienter nach der besten Lösung zu durchsuchen. Hierbei wird jeweils die Aktion gewählt, welche zu dem Unterbaum mit dem maximalen Gewinnwert führt.

3. Verlustminimierung

Der zweite hier behandelte Artikel (M. Zinkevich & Piccione, 2007) beschäftigt sich mit der *Verlustminimierung* (frei aus dem engl.: regret minimization) in umfangreichen Spielen (engl.: extensive games) mit unvollständigen Informationen. Um die Inhalte dieses Textes besser vermitteln zu können, folgen auch in diesem Abschnitt einige allgemeine Erörterungen. Interessant sind diese auch in Bezug auf das "Fiktive Spiel", da sich mit der im Anschluss verwendeten Terminologie auch diese Methode genauer spezifizieren lassen würde. Im Anschluss wird das Prinzip des "Kontrafaktischen Verlustes" und dessen Anwendung im Poker dargestellt.

3.1 Grundidee und Grundlagen

Wie der Name schon sagt, geht es darum den Verlust eines Spielers möglichst auf Null zu reduzieren, wobei es zuerst einmal nicht explizit um das Spiel Poker geht. Zusätzlich wird aber auch gezeigt, wie sich diese Minimierung nutzen läßt um ein Nashgleichgewicht für ein Spiel zu berechnen. Einige der im Folgenden genannten Beschreibungen zur allgemeinen

Spieltheorie wurden ergänzt mit der Hilfe eines weiteren Artikels (Osborne, 2006). Der Begriff des "Extensive Game" (EX) kommt aus der allgemeinen Spieltheorie. Der Kern hierbei ist zunächst einmal ein perfekter Spielbaum. Jeder Endzustand dieses Baumes (Blätter) ist mit einem bestimmten Gewinn/Verlust für alle Spieler assoziiert und jeder andere mit einem Spieler, welcher eine Aktion wählen muss. Der Zusatz "mit *unvollständigen Informationen*" gibt an, dass jedem Spieler nicht der gesamte (bisherige) Spielbaum während eines Spiels bekannt ist. Bestimmte Zustände kann er nicht voneinander unterscheiden und liegen daher in einer Informationsmenge (weil er z.B. im Poker die Karten der anderen Spieler nicht kennt). Genau definiert besteht ein "extensive game" aus mehreren Strukturen, welche in anschließender Liste beschrieben und in ihrer formalen Schreibweise wiedergegeben werden.

- Es gibt eine endliche Anzahl von Spielern, welche durch N ausgedrückt wird.
- Eine endlichen Menge H von Sequenzen, welche die möglichen *Historien* von Aktionen darstellen. Historien stellen Pfade durch den Spielbaum dar, welche leer sein können (Pfad "zum Root"), zu einem beliebigen Knoten im Baum verlaufen können oder bis zu einem terminalen Knoten (Blatt) verlaufen. Letztere bezeichnet man als terminale Historien und werden durch die (echte) Untermenge Z von H angegeben.
- Die Funktion $A(h)$ gibt die Menge der auswählbaren Aktionen nach (z.B. Call und Check) einer nicht terminale Historie $h \in H$ $(H \backslash Z)$ zurück.
- Die Funktion $P(h)$ weißt jeder Historie $h \in H \backslash Z$ einen Spieler aus N zu, welcher eine mögliche Aktion (aus $A(h)$) auswählen muss. Die Menge der Spieler wird allerdings um einen Spieler c erweitert. Ist $P(h) = c$ wird eine zufällige Aktion ausgewählt (z.B. wenn eine neue zufällige Karte in den Flop kommt).
- Die Funktion f_c assoziiert mit jeder Historie h mit $P(h) = c$ (der Zufall bestimmt die nächste Aktion) eine Wahrscheinlichkeitsmessung $f_c(\bullet|h)$ auf $A(h)$. Die Funktion f gibt also für jede nach einer Historie auswählbare Aktion die Wahrscheinlichkeit an, dass sie auch tatsächlich gewählt wird (also z.B. wie hoch die Wahrscheinlichkeit ist, dass ein Bube gewählt wird).
- Die Untergliederung (Informationspartition) I_i eines Spielers i beinhaltet alle Historien h in welchen $P(h) = i$ gilt. Alle Historien h' mit den gleichen auswählbaren Aktionen $(A(h) = A(h'))$ müssen hierbei in der gleichen Teilmenge der Informationspartition stehen. Diese Teilmengen werden mit $I_i \in \mathrm{I}_i$ dargestellt und als eine Informationsmenge von Spieler i bezeichnet. $A(I_i)$ gibt die Menge der auswählbaren Aktionen nach den Historien in I_i an und $P(I_i)$ den Spieler. Verdeutlichend beschrieben umfasst eine Informationsmenge die Historien, welche für einen Spieler auf Grund seines unvollständigen Wissens im Spielbaum nicht zu unterscheiden sind.
- Zuletzt gibt es für jeden Spieler einer Funktion u_i welche die terminalen Zuständen in Z auf relle Werte abbildet. Gleichen sich die Funktionswerte aller Spieler an jeweils allen terminalen Knoten auf Null aus, wird das Spiel als ein "zero-sum extensive Game" bezeichnet. Die maximale Spanne zwischen kleinstem und größten Wert der terminalen Knoten für Spieler i wird mit $\Delta_{u,i}$ dargestellt.

Bezeichnung	Beschreibung
σ	Beschreibt ein vollständiges Strategieprofil, also eine feste Strategie für jeden Spieler
σ_{-i}	Alle Strategien in σ außer σ_i
$\pi^{\sigma}(h)$	Wahrscheinlichkeit, dass die Historie h erscheint, wenn die Spieler entsprechend dem Profil σ ihre Aktionen wählen
$\pi_i^{\sigma}(h)$	Wahrscheinlichkeit, dass wenn Spieler i nach σ spielt, er in Historie h die entsprechende Aktion wählt wie in den Präfixen h' von h an denen er auch am Zug war ($P(h') = i$), wobei $\pi^{\sigma}(h) = \prod_{i \in N \cup \{c\}} \pi_i^{\sigma}(h)$ gilt (Produkt aller Einzelbeiträge der Spieler)
$\pi_{-i}^{\sigma}(h)$	Produkt der Beiträge aller Spieler, außer dem von i
$\pi^{\sigma}(I)$	Ist die Wahrscheinlichkeit, dass wenn alle Spieler nach dem Profil σ spielen, Informationsmenge I erreicht wird. Formal ausgedrückt gilt $\pi^{\sigma}(I) = \sum_{h \in I} \pi^{\sigma}(h)$ (Summe der Wahrscheinlichkeiten aller Historien in I)
$u_i(\sigma)$	Erwartete Auszahlung für Spieler i bei Profil σ in dem daraus resultierenden Endknoten, formal dargestellt durch $u_i(\sigma) = \sum_{h \in Z} u_i(h)\pi^{\sigma}(h)$ (Summe aller Gewinnwerte für Spieler i im Spielbaum, gewichtet mit der Wahrscheinlichkeit, dass die jeweilige terminale Historie in σ auftritt)

Table 1: Begriffsdefinitionen für Strategien in "Extensive Games"

3.1.1 Strategien und Nash Gleichgewicht

Eine Strategie σ_i ist eine Funktion, welche Aktionen aus $A(I_i)$ auswählt für alle Informationsmengen des Spielers i. Zur besseren Übersicht erfolgt die Definition über die weiteren verwendeten Zeichen im Zusammenhang mit Strategien in Tabelle 1.

Ein Nash Gleichgewicht wird hier entsprechend der in Tabelle 1 beschriebenen formalen Ausdrücke dargestellt. Bei zwei Spielern bedeutet dies:

$$u_1(\sigma) = max_{\sigma_1' \in \sum_1} u_1(\sigma_1', \sigma_2) \qquad \text{und} \qquad u_2(\sigma) = max_{\sigma_2' \in \sum_2} u_1(\sigma_1, \sigma_2')$$

Ein Nash Gleichgewicht ist demnach ein Profil, in dem jeder Spieler bei gegebener Strategie der Gegner (hier nur einer) von seinen möglichen Strategien ($\sum_i$) die Auswählt, welche seinen Gewinn maximiert. Keiner kann demnach mehr eine andere Strategie wählen, ohne seinen Gewinn zu schmälern. Interessant in diesem Zusammenhang ist die Eigenschaft von "Extensive Games", dass diese nicht zwingend ein Nash Gleichgewicht besitzen müssen oder sogar viele Existieren (für Beispiele siehe auch (Osborne, 2006)). Ein Nash Gleichgewicht bei dem das Maximum nicht vollständig erreicht wird, nennt man auch ϵ-Nash Gleichgewicht.

Zum besseren Verständniss werden die definierten Begriffe zunächst einmal durch das in Abbildung 1 auf der linken Seite dargestellte Beispiel aus (Osborne, 2006) eines "Extensive Games" mit vollständigen Informationen verdeutlicht. Das Spiel hat die terminalen Historien $Z = \{(X, w,), (X, x), (Y, y), (Y, z)\}$. Mit h als leere Historie ist $P(h) = 1$ (also Spieler 1) am Zug) und $A(h) = \{X, Y\}$. Wenn $h = \{X\}$ oder $h = \{Y\}$ ist $P(h) = 2$. Spieler 1 hat in diesem Beispiel nur zwei Strategien, und zwar X und Y. Spieler 2 hat vier

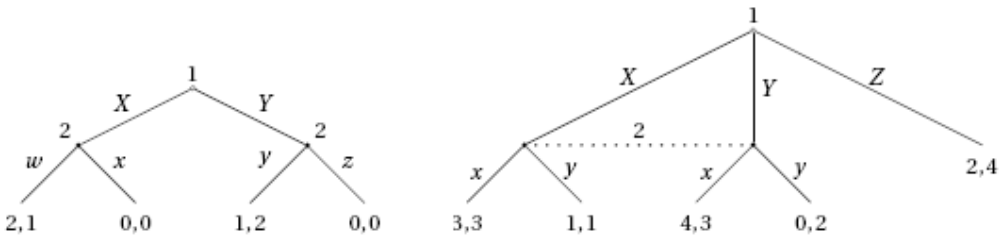

Figure 1: Beispiele für "Extensive Games" aus (Osborne, 2006)

Strategien und zwar wy, wz, xy, und xz, wobei zuerst immer die Wahl von Spieler 2 steht, wenn Spieler 1 X gewählt hat und dann die wenn Spieler 1 Y gewählt hat. Eine Strategie ist demnach immer eine vollständige Beschreibung der Handlungsweise in jedem Spielzustand. Nashgleichgewichte in diesem Spiel sind (X, wy), (X, wz), und (Y, xy), da in jedem Fall der gewählten Strategien kein Spieler eine andere Strategie wählen kann um seinen Gewinn zu vergrößern (z.B. im letzten Fall, wenn Spieler zwei sich schon für xy entschieden hat, würde Spieler 1 durch Wahl von X nicht erhalten). Es ist festzustellen, dass unter Berücksichtigung der Abhängigkeit der tatsächlichen Entscheidung von Spieler 1, nur das erste Nashgleichgewicht optimal ist. In der weiteren Betrachtung spielt dies jedoch keine Rolle. Auf der rechten Seite von Abbildung 1 sehen wir ein Beispiel für ein partitioniertes "Extensive Game" mit unvollständigen Informationen. Die Historien zu den durch eine Linie verbundenen Zuständen sind hierbei nicht unterscheidbar für Spieler 2. Sein Informationspartition ist demnach $\{\{X, Y\}, \{Z\}\}$, mit den zwei Informationsmengen $\{X, Y\}$ und $\{Z\}$ und X,Y,Z als Historien. Besonders zu beachten ist das $A(X) = A(Y) = \{x, y\}$ gilt wie vorgeschrieben. Strategien müssen jetzt auf Basis der Informationsmengen definiert werden, da die Wahl einer Aktion nach allen zugehörigen Historien gleich ist (aus Sicht des Spielers).

3.1.2 Verlustminimierung

Das Prinzip der Verlustminimierung ist ein allgemein bekanntes Spielkonzept, welches auf Lernen beruht. Betrachtet wird das wiederholte Spielen eines "Extensive Games", wobei σ_i^t die von Spieler i in Runde t verwendete Strategie darstellt. Der Gesamtverlust eines Spielers über alle Runden ergibt sich aus der Strategie, die im Schnitt über alle Runden die Differenz zum Nachgleichgewicht des Strategieprofils der Runden möglichst positiv werden läßt. Formal nach dem Paper bedeutet dies:

$$R_i^T = \frac{1}{T} max_{\sigma_i^* \in \sum_i} \sum_{t=1}^{T} (u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma^t))$$

Mit $\bar{\sigma}_i^T$ als durchschnittliche Strategie bis zum Zeitpunkt T für Spieler i wird für jede Informationsmenge des Spielers und jede auswählbare Aktion von dieser folgendes berechnet:

$$\bar{\sigma}_i^T(I)(a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)(a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}$$

Bezeichnung	Beschreibung
$u_i(\sigma, h)$	Erwarteter "Nutzen" für Spieler i nachdem Historie h erreicht wurde und alle Spieler die Strategie entsprechend von σ verwendet haben
$u(\sigma, I)$	Wird von den Autoren als "Kontrafaktischer Nutzen" (engl.: Counterfactual utility) bezeichnet. Es steht für den Nutzen welcher zu erwarten ist nachdem I erreicht wurde, wenn alle Spieler Strategie σ verwenden und Spieler i nicht gespielt hat um I zu erreichen.
$\sigma\|_{I \to a}$	Definiert für alle $a \in A(I)$ ein zu σ identisches Strategieprofil, außer dass Spieler i immer Aktion a wählt, wenn er in der Informationsmenge I ist.

Table 2: Begriffsdefinitionen für "Counterfactual Regret"

Also die durchschnittliche Wahrscheinlichkeit, dass a in I gewählt wird (wenn I überhaupt erreicht wird). Ein Algorithmus, welcher den Verlust eines Spielers möglichst minimiert, muss in jeder Runde t auf die Art eine Strategie σ_i^t für Spieler i wählen (unabhängig von der Strategien der anderen Spieler), dass R_i^T gegen Null geht, wenn t gegen Unendlich geht. Da der Verlust über die Differenz des Gewinns einer Runde zum Gewinn im Nash Gleichgewicht der Runde definiert ist und in einem Null-Summen Spiel sich Gewinn und Verluste auf Null ausgleichen, kann ein solcher Algorithmus auch direkt verwendet werden um ein gesamtes Nashgleichgewicht anzunähern (also eine "durchschnittliche Strategie" auf Basis des Wissens eines Spielers, welche ein Nashgleichgewicht annähert). In einem Nullsummenspiel ist $\bar{\sigma}^T$ (also das Strategieprofil zum Zeitpunkt T) demnach ein Nashgleichgewicht, wenn der Gesamtverlust beider Spieler kleiner ϵ ist (die Strategie ist dann ein 2ϵ-*Nash Gleichgewicht*)

3.2 Kontrafaktischer Verlust

Der von den Autoren das Papers (M. Zinkevich & Piccione, 2007) definierte Begriff des *Kontrafaktisches Verlustes* (engl.: Counterfactual Regret), beschreibt die Aufteilung des Gesamtverlustes auf einzelne Informationsmengen, welche unabhängig von einander minimiert werden können. Die Summe der Verluste der einzelnen Informationsmengen stellte eine Schranke für den Gesamtverlust dar. Im Folgenden werden die in Tabelle 2 definierten Begriffe verwendet.

Der Begriff "Kontrafaktisch" beschreibt den Umstand, dass man etwas betrachtet wie es hätte sein können. Kontrafaktisches Denken beschreibt somit zum Beispiel einen Gedankengang wie in etwa "Wenn ich damals Lotto gespielt hätte, wäre ich heute Millionär". Tatsächlich beschreibt der von den Autoren beschriebene *immediate counterfactual regret* (Abk.: ICR) den Verlust welcher ein Spieler in einer bestimmten Informationsmenge gehabt hätte, wenn er von Anfang an versucht hätte diesen zu erreichen. Formal dargestellt mit den in Tabelle 2 beschriebenen Mitteln sieht dies dann folgender Maßen aus:

$$R_{i,imm}^T(I) = \frac{1}{T} max_{a \in A(I)} \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I))$$

Der Verlust nach einer Informationsmenge I definiert sich also über die Aktion in $A(I)$, welche optimal zu wählen gewesen wäre in den vorherigen Runden, gewichtet mit der Wahrscheinlichkeit, dass I auch ohne das Wirken von Spieler i anhand der Strategien

der anderen Spieler, erreicht worden wäre. Laut den Autoren ist der tatsächliche Verlust eines Spielers R_i^T immer kleiner gleich $\sum_{I \in \mathrm{I}_i} R_{t,imm}^T(I)$, also kleiner als die Summe aller seiner ICR (der Entsprechende Beweis ist im Paper vorhanden).
Jetzt kann ein Nash Gleichgewicht gefunden werden, nur durch Minimierung der einzelnen ICR, was den Vorteil hat, dass sich diese minimieren lassen nur durch Kontrolle von $\sigma_i(I)$ (der Strategie eines Spielers). Für alle Informationsmengen I der Informationspartition eines Spielers i, wird für alle $a \in A(I)$ entsprechend der Formel für ICR folgendes berechnet:

$$R_i^T(I,a) = \frac{1}{T} \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I))$$

Dies entspricht dem Kontrafaktischen Verlust bei der Informationsmenge I und Aktion a, wie schon zuvor beschrieben. Die Autoren definieren jetzt $R_i^{T,+}(I,a) = max(R_i^T(I,a), 0)$, also auf den maximalen Kontrafaktischen Verlust und bilden daraus eine Strategie für Spieler i in der nächste Runde $T+1$. Auch dies erfolgt erst einmal in der formalen Definition des Papers:

$$\sigma_i^{T+1}(I)(a) = \frac{R_i^{T,+}(I,a)}{\sum_{a \in A(I)} R_i^{T,+}(I,a)}, \text{ wenn } \sum_{a \in A(I)} R_i^{T,+}(I,a) > 0, \ \frac{1}{|A(I)|} \text{ sonst.}$$

Wie auch im Paper treffend ausgedrückt bedeutet dies, dass "Aktionen proportional zu der Menge an positivem Kontrafaktischen Verlust ausgewählt werden, der Auftritt wenn man nicht diese Aktion wählt. Hat keine Aktion einen positiven Wert, wird zufällig eine ausgewählt". Da wenn ein Spieler seine Aktionen entsprechend der letzten Definition auswählt $R_{i,imm}^T(I) \leq \Delta_{u,i}\sqrt{|A_i|}/\sqrt{T}$ gilt und entsprechend auch $R_i^T(I) \leq \Delta_{u,i}|I_i|\sqrt{|A_i|}/\sqrt{T}$ (mit $|A_i| = max_{h:P(h)=i}|A(h)|$), kann diese Wahl der Strategie laut den Autoren verwendet werden ein Nash Gleichgewicht zu berechnen (die Beweise sind auch hier im Paper vorhanden). Mit eigenen Worten bedeutet dies, weil die ICR durch $\Delta_{u,i}\sqrt{|A_i|}/\sqrt{T}$ (Max. Spanne im Verlust des Spielers, mal der maximalen Anzahl aller auswählbaren Zustände in denen Spieler i nach einer Historie h am Zug ist, mal der Wurzel des Nummer der aktuellen Runde) beschränkt sind, gilt zusammen mit $R_i^T \leq \sum_{I \in \mathrm{I}_i} R_{t,imm}^T(I)$ (wie zuvor auch schon definiert), dass auch der tatsächliche Gesamtverlust beschränkt wird. Die Herleitung eines Nash Gleichgewichtes ergibt sich aus dem schon aufgeführten Grund, dass sich in einem Nullsummenspiel Gewinn und Verlust ausgleichen.

3.3 Kontrafaktischer Verlust im Poker

Nachdem nun im Allgemeinen beschrieben wurde wie man ein Nash Gleichgewicht berechnen kann, bezieht sich das Paper direkt auf das Spiel Poker zwischen zwei Spielern in der Variante Texas Hold'em.

3.3.1 Abstraktion

Zunächst einmal wird die Art der Abstraktion des Spiels beschrieben, deren Ziel es ist die Anzahl der Informationsmengen auf eine berechenbare Größe zu reduzieren. Die Wettregelen werden hierbei voll übernommen und die Abstraktion bezieht sich nur auf die gespielten Karten. Die Karten werden entsprechend der *quadrierten Handstärke* (engl.: hand strenght squared) gruppiert. Die Handstärke ist hierbei die Gewinnwahrscheinlichkeit nur nach den

Karten, die ein Spieler aktuell sehen kann. Mit der quadrierten Handstärke ist das Quadrat der Handstärke nach der zuletzt aufgedeckten Karte gemeint. Durch das Quadrieren erhalten starke Blätter einen zusätzlichen Vorteil. Für die Abstraktion werden zu Beginn die aus den Startkarten resultierenden Sequenzen (Historien, z.B. Bube und Dame auf der Hand durch den Zufallsspieler) entsprechend ihrer quadrierten Handstärke in einen von 10 gleich großen Buckets einsortiert. Dann werden alle Sequenzen der Runde zwei, welche Präfixe aus den gleichen Buckets der Runde eins haben, in einen von 10 neuen Buckets einsortiert (wieder nach quadrierter Handstärke). In Runde zwei sind demnach alle Sequenzen einer Partition als ein paar von Zahlen darstellbar: Der Nummer des Buckets der vorherigen Runde und dem in der aktuellen (welcher von dem der vorherigen abhängt). Die wird nun in jeder Runde wiederholt, wobei die Einordnung immer wieder nach der Übereinstimmung in den vorherigen Buckets geschieht, wodurch Kartensequenzen in *Bucketsequenzen* (im Grunde Informationsmengen, also nicht mehr unterscheidbaren Historien) partitioniert werden. Das hieraus resultierende, abstrakte Spiel hat nach den Autoren eine Anzahl von 10^{12} Zuständen und 10^7 Informationsmengen.

3.3.2 Minimierung des Kontrafaktischen Verlustes im Poker

Die Minimierung wird nun auf dem abstrakten Spiel durchgeführt. Im Prinzip Spielen zwei Gegner ständig gegeneinander und benutzen die abgeleitete Strategie jeweils für die Runde $T + 1$. Nach einer bestimmten Anzahl Iterationen wird die auf diese Weise berechneten Strategien $(\bar{\sigma}_1^T, \bar{\sigma}_2^T)$ als das angenährte Ergebnisgleichgewicht verwendet. Im Grunde müssten alle $R_i^t(I, a)$ bei jeder Iteration gespeichert und danach aktualisiert werden. Hier verwenden die Autoren jedoch einen zusätzlichen Trick, welcher die Anzahl an zu aktualisierenden Informationsmengen deutlich reduziert. Kern ist hierbei der Zufallsspieler c, dessen Handlungen (da sie zufällig sind) keinen Einfluss auf die Strategien der Spieler haben (Zumindest in Bezug auf die Abstraktion durch die Buckets). Die Handlungen der echten Spieler werden daher in einer als "Joint Bucket Sequenz" bezeichneten Struktur zusammengefasst, für welche anschließend nur noch die Updates gemacht werden müssen. Genauere Angeben werden hierzu leider nicht gemacht.

4. Vergleich und Nutzen

Die beiden vorgestellten Algorithmen, "Fiktives Spiel" und "Verlustminimierung", schließen beide mit einem Abschnitt ab, indem sie ihren jeweiligen Algorithmus gegen jeweils andere antreten lassen. Leider gibt es in beiden keinen direkten Vergleich untereinander. Beide messen sich allerdings mit einem gemeinsamen Gegner mit dem Namen PsOpti. Ebenfalls gewinnen beide gegen diesen Gegner. Da die vorgestellten Ergebnisse allerdings sehr verschieden sind, lässt sich nur auf Grund der Paper keine konkrete Aussage darüber machen welcher der bessere ist. Einzig die höhere Komplexität der Abstraktion bei der Verlustminimierung spricht dafür, dass dieser Algorithmus im direkten Vergleich besser sein könnte.
Was wir von beiden Algorithmen lernen können ist, dass die als "Bucketing" (oder erstellen von Informationsmengen) bezeichnete Methode zur Abstraktion des Spielbaumes, allgemein sinnvoll zu sein scheint. Während der Text zum "Fiktiven Spiel" hier mehr allgemeine Erklärungen liefert, erhalten wird bei dem Text zur Verlustminimierung sehr exakte

Definitionen und für den der den Algorithmus genau überprüfen will auch die nötigen Beweise. So erfahren wir, dass Buckets nicht anders als Informationsmengen sind (nach dem Wortlaut von (M. Zinkevich & Piccione, 2007)), welche verschiedene Pfade durch den Spielbaum als einen erscheinen lassen. Letzten Endes verwenden beide Ansätze ein ähnliches Gedankengut, nur dass es im Fall der Verlustminimierung genauer spezifiziert wird. Was im Fiktiven Spiel als "Chance Node Elimination" bezeichnet wird und durch Konvertierungsmatrizen aufgelöst wird, stellt auch nur eine Möglichkeit dar Informationsmengen zu erstellen und so Historien zusammenzulegen und den Zustandraum zu reduzieren. Allgemein ist leicht zu sehen, dass irgendeine Art der Abstraktion notwendig ist, um die hohe Komplexität des Spiels in den Griff zu bekommen. Auch wenn der Autor von "Fiktives Spiel" keine Angaben zur Laufzeit seines Algorithmus macht, dürfte auch hier sehr viel Zeit benötigt werden. Für die Technik der Verlustminimierung wurde eine Anzahl von 100 Millionen Iterationen in 33 Stunden durchgeführt um auf eine stabile Lösung zu kommen. Ohne eine angemessene Abstraktion wird also (noch) kein Algorithmus erfolgreich eine Lösung in absehbarer Zeit berechnen. Für den Ansatz der Verlustminimierung spricht an dieser Stelle allerdings auch, dass keine aufwendige Berechnung von Konvertierungsmatrizen notwenig ist um den Übergang zwischen verschiedenen Buckets zu bestimmen.
Als abschließendes Wort sei bemerkt, dass der Ansatz der Verlustminimierung in Hinsicht auf eine mögliche Nachimplementation die bessere Wahl zu sein scheint. Zwar wirkt die Theorie komplexer, aber bedingt durch die genauen Spezifikationen, fällt eine Umsetzung, wenn der Stoff erst einmal verstanden ist, entsprechend leichter und im Anhang ist sogar fertiger Programmkode vorhanden.

References

Duziak, W. (2006). Using fictitious play to find pseudo-optimal solutions for full-scale poker. *In Proceedings of the 2006 International Conference on Artificial Intelligence (ICAI-2006)*.

M. Zinkevich, M. Bowling, M. J., & Piccione, C. (2007). Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems (NIPS 2007)*, Seiten 374–380.

Osborne, M. J. (2006). Strategic and extensive games. *University of Toronto, Department of Economics in its series Working Papers with number tecipa-231*, Seiten 1–7 und 15–23.

Stockhammer, P. (2006). Einführung in bayessche netzwerke. *Hauptseminar Wrtschaftsinformatik - TU Claustal*.

Seminararbeit über Abstraktion in Texas Hold'em Poker

Marian Wieczorek

Abstract

Diese Seminararbeit ist eine Ausarbeitung der Paper [Billings07] und [Gilpin06]. Die Paper werden getrennt behandelt und am Schluss einander gegenüber gestellt. Da beide Paper lineares Programmieren nutzen um eine möglichst optimale Strategie zu berechnen, widmet sich das erste Kapitel grob der Formulierung von Texas Hold'em Heads Up als lineares Programmierproblem. Im zweiten Kapitel wird die Problemstellung des ersten Papers erklärt. Darauf hin folgt in Kapitel drei die Bewertung der angedachten Abstraktionen. Die Bewertungen ergeben sich aus Experimenten und Berechnungen der Autoren. Im letzten Kapitel zum ersten Paper werden die Vereinfachungen in drei Spielprogrammen zusammengeführt. Das erste Kapitel zum zweiten Paper arbeitet dessen Problemstellung heraus. Im sechsten Kapitel wird die Suche einer optimalen Strategie für die ersten zwei Runden und im siebten Kapitel für die letzten zwei Runden erleutert.

Das erste Paper stellt unterschiedliche Stufen der Abstraktion für die Zweispielervariante von Texas Hold'em Poker, Heads up genannt, vor. Dabei wird der ursprüngliche Entscheidungsbaum um hundert Billionen Blätter reduziert. Auf der vereinfachten Variante, die offline berechnet wird, wird mit linearer Programmierung eine optimale Strategie berechnet. Einige Abstraktionen arbeiten die Autoren als wertvoll heraus und erstellen auf ihnen drei Spielprogramme (Bots) *psOpti0*, *psOpti1* und *psOpti2*.

Das zweite Paper nimmt sich zur Aufgabe Heads up Hold'em mit minimiertem „Expertenwissen“ zu spielen. Die Autoren entwickeln einen Bot *GS1*, der zunächst eine große Menge an Informationen vorberechnet und eine nahezu optimale Strategie bis zum Flop entwickelt. Für die Runden Turn und River wird die Strategie hauptsächlich durch Echtzeit (online) lineares Programmieren gefunden.

1. Poker als lineares Programmierproblem

Texas Hold'em Poker hat eine einfache Struktur. Ein Spiel besteht aus mehreren Runden. Es gibt Signalrunden, in denen die Spieler Karten erhalten, oder eine oder mehrere Board-Karten aufgedeckt werden und Wettrunden in denen die Teilnehmer gegeneinander spielen. In den Wettrunden wählen die Spieler verschiedene Strategien. Sie können in geregelten Reihenfolgen checken (check), wetten (bet), austeigen (fold), mitgehen (call) und erhöhen (raise). In Limit Poker sind die Anzahl der Wetten begrenzt. Durch Aufzählung aller Strategiekombinationen lassen sich theoretisch für jede Runde die Erwartungswerte aller Spieler berechnen.

Bei bekannten Erwartungswerten kann eine Spielmatrix $M \in \mathbb{R}^{m \times n}$ aufstellt werden, die zu jeder Strategie der Spieler den Gewinn beziehungsweise Verlust (negativer Gewinn) angibt. m ist die Anzahl der Strategien von Spieler A und n die Anzahl der Strategien von Spieler B. Wählt Spieler A eindeutig eine Zeile der Spielmatrix (also eine Strategie), nennt man dies „pure strategy“. Der Gegner B versucht dann durch Wahl einer geeigneten Spalte (seine Gegenstrategie) mit seinem Zug den Gewinn für Spieler A zu minimieren (m_{ij} ist der Gewinn von Spieler A und Verlust von Spieler B). Dies bedeutet bei einem Zweispieler Zero-Sum Spiel wie Poker, dass sich der Gewinn für Spieler B erhöht. Entscheidet Spieler

A auf Grund einer Verteilung, welche Strategie er wählt, nennt man dies „mixed strategie“. Dieses Vorgehen hat den Vorteil, dass Spieler A nun seinen erwarteten Gewinn durch die Wahl einer geeigneten Verteilung statistisch vergrößern kann. Spieler A wählt immer eine Zeile und Spieler B eine Spalte. Haben beide Spieler ihre Strategie gewählt, lässt sich der Gewinn/Verlust in der Matrix ablesen.

Von Neumann bewies, dass optimales Spielen dem Spieler mindestens den Spieletheoretischen Wert des Spiels ermöglicht. Optimales Spielen mit einer „mixed strategie“ stellt also die Aufgabe, die beste Verteilung über den Strategien zu ermitteln. Dabei versucht man den minimalen Gewinn zu maximieren, was zu einem linearen Programmierproblem führt [Wayne07]. Grundlegend ist jedoch, dass die Spielmatrix bekannt ist. Bei vollständigem Heads up Hold'em ist der Entscheidungsbaum jedoch viel zu groß.

2. Problemstellung

[Billings07] sucht eine optimale Strategie für Texas Hold'em Heads up. Da der Entscheidungsbaum zu groß ist (ca. 10^{18} Blätter), werden verschiedene Abstraktionen untersucht. Ziel der Abstraktion ist es den Suchbaum so zu verkleinern, so dass auf diesem mit linearer Programmierung und erschöpfender Berechnung der abstrahierten Erwartungswerte eine optimale Strategie berechnet werden kann. Die Autoren erkennen, dass die Strategien auf dem ursprünglichen Problem keine Optimalität gewährleisten und nennen sie deshalb „pseudo-optimal“. Die Qualität der Abstraktion ist entscheidend. Aus diesem Grund werden verschiedene Abstraktionen analysiert. Es stellt sich heraus, dass Abstraktionen oft nicht effektiv genug sind, um den Suchbaum entscheidend zu verkleinern. Durch Andere verliert das Spiel an taktischen Eigenschaften, die den resultierenden Computer-Spieler schwach machen.

3. Abstraktionen

In diesem Abschnitt werden die verschiedenen Abstraktionen vorgestellt. Sowohl für Hände als auch für die Wettrunden gibt es gewinnbringende Vereinfachungen.

3.1 Abstraktion über das Blatt

Die Autoren stellen kurz übliche Reduktionen vor. Naheliegend ist die Abstraktion über Farben und Rang der Hände. Die Farbe spielt beim Pokern für die Wertung keine Rolle. Bei Rangabstraktionen fasst man Hände zusammen, die gleiche Wertigkeit aber verschiede Kombinationen aufweisen. Diese beiden Reduktionen bieten nicht viel. Durch Reduktion der Farben verkleinert man das Problem um den Faktor 24!. Gleichwertige Kombinationen treten im Poker zu selten auf, als dass sich die Abstraktion lohnt.

Eine starke Verkleinerung des Suchbaums würde sich ergeben, wenn man das Deck von 52 Karten verringern würde. Experimente haben ergeben, dass optimale Strategien auf dem reduzierten Suchbaum im tatsächlichen Spiel zu ungenau sind.

3.2 Abstraktion der Wettrunden

Vielversprechend ist die Abstraktion der Wettrunden. Man kann sich zu nutze machen, dass selbst bei unlimit Poker selten mehr als vier Runden gewettet wird. Eine vollständige Aufzählung der möglichen Strategien bei vier Runden ergibt 19 verschiedene Möglichkeiten. Fold und check gefolgt von einem Fold treten in der Praxis nicht auf. Endet eine Wettrunde mit einem Fold, ist das Spiel vorbei. Die Autoren schlagen vor die übrigen neun Kombinationen auf drei Runden zu verringern. Dieses Vorgehen birgt noch keine erkennbaren Verluste. In Experimenten wurde jedoch nachgewiesen, dass die Reduktion um weitere Runden auf einmal einen deutlichen Einbruch an Genauigkeit aufweist. Schon mit der nur drei statt vier Runden kann der Suchbaum um ca. 10^{11} $(= 2 * \binom{48}{3} * 9 * 45 * 9 * 44 * 19 + 2 * 45 * 9 * 44 * 19 + 2 * 44 * 19 + 4)$ auf ca. 10^7 Blätter reduziert werden.

3.3 Einschränken der Runden

Aus der Reduktion von Runden werden verschiedene Modelle abgeleitet. Die Autoren stellen die Möglichkeiten vor den River oder Preflop zu ignorieren. Daraus entstehen entsprechend Pre- und Postflop-Modelle. In Preflop Modellen verliert man nur strategische Raffinessen. Um nach drei Runden nicht auf den Show down zu setzen, können die Erwartungswerte über das Turn und River vorberechnet und für das lineare Programmieren genutzt werden. Für vereinfachte Pokervarianten wurden die Runden sogar bis auf eine reduziert. Hierbei verliert man sämtliche Taktiken für folgende Runden. Die Autoren benutzen ein solches Modell von Alex Selby um Strategien in einem Postflop-Modell exakter zu berechnen. Werden keine weiteren Vereinfachungen getroffen, muss das Preflop-Modell auf eine Runde beschränkt werden, damit das Problem lösbar wird. Eine gute Vereinfachung bietet Bucketing (siehe dazu Abschnitt 3.4), wodurch auch Dreirundenmodelle möglich sind.

Der Vorteil an postflop Modellen ist, dass sie den hohen Branching-Faktor des Preflops vernachlässigen. Mit anschließender Wettrunde erhält man eine Reduktion um $\binom{52}{2}\binom{50}{2} * 9 \approx 10^7$ Blätter. Der Spielverlauf ist jedoch eigentlich für Entscheidungen in späteren Runden entscheident. Bei Postflop- versucht man wie bei Preflop-Modellen nicht Erwartungswerte aber Eingangswahrscheinlichkeiten zu ermitteln. Die Autoren konstruieren für die Bots *psOpti0*, *psOpti1* und *psOpti2* verschiedene Verfahren, um die Eingangswahrscheinlichkeiten zu berechnen und messen deren Brauchbarkeit.

Anstatt Runden zu beschränken wurde angedacht, Runden zusammenzufassen. Die Modelle ergaben grobe Ungenauigkeiten, weswegen sie verworfen wurden.

3.4 Abstraktion der Strategien

Poker ist ein Spiel mit lückenhafter Information. Manche Kombinationen von Karten sind stärker, andere stellen sich im Laufe des Spiels als stark heraus. Auch Bluffen ist auf dieser Grundlage möglich. Die möglichen Varianten einer Hole-Boardcard-Kombination sind in jeder Runde bestimmbar. Das Zusammenfassen strategisch gleicher Kombinationen nennt man Bucketing. Die Autoren erkennen, dass sich die Stärke einer Kombination durch mehr als eine Dimension auszeichnet. Sie stellen die Projektion auf zwei Dimensionen vor. Erste ist der erwartete, mögliche Gewinn durch die Kombination. Als zweite Dimension wählen sie das Potential der Kombination. Dies beschreibt die Wahrscheinlichkeit, dass sich aus

der momentanen Situation eine Verbesserung ergibt. Jede zu einer Runde möglichen Kombination lässt sich als Punkt in die Ebene projezieren. Clustering-Methoden ergeben die Einteilung in die so genannten „Buckets“. Die Autoren beobachten, dass die Buckets ungleich gefüllt sind. Buckets mit hohem Erwartungswert die Hand zu gewinnen, sind weniger gefüllt als jene mit niedrigem Erwartungswert. Karten mit niedrigen Erfolgsaussichten scheinen also auch strategisch ähnlich und somit wertloser zu sein. Sichert man einem Bucket Kombinationen, die schwach im Erwartungswert sind, aber hohe Verbesserungsaussichten haben zu, kann man beobachten, dass dieser Bucket am häufigsten in Bluff-Situationen eingesetzt wird. Die Autoren wählen lediglich sechs Buckets, von denen einer für das Bluffen bereitgestellt wird. Dies ist eine sehr grobe aber trotz dessen erfolgreiche Abstraktion. Dass es keinen Unterschied macht in unterschiedlichen Runden eine unterschiedliche Anzahl von Buckets zu benutzen, wurde in Experimenten festgestellt.

Hat man für jede Runde die Buckets berechnet, wird auch jedem Gegner ein Bucket zugeteilt. Da die Karten nun abstrahiert sind, werden Übergänge zwischen den Runden nur noch als Transitionen zwischen den $(\underbrace{n \times n \times \ldots \times n}_{\text{Anzahl der Spieler}})$ Buckets betrachtet. Dazu müssen Übergangswahrscheinlichkeiten ermittelt werden. In jeder Runde ist ein Blatt mit einer gewissen Häufigkeit in allen Bucket-Tupel dieser Runde vertreten. Durch austeilen einer weiteren Karte, wechselt dies mit den angegebenen Übergangswahrscheinlichkeiten in die Bucket-Tupel der nächsten Runde. Bucketing reduziert die Komplexität von Heads up Texas Hold'em stark. Das gesamte Spiel hat nun nur noch $(6^2)^4 * 9^3 * 19 \approx 10^{10}$ und ein dreistufiges Preflop-Modell 10^7 Blätter.

4. Kombination von pre- und postflop Modellen

Generell wird nach einer optimalen Auswahl der Strategien gesucht. Diese gewinnt man durch Lösen des Minimax-Problems mit Hilfe von linearem Programmieren auf dem abstrahierten Suchbaum. Als Abstraktionen stehen pre- und postflop Modelle zur Verfügung. Es liegt nahe für jede Runde einer Hand unabhängig eine Strategie zu berechnen. Der Spielverlauf ist aber maßgeblich für alle Entscheidungen, die während eines Spiels getroffen werden. In geringem Maße ist Dekomposition jedoch möglich.

Strategie für *psOpti0* Für die Strategie von *psOpti0* in den Wettrunden wird das lineare Programmierproblem auf einem dreistufigen postflop Modell berechnet. Dieses nutzt die Abstraktionen der Reduktion in den Wettrunden und Bucketing. Da das Preflop für die Optimierung der Strategie nicht benutzt wird, müssen die Eingangswahrscheinlichkeiten abgeschätzt werden. Hier wird für *psOpti0* Gleichverteilung der Strategien im Preflop angenommen. Spielt *psOpti0* Poker, wird es in der ersten Runde die Aktionen des Gegners stets mit call abnicken. Im weiteren Verlauf nutzt es die optimierte Strategie.

Strategie für *psOpti1* *psOpti1* geht bei der Berechnung der Strategie für das Postflop-Modell ebenfalls von einer Gleichverteilung aus. Es berechnet vier Modelle für Potgrößen von zwei, vier, sechs und acht bets. Im Gegensatz zu *psOpti0* spielt *psOpti1* das Preflop mit dem Einrundenmodell von Alex Selby (siehe Abschnitt 3.3) und wechselt danach zum zur Potgröße passenden postflop Modell. Dieses Aneinandersetzen ist

wie bereits bemerkt spieltheoretisch gesehen falsch. Es wird jedoch das beste Ergebnis erziehlen.

Strategie für *psOpti2* Für *psOpti2* werden sieben postflop Modelle berechnet. Ihre Eingangswahrscheinlichkeiten ergeben sich aus einem von Profis „handgefertigten“ Preflop-Modell. Im Gegensatz zum Einrundenmodell wird diesmal ein Dreistufiges benutzt. Der enorme Branching-Faktor der Preflop-Phase wird durch Bucketing reduziert. Die Autoren erhoffen sich dadurch eine bessere Approximation der Anfangsbedingungen.

Alle Programme benutzen sechs Buckets pro Runde.

5. Problemstellung

[Gilpin06] behandelt das Spielen von Poker mit nur minimalem Vorwissen über das Spiel. Um nach wie vor in annehmbarer Zeit Strategien zu berechnen, lagern die Autoren aufwendige Berechnungen einmal in Datenbanken aus. Der Artikel geht nicht ausführlich auf die Abstraktionen und ihre Wirkung ein, benutzt sie aber implizit. Der Einfluss der meisten, benutzten Vereinfachungen wurde bereits in den vorherigen Kapiteln erleutert. Ist der Entscheidungsbaum auf handhabbare Größe reduziert, wird wiederum lineares Programmieren für Suche nach einer optimalen Strategie verwendet. Zur Reduktion des Entscheidungsbaums wird der Algorithmus *GameShrink* benutzt. Durch Angabe von einem Abstraktionsgrad und Zusatzinformationen über Beziehungen der Knoten, liefert *GameShrink* ein vereinfachtes Spiel zurück. Da die Autoren nicht auf Expertenwissen setzen, verbessert sich der Algorithmus automatisch mit dem Stand der Technik. Tiefergehende Abstraktionen können berechnet werden, während bei anderen Spielprogrammen durch neue Erkenntnisse in der Forschung auf Poker-Wissen basierende Algorithmen komplett verworfen werden müssen. Im Gegensatz zu *psOpti* berechnet *GS1* Strategien sowohl off- als auch online. Dies bringt neue Herausforderungen mit sich.

6. Der iterative Aufbau von *GS1*

Dieser Abschnitt beschäfftigt sich mit den einzelnen Schritten in denen *GS1* berechnet wird. Wie bei den vorherigen Bots werden die Runden getrennt behandelt. Das Postflop-Modell wird jedoch nicht offline auf der Basis von Preflop-Wissen generiert. Vielmehr wartet *PS1* ab, wie sich das Spiel bis zum Turn entwickelt und berechnet daraufhin von seinem Standpunkt die Strategie für den weiteren Verlauf.

6.1 Strategie für preflop und flop

Um eine Abstraktion auf den ersten zwei Runden mit *GameShrink* durchzuführen, müssen zunächst einige Vorberechnungen stattfinden. Ohne die Auslagerung von Ergebnissen wäre die Rechenzeit nicht tragbar. *GameShrink* ist in der Lage einen Spielbaum nur durch Angabe eines Grenzwerts auf strategische Gleichheit zu vereinfachen. Doch dauert dies häufig sehr lange, wenn das Programm selbst entscheiden muss, wie ähnlich sich Instanzen sind. Damit die *GameShrink* auf Hold'em anwendbar ist wird ein Maß für Ähnlichkeit definiert und eine Datenbank angelegt mit der *GameShrink* effizient einzelne Hände vergleichen kann.

Zu jeder zwei-Karten Kombination lassen sich die mittlere Anzahl an Erfolgen und Misserfolgen bestimmen. Zwei Hände werden ähnlich genannt, wenn der Abstand der Erfolge zusammen mit dem Abstand der Misserfolge niedrig ist. Diese Relation ähnelt dem Bucketing des ersten Papers. Damit *GameShrink* die Relationen schnell vergleichen kann, werden die Werte für den mittleren Erfolg und Misserfolg für alle preflop-flop-Kombination in eine Datei ausgelagert. Diese enthält $\binom{52}{2}\binom{50}{3} = 25\ 989\ 600$ Einträge. Die Berechnung dieser Datei benötigt dabei eine weitere Datenbank mit $\binom{52}{7} = 133\ 784\ 560$ Werten, die lediglich den Rang aller Hände speichert.

Von dort an, da *GameShrink* auf den Datenbanken in nur vier Stunden neue Abstraktionen berechnen kann, haben die Autoren einige reduzierte Spielbäume evaluiert. Für das weitere Vorgehen wurde eine Vereinfachung gewählt, die 169 Klassen für das preflop und 2 465 Klassen für das flop benutzt. Die 169 Klassen für das preflop entsprechen dabei einer verlustfreien Reduktion um Farbäquivalenz der hole-Karten Kombinationen.

Da die Erwartungswerte für die lineare Programmierung notwenig sind, werden schlussendlich auch diese in einer Datenbank vorberechnet zu Verfügung gestellt. Ohne die Auslagerung wäre eine ausreichende Evaluierung unterschiedlicher Abstraktionen nicht möglich. Die Datenbank besteht aus $\frac{\binom{52}{2}\binom{50}{2}}{2}\binom{48}{3} = 14\ 047\ 378\ 800$ Einträgen, sie wird stets Stückweise geladen und konnte auf die Hälfte ihrer eigentlichen Größe reduziert werden, da der Gewinn des Gegners gleich dem eigenen Verlust ist. Erwartungswerte zweier Hände können also durch Anpassen der Vorzeichen einfach den Spielern zugewiesen werden. Mit dieser Datenbank ist es möglich mit linearem Programmieren eine optimale Strategie in rund einer Woche zu finden.

7. Strategie für turn und river

Die Suche nach einer optimalen Strategie besteht wie in den ersten zwei Runden aus einer Vereinfachung mit *GameShrink* und anschließendem linearem Programmieren. Allerdings wird Letzteres diesmal zur Spielzeit durchgeführt. Bei der Abstraktion wird an einer Stelle zum ersten Mal spezielles Wissen eingesetzt. Die Autoren erkennen, dass die Strategie von späteren Runden nie auf vorangegangene Wetten aufbaut. Es ist also nur der Spielverlauf der Karten entscheidend. Außerdem nutzen die Autoren die Farbäquivalenz der Karten aus. Eine Reduktion, die im vorherigen Paper nicht nötig war, bringt nun bei Echtzeitberechnung wertvolle, weitere Sekunden. Auf allen möglichen $\binom{52}{4}$ Flop-Resultaten, mit dem hinzugekommenen Turn, abzüglich der Farbäquivalenz werden nun Abstraktionen berechnet. Da der Grenzwert für jede Folge ein Anderer sein kann, definieren die Autoren einen gewünschten Abstraktionsgrad. Dieser wird dann mit binärer Suche über den Grenzwerts mit *GameShrink* möglichst genau angenähert. Die Abstraktionen von turn und river werden getrennt berechnet.

Damit die Strategieberechnung representativ ist, werden wieder die Eingangswahrscheinlichkeiten abgeschätzt. In diesem Paper verwenden die Autoren die Regeln von Bayes. Auf der Basis des bekannten, approximierten Verlaufs werdem die Wahrscheinlichkeiten zu jeder hole-Karten Kombination des Gegners berechnet. Es gilt: $Pr[\theta|h,s] = \frac{Pr[h|\theta,s]Pr[\theta]}{\sum_{\theta' \in \Theta} Pr[h|\theta',s]}$, wobei θ das betrachtete Hole des Gegners, Θ die Menge aller hole-Karten Paare, h die History und s die bisherige Strategie des Gegners ist. $Pr[h|\theta,s]$ lässt sich einfach aus dem beobach-

teten Spielverlauf berechnen. In jeder Runde müssen alle Wahrscheinlichkeiten berechnet werden. Daraufhin kann die Echtzeitstrategiesuche erfolgen. Ist *GS1* an der Reihe ohne ein entgültiges Ergebnis erziehlt zu haben, handelt er nach dem Derzeitigen und rechnet von diesem Punkt weiter. Findet *GS1* das Ergebnis früher, fängt er bereits mit Berechnungen für die nächste Runde an. Dabei ist zu bemerken, dass zu jeder Zeit lineares Programmieren eine Lösung bereitstellt. Diese muss aber während der Rechenzeit nicht die optimale sein.

8. Vergleich der Verfahren

Die vorgestellten Verfahren sind schnell im Spielen. Selbst die Variante, die eine optimale Strategie für das turn und river in Echtzeit sucht, braucht im Mittel nur neun Sekunden. Jedoch ist das berechnen der Datenbanken und abschätzen der Erwartungswerte sehr aufwendig. Diese betrugen Wochen und Monate auf zum Teil mehr als einem Computer.

Das zweite Verfahren stellt einen Computer-Spieler vor, der einige state-of-the-art Programme schlägt. Dies zeigt, dass Poker ohne Expertenwissen erfolgreich gespielt werden kann. Die *psOpti* Programme untersuchen verschiedene Approximation von Postflop Modellen. Dabei stellt sich raus, dass *psOpti1* überraschender Weise überragt. *psOpti2*, dessen Eingangswahrscheinlichkeiten auf einem, von einem Experten konstruiertem, Preflop Modell basiert, enthält nach Einschätzung der Entwickler taktische Fehler. *psOpti1* erziehlt selbst gegen Weltklasse menschliche Spieler gute Ergebnisse. Gegen „thecount“ scheint *psOpti1* zunächst zu Gewinnen. Die entscheidende Wende ergibt sich, als die Autorem dem Poker-Meister offenbaren, dass *psOpti1* nicht lernt. Dadurch war es ihm möglich schwächer zu spielen und dabei die Schwächen des Programms zu erforschen. Nach kurzer Zeit unterlag *psOpti1*.

Keines der vorgestellten Programme lernt. Dies scheint zunächst von der psychologischen Seite von Poker ein großer Nachteil zu sein, wird aber durch die Versuche widerlegt. Beide Verfahren untersuchen Heads Up Hold'em. Die Abstraktion für Mehrspieler Partien ist um einiges schwieriger.

Literatur

Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., & Szafron, D. (2007). Approximating game-theoretic optimal strategies for full-scale poker. *International Joint Conference on Artificial Intelligence (IJCAI)*, 661–668.

Gilpin, A., & Sandholm, T. (2006). A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation..

Wayne, K. (2007). Lineare programming lecture iii..

Opponent Modelling in RoShamBo

Timo Bozsolik TIMO.BOZ@GMX.DE
FB Informatik, TU Darmstadt

Abstract

Diese Arbeit befasst sich mit dem Einsatz von computerisierten Spielern in RoShamBo, besser bekannt als "Stein, Schere, Papier". Dieses einfache Spiel eignet sich besonders gut, um die generellen Techniken in Bezug auf die Modellierung und Analyse von Gegenspielern in einer Spielsituation aufzuzeigen und zu untersuchen. Der erste internationale RoShamBo-Programmierwettbewerb hat verschiedene Strategien für das *Opponent Modelling* hervorgebracht, von denen einige hier erläutert werden sollen. Die meisten dieser Vorgehensweisen lassen sich auch auf andere Spiele wie zum Beispiel Poker übertragen und sind daher von generellem Interesse für die Entwicklung computerisierter Spieler.

1. Einleitung

RoShamBo oder *"Stein, Schere, Papier"* ist ein weit verbreitetes Knobelspiel, bei dem zwei Spieler gegeneinander antreten und mit Handzeichen jeweils gleichzeitig eines der Symbole Stein, Schere oder Papier formen. Je nachdem, welches Symbol der eine und welches der andere Spieler wählt, gewinnt einer der beiden die aktuelle Runde; das gesamte Spiel gewinnt dann derjenige Spieler, der die meisten Runden für sich entscheiden konnte. Die genauen Regeln können Tabelle 1 entnommen werden. Wählen beide Spieler das gleiche Symbol, ergibt sich ein Unentschieden; wählt etwa der erste Spieler Schere und der zweite Papier, so gewinnt ersterer ("Schere schneidet Papier").

	Stein	Schere	Papier
Stein	—	Stein	Papier
Schere	Stein	—	Schere
Papier	Papier	Schere	—

Tabelle 1: Regeln bei RoShamBo

Da jedes Symbol gegen je ein anderes gewinnt und gegen eines verliert (beispielweise gewinnt Stein gegen Schere, verliert jedoch gegen Papier), sind alle Symbole spieltechnisch gleichwertig und führen in jeder Runde zunächst zu den gleichen Gewinnchancen. Ist jedoch das Verhalten des Gegners beim RoShamBo bekannt oder vorhersehbar (Gegner spielt immer Stein), so kann immer und eindeutig die optimale Gegenmaßnahme ergriffen werden (spiele Papier), welche garantiert zu Erfolg führt. Auf diese Weise entwickelt das auf den ersten Blick trivial und rein zufällig wirkende Spiel eine unerwartete strategische Komplexität und stellt so ein interessantes Forschungsfeld für die Techniken des *Opponent Modelling* dar.

Im Folgenden werden in Kapitel 2 zuerst einige theoretische Überlegungen zur optimalen Strategie in RoShamBo vorgenommen und darauf in Kapitel 3 einige einfache,

aber sub-optimale Vorgehensweisen in Form von Standard-Bots vorgestellt und entsprechende Gegenmaßnahmen analysiert. Hierauf wird im Abschnitt 4 mit *Iocaine Powder* ein komplexer Computerspieler und der Gewinner des ersten internationalen RoShamBo-Programmierwettbewerbs genauer untersucht. Zum Schluss wird in Kapitel 5 auf Grundlage der gewonnenen Informationen die Frage geklärt, inwieweit sich die Ergebnisse des Opponent Modelling in RoShamBo sich auf das Pokerspiel beziehen lassen.

2. Theoretische Betrachtung

Aufgrund der theoretischen Gleichwertigkeit aller Symbole ist die optimale Strategie beim RoShamBo die, schlicht in jeder Runde eine zufällige Entscheidung zu treffen. Gegen diese Strategie gibt es keinerlei erfolgsversprechende Gegenmaßnahmen, da das Verhalten bei einem guten Zufallsgenerator komplett indeterministisch ist und vom Gegner nicht eingeschätzt werden kann. Damit unterliegt der Ausgang einer Runde ebenfalls dem Zufall. Eine genügend große Anzahl von Runden vorausgesetzt, verliert und gewinnt ein Spieler, der diese Strategie anwendet, in etwa gleicher Anzahl, egal welches Spiel und welche Analysen der Gegner verfolgt.

Weiß man jedoch genau, nach welcher Taktik der Gegenüber handelt, so ist es leicht, entsprechend darauf zu reagieren und man wird in jeder Runde gewinnen. Damit ist das Endergebnis in diesem Fall also entscheidend besser als das der optimalen Strategie. Dies zeigt, dass die zufällige Auswahl zwar global gesehen nicht geschlagen werden kann. Jedoch kann diese wiederum selbst auch einfache Strategien nicht mit einer signifikanten Differenz besiegen. Betrachtet man beispielsweise einen Computerspieler, der diese zufällige Strategie verfolgt, und lässt diesen in 1000 Runden gegen einen Bot antreten, welcher stets Stein spielt, so wird ersterer mit hoher Wahrscheinlichkeit in etwa 333 Runden gewinnen, in 333 Runden verlieren und 333 Runden werden unentschieden ausgehen. Die Differenz wird also mit einer großen Wahrscheinlichkeit betragsmäßig nahe bei 0 liegen. Ein Spieler, welcher die Tatktik des Stein-Bots von vorneherein erkennt, wird alle 1000 Runden gewinnen und so im Vergleich besser abschneiden.

Analog zu den obigen Ausführungen hat der erste internationale RoShamBo - Programmierwettbewerb gezeigt, dass es beim Entwurf eines Computerspielers nicht darauf ankommt, dass dieser gegen alle Spieler die gleichen Gewinnchancen hat. Vielmehr gilt es, die anderen sub-optimalen, weniger guten Computerspieler möglichst hoch zu schlagen (siehe [1]). Idealerweise muss das System des Gegenspielers also erkannt oder zumindest angenähert werden, wobei das eigene Verhalten möglichst nicht preisgegeben werden darf, da der Gegner das gleiche Ziel verfolgt. Um einen Eindruck in die Vielfalt der Gegnerstrategien zu geben und damit deren Erkennung zu ermöglichen, soll im Folgenden ein Überblick über verschiedene Basisstrategien gegeben und mögliche Gegenmaßnahmen diskutiert werden.

3. Die Dummy-Bots

Um mögliche Gegner zu simulieren und den Teilnehmern eine grundsätzliche Vielfalt an zu erkennenden Strategien vorzugeben, wurden beim ersten internationalen RoShamBo-Programmierwettbewerb 11 sogenannte *Dummy-Bots*, also Computerspieler mit relativ einfachen Strategien, eingesetzt (siehe [1]). Diese waren von Anfang an nicht darauf ausgelegt,

die anderen Teilnehmer zu schlagen und das Turnier zu gewinnen, sondern sollten vielmehr grundsätzliche Verhaltensweisen simulieren, die von den anderen Bots erkannt und deren Schwachstellen ausgenutzt werden sollten. Daher agieren diese Bots meist unabhängig von ihrem aktuellen Gegner nach mehr oder weniger vorbestimmten Mustern, betreiben also selbst keinerlei *Opponent Modelling*. Trotzdem ist deren Analyse unabdinglich, da diese verschiedene Spielweisen erkannt werden und entsprechende Gegenstrategien entwickelt werden müssen. Aus diesem Grund sollen hier einige dieser *Dummy-Bots* auf deren grundsätzliche Strategie hin analysiert und auf deren Erkennung eingegangen werden.

Die optimale Strategie in Form eines **Random-Bots**, der jede Aktion völlig zufällig wählt, wurde schon im letzten Abschnitt diskutiert. Eine Vorhersage des nächsten Spielzuges ist hier - ein guter Zufallsgenerator vorausgesetzt - grundsätzlich selbst mit maximaler Informationsmenge nicht möglich, weshalb jede mögliche Gegenstrategie im Durchschnitt ein ähnliches Ergebnis (ein Unentschieden) produziert. Wie erwartet konnte dieser Bot im Rahmen des ausgerichteten Turniers einen mittleren Platz erreichen, da er gegen keinen der anderen Teilnehmer weder mit einer größeren Differenz gewann noch verlor. Eine Alternative zu diesem Bot sind pseudo-zufällige Strategien, welche zwar nach einem festen und deterministischen Muster handeln, welches jedoch nicht offensichtlich ist. Beispiele dafür sind der **PI-Bot**, welcher seine Aktionen anhand der Ziffern der Zahl PI (modulo 3) bestimmt, oder der **Text-Bot**, welcher dafür die Buchstaben eines beliebigen Textes benutzt. Letzterer konnte aufgrund der in der menschlichen Sprache auftretenden Muster jedoch leicht von besseren Pattern-Detektoren geschlagen werden. Diese beiden Bots sind in ihren Aktionen vollständig vorhersehbar und damit leicht zu schlagen, jedoch nur, wenn man die zugrunde liegende Zahl bzw. den Text kennt. Daher können effektive Gegenstrategien auch nur auf diese Daten ausgelegt sein und sind im Endeffekt nicht tragbar.

Auf den ersten Blick genauso zufällig, jedoch völlig deterministisch und nach einem erkennbaren Modell arbeitend ist der **De Brujin-Bot**. Dieser wählt seine Aktionen nach einer Zeichenkette bestimmter Länge, welche nach einem festen System generiert wird. In einer solchen De Bruijn-Sequenz kommen alle Kombinationen einer bestimmten Länge aus einer bestimmen Anzahl von Zeichen genau einmal vor. Nimmt man beispielsweise die Zeichen “r” (für Stein), “s” (für Schere) und “p” (für Papier) als Grundlage, so enthält die Sequenz “rrpsppprss” jede Sub-Sequenz der Länge 2 über diesem Alphabet genau einmal (also “rr”, “rp” etc.). Dieser Bot schnitt erstaunlicherweise als einer der besten im Feld ab, was daran liegen könnte, dass die Aktionen hier gleichmäßiger verteilt sind als beim **Random-Bot** und dass kein Muster zweimal vorkommt, was jede Erkennung von Patterns von vorneherein unnütz und sogar kontraproduktiv werden lässt.

Während die bisher vorgestellten Bots bei der Wahl ihres nächsten Spielzuges keinerlei Informationen über den bisherigen Spielverlauf zugrunde legen, richten sich die folgenden Bots nach den letzten selbst getroffenen Entscheidungen. Der **Switch-a-lot-Bot** etwa wählt mit einer hohen Wahrscheinlichkeit immer ein Symbol, welches unterschiedlich zum letzten Zug ist. Ist diese Strategie ersteinmal erkannt, gestalten sich die Abwehrmaßnahmen denkbar einfach. War der letzte Spielzug dieses Bots etwa Schere, so wird der nächste entweder Stein oder Papier sein. Die Reaktion mit Papier wird in der Hälfte der Fälle ein Unentschieden ergeben und in der anderen Hälfte zum Gewinn der Runde führen, jedoch nie mit einem Verlust. Ein ähnlich vorhersehbares Verhalten liegt dem **Flat-Bot** zugrunde, welcher seine Aktionen zwar auch zufällig wählt, jedoch durch Analyse seiner bisherigen Spielzüge

versucht, die Anzahl der jeweils gespielten Symbole, in etwa gleich zu halten. So wählt dieser Bot in 80% der Fälle immer die bisher seltenste Aktion. Ist ein Gegner in der Lage, dieses Verhalten zu durchschauen, so kann die optimale Reaktion leicht berechnet werden. Durch ein solches Verhalten werden wie auch beim **Switch-a-lot-Bot** in gewisser Weise menschliche (meist unbewusste) Strategien simuliert, da auch der Mensch lange Sequenzen des gleichen Symbols eher vermeidet und so die aktuelle Entscheidung von den vorherigen abhängig macht.

Der komplexeste und einzige auf Erfolg ausgelegte Dummy-Bot im ausgerichteten Turnier, war der **Anti-rotn-Bot**. Dieser analysiert die gegnerischen Spielzüge in Form von Rotationen (aufeinanderfolgende gleiche Symbole werden als 0, verschiedene als +1 bzw. -1 gewertet) und trifft seine Entscheidungen auf Basis dieser Historie. Ähnlich wie bei der optimalen Gegenstrategie beim **Switch-a-lot-Bot** richtet sich dann der nächste Spielzug nicht nach dem wahrscheinlichsten nächsten Spielzug des Gegners, sondern nach dem mit der geringsten Wahrscheinlichkeit. Wird beispielsweise Stein als unwahrscheinlichste nächste Aktion des Kontrahenten angenommen, so wird die Reaktion mit Schere gegen die beiden vermutlich häufiger auftretenden Symbole Schere und Papier nie verlieren. Zusätzlich werden hier die eigenen Verluste mitgezählt und ab einem Gesamtverlust von 4% nur noch nach zufälliger Strategie gespielt, um hohe Niederlagen zu vermeiden.

Die in diesem Kapitel besprochenen *Dummy-Bots* verfolgten alle **eine** einfache, relativ leicht zu erkennende Spielweise. Ein großer Teil der eigentlich am Turnier teilnehmenden Computerspieler bediente sich dem Konzept der *gemischten Strategien*. In diesem Fall werden verschiedene einfache Spielweisen parallel implementiert, in einer Vorentscheidung die nächste Aktion jeder einzelnen berechnet und dann diejenige mit dem größten Erfolg in der Vergangenheit endgültig verwendet. Zwar kann sich dieser Ansatz leicht an eine Vielzahl an Gegnern anpassen, jedoch richtet sich die gesamte Qualität nach der der besten Teil-Strategie, weshalb sich im Turnier allgemeinere Verfahren (siehe z.B. *Iocaine Powder* in Kapitel 4) eher durchgesetzt haben. Eine weitere Unterteilung der Bots kann man in Bezug auf die Analyse der bisherigen Spielrunden treffen (siehe [4]). Hier existieren verschiedene statistische Methoden (wie etwa beim **Flat-Bot**) sowie Formen der direkten Analyse der Historie, wie etwa die Erkennung von Mustern. Damit die eigene Strategie außerdem verschleiert und vor hohen Verlusten gegen starke Gegner bewahrt wird, wurde in vielen Ansätzen ein gewisses Rauschen in Form von zufälligen Aktionen eingeführt. Viele Bots weichen sogar erst von der randomisierten Spielweise ab, wenn diese eine signifikante Schwäche beim Gegner erkennen. Eine kurze Vorstellung der besten Computerspieler im Turnier und die kompletten Ergebnisse finden sich in [3].

4. Iocaine Powder

Mit dem Gewinner des ersten internationalen RoShamBo-Programmierwettbewerbs, *Iocaine Powder* (siehe [2]), soll nun ein komplexer, sehr spielstarker Bot detailliert untersucht werden. Dessen Strategie ist sehr generischer Natur und zeigt, dass ein guter Prognose-Algorithmus auch auf viele andere Probleme angewendet werden kann, wie etwa Datenkompression, eMail-Klassifizierung oder Data Mining.

4.1 Metastrategien

Jeder direkte Algorithmus zur Vorhersage der Aktionen eines Spielers hat eine gravierende Schwäche: der Gegner kann die eigene Vorgehensweise erkannt haben und selbst entsprechend handeln bzw. agieren. Um diesem Nachteil entgegenzuwirken, werden in *Iocaine Powder* sogenannte Meta-Strategien eingesetzt. Diese sechs übergeordneten Taktiken erweitern dann ein gegebenes Prognose-Modul, so dass auf Basis von dessen Entscheidung eine neue getroffen wird, die wieder weitere Faktoren berücksichtigt. Der entscheidende Aspekt ist hier die Frage, wie man dagegen vorgehen kann, dass ein Gegner die eigene Spielweise durchblickt. Dazu wird im Folgenden von einem Vorhersage-Modul **P** ausgegangen, dass die nächste Aktion des Gegners unabhängig voraussagt.

Die einfachste mögliche Meta-Strategie **P.0** verkörpert die unmittelbare Anwendung des Vorhersagers und versucht direkt, das von **P** bestimmte Symbol zu schlagen. Beispielsweise würde **P.0** also Papier liefern, um einen von **P** prognostizierten Stein des Gegners zu besiegen. Einen Schritt weiter geht **P.1**, die Abwehr des sogenannten *second-guessing*, bei der davon ausgegangen wird, dass der Gegner die eigene Vorhersage kennt und entsprechend handelt. Würde **P** also Stein liefern und unser Gegner die Funktionsweise von **P** kennen, würde er damit rechnen, dass wir Papier spielen. Daher wird der Gegner in dieser Situation selbst Schere spielen, um dieses zu schlagen. **P.1** liefert nun Stein, um wiederum die Schere des Gegners zu egalisieren. Wieder eine Stufe weiter geht **P.2**, bei der davon ausgegangen wird, dass der Gegner das eigene Verfahren von **P.1** kennt, wiederum einen Schritt weiter denkt und dieses versucht abzuwehren. Da **P.1** als nächste Aktion Stein empfohlen hat und man annimmt, dass der Gegner auch **P.1** durchblickt hat und als Reaktion entsprechend Papier spielen würde, begegnet man diesem mittels **P.2** mit Schere. Diese Kette könnte nun unendlich fortgesetzt werden, in dem man sich weiter die Frage stellt, was ist, wenn der Gegner die jeweils vorherige Strategie durchschaut hat und dies wiederum abwehren will. Jedoch empfiehlt **P.0** bereits das Spiel von Papier, **P.1** liefert Stein und nach **P.2** sollte Schere gespielt werden, so dass alle möglichen Aktionen bereits abgedeckt sind und weitere Schritte wieder nur die gleichen Spielzüge liefern würden (es gilt **P.3**=**P.0**, **P.4**=**P.1** usw.).

Umgekehrt zur bisherigen Annahme, nämlich dass die Aktionen des Gegners von **P** bestimmt werden, gehen die folgenden drei Strategien davon aus, dass der Gegner selbst **P** benutzt. Daher wird hier auch nicht **P** zur Vorhersage

herangezogen, sondern **P'**, bei welchem die Position des eigenen Bots und des Gegners vertauscht ist. Damit liefert **P'** quasi eine Vorhersage des eigenen Spiels. Lautet diese beispielsweise Stein, so wird angenommen, dass der Gegner Papier spielt, weswegen **P'.0** Schere suggeriert. **P'.1** und **P'.2** sind dann analog zu **P'.1** und **P'.2** wieder genau die Taktiken, die davon ausgehen, dass der Gegner das eigene Vorgehen von **P'.0** und **P'.1** kennt, entsprechend einen Schritt weiter gehen und darauf reagieren.

Die diskutierten Ansätze stellen eine wirkungsvolle Abwehr gegen intelligente Gegenspieler dar, jedoch kennt man die Taktik des Gegners und vor allem dessen Level des "Weiterdenkens" nicht. Damit kann jede dieser Vorgehensweisen zu einem gegebenen Vorhersage-Modul die beste sein, jedoch ist a priori nicht bekannt, welche. Und da es sechs verschiedene Strategien gibt, wovon jeweils zwei Stein, zwei Schere und zwei Papier empfehlen, kann die Wahl der nächsten Aktion auch nicht in irgendeiner Weise eingeschränkt werden. Es wird jedoch angenommen, dass die Aktionen des Gegners konsistent sind und dieser sein grundsätzliches

Verhalten nicht (oder nur sehr langsam) verändert. Daher kann man die zurückliegenden Spielzüge benutzen, um zu entscheiden, welche dieser sechs Strategien bisher die beste war und diese auch im nächsten Zug einsetzen. Aufgrund der im Nachhinein ja bekannten Erfolgsquote jeder Strategie wird also diejenige mit der besten Performance ausgewählt und angewandt.

4.2 Algorithmen zur Vorhersage

Jede noch so durchdachte Meta-Strategie ist nutzlos ohne einen Basis-Algorithmus zur Vorhersage des nächsten Zuges des Gegners, auf deren Grundlage dann die weitere Entscheidung abgeleitet werden kann. Hier benutzt *Iocaine Powder* drei verschiedene Algorithmen, wobei die aktuell anzuwendende Variante wieder durch den Vergleich der Güte in den vorangegangenen Spielzügen bestimmt wird.

Eine einfache statistische Methode, die nächste Aktion des Gegners vorauszusehen, stellt die **Frequenzanalyse** dar. Diese findet schlicht das in der Vergangenheit am häufigsten eingesetzte Symbol des Gegners und sagt dieses auch für den nächsten Zug voraus. Damit kann zum einen das Verhalten extrem einfacher Bots vorhergesagt werden, auf der anderen Seite werden in Kombination mit den bereits diskutierten Meta-Strategien Bots, die selbst eine Frequenzanalyse einsetzen, leicht geschlagen.

Im Gegensatz dazu analysiert die Methode des **History-Matchings** direkt die bereits gespielten Runden und sucht dort nach auftretenden Mustern. Bestanden die letzten Runden beispielweise aus Stein gegen Schere, Stein gegen Papier und Papier gegen Schere, so sucht dieser Algorithmus nach weiteren Vorkommen dieser Sequenz in der Vergangenheit und sagt das Symbol voraus, welches dem am häufigsten folgenden nächsten Zug des Gegners entspricht. Dabei werden Vorkommen als aussagekräftiger bewertet, je länger die übereinstimmende Abfolge und je öfter diese bereits aufgetreten ist.

Eine Absicherung gegen die erfolgreiche Analyse und Vorhersage des eigenen Spiels durch einen stärkeren Gegner bildet das **zufällige Raten** des nächsten Zuges. Falls ein Gegenspieler in der Lage ist, die oberen beiden Vorhersager und alle Variationen durch die Meta-Strategien abzuwehren, erreichen diese für die vergangenen Runden eine schlechte Erfolgsquote. Als Konsequenz ist die Quote dieses Algorithmus die beste und gegen besonders gute Gegner wird einfach nach dem Zufallsprinzip gespielt. Damit wird *Iocaine Powder* auch gegen wesentlich stärkere Gegner nicht besonders hoch verlieren oder sogar gewinnen, da es gegen diese Taktik keine wirksame Gegenstrategie gibt.

4.3 Zusätzliche Erweiterungen

Die Kombinationen aus drei von Grund auf verschiedenen Algorithmen zur Vorhersage des Gegnerverhaltens in Kombination mit den sechs Meta-Strategien, die die Verwendung von ähnlichen Möglichkeiten der Vorhersage bei Gegnern kompensieren sollen, hat sich insgesamt als gute Gesamt-Strategie herausgestellt. Die Annahme, dass der Gegenspieler sein Verhalten im Zeitverlauf nicht ändert, kann jedoch offensichtlich nicht immer getroffen werden. Daher wurde eine weitere Ebene von Meta-Meta-Strategien eingeführt, welche den Bewertungshorizont zur Auswahl der eigentlichen Meta-Strategie (z.B. **P.0** oder **P.1** etc.) ändert. Dazu wird wiederum die Erfolgsquote jeder Entscheidung basierend auf den letzten 1000, 100, 10 etc. Zügen verglichen und der Zeitraum, der im gesamten Spiel am

besten war, ausgewählt. Damit können auch Gegner, die ihr Spiel in einer bestimmten Regelmäßigkeit ändern, besser eingeschätzt werden. Weiterhin werden auch die Zeithorizonte der Vorhersage-Algorithmen variiert, verglichen und die beste Variante zur Bestimmung des nächsten Zuges gewählt.

Insgesamt hat sich herausgestellt, dass diese generische Vorgehensweise aus den drei Ebenen der Vorhersage-Algorithmen, der Meta-Strategien und der Variation der Bewertungszeiträume als sehr generische Variante den hoch spezialisierten Algorithmen überlegen ist. Dabei wurden in *Iocaine Powder* keinerlei Mechanismen zur Verschleierung der eigenen Spielweise eingebaut, so dass davon auszugehen ist, dass ein Bot, der speziell darauf programmiert ist, *Iocaine Powder* zu schlagen, darin auch erfolgreich sein wird. Im Hinblick auf die Vielzahl der möglichen Strategien, die ein Gegner anwenden kann, kann dieser Ansatz jedoch als sehr robust und effektiv bewertet werden.

5. Schlussfolgerungen

Zusammenfassend kann man sagen, dass sich hinter dem zunächst trivial erscheinenden Spiel RoShamBo ein auf den zweiten Blick komplexes und vielschichtiges Szenario verbirgt. Dieses Spiel kann als reine Anwendung des Opponent-Modelling angesehen werden, weshalb es für dessen Erforschung besonders nützlich sein kann. Die hier entwickelten Techniken zur Vorhersage des Gegnerverhaltens können dann direkt oder in Abwandlungen auch in anderen Spielen mit unvollkommener Information wie Poker eingesetzt werden. Natürlich kommen hier weitere Spielfaktoren hinzu und es müssen mehr Aspekte als nur der wohl wahrscheinlichste nächste Zug des Gegners beachtet werden. Beispielsweise ist dieser unwichtig, wenn man beim Poker das definitiv beste Blatt am Tisch besitzt. Da diese Situation jedoch sehr selten vorkommt, ist es von Vorteil zu wissen, was der Gegner vermutlich als nächstes tun wird, um so zum Beispiel unnötige Bluffs zu vermeiden. Außerdem weiß jeder Pokerspieler, dass man auch mit zwei Assen auf der Hand verlieren kann und daher jede verfügbare Information bei seinem Spiel beachten sollte. Grundsätzlich ist es also ein wichtiger Faktor zu wissen, wie sich ein Spieler in einer bestimmten Situation verhält, um sein eigenes Spiel darauf ausrichten zu können. Gerade in einem Kontext wie dem Heads-Up-Game spielt es auch eine große Rolle, zu ahnen was der Gegner denkt und was er denkt, was man selbst denkt etc. In diesem Fall könnten prizipiell die Überlegungen zu den Meta-Strategien von *Iocaine Powder* nützlich sein.

Literatur

[1] Billings, D. Thoughts on RoShamBo, 2000. ICGA Journal, Vol. 23, No. 1, pp. 3-8.

[2] Egnor, D. Iocaine Powder, 2000. ICGA Journal, Vol. 23, No. 1, pp. 33-35.

[3] Billings, D. The First International RoShamBo Programming Competition, 2000. ICGA Journal, Vol. 23, No. 1, pp. 42-50.

[4] Billings, D. The Second International RoShamBo Programming Competition, 2001. `http://www.cs.ualberta.ca/~darse/rsbpc.html`.

Opponent Modeling im Poker

Lars Meyer atomic_ant@freenet.de
Thomas Görge thgoerge@gmx.de
Technische Universität Darmstadt

Abstract

Eine starke Poker-KI zu entwickeln ist eine anspruchsvolle Aufgabe. Diese muss in der Lage sein, jeden ihrer Gegner einzuschätzen und ihr Spiel bestmöglich an diesen anzupassen um die maximale Gewinnrate zu erzielen. In der Einführung dieses Papers wird zunächst erläutert, warum das Modellieren des Gegners im Poker von großer Bedeutung ist. Danach werden die Ansätze beschrieben, die von den Programmen Loki bzw. Poki und Vexbot verwendet werden, um ihre Gegner zu beurteilen. Hierbei zeigen wir auch typische Probleme und Lösungsmöglichkeiten für diese auf. Abschließend werden die Ergebnisse von Experimenten vorgestellt, in denen die genannten Programme gegen andere Programme oder menschliche Gegner gespielt haben.

1. *Einführung*

Für das Spiel Schach wurden in der Vergangenheit bereits sehr starke Programme entwickelt, die die besten menschlichen Spieler schlagen können. Das Gleiche für ein Programm, das Poker spielt, zu bewerkstelligen, ist das Ziel vieler wissenschaftlicher Studien. Poker unterscheidet sich jedoch in einem wichtigen Punkt von Schach. Dem Spieler stehen nicht alle für seine Entscheidung relevanten Informationen zur Verfügung, denn man kennt das Blatt des Gegners nicht. Die optimale Entscheidung zu treffen bedeutet im Poker genau die Aktion durchzuführen, die man auch ausführen würde, wenn man die Hände der anderen Spieler kennt. Um also eine möglichst optimale Entscheidung zu treffen, muss man abschätzen, welche Karten der Gegner gerade auf der Hand halten könnte. Um dies durchzuführen müssen die Aktionen des Gegners beobachtet werden, um ein Modell seiner Spielweise zu erstellen. Das eigene Spiel hängt also sehr stark von den Strategien der anderen Spieler ab. Dies stellt einen Gegensatz zu Schach dar, denn hier ist die Spielweise des Gegners eher unbedeutend und es genügt den objektiv besten Zug auszuführen.

Da Poker eine mathematische Struktur zu Grunde liegt, existiert in der Theorie eine optimale Strategie, welche sich aus einem Nash Equilibrium ergibt. Es ist jedoch zur Zeit noch nicht möglich, eine solche Strategie mit Hilfe von Computern zu berechnen. Es gibt Ansätze, in denen eine Annäherung an diese optimale Strategie verwendet wird, um Texas Hold'em mit zwei Spielern zu spielen. Da dies jedoch nur eine Annäherung ist, können die Fehler in dieser durch starke Spieler entdeckt und ausgenutzt werden.

Selbst wenn eine optimale Strategie gefunden wird, reicht es nicht diese anzuwenden. In einem Nash Equilibrium hat keiner der Spieler den Anreiz seine Strategie zu wechseln, da dies nur zu einem schlechteren Ergebnis führen kann. Es wird also das minimale Ergebnis des Spielers maximiert, der eine solche Strategie anwendet. Die Equilibrium-Strategie für das Spiel Stein-Schere-Papier wäre zum Beispiel, rein zufällig eine der drei Möglichkeiten zu

wählen. Ein Spieler, der diese Strategie anwendet, kann auf lange Sicht niemals verlieren. Angenommen der Gegner wäre aber sehr schwach und würde immer nur die Möglichkeit Papier wählen. Gegen einen solchen Spieler würde man mit der Equilibrium-Strategie nicht gewinnen. Beobachtet man seinen Gegner jedoch, sieht man, dass er die ersten 10 mal nur Papier genommen hat und nimmt daher die nächsten Male immer Stein, um seine Schwäche auszunutzen.

Die Equilibrium-Strategie ist also eine defensive Strategie, die perfekt spielende Gegner annimmt. Dies ist aber beim Poker nicht der Fall und daher ist es einer der wichtigsten Aspekte, seinen Gegner zu beobachten und einzuschätzen um seine Schwächen schließlich auszunutzen. Eine Beobachtung könnte zum Beispiel die Häufigkeit sein, mit der ein Spieler vor dem Flop erhöht. Angenommen ein Spieler erhöht in 30% der Fälle vor dem Flop. Dann kann man davon ausgehen, dass er im Schnitt eine Hand hat, die zwischen den besten 15% und den besten 16% aller Hände liegt. Hält man selber eine Hand, die zu den 10 % der stärksten Hände gehört, kann man also nochmals erhöhen. Gegen einen Spieler der nur in 10 % der Fälle vor dem Flop erhöht, würde man die selbe Hand eher folden.

2. Einschätzen der eigenen Karten

Ein guter Poker Spieler muss die Situation, in der er sich befindet, sehr gut einschätzen können. Hierbei sollte er sich zunächst ein Bild über die Stärke sowie das Potential der eigenen Hand machen. Hiernach muss der Gegner betrachtet werden, um abhängig von dessen Spielweise eine möglichst optimale Aktion zu wählen. Im folgenden Abschnitt wird zunächst aufgezeigt, wie die Stärke sowie das Potential der eigenen Hand berechnet werden kann. Anschließend werden Verfahren beschrieben, um ein Modell des jeweiligen Gegners zu gewinnen und so das Spiel optimal auf diesen einzustellen. Hierbei werden die Ansätze der Bots Loki(Billings, Papp, Schaeffer, & Szafron, 1998)/Poki(Davidson, Billings, Schaeffer, & Szafron, 2000) sowie Vexbot(Davidson, Darse Billings, & Holte, 2004) verdeutlicht. Loki/Poki sind für die Poker Variante Fixed-Limit mit mehreren Mitspielern entwickelt worden, während Vexbot für das Heads-Up Spiel im Fixed-Limit entworfen wurde.

2.1 Einschätzen der eigenen Hand

Um die eigene Hand einzuschätzen müssen im wesentlichen zwei Eigenschaften dieser untersucht werden. Zum einen die momentane Stärke der Hand und zum anderen das Potential, dass die Hand sich noch verbessert.

2.1.1 Stärke der Hand

Mit der Stärke der Hand wird die Wahrscheinlichkeit bezeichnet, dass ein Spieler momentan die beste Hand hat. Die Stärke einer Hand muss in jeder Setzrunde neu bewertet werden. Angenommen man hat die Karten Ad [1] und Qc und der Flop wäre 3h-4c-Jh. Um die Stärke der eigenen Hand zu berechnen, kann man alle möglichen Hände, die der Gegner haben könnte durchgehen und zählen, wie viele davon besser, schlechter oder genauso gut sind wie die eigenen Karten. Der Gegner kann $\binom{47}{2}$= 1081 Kartenkombinationen auf der Hand halten. Hiervon sind 444 besser, 9 gleich gut und 628 schlechter als die eigene Hand.

1. Die Buchstaben d,h,s und c stehen für die Farbe der Karte (d = Karo, h = Herz, s = Pik, c = Kreuz).

Gegen zwei zufällige Karten ergibt sich also eine Chance von 58,5% für die eigene Hand besser zu sein. Diese Wahrscheinlichkeit bezieht sich auf das Spiel gegen einen Gegner. Um die Wahrscheinlichkeit gegen mehrere Gegner zu berechnen, kann man die gegen einen Spieler bestimmte Wahrscheinlichkeit mit der Anzahl der Gegner potenzieren. Gegen 5 Gegner würde sich also eine Handstärke von $0,585^5 = 0,069$ ergeben. Die Chance hier die momentan besten Karten auf der Hand zu halten wäre also gerade mal 6,9%.

2.1.2 Potential der Hand

Die momentane Stärke der Hand reicht nicht aus, um ihre Qualität zu beurteilen. Angenommen die eigenen Karten sind 7h-8h und der Flop wäre Js-6h-5h. Hier gibt es viele Karten, die noch auf dem Turn oder dem River kommen können und welche die eigene Hand erheblich verbessern (zur Straße oder zum Flush). Obwohl man am Flop wahrscheinlich hinten liegt, gewinnt man zu 65,1% gegen zwei zufällige Karten des Gegners. Daher berechnet man die Wahrscheinlichkeit, dass sich eine Hand, die momentan hinten liegt, noch zur besten Hand verbessert (P_{pot}). Mit Hilfe der zwei gesammelten Informationen (Handstärke und Potential der Hand) kann man die Effektive Handstärke (EHS) berechnen. Diese ist die Summe der Wahrscheinlichkeit, dass man momentan vorne liegt und der Wahrscheinlichkeit, dass sich die eigenen Karten noch zur stärksten Hand verbessern.

$$EHS = HS_n + (1 - HS_n) * P_{pot}$$

3. Einschätzen des Gegners

Für die Setzstrategie können jetzt die EHS und die Pot odds in Betracht gezogen werden. Jedoch ist die Annahme falsch, dass alle Karten, die der Gegner halten kann, gleich wahrscheinlich sind. Im obigen Beispiel (eigene Karten: Ad-Qc, Flop: 3h-4c-Jh) wurde beispielsweise gesagt, dass die Handstärke gegen einen Gegner bei 0,585 liegt. Tatsächlich wird man jedoch weitaus öfter als in 58,5% der Fälle vorne liegen, da die meisten Hände, die Ad-Qc hier schlagen, höchstwahrscheinlich vor dem Flop weggeworfen worden wären, da sie eigentlich sehr schwach sind. Das Ziel ist es also, dem Gegner eine Menge von möglichen Karten zuzuweisen, die er auf der Hand halten könnte, um dann seine Chancen zu berechnen. Um diese möglichst effizient bestimmen zu können, müssen die Informationen, die man während des Spiels gegen ihn gewonnen hat, in Betracht gezogen werden.

3.1 Der Ansatz in Loki

Loki weist jeder möglichen Kartenkombination eines Gegners ein Gewicht zu, welches beschreiben soll, wie wahrscheinlich es ist, dass er diese Karten auf der Hand hält. Hierbei wird jedem Gegner ein eigenes Set an Gewichten zugewiesen, da sich die Spieler sehr stark in Bezug auf ihre Strategie unterscheiden können. Desweiteren werden die Gewichte durch die Aktionen des Gegners beeinflusst. Wenn ein Spieler beispielsweise erhöht, gibt dies Aufschluss über die Stärke seiner Hand, und die Hände, die zu diesem Zeitpunkt sehr stark sind, müssen höher gewichtet werden.

Zum einen müssen also die initialen Gewichte für die Karten jedes Gegners bestimmt werden, und zum anderen müssen diese Gewichte im Laufe einer Hand aufgrund der Aktio-

nen des Gegners angepasst werden. Im Folgenden werden die Ansätze der Entwickler von Loki beschrieben, um diese Probleme zu lösen.

3.1.1 Berechnung der initialen Gewichte

Die Evaluierung der eigenen Karten vor dem Flop ist im Poker ein maßgebender Faktor, um erfolgreich zu spielen. Es gibt $\binom{52}{2} = 1326$ unterschiedliche Startkombinationen, die sich in 169 verschiedene Handtypen unterteilen lassen. Für jede dieser Handtypen wurden 1.000.000 Spiele gegen jeweils 9 zufällige Hände durchgeführt. Aus den Ergebnissen konnte dann die sogenannte *income rate* für jede Hand berechnet werden. Diese entspricht der durchschnittlichen Anzahl von Einsätzen, die mit dieser Hand gewonnen werden. Die höchste *income rate* hat beispielweise ein Paar Asse, während eine 7 und eine 2 in unterschiedlichen Farben die geringste *income rate* hat.

Mit Hilfe des Wissens über die *income rate* der Karten, sowie über die Beobachtung der Häufigkeit, mit der ein Gegner vor dem Flop wegwirft, mitgeht oder erhöht, kann eine sinnvolle Abschätzung der Gewichte der gegnerischen Hände getroffen werden.

Angenommen ein Gegner würde mit 30% aller Hände vor dem Flop mitgehen. Das bedeutet, dass die Hände, mit denen er mitgeht, im Durchschnitt eine *income rate* von 200 haben. Diesen Wert nennt man Median einer Hand und er wird mit μ bezeichnet. Desweiteren nimmt man eine bestimmte Varianz σ an. In diesem Beispiel soll sich durch die Varianz eine Schwankung der income rate der vom Gegner gespielten Hände um 100 ergeben. Allen Händen deren *income rate* größer als 200 + 100 = 300 ist wird das Gewicht von 1.0 zugeweisen, da sie sehr wahrscheinlich sind. Im Gegensatz dazu wird allen Händen deren *income rate* kleiner als 200 - 100 ist das Gewicht 0.01 zugewiesen, da es sehr unwahrscheinlich ist, dass der Gegner mit einer solch schlechten Starthand mitgeht. Die Gewichte der Hände mit einer *income rate* zwischen 100 und 300 werden linear interpoliert. Die Median-Hand erhält hierbei das Gewicht 0.5.

Es gibt viel Potential, diesen Ansatz zu verbessern, da spezielle Verhaltensweisen eines Spielers nicht berücksichtigt werden. Vielleicht erhöht ein Spieler öfter gegen Gegner, die von ihm als schlecht eingestuft werden. Erhöht er dann gegen einen starken Spieler, hat er höchstwahrscheinlich eine sehr starke Hand. Da das Programm aber nur die Durchschnittswerte betrachtet, wird es die Hand des Spielers schlechter einschätzen als sie wirklich ist.

3.1.2 Anpassen der Gewichte

Die Aktionen des Gegners, während eine Hand gespielt wird, führen dazu, dass die Gewichte seiner Hände angepasst werden. Diese werden unterteilt in die Aktion, die er durchführt (Fold, Check/Call, Bet/Raise), wie viele Bets er zahlen müsste um mitzugehen (0 Bets, 1 Bet oder > 1 Bet) und wann die Aktion ausgeführt wird (Pre-Flop, Flop, Turn, River). Aktionen können also in 36 verschiedene Kategorien unterteilt werden. Für jede Kategorie wird die Häufigkeit bestimmt, mit der der Gegner die entsprechende Aktion ausführt. Diese Werte werden dafür verwendet, um die Anpassung der Gewichte vorzunehmen.

Beispielsweise könnte man durch die Analyse dieser Häufigkeiten herausfinden, dass ein Gegner eine effektive Handstärke (EHS) von durchschnittlich 0.6 braucht, um am Flop nach einer Erhöhung mitzugehen. Unter Einbeziehung der Varianz kann die untere Grenze 0.4

Hand	Weight	EHS	Rwt	Nwt
Jd-4h	0.01	0.99	1.00	0.01
Ac-Jc	1.00	0.94	1.00	1.00
5s-5h	0.70	0.74	0.85	0.60
Qs-Ts	0.90	0.22	0.01	0.01

Table 1: Anpassung der Gewichte beispielhafter Hände nach dem Flop 3h-4c-Jh

und die obere Grenze 0.8 sein. Jetzt werden die Gewichte aller Hände, deren EHS kleiner als 0.4 ist, mit einem sehr kleinen Faktor (z.B. 0.01) multipliziert, da der Gegner mit solchen Händen wahrscheinlich nicht mitgehen würde. Die Gewichte aller Hände mit einer EHS $>$ 0.8 werden mit dem Faktor 1 multipliziert, bleiben also gleich. Die Faktoren für die Hände mit einer EHS zwischen 0.4 und 0.8 werden durch linerare Interpolation bestimmt. Die Anpassung der Gewichte wird für jede Setzrunde durchgeführt, so dass am Ende nur noch sehr wenige Hände ein hohes Gewicht haben.

In Tabelle 1 werden die Werte, die für die Anpassung der Gewichte wichtig sind, für einige interessante Beispielhände dargestellt. Die Werte stehen für einen Gegner, der vor dem Flop mitgegangen ist und jetzt am Flop (3h-4c-Jh) setzt. Weight bezeichnet das Gewicht, das der Hand vor dem Flop zugewiesen wurde, EHS die effektive Stärke der Hand, Rwt den Faktor, mit dem das Gewicht multipliziert wird und Nwt das neue Gewicht der Hand.

Die Hand Jd-4h wäre die stärkste Kartenkombination im Beispiel. Jedoch ist sie vor dem Flop schwach und erhält daher ein Gewicht von 0.01, wenn man davon ausgeht, dass der Spieler in 30% der Fälle vor dem Flop mitgeht. Da der EHS der Hand über 0.8 liegt beträgt der Faktor zur Anpassung der Gewichte 1, so dass das neue Gewicht bei 0.01 bleibt.

Eine starke Hand, die zudem sehr wahrscheinlich ist, ist Ac-Jc. Hier beträgt das Gewicht am Ende weiterhin 1.

Qs-Ts sind vor dem Flop relativ gute Karten, was sich in einem Gewicht von 0.90 ausdrückt. Jedoch wurde auf dem Flop nichts getroffen und daher beträgt der EHS der Hand am Flop nur 0.22. Aufgrund der genannten Beobachtungen gehen wir wieder davon aus, dass die Durchschnittsstärke einer Hand 0.6 betragen muss, damit der Gegner setzt und sich durch die Varianz eine Schwankung von $\pm$0.2 ergibt. Da die EHS von Qs-Ts kleiner als 0.4 ist, wird das Gewicht der Hand mit dem Faktor 0.01 multipliziert.

Die hier aufgezeigten Verfahren bieten zwar schon gute Ansätze, man muss jedoch beachten, dass es noch viel Raum für Verbesserungen gibt. Gegner ändern beispielsweise ihre Taktik und spielen am Anfang des Spiels sehr konservativ und tight um dieses Image im späteren Verlauf des Spiels auszunutzen. Desweiteren ist die Kategoriesierung der Situationen, für die die Häufigkeiten der Aktionen gezählt werden, sehr grob. Viele relevante Informationen, wie beispielsweise die eigene Position zu den Gegnern, werden außer Acht gelassen.

3.1.3 Weiterentwicklung von Loki durch Poki

Wie in dem obigen Abschnitt bereits erwähnt wurde, werden in Loki die Häufigkeiten der Aktionen des Gegners in unterschiedlichen Situationen gemessen. Es wurde eine sehr ein-

	FOLD	CALL	RAISE	%
FOLD	13.0	0.3	0.3	13.6
CALL	0.0	58.4	3.3	61.8
RAISE	0.0	10.5	14.1	24.7
%	13.0	69.3	17.7	85.6

Table 2: Genauigkeit der Vorhersagen des Neuralen Netzes

fache Kategorisierung vorgenommen, welche 12 verschiedene Typen von Situationen unterscheidet, in denen ein Spieler jeweils wegwerfen, mitgehen oder erhöhen kann. Die Kategorien ergaben sich aus der Setzrunde (Pre-Flop, Flop, Turn, River) und der Anzahl der Bets, die eingesetzt werden müssen, um mitzugehen (0,1,2 oder mehr). Diese Kategorisierung ist sehr vereinfacht und lässt viele relevante Details außer Acht. Es ist beispielsweise sehr wichtig, wie viele Spieler sich zu dem Zeitpunkt noch im Spiel befinden. Desweiteren ist etwa das Setzen eines Spielers niedriger zu bewerten, wenn dieser in letzter Position sitzt und vorher alle Gegner gecheckt haben.

Jedoch kann nicht jeder Faktor für die Kategorisierung verwendet werden, da diese dann viel zu komplex wird. Um festzustellen welche Faktoren am relevantesten sind, wurde eine frühere Studie betrachtet, in der mit Hilfe eines *artificial neural networks* (ANN) die nächste Aktion des Gegners vorausgesagt werden sollte. Als Trainingsdaten standen mehrere hundert Hände dieses Spielers zur Verfügung. Die Eingangsknoten des ANN waren Parameter, die den Kontext der Situation beschreiben (beispielsweise ob die letzte Aktion ein Call oder ein Raise war oder ob ein Ass auf dem Board liegt). Die Ausgangsknoten gaben die Vorhersage für die Aktion des Gegners an (Fold,Bet,Raise). Tabelle 2 zeigt die Genauigkeit der Vorhersagen eines typischen ANN. Die Spalten geben hierbei an, welche Aktion vorausgesagt wurde, während die Zeilen darstellen was der Gegner wirklich gemacht hat. In diesem Fall hat der Gegner zum Beispiel in 13.6% aller Fälle gefoldet. In 13.0% aller Fälle wurde ein Fold korrekt vorhergesagt, während ein Call und ein Raise jeweils zu 0.3 % der Fälle vorausgesagt wurde, wenn der Spieler in Wirklichkeit gefoldet hat. Insgesamt konnten hier zu 85.6 % die Aktion des Spielers richtig vorhergesagt werden.

Ein solches ANN kann nicht während des Spiels eingesetzt werden, da die Berechnung zu lange dauert. Jedoch erhielt man als Ergebnis dieser Studie eine Gewichtung für die Relevanz der Eingangsparameter. Zwei Parameter stellten sich hierbei als besonders wichtig heraus. Hierbei handelte es sich um die vorangegangene Aktion des Gegners sowie um die Anzahl der Bets, die der Gegner vorher aufbringen musste um mitzugehen. Die Kategorisierung der Situationen wurde um diese Parameter erweitert.

Durch diese Erweiterungen konnte die Gewinnrate des Bots signifikant verbessert werden. Im Abschnitt **Experimente** werden wir genauer hierauf eingehen.

3.2 Vexbot

Im folgenden Abschnitt wird das Programm VEXBOT vorgestellt, dass mit Hilfe von Suchbäumen den zu erwartenden Gewinn bzw. Verlust der verschiedenen Handlungsmöglichkeiten berechnet, um eine Entscheidung zu treffen. Dabei werden nicht nur die eigene Handstärke und statischen Gewinnwahrscheinlichkeiten, sondern auch Erfahrungswerte über den Geg-

ner berücksichtigt, um eine bessere Gewinn- bzw. Verlustschätzung zu erreichen. Eine Neuerung von VEXBOT ist hier seine Fähigkeit trotz unvollständiger Informationen selbstständig gewinnmaximierende Konterstrategien zu entwickeln. VEXBOT wurde speziell für das Spiel gegen genau einen Gegner entwickelt und stellt neben dem Sieg gegen die vormals beste KI "PsOpti" auch eine echte Herausforderung für starke menschliche Spieler dar.

Die gesamten Berechnungen von VEXBOT beruhen dabei auf einer Baumdarstellung in der alle möglichen Spielverläufe von der aktuellen Situation bis zum Ende der Runde enthalten sind. Im Baum wird zwischen Knoten unterschieden bei denen das Programm selbst eine der drei möglichen Aktionen: "fold", "check/call", "bet/raise" auswählen muss, den sog. "Program decision nodes" und solchen bei denen der Kontrahent eine Entscheidung zu treffen hat: "Opponent decision nodes". Blattknoten sog. "Leaf nodes" repräsentieren das Ende einer Runde, das entweder durch den "fold" eines Spielers oder durch den "showdown" zu stande kommt. Abschließend stellen "Chance Nodes" die zufällige Komponente von Poker nämlich das Aufdecken der öffentlichen Karten dar.

Für jeden dieser Knoten lässt sich ein erwarteter Gewinn berechnen, der als Entscheidungskriterium dient. Beim "Chance Node" ergibt sich dieser einfach aus der Wahrscheinlichkeit, dass eine bestimmte Karte C_i aufgedeckt wird: $Pr(C_i)$ und dem erwarteten Gewinn des entsprechenden Teilbaums: $EG(C_i)$

$$EG(C) = \sum_{1 \leq i \leq n} Pr(C_i) \times EG(C_i)$$

Auch beim "Opponent decision node" lässt sich der erwartete Gewinn als gewichtete Summe je nach getroffener Entscheidung O_i des Kontrahenten berechnen:

$$EG(O) = \sum_{i \in \{f,c,r\}} \sum Pr(O_i) \times EG(O_i)$$

Bei "Program decision nodes" kann der erwartete Gewinn unter Verwendung der sog. "Miximax" Strategie wie folgend angegeben werden:

$$EG(U) = \max(EG(U_{fold}), EG(U_{check/call}), EG(U_{bet/raise}))$$

Hier ist jedoch auch eine sog. "Miximix" Strategie mit einer beliebigen Gewichtung ähnlich wie bei "Chance Nodes" und "Opponent Nodes" denkbar, die zu einer zufälligen Wahl der Aktion führt.

Zu guterletzt lässt sich noch der erwartete Gewinn der Blattknoten in Abhängigkeit von der Gewinnwahrscheinlichkeit, der Potgröße und der in den Pot selbst eingezahlten Geldmenge beschreiben:

$$EG(L) = (P_{win} \times L_{\$pot}) - L_{\$cost}$$

Da an jedem inneren Entscheidungsknoten drei Handlungsmöglichkeiten bestehen und in jeder der 4 Spielphasen: "Preflop", "Flop", "Turn" und "River" 4 mal der Einsatz erhöht werden darf, kommt es leicht zu Bäumen mit mehreren Millionen Knoten. Dies macht eine effiziente Datenstruktur und das Caching von Zwischenergebnissen notwendig, um die erwarteten Gewinne trotzdem noch in annehmbarer Zeit zu berechnen.

Die besondere Herausforderung bzgl. des Opponent Modeling ist zum einen die Wahrscheinlichkeit mit der der Gegner bei einem "Opponent decision node" eine bestimmte Aktion

ausführt möglichst genau zu berechnen und zum anderen den erwarteten Gewinn eines Blattknotens zu bestimmen. Wird die Runde durch das folden eines Teilnehmers beendet, so ist die Berechnung relativ trivial, da die Gewinnwahrscheinlichkeit entweder 0% oder 100% beträgt. Sobald es jedoch zu einem showdown kommt, muss hier mit Hilfe des Opponent Modelings eine möglichst zutreffende Schätzung erstellt werden. VEXBOT versucht dies bestmöglich zu gewährleisten indem es das Setzverhalten die sog. "bet-sequence" zusammen mit der Handstärke des Kontrahenten von jedem gesehenen Showdown speichert. In einem späteren Spiel wird dann geprüft, ob bereits eine ähnliche "bet-sequence" beobachtet wurde und wie stark seine Hand in diesen Fällen war. Die Handstärke wird bei VEXBOT in Intervalle zwischen 0 und 1 unterteilt und deren Auftrittshäufigkeit bei einer bestimmten "bet-sequence" in einem Histogramm gespeichert. Sollte von einem Gegner bei einer bestimmten "bet-sequence" 7 mal eine extrem starke und nur einmal eine sehr schwache Hand beobachtet werden so käme bei 10 Intervallen folgendes Histogramm zu stande: [0 1 0 0 0 0 0 0 7 0]. Man erhält somit die Information, dass er bei dieser "bet-sequence" in 87,5% der Fälle tatsächlich sehr gute Karten einer Stärke vom 0,8-0,9 und nur in 12,5% der Fälle mit einer Handstärke von nur 0,1-0,2 geblufft hat.

In dem eben genannten Verfahren werden allerdings nur Beobachtungen berücksichtigt bei denen es zu einem Showdown kommt, damit man die Handstärke des Kontrahenten auch erfährt. Sehr häufig ist aber gerade dies nicht der Fall, da oft einer der beiden Spieler die Runde vorzeitig durch folden beendet. Auch diese Situationen werden von VEXBOT berücksichtigt, da man trotz unbekannter Handstärke des Gegners die Häufigkeiten für ein bestimmtes Verhalten bzw. bestimmte "bet-sequences" beobachten kann. Leicht kann man sich verdeutlichen, dass die Information, dass ein Spieler immer foldet sobald sein Kontrahent raised sehr wertvoll ist, obwohl man nie seine Handstärke erfährt. Diese Häufigkeiten bieten somit einen Anhaltspunkt dafür, wann und wie der Kontrahent entscheidet.

3.2.1 Anwendungsbeispiel

Im Folgenden wird versucht die Vorgehensweise von VEXBOT durch ein kurzes Beispiel etwas anschaulicher darzustellen und seine Funktionen Schritt für Schritt aus dessen Sichtweise zu erklären. In der dargestellten Situation befinden wir uns in der letzten Setzrunde direkt nachdem der River aufgedeckt wurde. Der Pot entspricht zu Beginn der Runde der vierfachen Größe des Big Blinds(BB). Nach einem bet unsererseits (Spieler1) reagiert der Gegner (Spieler2) mit einem raise, wir befinden uns nun also an dem rot markierten Knoten in Figur1. Um zu entscheiden wie wir nun reagieren sollen werden die erwarteten Gewinne der möglichen Aktionen berechnet: **EG(fold)**, **EG(call)** und **EG(raise)**.

EG(fold) : Ein fold unsererseits führt offensichtlich zum Verlust von allem was wir bisher gesetzt haben, also von 2 BB vor dem River und 2 BB durch unseren bet zu einem EG(fold)= -4 BB.

EG(call) : Um den EG(call) berechnen zu können benötigen wir nun allerdings Informationen wie die Handstärke des Gegners, mit der er typischerweise ein solches Verhalten gezeigt hat und natürlich die Stärke der eigenen Hand. Angenommen durch vorherige Spiele liegt uns bereits folgendes Histogramm vor, das Beobachtungen über exakt den selben Spielverlauf bzw. "bet-sequence" enthält: [1 1 0 0 0 0 0 0 4 4

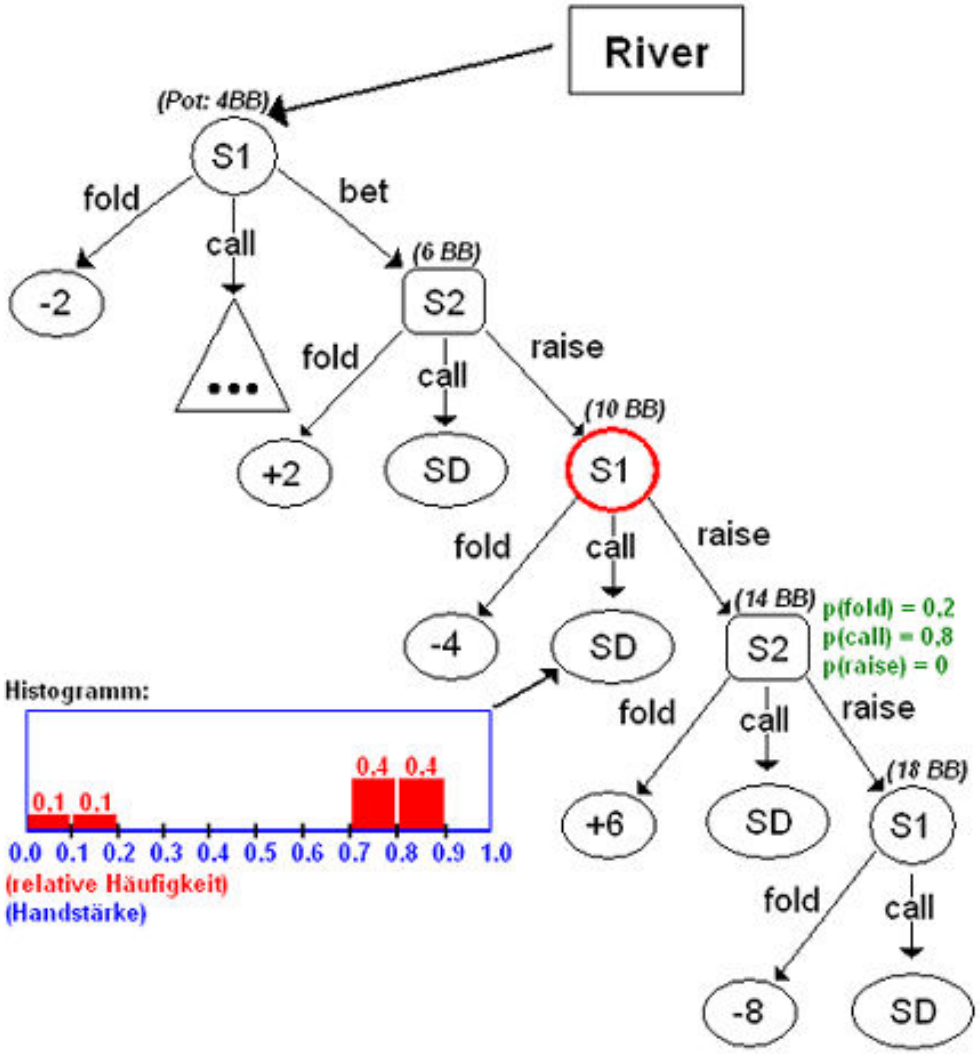

Figure 1: beispielhafter Ausschnitt des Suchbaums

0]. Dies bedeutet, dass der Gegner in der Vergangenheit in 20% der vergleichbaren Fälle nur eine Handstärke zwischen 0 und 0,2 hatte(siehe [1 1 ...]) und in 80% der Fälle eine Handstärke von 0,7 bis 0,9 (siehe [... 4 4 0]). Entsprechen unsere eigenen Karten nun zum Beispiel einer Handstärke zwischen 0,2 und 0,7 so könnten wir besten Wissens nur gegen einen Bluff gewinnen, der bisher in 20% der Fälle auftrat. Folglich liegt unsere Gewinnwahrscheinlichkeit bei einem call Pr(win—call) = 20%. Da sich der Pot durch unseren call nun auf 12 BB ausgeweitet hat, von denen 6 BB vom Gegner stammen, würden wir durch einen Sieg also genau 6 BB gewinnen: $Pr(win|call) * 6BB = 0,2 * 6BB = 1,2BB$. Nicht zu vergessen ist allerdings die Tatsache, dass wir in 80% der Fälle auch die 6 von uns gesetzten Big Blinds

HS	Pr(win/call)	EG(call)	Pr(win/raise,call)	EG(raise)	beste Aktion
0,70	0,2	-3,6	0,0	-5,2	call
0,75	0,4	-1,2	0,25	-2,0	call
0,80	0,6	+1,2	0,5	+1,2	call oder raise
0,85	0,8	+3,6	0,75	+4,4	raise
0,90	1,0	+6,0	1,0	+7,6	raise

Table 3: Erwartete Gewinne (EG) von call und raise in Abhängigkeit der eigenen Handstärke (HS)

verlieren: $Pr(loose|call) * -6BB = 0,8 * -6BB = -4,8BB$. Dies führt insgesamt zu einem erwarteten Gewinn von: $EG(call) = 1,2BB - 4,8BB = -3,6BB$ und stellt im Gegensatz zum Folden eine Verbesserung um 0,4 BB dar. Sollte unsere eigene Handstärke allerdings nur 0,1 betragen ergibt sich unser erwarteter Gewinn zu $EG(call) = -4,8BB$ und es wäre besser für uns zu folden. Eine Übersicht über die erwarteten Gewinne im Falle eines Calls oder Raises in Abhängigkeit der eigenen Handstärke sind in Tabelle1 dargestellt.

EG(raise) : Um den erwarteten Gewinn eines re-raises unsererseits zu berechnen, müssen wir zunächst die erwarteten Gewinne der sich daraus ergebenen Möglichkeiten ermitteln und diese dann entsprechend ihrer Eintrittswahrscheinlichkeit gewichten. Diese Wahrscheinlichkeiten werden hier anhand der bisher beobachteten Häufigkeiten für bestimmte Aktionen unter vergleichbaren Umständen gewonnen. Der Einfachheithalber setzen wir die Wahrscheinlichkeit eines re-raises durch den Gegner auf 0% wodurch die Fälle c) und d) im weiteren Verlauf vernachlässigt werden können. Fall a) tritt zu 20% und b) folglich zu 80% auf.

Grundsätzlich kann es nach unserem re-raise zu 4 verschiedenen Szenarien kommen:

a) Der Gegner steigt aus: fold. In diesem Fall kommt es zu keinem Showdown, da der Gegner die Runde vorzeitig beendet. Zusammen mit einer beobachteten Wahrscheinlichkeit von 20% für diese Aktion berechnet sich der gewichtete, erwartete Gewinn also zu $0,2 * EG(raise, fold) = 0,2 * 6BB = 1,2BB$ und ist unabhängig von unserer Handstärke.

b) Der Gegner geht mit: call. Hier muss wie schon bei EG(call) beschrieben mit Hilfe der Handstärken eine Gewinnwahrscheinlichkeit und damit nun der erwartete Gewinn berechnet werden. Da der Pot hier schon auf 16 BB gewachsen ist beträgt er: $EG(raise, call) = 16 * Pr(win|raise, call) - 8$, also zum Beispiel: $EG(raise, call) = 16 * 0,0 - 8 = -8BB$ bei einer Gewinnwahrscheinlichkeit von 0%. Wir gehen also davon aus, dass er nur mit einer sehr starken Hand callen wird und wir hier keine Aussicht auf einen Sieg haben. Zusammen mit der Eintrittswahrscheinlichkeit dieses Falles von 80% ergibt sich der Anteil von b) an EG(raise) mit $0,8 * -8BB = -6,4BB$.

c) Der Gegner erhöht erneut, woraufhin wir folden: re-raise, fold.
$\Rightarrow 0 * EG(raise, raise, fold) = 0BB$

d) Der Gegner re-raised uns woraufhin wir callen: re-raise, call.
$\Rightarrow 0 * EG(raise, raise, call) = 0BB$

Mit Hilfe der einzelnen Anteile von EG(raise,fold), EG(raise,call), EG(raise,raise,fold) und EG(raise,raise,call) ergibt sich EG(raise) also als:

$$EG(raise) = 0,2{*}EG(raise, fold){+}0,8{*}EG(raise, call){+}0{*}EG(r, r, f){+}0{*}EG(r, r, c)$$

$$EG(raise) = 1,2BB + -6,4BB + 0BB + 0BB$$

$$EG(raise) = -5,2BB$$

Hier kann man ablesen, dass sich VEXBOT unter Verwendung der Miximax-Strategie bei einer eigenen Handstärke von 0,7 für einen call entscheidet, da hier mit EG(call) = -3,6BB der größte Gewinn zu erwarten ist. Sollte auf lange Sicht die Verwendung des jeweils größten Wertes zu vorhersehbaren Handlungen führen, ist hier allerdings auch eine Miximix-Strategie denkbar. Diese wählt zufällig eine Aktion aus, wobei zwar Aktionen mit größerem erwartetem Gewinn auch wahrscheinlicher durchgeführt werden, es einem Gegner aber trotzdem erschwert eine genaue Vorhersage über das eigene Verhalten zu treffen.

3.2.2 Probleme

Das Opponent Modeling beim Pokern ist eine große Herausforderung, die einige grundsätzliche Probleme mit sich bringt.

1. Das Modell des Kontrahenten muss hier sehr schnell erstellt werden, da ein durchschnittlich langes Match beim Pokern gegen menschliche Spieler meist aus weniger als 100 Runden besteht. Hier ist das Ziel möglichst alle zur Verfügung stehenden Informationen zu nutzen, um so den Lernprozess zu beschleunigen.

2. Erfahrene Pokerspieler wechseln häufig ihr Spielverhalten, um möglichst unberechenbar zu bleiben. Dies führt dazu, dass aus Vergangenheitsdaten nicht mehr auf sein aktuelles Verhalten geschlossen werden kann und der Lernprozess von vorne beginnen muss. Ein solcher Strategiewechsel des Gegners muss auch desshalb möglichst schnell erkannt werden da das bisher für ihn aufgestellte Modell zu Fehlentscheidungen führen würde und so viel Schaden anrichten kann.

3. Eine weitere Schwierigkeit mit der man beim Erstellen einer guten Poker KI konfrontiert ist, stellen die oft nur unvollständigen Informationen dar. In vielen Spielsituationen kommt es dazu, dass Spieler folden und so ihre Karten nicht aufgedeckt werden. Hier kann man nicht wissen ob sie wegen einer schwachen Hand kaum eine andere Möglichkeit hatten oder einen Fehler gemacht haben indem sie ein stärkeres Blatt weglegten. Auch der Fall, dass der Spieler ein relativ gutes Blatt weglegt, da er einen Kontrahenten anhand seiner Beobachtungen als stärker einschätzt, bleibt der KI verborgen.

Bei VEXBOT ist das 1. Problem besonders schwerwiegend, da hier Informationen immer in Verbindung mit genau einer “bet-sequence” gespeichert werden. Wie oben bereits beschrieben gibt es sehr viele Variationsmöglichkeiten bzgl. der “bet-sequences” was

leicht zu Suchbäumen mit vielen Millionen Knoten führt. Das Problem daran ist, dass somit jede dieser vielen "bet-sequences" mindestens einmal beobachtet werden muss, um für jeden Fall auch eine Prognose aufstellen zu können. Dies ist natürlich unter normalen Umständen kaum gewährleistet, da hier die Anzahl der Runden eines Spieles weit geringer ist als die Anzahl der unterschiedlichen "bet-sequences". Um diesem grundsätzlichen Problem zu begegnen hat man ein Abstraktionssystem entwickelt, dass je nach Umfang der zur Verfügung stehenden Informationen eine geeignete Abstraktionsschicht auswählt. Die Abstraktion bezieht sich dabei auf die Unterscheidung zwischen verschiedenen "bet-sequences" und weist solche mit gemeinsamen Eigenschaften einer gemeinsamen Gruppe zu. Dies führt dazu, dass nicht jede einzelne "bet-sequence" sondern nur jede Gruppe von "bet-sequences" beobachtet werden muss, um eine Prognose zu liefern. Je weniger Informationen verfügbar sind desto stärker muss abstrahiert also desto weniger Gruppen können unterschieden werden. Eine Gruppe, die zum Beispiel alle "bet-sequences" mit gleicher Anzahl an bets und raises enthält ohne zwischen der genauen Abfolge zu unterscheiden, enthält logischerweise viel mehr Beobachtungen als dies mit Berücksichtigung der Reihenfolge der Fall ist. Sie gibt somit zwar keine genaue Auskunft über sein Verhalten in einer bestimmten, vielleicht bisher noch nie beobachteten Situation, kann aber durch die Verfügbarkeit von ähnlichen Situationen wenigstens eine Schätzung abgeben an der man sich orientieren kann.

Hier kommt es also zu einem Trade-Off zwischen der mit größerer Abstraktion abnehmenden Relevanz der Gruppe für den aktuellen Einzelfall und der zunehmenden Anzahl der in der Gruppe enthaltenen Beobachtungen, die durch eine größere Datenbasis eher Rückschlüsse auf sein prinzipielles Verhalten zulassen. Bei Tests hat sich herausgestellt, dass es oft vorteilhafter ist eine stärker abstrahierende Unterteilung mit vielen Daten pro Gruppe zu verwenden, da die Prognose andernfalls zu stark von einzelnen, nicht repräsentativen Entscheidungen beeinflusst wird. Ein intuitiver Ansatz wäre also die Abstraktionsebene mit dem größten Informationsgehalt zu ermitteln und als Grundlage für das Opponent Modelling zu verwenden. Dies wird bei VEXBOT allerdings in einer leicht abgewandelten Form umgesetzt, indem vor jeder Entscheidung nicht nur eine, sondern alle Abstraktionsschichten, nach ihrem jeweiligen Informationsgehalt gewichtet, berücksichtigt werden.

Nehmen wir beispielsweise an, dass für eine aktuelle Showdown-Situation genau 5 Beobachtungen vorliegen, die eine identische "bet-sequence" besitzen. Diese sehr konkrete Information der wir ein Abstraktionslevel (A0 $\Rightarrow$ gar keine Abstraktion) zuweisen, würde von VEXBOT zum Beispiel für $m = 0,95$ mit $(1 - m^5) = 0,23$ des Gesamtgewichts berücksichtigt werden. Unter der Annahme, dass auf dem nächst höheren Abstraktionslevel (A1) 20 Beobachtungen vorhanden sind, die sich zum Beispiel in der Anzahl der bets und raises der jeweiligen Spieler gleichen, würde die hier gewonnene Information mit $(1 - m^{20}) = 0,64$ des noch verbliebenen Gewichts, also mit $(1 - 0,23) * 0,64 \approx 50\%$ des Gesamtgewichts berücksichtigt werden. Die nächst höhere Abstraktion (A3) enthält vielleicht 75 Beobachtungen und würde also mit $(1 - m^{75}) = 0,98$ beinahe das gesamte restliche Gewicht ca. 27% erhalten. Dies lässt sich so lange weiterführen, bis allen vorgesehenen Abstraktionsschichten ein Gewicht zugeordnet wurde, das deren Relevanz für die aktuelle Situation möglichst gut widerspiegelt.

Das zweitgenannte Problem, dass gute Spieler oft ihre Vorgehensweise ändern, berücksichtigt VEXBOT indem es neuere Daten stärker gewichtet und ältere Daten entsprechend einer

Methode exponentieller Glättung immer mehr vernachlässigt. Die zu Grunde liegende Funktion hängt dabei von einem Parameter h ab, der die genaue Gewichtung festlegt. Bei h = 0.95 würde zum Beispiel die neueste Beobachtung mit 5% und die letzten 1/(1-h) = 20 Beobachtungen mit insgesamt (1-1/e) = 63% des Gesamtgewichts in die Berechnung eingehen.

4. Experimente

In den folgenden Abschnitten werden die Testergebnisse der hier vorgestellten Bots beschrieben. Dabei unterscheiden wir die durchgeführten Experimente je nach Anzahl der Spielteilnehmer.

4.1 Mehrspieler-Matches

Poki wurde auf einem Online Poker Server im Internet Relay Chat (IRC) getestet. Dort spielen menschliche Spieler in verschiedenen Channels gegeneinander. Obwohl nicht um echtes Geld gespielt wird, spielen die meisten Spieler ernsthaft und somit handelt es sich hierbei um eine gute Testumgebung für Poki. Es ist allen Spielern erlaubt an den Einsteiger-Spielen teilzunehmen (#holdem1). Ein Spieler, der genug virtuelles Geld verdient hat, darf dann an Spielen mit fortgeschrittenen Gegnern teilnehmen (#holdem2).

Ein Ziel war es zu testen, ob sich die Gewinnrate durch das verbesserte System, mit dem Poki die Gegner modelliert, signifikant verbessert. Daher wurde Poki sowohl mit dem alten System, wie es von Loki verwendet wurde, als auch mit dem verbesserten System in den Einsteiger- und den Fortgeschrittenen-Spielen getestet. Zusätzlich wurde auch eine Version getestet, die kein Opponent Modeling verwendet. Um möglichst aussagekräftige Ergebnisse zu erzielen, sind jeweils über 20.000 Hände gespielt worden. Die Gewinnrate wird in Small Bets pro Hand (sb/h) gemessen.

Die Version ohne Opponent-Modeling erzielte in den Einsteiger-Spielen weder Gewinne noch Verluste. Das Programm mit dem alten System (Poki_s1) erzielte in den Einsteiger-Spielen eine Gewinnrate von 0.09 sb/h, während das Programm mit dem verbesserten Opponent Modeling (Poki_s2) sogar eine Gewinnrate von 0.22 sb/h erreichte. Die Gewinnrate eines professionellen Pokerspielers liegt in etwa zwischen 0.05 und 0.10 sb/h. Daher sind die Ergebnisse sehr gut, auch wenn die Gegner eines professionellen Pokerspielers weitaus stärker sind.

In den Fortgeschrittenen-Spielen erzielte Poki_s1 eine Gewinnrate von 0.09 sb/h und Poki_s2 eine von 0.08 sb/h. Allerdings kam es bei Poki_s2 scheinbar zu einer Anomalie zwischen der 5.000 und der 10.000 Hand, da der Gewinn hier fast komplett verloren wurde. Hierfür gibt es außer Pech keine sinnvolle Erklärung. Zwischen der 10.000 und der 20.000 Hand wurde wieder fast genauso viel virtuelles Geld verdient, wie Poki_s1 in dem Zeitraum von 20.000 Händen gewonnen hat. Abbildung 1 zeigt diese Ergebnisse. Falls diese Anomalie tatsächlich aufgrund von Pech entstanden ist, würde die Gewinnrate von Poki_s2 mindestens 0.12 sb/h betragen.

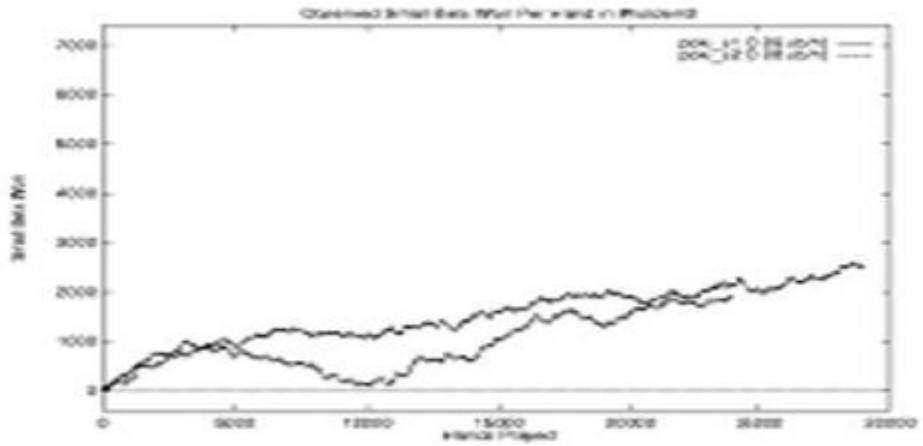

Figure 2: Ergebnisse von Poki_s1 und Poki_s2 in #holdem2

4.2 2-Spieler-Matches

Im Verlauf des 2-Spieler-Experiments spielten im Rahmen eines round-robin Tuniers alle teilnehmenden Pokerprogramme mindestens 40.000 Hände gegeneinander. Neben VEXBOT wurden zusätzlich noch folgende Programme im Experiment untersucht:

1. SPARBOT ist die öffentlich zugängliche Version von PsOPTI-4 und war bisher in Limit Hold'em mit 2 Spielern das stärkste bekannte Programm.

2. POKI ist die erweiterte Version von Loki und stellt ein Formelbasiertes Programm dar, das mit Hilfe von Opponent Modeling besser auf das Verhalten des Gegners reagieren kann. Es ist das stärkste bekannte Programm in der 10-Spieler-Variante von Limit Hold'em Poker und wurde ja im obigen Abschnitt bereits detailliert beschrieben.

3. HOBBYBOT ist ein privat entwickelter Bot, der sich nur langsam auf seinen Gegner einstellt, aber spezielle Schwächen von Poki im 2-Spieler-Match ausnutzt.

4. JAGBOT ist ein sehr einfaches, formelbasiertes Programm, das sich auf rein rationales Spielen konzentriert ohne besondere Schwächen des Gegners herausfinden und ausnutzen zu können.

5. ALWAYS CALL BOT & ALWAYS RAISE BOT wurden hier als Extrembeispiel von schwachen Spielern genutzt, deren Spiel einfach vorhersagbar ist und von besseren Bots schnell durchschaut werden sollte. Sie wurden hier verwendet, um eine bessere Vergleichbarkeit der verschiedenen Bots in Bezug auf die Gewinnmaximierung gegen schwache Gegner zu gewährleisten.

Die in Tabelle2 zu beobachtenden Resultate des Experiments sind alle statistisch signifikant und werden in Form von durchschnittlich gewonnenen Big Blinds pro Hand dargestellt. Generell konnte man dem Experiment entnehmen, dass Vexbot nicht nur gegen alle anderen

Bots gewonnen hat, sondern von diesen jeweils auch noch am meisten Gewinne abschöpft. Auch bei Spielen gegen menschliche Spieler zeigte sich Vexbot größtenteils überlegen und gegen Experten mindestens ebenbürtig. Diese Ergebnisse haben allerdings keine so große Aussagekraft, da hier die menschlichen Spieler nicht die Geduld hatten eine statistisch signifikante Anzahl von Runden zu spielen und so glücksbedingte Schwankungen einen zu großen Einfluss haben. Beobachtet werden konnte allerdings ein Anstieg der Gewinnrate von Vexbot gegen menschliche Spieler nach ca. 200-400 Runden, was auf einen positiven Einfluss des Opponent Modelings schließen lässt.

Programm	Vexbot	Sparbot	Hobbot	Poki	Jagbot	Always Call	Alway Raise
Vexbot	-	+0.052	+0.349	+0.601	+0.477	+1.042	+2.983
Sparbot	-0.052	-	+0.033	+0.093	+0.059	+0.474	+1.354
Hobbot	-0.349	-0.033	-	+0.287	+0.099	+0.044	+0.463
Poki	-0.601	-0.093	-0.287	-	+0.149	+0.510	+2.139
Jagbot	-0.477	-0.059	-0.099	-0.149	-	+0.597	+1.599
Always Call	-1.042	-0.474	-0.044	-0.510	-0.597	-	0.000
Always Raise	-2.983	-1.354	-0.463	-2.139	-1.599	0.000	-

Table 4: Durchschnittlich gewonnene Big Blinds pro Hand

5. Fazit

Im Verlauf dieses Papers wurden einige Verfahren vorgestellt wie das Opponent Modeling realisiert werden kann. Dabei haben wir versucht aufzuzeigen, dass sowohl Poki wie auch Vexbot zwar sehr vielversprechende Ansätze, allerdings auch einige Unzulänglichkeiten mit sich bringen. Bei Vexbot zum Beispiel führt der aufwändige Aufbau des Suchbaumes dazu, dass diese Vorgehensweise für mehr als 2 Spieler kaum noch mit annehmbarer Rechenzeit umzusetzen ist. Das in Zusammenhang mit Vexbot vorgestellte Abstraktionsmodell ist allerdings eine sehr gute Möglichkeit trotz einer begrenzten Anzahl an Daten einen möglichst guten Eindruck des Gegnerverhaltens zu gewinnen. Poki andererseits bietet eine gute Herangehensweise, da hier durch spezielle Analysen die für die Einschätzung des Gegners signifikantesten Parameter ermittelt und auch nur diese in der Berechnung verwendet wurden. Doch obwohl die Entwicklung immer weiter voranschreitet, ist das Problem einer optimalen Poker-KI noch weit davon entfernt gelöst zu werden.Es bedarf noch vieler Verbesserungen, vor allem in Bezug auf das Opponent Modeling, da hier noch immer viele Informationen nur unzureichend genutzt werden. Poki zum Beispiel ignoriert vollkommen die in einem Showdown aufgedeckten Karten und die damit verbundenen Informationen über das Spielverhalten des Gegners. Hier könnte man zum Beispiel in dem Wissen seiner Handstärke seine in der selben Runde getroffenen Entscheidungen rückwirkend analysieren und so mögliche Fehler und Schwächen des Gegners aufdecken. Während andere Aspekte der Pokerprogramme möglicherweise nahezu Perfektion erreicht haben, bleibt die essentielle Funktion des Opponent Modelings mit großer Wahrscheinlichkeit auch in Zukunft eine große Herausforderung.

Abschließend kann man aber festhalten, dass die im Laufe dieses Papers vorgestellten Architekturen und Algorithmen bereits schon heute sehr starke Pokerprogramme hervorge-

bracht haben von denen menschliche Spieler mit Sicherheit vieles lernen können. Beobachtet man die aktuelle Entwicklung, liegt sogar die Vermutung nahe, dass Computerprogramme in absehbarer Zeit allen menschlichen Spielern in ihrer Spielstärke überlegen sein werden.

References

Billings, D., Papp, D., Schaeffer, J., & Szafron, D. (1998). Opponent modeling in poker. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pp. 493–499, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Davidson, A., Billings, D., Schaeffer, J., & Szafron, D. (2000). Improved opponent modeling in poker. In *Proceedings of The 2000 International Conference on Artificial Intelligence (ICAI'2000)*, pp. 1467–1473.

Davidson, A., Darse Billings, Jonathan Schaeffer, D. S. T. S. N. B. M. B., & Holte, R. (2004). Game tree search with adaptation in stochastic imperfect information games..

Evolutionäre Algorithmen

Michael Herrmann mherrmann@gmx.org
Arno Mittelbach mail@arno-mittelbach.de

Abstract

Erste Versuche evolutionäre Algorithmen für die Programmierung von Agenten für Spiele mit unvollständigen Informationen einzusetzen zeigen, dass solche Agenten ihren statischen Alternativen weit überlegen sind. Auf den folgenden Seiten werden die Ergebnisse und die für die Implementierung eines Agenten wichtigen Details der Artikel (Kendall & Willdig, 2001), (Barone & While, 1997), (Barone & While, 1999), (Noble & Watson, 2001) und (Noble, 2002) zusammengefasst.

1. Einleitung

Die vorliegenden Artikel über evolutionäre Algorithmen in Pokerspielen lassen sich in drei größere Kategorien einteilen, an Hand derer wir auch diese Zusammenfassung strukturiert haben. Zunächst werden wir einen einfachen lernbasierten Algorithmus (Kendall & Willdig, 2001) beschreiben, um daran anschließend auf evolutionäre Strategien einzugehen (Barone & While, 1997), (Barone & While, 1999). Im Anschluss daran beschäftigen wir uns mit Pareto-Koevolutionären Algorithmen, behandelt in (Noble & Watson, 2001) und (Noble, 2002), in denen versucht wird einige Probleme mit evolutionären Ansätzen zu lösen. Abschließend werden wir in einem Fazit die für die Implementierung wichtigen Details und Hinweise noch einmal kurz und bündig zusammenzufassen.

2. Lernbasierter Ansatz

Der einfachste Ansatz um einen sich entwickelnden Pokerspieler zu generieren, ist durch einen lernbasierten Algorithmus. Ein solcher Algorithms wird in (Kendall & Willdig, 2001) beschrieben. Es wird gezeigt, dass ein solcher Algorithmus einem statisch programmierten Spieler überlegen ist.

Als Pokervariante wird eine Form des *Draw Poker*[1] mit zwei Wettrunden verwendet.

2.1 Statisch programmierte Spieler

Um die vier klassischen Pokerspieler zu simulieren wird jedem Spieler statisch vorgegeben was er bei welcher Handstärke machen soll (Fold, Call oder Raise und ggf. um wieviel erhöht werden soll).

Als Beispiel sei hier der Unterschied zwischen einem Loose Aggressive Spieler und einem Tight Passive Spieler angegeben. Der Loose Aggressive Spieler steigt aus dem Spiel aus, wenn er nichts besseres als ein Achter-Pärchen hat. Der Tight Passive Spieler steigt hingegen bei allem bis einschließlich einem Ass-Pärchen aus.

1. Siehe http://de.wikipedia.org/wiki/Draw_Poker

Weiterhin unterscheiden sich die statischen Spieler im Setzverhalten. Der Loose Aggressive Spieler geht im Spiel mit (Call) bei allem zwischen einem Neuner- und einem Königs-Pärchen. Der Tight Passive Spieler geht erst ab zwei Pärchen und einer Dreier-Karte bis einer Straße mit Ass als Kicker mit.

2.2 Sich entwickelnder Spieler

Der sich entwickelnde Spieler speichert für jede mögliche Hand einen Lernwert. Dieser wird mit 10 initialisiert, und kann keine Werte größer als 10 und kleiner als 0 annehmen. Der Lernwert wird so interpretiert, dass eine hohe Zahl für eine gute Hand und eine kleine Zahl für eine schlechte Hand steht.

Hat der Spieler nun eine bestimmte Hand, dann holt er sich den dazugehörigen Lernwert und generiert nach einem Algorithmus[2] seine Spielentscheidung. Nach Ende des Spiels wird überprüft ob der Spieler mit der Hand erfolgreich war oder nicht. War er erfolgreich, so wird der Lernwert zur Hand um 0,1 erhöht. Im Gegenzug wird er bei Misserfolg um 0,1 verringert. Dadurch werden gute Hände über die Zeit einen hohen Lernwert behalten wobei schlechte Hände einen niedrigen Lernwert bekommen.

In den Algorithmus, der die Spielentscheidung generiert, gehen verschiedene Größen ein. Diese sind z.B. der Lernwert der entsprechenden Hand, die momentane Potgröße und die Anzahl der Spieler, die im Spiel verblieben sind.

2.3 Experimentergebnis

Da der sich entwickelnde Spieler mit einem Lernwert von 10 für jede Hand startet, wird er am Anfang bei jeder Hand spielen und erhöhen. Er benötigt also eine gewisse Anlaufzeit, um zu lernen, welche Hände schlecht und welche gut sind. Deswegen verliert er am Anfang sehr viel Geld.

Für das Ergebnis ist es egal, ob man den Spieler gegen Loose Aggressive oder Tight Aggressive Spieler spielen lässt. Der grobe Ablauf bleibt der gleiche. Am Anfang verliert der Spieler, aber nach einer gewissen Anzahl an gespielten Händen gewinnt er sein zuvor verlorenes Kapital zurück. Nach diesem Zeitpunkt gewinnt der Spieler stetig mehr Geld. Das lässt sich dadurch erklären, dass die Lernwerte sich inzwischen gut entwickelt haben und der Spieler nur noch bei Spielen mitspielt bei denen er realistische Gewinnchancen hat. Somit ist klar, dass der Spieler die statischen Spieler schlagen kann.

Für genauere Informationen, wie Spieltabellen und Testergebnisse, sei der Leser auf den Artikel (Kendall & Willdig, 2001) verwiesen.

3. Evolutionärer Ansatz

Ein weiterer Ansatz einen sich entwickelnden Pokerspieler zu schreiben, ist die Verwendung von evolutionären Algorithmen. In diesem Abschnitt wird auf die Artikel (Barone & While, 1997) und (Barone & While, 1999) eingegangen.

2. Der genaue Algorithmus ist in (Kendall & Willdig, 2001) angegeben

3.1 Einleitung

Ein evolutionärer Algorithmus ist ein Optimierungsverfahren, das als Vorbild die biologische Evolution hat. Dabei durchlaufen die Individuen einer bestehende Population immer wieder folgende Aktionen, um effektivere Lösungsansätze zu finden:

Selektion Durch die Selektion wird die Richtung vorgegeben in der sich das Erbgut (die Repräsentation des Lösungsansatzes) weiterentwickelt.

Rekombination Die Rekombination oder das *Crossover* von Erbgut liegt hinsichtlich ihres Beitrags zur Zielfindung zwischen Mutation und Selektion. Um gute Ansätze noch zu verbessern wird beispielsweise das Erbgut zweier guter Individuen kombiniert, in der Hoffnung daraus eine noch bessere Lösung zu erhalten.

Mutation Die Aufgabe der Mutation ist es neue Varianten und Alternativen zu erzeugen, um dadurch lokale Maxima zu überwinden.

In den beiden behandelten Artikeln wird eine $(1 + 1)$ Evolutionsstrategie angewandt. Das bedeutet, dass zur Erzeugung von einem Kind ein Elternelement benutzt wird, also asexuelle Evolution stattfindet. Bei den Pareto-Koevolutionsansätzen (Abschnitt 4) hingegen kommt eine $(2 + 2)$ Evolutionsstrategie zum Einsatz. Hier werden also aus zwei Eltern zwei Nachkommen generiert.

Für eine weitere Beschreibung von evolutionären Algorithmen verweisen wir auf (Wikipedia, 2008a).

3.2 (Barone & While, 1997)

Barone und While unterteilen ihren Pokerspieler (Pokeragenten) in mehrere Komponenten. Jede dieser Komponenten kümmert sich um eine bestimmte Aufgabe. Der vorgeschlagene Agent benutzt die Komponenten Handstärke, Position und Risikomanagement. Diese werden durch den *Resolver* befragt, wenn eine Spielentscheidung ansteht. Die Komponenten unterrichten den *Resolver* über ihre Spieleinschätzung, also mit welcher Wahrscheinlichkeit sie mitgehen, erhöhen oder aus dem Spiel rausgehen wollen. Der Resolver gewichtet diese einzelnen Meinungen und trifft eine Entscheidung. Wie er diese Komponentenmeinungen am besten gewichtet, wird ebenso, wie die Einschätzungen der einzelnen Komponenten, bei einem Evolutionsschritt angepasst.

3.2.1 Population der Pokerspieler

Die Anzahl an Pokeragenten variiert abhängig vom Experiment. Jeder Pokeragent hat eigene Komponenten für Handstärke, Position und Risikomanagement sowie einen *Resolver*. Jede Simulation geht über 500 Generationen und jede Generation spielt 100 Hände. Bei dem sich entwickelnden Spieler findet nach jeder Generation eine Mutation statt.

3.2.2 Experimentergebnis

Bei den Experimenten zeigt sich, dass die sich entwickelnden Pokerspieler zu Beginn den statischen unterlegen sind. Das kommt daher, weil die sich entwickelnden Pokerspieler am

Anfang durch Zufallswerte zusammengesetzt sind. Nach einer gewissen Zeit werden sie aber besser und schlagen die statischen Spieler. Dadurch ist es ihnen möglich ihr zuvor verlorenes Spielgeld wieder zurück zu gewinnen und am Experimentende sogar einen Überschuss erspielt zu haben.

Ein sich entwickelnder Pokerspieler der gegen *loose aggressive* Spieler spielt entwickelt ein risikofreudigeres Spiel als ein Pokerspieler der gegen *tight aggressive* Spieler spielt.

Beide sich entwickelnden Pokerspieler gehen bei einem *Straight Flush* aus dem Spiel. Da dieses Blatt so selten ist, konnten sie nicht lernen, dass hier ein sehr gutes Blatt vorliegt.

Ein überraschendes Ergebnis tritt auf, wenn ein sich entwickelnder Spieler gegen *loose aggressive* Spieler spielt, wird er mit hoher Wahrscheinlichkeit aus dem Spiel herausgehen, wenn er in später Position sitzt.

Für die vollständigen Ergebnisse verweisen wir auf (Barone & While, 1997).

3.3 (Barone & While, 1999)

Im Gegensatz zu (Barone & While, 1999) wird hier die Population nicht über Pokeragenten erstellt, sondern durch Kandidaten. Jeder Kandidat besteht aus drei Funktionen. Eine für die Spielaktion Mitgehen, eine für aus dem Spiel gehen und eine für Erhöhen. Diese Funktionen erhalten als Argument einen zu der Gewinnwahrscheinlichkeit des aktuellen Spielstandes korrelierenden Wert. Zudem besteht jede dieser Funktionen aus weiteren Konstanten, die die Form der Funktion definieren.

3.3.1 Evolutionäre Struktur

Jeder sich entwickelnde Pokerspieler besteht aus einem Hypercube, der in zwei Dimensionen unterteilt wird. Eine Dimension steht für das Riskomanagement und eine andere für die Position. Die Positionskomponente ist noch einmal in drei Divisionen unterteilt (eine für frühe, mittlere und späte Position). Die Risikomanagementkomponente ist in vier Divisionen[3] unterteilt. Dies führt zu zwölf Hypercubeelementen. Jedes Hypercubeelement besteht aus N Kandidaten und jeder dieser Kandidaten aus sieben reellen Zahlen. Diese Zahlen stehen für die nötigen Konstanten, die die Form der oben genannten Funktionen definieren.

An jedem Punkt, an dem der Pokerspieler eine Spielentscheidung treffen muss, wählt er zuerst das entsprechende Element aus dem Hypercube durch Herausfinden der aktuellen Position und Anzahl der Wetten bei denen er mitgehen müsste. Danach nimmt er einen Kandidaten und wertet damit die oben genannten Funktionen aus, um den nächsten Spielzug zu berechnen.

Die N Kandidaten eines Hypercubeelementes werden der Reihe nach befragt.

Evolution findet dann statt, wenn alle Kandidaten oft genug befragt wurden. Dabei werden die besten $N/2$ Elemente der Population behalten und generieren $N/2$ Nachfolger, die der Population dann hinzugefügt werden. Nachfolger werden durch Mutation der sieben Zahlen eines Kandidaten generiert.

3. eine für keine Wette, eine für eine Wette, eine für zwei Wetten und eine für drei oder mehr Wetten, die mitzugehen sind

3.3.2 Experimentergebnis

Bei allen Experimenten werden wieder die üblichen vier Standardpokerspieler verwendet. Für die Experimente werden zwei Tische definiert. Am ersten Tisch (genannt *Loose Table*) sitzen neun *loose aggressive* Spieler und ein sich entwickelnder Spieler. Am zweiten Tisch (genannt *Tight Table*) sitzen neun *tight passive* Spieler und ein sich entwickelnder Spieler.

Die sich zu entwickelnten Pokerspieler setzen sich nach einer anfänglichen Verlustphase gegen die statischen Spieler durch.

Im nächsten Experiment werden trainierte Pokerspieler untersucht. Ein Spieler der am *Loose Table* gespielt hat (Spieler A^{loose}) zeigt wesentliche Unterschiede zu einem Spieler der am *Tight Table* (Spieler A^{tight}) gespielt hat. A^{tight} hat beispielsweise gelernt, dass wenn ein Gegenspieler erhöht, dass dies ein Anzeichen dafür ist, dass er vermutlich eine starke Hand hat und er selbst besser aus dem Spiel gehen sollte. A^{loose} hingegen hat gelernt, dass er bei Erhöhungen der Gegenspieler ruhig mitgehen kann, wenn er selbst eine starke Hand hat, da die Gegenspieler oft schwache Hände spielen.

In einem weiteren Experiment wird untersucht, ob die Pokerspieler A^{tight} und A^{loose}, an einem für ihren antrainierten Spielstiel besseren Tisch auch besser abschneiden. Diese naheliegende Vermutung wird durch das Experiment bestätigt.

Für die genauen Ergebnisse und Tabellen verweisen wir den Leser auf (Barone & While, 1999).

4. Pareto-Koevolution

Ein großes Problem von Spielstrategien, die von evolutionären Algorithmen dadurch gewonnen werden, dass die Individuen der Population ausschließlich gegen sich selbst antreten ist, dass diese häufig zu sehr auf einen kleinen Ausschnitt des Lösungsraumes spezialisiert sind. Eine wünschenswerte Strategie hingegen, sollte allgemein genug sein, um gegen viele verschiedene Spieler erfolgreich abzuschneiden.

Dieses Problem lässt sich unter anderem dadurch erklären, dass die Überlegenheitsrelation (Strategie A gewinnt gegen Strategie B) nicht transitiv ist. Dies lässt sich an einem kleinen Beispiel auf Grundlage des Spiels Stein-Schere-Papier leicht erklären: Nehmen wir an, Spieler A spielt immer Papier, Spieler B immer Stein und Spieler C immer Schere. Es gilt nun, dass Spieler A immer gegen Spieler B gewinnt, der wiederum immer gegen Spieler C gewinnt. Jedoch wird Spieler A immer gegen Spieler C verlieren, obwohl er den gegen C überlegenen Spieler B immer schlägt.

Die Existenz einer solchen, nicht transitiven Überlegenheitsrelation kann bedeuten, dass eine Suche zwar ständig neue Strategien findet, die besser als die vorherigen sind, jedoch keine Strategie findet die gegen viele verschiedenen Strategien im Allgemeinen gewinnt.

Diesem Problem soll mit Hilfe der *Pareto-Koevolution* entgegengewirkt werden.

4.1 Begriffserklärung

Bevor wir uns mit *Pareto-Koevolution* in evolutionären Algorithmen beschäftigen, wollen wir uns kurz den Begriffen *Pareto* und *Koevolution* widmen.

4.1.1 Koevolution

Koevolution, auch Coevolution, bezeichnet im Rahmen der biologischen Evolutionstheorie einen evolutionären Prozess der wechselseitigen Anpassung zweier stark interagierender Arten aufeinander, der sich über sehr lange Zeiträume in der Stammesgeschichte beider Arten erstreckt (Wikipedia, 2008c).

Übertragen auf evolutionäre Algorithmen in Spielen würde dies bedeuten, dass sich verschiedene Strategien gegenseitig beeinflussen und sich wechselseitig anpassen. Das ist ein Unterschied zu rein evolutionären Ansätzen, bei denen sich Lösungsansätze nur ihrer Umgebung anpassen.

4.1.2 Pareto-Optimum

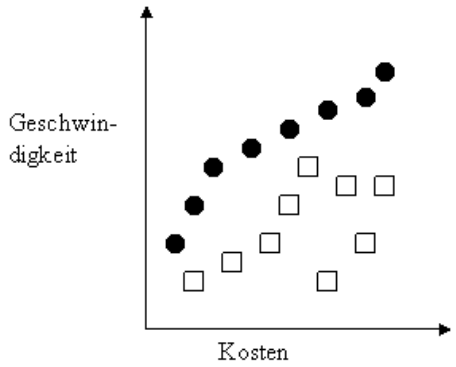

Figure 1: Lösungen des hypothetischen Herrstellungsproblems eines Autos. Die schwarzen Kreise beschreiben die Pareto-Menge.

Das Pareto-Optimum ist ein Begriff aus der VWL und beschreibt all die Lösungen eines Problems mit mehreren Freiheitsgraden, in denen keine relevante Größe (oder Dimension) weiter verbessert werden kann, ohne dass sich eine andere verschlechtert. Als Beispiel soll uns hier die Herstellung eines Autos dienen, wobei die für uns relevanten Dimensionen Kosten und Höchstgeschwindigkeit sind. Ein mögliches Pareto-Optimum könnte zum Beispiel bei 15.000 EUR und 160 km/h liegen. Dies würde bedeuten, dass man für 15.000 EUR kein Auto herstellen kann, dass schneller als 160 km/h fährt und dass man für ein Auto, dass 160 km/h fahren soll mindestens 15.000 EUR zahlen muss. Ein anderes Optimum kann zum Beispiel bei 20.000 EUR und 200 km/h liegen. All jene Lösungen (siehe Figure 1), dieser optimalen Kompromisse bezeichnet man auch als *Pareto-Menge* (im Englischen Pareto-optimal set oder Pareto-front). In den meisten Fällen wird diese Menge mehr als ein Element enthalten, so dass sie beispielsweise als Auswahlkriterium für evolutionäre Algorithmen benutzt werden könnte.

Ein weiterer wichtiger Begriff in diesem Zusammenhang ist der, der *Pareto-Dominanz* (oder der *Pareto-Superiorität*). Man sagt eine Lösung Pareto-dominiert eine andere Lösung, falls sie in allen relevanten Größen (Dimensionen) mindestens genauso gut und in mindestens

einer relevanten Größe besser ist als die andere Lösung. So wird beispielsweise dass Herstellungsverfahren, bei dem 160 km/h schnelle Autos für 20.000 EUR hergestellt werden von dem Design Pareto-dominiert, dass es erlaubt 160 km/h schnelle Autos für 15.000 EUR herzustellen.

4.2 Optimieren mit Hilfe des Pareto-Optimums

Im Gegensatz zu gewöhnlichen Herangehensweisen an Optimierungsprobleme können Pareto-Optimale Lösungen, nach dem Pareto Ansatz nicht mit Hilfe einer Kostenfunktion bewertet werden, so dass zwei Pareto-optimale Lösungen zunächst einmal nebeneinander existieren ohne dass eine Aussage getroffen werden kann, welche der beiden besser ist. Nehmen wir zum Beispiel an, dass 10 Äpfel auf 2 Personen aufgeteilt werden sollen, so sind die Lösungen (10,0) und (5,5) zwei mögliche Verteilungen. Beide Lösungen sind Pareto-optimal, da keine Dimension des Problems weiter verbessert werden kann (eine Person erhält weitere Äpfel), ohne dass eine andere Dimension verschlechtert wird (eine Person bekommt Äpfel abgenommen). Jedoch ist die zweite Lösung "gerechter" als die erste (Wikipedia, 2008d).

Der Pareto-Ansatz in Optimierungsproblemen ist daher nicht die Bestimmung der "besten" Lösung anhand einer Kostenfunktion, sondern die Bestimmung der Pareto-Menge, damit ein menschlicher Entscheidungsträger, oder eine andere Instanz aus dieser Menge geeignete auswählen kann.

4.3 Pareto-Koevolution in evolutionären Algorithmen

Wie bereits in der Einleitung erwähnt, kann die nicht transitive Überlegenheitsrelation von Spielen dazu führen, dass evolutionäre Algorithmen sehr spezielle, statt allgemein gute Strategien entwickeln. Die Idee ist nun, verschiedene Spieler als unterschiedliche Dimensionen (Individuen der Population) eines mehrdimensionalen Optimierungsproblems anzusehen. Darauf sollen nun Prinzipien, wie die Pareto-Dominanz angewandt werden, um allgemeingültige und robuste Strategien zu entwickeln. Ein Individuum ist unter diesem Ansatz umso besser, je mehr Spieler "es schlägt" (hierfür wird im folgenden noch versucht eine Definition zu finden).

Man beachte den sich von "normalen" evolutionären Algorithmen unterscheidenden Ansatz, die ein Individuum umso besser bewerten, je höher der Wert einer Fitness-Funktion (zum Beispiel die Anzahl der Chips beim Pokern) ist.

4.4 Pareto-Koevolution in Texas Hold'em

Im folgenden werden die Ergebnisse des Artikels (Noble & Watson, 2001), in dem an Hand einer einfachen Implementierung untersucht werden soll, ob Pareto-Koevolution evolutionäre Algorithmen verbessern kann, zusammengefasst. Im Anschluss daran werden wir den Folgeartikel (Noble, 2002) diskutieren, bei dem einige Ansätze untersucht werden, um Algorithmen mit Pareto-Koevolution weiter zu verbessern. Im Gegensatz zu allen bisher besprochenen Artikeln (über evolutionäre Algorithmen), wird in beiden Artikeln Texas Hold'em Poker nicht vereinfacht, in einer Limit 2$/4$ Version gespielt.

4.4.1 (Noble & Watson, 2001)

Noble und Watson wählen in ihrer Untersuchung eine einfache Datenstruktur, um verschiedene Pokerstrategien abzubilden. Sie betrachten lediglich die Handstärke sowie die aktuelle Wetthöhe, um sich für Fold, Call oder Raise zu entscheiden. Somit lassen sie viele Feinheiten des Pokerspiels, wie Position, ob die beiden Pre-Flop-Karten connected oder suited sind etc. außer Acht. Dieses zusätzliche Wissen ist aber für die Fragestellung des Artikels auch nicht weiter von Bedeutung, da es den Autoren um die Bewertung der Pareto-Selektion und nicht um die Erstellung eines guten Pokerspielers geht.

Die Strategien werden mit 2 Wahrscheinlichkeiten und 24 Integern (6 für jede Wettrunde) kodiert. Die beiden Wahrscheinlichkeitswerte geben an, mit welcher Wahrscheinlichkeit der Spieler blufft bzw. mit welcher Wahrscheinlichkeit er sich für ein Check-Raise anstatt eines normalen Raise entscheidet (falls er in die Situation kommt). Bei den Integern geben zwei die Mindeststärke einer Hand an, bei der ein Spieler im Spiel bleibt. Zwei weitere beschreiben die Karten, die ein Spieler als eine starke Hand ansehen würde. Die letzten beiden Integer beschreiben den Einsatz, den der Spieler zu dieser Zeit gerne setzen würde und den Wert den er bereit ist maximal mitzugehen. Zusätzlich wurden vier Binärwerte eingeführt, die das Spielverhalten der Spieler in jeder Wettrunde modifizieren können. Ein Bit gibt zum Beispiel an, ob ein Spieler, hält er starke Karten auf der Hand, seine eigentliche Strategie für diese Runde nicht beachtet und stattdessen den maximal möglichen Betrag setzt.

Für eine ausführliche Beschreibung der verwendeten Zahlen verweisen wir den Leser auf (Noble & Watson, 2001).

Ein einfacher Pareto-Koevolutions-Algorithmus Als Algorithmus wurde ein einfacher genetischer Algorithmus gewählt der Reproduktion mit Hilfe von multi-point Crossover und Mutation simuliert. Die Pareto-Dominanz wurde für die Selektion herangezogen.

Es wurde mit einer Population von 100 zufälligen Pokerstrategien angefangen, von denen je 10 zufällig ausgewählt wurden, um 50 Hände Poker zu spielen. 200 solcher Spiele wurden gespielt (so dass jede Strategie im Schnitt 1000 Hände spielen konnte), bevor sich die nächste Generation entwickelt. Die Ergebnisse der 10.000 Hände einer Generation werden für die Selektion in einer Matrix akkumuliert. Mit Hilfe dieser Matrix kann nun die Pareto-Menge, durch paarweise Vergleiche (welcher Spieler hat wie viele Chips von wem gewonnen) entwickelt werden, die dem Algorithmus als Selektionsgrundlage dient.

Ergebnisse im Vergleich mit regulären, koevolutionären, genetischen Algorithmen Als Vergleichsalgorithmus[4] wurde ein ähnlich gestalteter genetischer Algorithmus herangezogen, der für die Selektion jedoch nicht die Pareto-Menge, sondern die verbleibenden Chips der Spieler nutzt. Um die beiden Ansätze miteinander vergleichen zu können, wurden zwei Referenztische mit je 5 handgeschriebenenen Strategien[5] gebildet. Jedes Individuum aus den Endpopulationen (Pareto und regulär) musste nun für ein Spiel, mit 1000 Händen, an beiden Referenztischen antreten.

4. Die Individuen des Vergleichsalgorithmus durchlaufen die gleiche Trainingsphase, wie der Pareto-Algorithmus.
5. Die Referenzspieler wurden mit Hilfe der gleichen Datenstrukturen kodiert.

Das Ergebnis legt nahe, dass Strategien, die mit Hilfe von Pareto-Koevolution entwickelt werden den Strategien überlegen sind, die durch reguläre, genetische Algorithmen erzeugt werden. So schnitten die Pareto Spieler gegen beide Referenzgruppen im Schnitt deutlich besser ab, als die Vergleichsstrategien.

In der Untersuchung aufgetretene Probleme Neben der relativ eindeutigen Aussage, dass Pareto-Strategien im Schnitt besser abschneiden, als ihre rein genetischen Verwandten, wurden auch einige Probleme offensichtlich.

Zum einen zeigt sich, dass die entwickelten Strategien nicht als besonders stark angesehen werden können. Sowohl die Individuen der Pareto-Population, wie auch die Individuen der Vergleichspopulation schafften es nicht an einem der Referenztische Gewinn zu erspielen[6]. Zum anderen scheinen die Pareto-Strategien nicht an ihrem Kollektiv-Wissen festhalten zu können. Dies zeigt sich durch eine sehr kurze Verweildauer von Strategien in der Pareto-Menge (im Schnitt zwei Generationen) und keinem stetigen Wachstum der Spielstärke.

Für eine ausführliche Diskussion der entwickelsten Strategien und der aufgetretenen Probleme sei auf den Artikel (Noble & Watson, 2001) verwiesen.

4.4.2 (Noble, 2002)

In seinem Folgeartikel (Noble, 2002) versucht Jason Nobel die in (Noble & Watson, 2001) aufgetretenen Probleme (siehe 4.4.1) anzugehen, um robuste Texas Hold'em Poker Strategien zu entwickeln. Hierbei wurden die wesentlichen Strategien beibehalten, jedoch kommt eine weitaus feinere Darstellung der Pokerstrategien und eine Technik zur Erhaltung der Vielfalt (*Deterministic Crowding*) zum Einsatz.

Als Darstellungsform der Strategien wurde diesmal auf neuronale Netze[7] zurückgegriffen. Eine Strategie besteht hierbei aus zwei Netzen: Eines für das Pre-Flop-Spiel, bestehend aus 69 Eingängen und 5 versteckten Neuronen und eines für das Spiel auf dem Flop, Turn und River, mit 109 Eingängen und ebenfalls 5 versteckten Neuronen. Beide Netze verfügen über drei Ausgangsneuronen (für Fold, Call und Raise), die über die Strategie des Spielers entscheiden. Um die Komplexität jedoch etwas zu reduzieren wurden die Netze nicht vollständig verbunden[8]. Zu jeder Zeit existieren jeweils nur 50 Verbindungen. Für Informationen zu Mutations- und Rekombinationsstrategien sei der Leser auf (Noble, 2002) verwiesen.

Die Selektion wurde ähnlich, wie in (Noble & Watson, 2001) implementiert (siehe 4.4.1), jedoch wurde die Populationsgröße auf 20 Individuen beschränkt. Da Poker einen nicht zu vernachlässigenden stochastischen Anteil besitzt, wurde eine 100$ Fehlergrenze eingeführt. Das bedeutet, dass Spieler A Spieler B dann dominiert, falls er nicht mehr als 100$ schlechter als Spieler B gegen alle anderen Spieler spielt und besser als 100$ gegen mindestens einen Spieler als Spieler B.

6. Jedoch war der gemittelte Verlust der Pareto-Population deutlich geringer.
7. Für weitere Informationen siehe (Wikipedia, 2008b)
8. Währen sie vollständig verbunden, müssten für das Pre-Flop-Netzwerk 567 Gewichte und für das Post-Flop-Netz 887 Gewichte gepflegt werden.

Deterministic Crowding ist ein einfacher Mechanismus, der versucht die Vielfalt in den Lösungen zu erhalten. Hierbei werden zwei Eltern zufällig ausgewählt, um zwei Nachkommen, mit Hilfe von Mutation und Crossoverfunktionen zu erzeugen. In der folgenden Epoche treten die Nachkommen jeweils gegen den Elter an, dem sie am meisten ähneln. Um den Elter zu ersetzen, müssen die Nachkommen ihren zugewiesenen Elter Pareto-dominieren. Somit sind zu jeder Zeit, mit Ausnahme der ersten Epoche, 10 Individuen in der Pareto-Menge und 10 Nachkommen kämpfen darum, ihren zugewiesenen Elter zu ersetzen.

Als Ergebnis lässt sich festhalten, dass Pareto-Koevolution den einfachen genetischen Algorithmen deutlich überlegen ist. Es wurden verschiedene Experimente mit unterschiedlichen Ausgangslagen[9] durchgeführt. Als Vergleichsgruppe wurde wieder ein ähnlicher einfacher Koevolutionsalgorithmus ohne Pareto-Selektion gewählt. Die Individuen der Endpopulationen traten nun wieder gegen die Referenzspieler aus (Noble & Watson, 2001) (siehe 4.4.1)[10] an. In allen Experimenten waren die Pareto-Spielern ihren einfachen evolutionären Kollegen überlegen. Auch die Einführung von *Deterministic Crowding* (siehe 4.4.2) scheint sich rentiert zu haben. Die Pareto-Population akkumuliert stetig ihr Wissen über neue Generationen hinweg und die durschnittliche Aufenthaltszeit eines Individuums in der Pareto-Menge wird mit zunehmender Generationenzahl immer länger, was sich mit den immer stärker spielenden Individuen gut erklären lässt.

Probleme Auch wenn die Ergebnisse zunächst sehr vielversprechend klingen, so gibt es dennoch einige Beeinträchtigungen. So wurden zwar im Grunde die Ziele erreicht, jedoch erspielen nur die Spieler aus Experiment 3 und 4 (die Startpopulation fing nicht mit zufälligen Spielern sondern mit handkodierten neuronalen Netzen an) am Referenztisch einen Gewinn. Andererseits scheinen die stärksten Individuen aus Experiment 4 sehr gute und robuste Strategien entwickelt zu haben[11].

Für eine ausführlichere Diskussion der Experimente und der aufgetretenen Probleme sei auf den Artikel (Noble, 2002) hingewiesen.

5. Fazit

Als Ergebnis dieser Seminararbeit lässt sich zusammenfassen, dass alle drei vorgestellten Ansätze rein statischen Spielern nach einer Trainingsphase überlegen sind. Dabei zeigt sich, dass evolutionäre Ansätze rein Lernbasierten überlegen sind, jedoch die Pareto-Koevolution der vielversprechenste Ansatz für die Implementierung eines Pokeragenten ist. Besonders der Artikel (Noble, 2002) liefert wichtige Erkenntnisse über eine mögliche Implementierung.

Trotz aller guten Ansätze liegen keinerlei Erkenntnisse über evolutionäre Algorithmen für Texas Hold'em No Limit (sowie Turnierspiele) vor. Die Ansätze können daher nicht direkt auf das Praktikum übertragen werden. Wir erwarten jedoch, dass die oben genannten

9. Experiment 1: Untrainierte Spieler spielen gegeneinander.
   Experiment 2: Es sitzen handgeschriebene Referenzspieler mit am Tisch.
   Experiment 3: Die neuronalen Netze wurden "sinnvoll" vorbelegt.
   Experiment 4: Kombination von 2 und 3.
10. Die Spieler traten nur an dem stärkeren der beiden Referenztische an.
11. Der Autor des Artikels muss zugeben gegen den besten Spieler aus Experiment 4 im Schnitt zu verlieren.

Artikel auch für das Praktikum und die Implementierung eines Pokeragenten, für Texas Hold'em No Limit, eine gute Ausgangsbasis bilden.

References

Barone, L., & While, L. (1997). Evolving computer opponents to play a game of simplified poker. In Royle, G., & Backman, R. B. (Eds.), *8th University of Western Australia Computer Science Research Conf.*

Barone, L., & While, L. (1999). An adaptive learning model for simplified poker using evolutionary algorithms. In Angeline, P. J., Michalewicz, Z., Schoenauer, M., Yao, X., & Zalzala, A. (Eds.), *Proceedings of the Congress on Evolutionary Computation*, Vol. 1, pp. 153–160 Mayflower Hotel, Washington D.C., USA. IEEE Press.

Kendall, G., & Willdig, M. (2001). An investigation of an adaptive poker player. In *Australian Joint Conference on Artificial Intelligence*, pp. 189–200.

Noble, J. (2002). Finding robust texas hold'em poker strategies using pareto coevolution and deterministic crowding..

Noble, J., & Watson, R. A. (2001). Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection. In Spector, L., Goodman, E. D., Wu, A., Langdon, W. B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M. H., & Burke, E. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 493–500 San Francisco, California, USA. Morgan Kaufmann.

Wikipedia (2008a). Evolutionärer algorithmus — wikipedia, die freie enzyklopädie.. [Online; Stand 27. März 2008].

Wikipedia (2008b). Künstliches neuronales netz — wikipedia, die freie enzyklopädie.. [Online; Stand 26. März 2008].

Wikipedia (2008c). Koevolution — wikipedia, die freie enzyklopädie.. [Online; Stand 25. März 2008].

Wikipedia (2008d). Pareto-optimum — wikipedia, die freie enzyklopädie.. [Online; Stand 25. März 2008].

Lagging Anchor Algorithm: Reinforcement Learning with Imperfect Information and its application to poker games

Bastian Christoph bc1001.rbg.informatik.tu-darmstadt.de
Knowledge Engineering Group
Department of Computer Science
Technische Universität Darmstadt, Germany

Abstract

This paper describes how currently existing techniques for reinforcement learning with perfect information such as gradient-search based methods and linear programming methods can be extended towards the usage for solving games with imperfect information. It referes to the work of Frederick A. Dahl who developed the technique in his group and showed the application towards playing poker.

1. Introduction

Since reinforcement learning is widely developed in the case of games with perfect information there exist some well known techniques for solving games of the two-player zero-sum type in game theory.

The commonly known gradient search based techniques like TD-learning which developed over are few years and are used in a lot of tools. By changing the view to games, where not all information are visible in each game state, there has to be a technique for solving these problems in a more approximatively way. The TD-learning method is used later in the "truncated sampling"-part of the algorithm-implementation for extensive-form games.

The author analyzed the common gradient-search based method and the common linear programming method from the view of the psychological oriented human learning model and the machine learning model and used some new assumptions for changing the methods to a different way, so he showed how there can be a new technique, by him called the lagging anchor algorithm to give a quite good solution to reach the game theoretic optimal solution point.

2. Lagging Anchor Algorithm

General Description: The author describes that the fact, that simple gradient search may result in oscillations around the solution points, which can be exploited to define a better algorithm. The algorithm exhibits an exponential convergence for at least some special cases of matrix games. The general idea is to have an "anchor" for each player, which is "lagging" behind the current value of the paramter states v and w, which is dampening the oscillation seen with the basic algorithm. The anchor is drown towards the corresponding parameter value, proportionally to the distance between the parameter state and the anchor. Before he comes to this point, he gives a formal definition of the basic algorithm.

Formal Definitions: At first, the author gives the expected payoff and the strategies for the Blue and Red player. The basic algorithm is given by assuming that Blue measures the gradient of the exponential payoff with respect to the parameter set and updates the parameters in that direction. Red performs a similiar update with the opposite sign. He defines the basic algorithm by the rule (1) and the initial values, thereby using a convex projection towards a non-empty and closed convex set A. The goal is to find a minimax solution point. The necessary condition is, that neither side can improve his payoff locally. This is satisfied for a special fixed point for the mapping (1).

Definition of the Algorithm: So the scene is set for the definition of the lagging anchor algorithm. The anchors for v and w are v' and w', while the anchor drawing factor is given. The algorithm is given by (2) with the intial conditions. For a special case, there's no need for convex projections. Because convex projections are well defined, the shown sequence is determined by (2). The actual implementations demand to calculate gradients and convex projections, which my be non-trivial. In the following sections, the implementation in the matrix-form and in the extensive-form is shown. This will include the convergence result, too.

3. Usage towards Matrix-Form Games

General Description: The matrix M is given by m_{ij}, where the value is the expected outcome of the player Blue choosing i and the player Red choosing j. Linear and complete parameterization of the strategies are used. The parameter sets V and W for Blue and Red are given, while the sets are compact and convex. The interior points are sets in the affine subspaces of R^n and R^m, where the components sum to 1. The interior points there are points with positive value for all the components. We refer to the game as a fully mixed solution, if the game solution is an interior point. The information sets of the game do not need to be handled explicitly. The probability of Blue taking action i is represented as $B_v(i) = v_i$ and the probability of Red taking action j is represented as $R_w(j) = w_j$. With this, Blue has the expected payoff $E(v, w) = v^T * M * w$. The differentation of these gives the gradients, so that the learning rule given before changes to the new learning rule. These changed rule of the basic algorithm conincides the gradient update, modified by Selten. The simple algorithm for calculating convex projections is given by the Proposition 2, shown in the refered papers. It suffices to describe the computation of C_v, because the sets V and W have an identical structure. For an index set I, Q_I is defined. In the bi-matrix case, selten showed that the basic algorithm fails to converge towards fully mixed solutions. By stating that the gradient update is orthogonal to the error vector, the used Proposition 3 refines the result. The process keeps a constant distance to the solution point, when α approaches zero. For the notational convenience x^T is defined.

Lagging Anchor Algorithm: We are now moving the the lagging anchor algorithm for matrix games. The learning rule of (2) can be implemented towards (4), using the calculation procedure for C_v and C_w, which are given in the proposition (2). The proposition (4) therefore gives the convergence result. For general matrix games, the convergence has not been proven. But the algorithm has worked well on a large sample of test games. Therefore, random matrices were tested with dimensions up to 100 and the entries where uniformly distributed in the interval $[-1, 1]$. There, the exponential convergence has been

observed. The anticipatory learning rule of selten is given in the new notation. With the experiments, it giives a exponential convergence for large random matrices. In general, the idea of the algorithm is to produce approximate solutions to large games, using non-linear and incomplete parameterization. Rather than compete with existing efficient algorithms for linear programming, which are the standard for solving the matrix games. Here, the lagging anchor algorithm appears to have similarities with the interior point methods for the linear programming. It searches the interior of the set of admissible solution points. And it works to solve the primal and the dual problem simultaneously. There, the primal problem is the optimization problem of Blue, while the dual problem is the one of Red. Because the interior point methods consist of finite sequences of projections, rather than incremental search that converge to a solution, the similarities end at that point. There, the main fact is that the interior point methods can not be extended beyond the context of matrix games.

4. Usage towards Extensive-Form Games

General Description: In [2] the presented agent design allows non-linear and incomplete stategy representations in extensive form games. So the algorithm can be adjusted to that context. At last the stability result is given. The learning rule in (2) contains non-trivial operation of evaluating the gradient of the players. The author explained, how this is handled and presented the algorithm in pseudocode. The learning rule contains convex projections, which are needed if the domains of the players' parameters are restricted. Because the different parameter sets require different methods, there is no ability to specify a general algorithm. So the general optimization techniques like the solution of the Kuhn-Tucker conditions should be useful.

Main changes: In general, the process of estimating the gradient is split into two. In the first process, the gradient of the expected payoff is estimated with respect to the action probabilities (dE/dB). In the second process, the gradient of the action probabilities is calculated with respect to the parameters (dB/dv). When combining the gradient using the chain rule of differentiation, we get the estimated gradient of the agent's expected payoff with respect to the parameters (dE/dv). At last the agent's parameters are updated in the gradient direction, multiplied with the step size α.

4.1 Training Patterns

For explaining the algorithm, the notion of "training patterns" is used, which is commonly used in the context of neural networks. There, the training pattern is a combination of the input and the feedback. The application of the training pattern is implemented as the gradient search step (length proportional to α) in the parameter space V towards a minimization. The parameter state of B is modified to make the output closer to the feedback of the given input. The second part of the gradient calculation (dB/dv) is integrated with the parameter update. There we are assuming that the derivative of the payoff with respect to the out of B on input is estimated to be d. The training with the pattern implements both, the calculation of the gradients and the parameter update in the gradients direction.

4.2 Sampling the consequences of the alternative actions

General Description: In general, the problem of calculating the gradients of the player's payoffs analytically is far too complex to be considered. So we want to estimate them based on the outcomes of the sample games, where the Blue and Red agents are playing. This leads us the denotion "reinforcement learning". For simplificationn of the presentation, we describe the estimation of the Blue's gradient only. The method, which is developed therefore relates to the "what-if" analysis. At first the sample game is completed, then the course of the game is analyzed. For each visited decision node, the hypothetical consequences of the different available actions are estimated. So later, the convertion of the estimates into the training patterns is shown. The estimation if the consequence of action in the descision node is equivalent to the estimation of the expected payoff for the resulting node. So is seems, that there is an contradiction. It is mentioned, that the nodes of the game tree may not have unique values. Now it seems like we are estimating this. Given current strategies of Blue and Red, we are attemping to estimate expected payoff for nodes, so we do not. Given the players Blue and Red, there's no problem on defining the value of a node as its average payoff. The simplest way of estimating the expected payoff for a node is by doing sampling. There one simulates one or more games from the nodes in question, and then takes the sample average as the estimate. So this is clearly unbiased. Consider the case, where only one additional game is completed for each decision as "canonical form" of the algorithm. We can take the outcome of the original sample game as estimated consequence of action made in game, only actions deviating from course of game need to be sampled. In the figure the author shows an illustration of the algorithm. The circular and square nodes represent the decision nodes for the Blue and Red player, while the triangle nodes represent terminal states where the game rules define the payoff of the Blue player. The vertical path represents the course of the game. In the top node, Blue has two deviating actions, leading to game states which are labelled with "X". To estimate the utility, an additonal sample game is completed from each of these states, indicated in the figure. In the central node of the figure, the Blue player has two alternatives to the action, taken in the original game, leading to the states which are labelled "Y". The numbers which are attached to the arcs in the figure are giving the estimated payoffs resulting form the action taken by the Blue player in the preceding nodes. These numbers are turned into the training patterns later.

4.3 Truncated sampling using the TD-learning

General Description: The canonical form of the algorithm requires one sample game to be completed for each decision that player Blue can make throughtout the original sample game. The number of additional games that must be completed to estimate the gradient from one sample game is less than the branching factor multiplied by the tree depth. The sampling of the results from alternative decisions slows down the algorithm by a linear factor of thre tree depth, which is no desaster from the viewpoint of the computational complexity. The computational burden from the canonical form of the algorithm can be very significant in practice. It introdues the considerable noise in the estimed gradients.

Truncated Sampling: Therefore, the "truncated sampling" is introduced by the author. The general idea is to use an external function for the estimation of the node values. The purpose is only to estimate the consequences of the alteernative decisions, not to take

partin the game playing. In the example figure 4 in [2], this means that nodes which are labelled "X" and "Y" are evaluated by a function indead of being played out. Given the players Blue and Red, the course of the game is a Markov process, where the states are the game-tree nodes. The fact that the players have just imperfect information and that they act on the information sets is not a problem, as the pair of mixed strategies for Blue and Red introduces the Markov Process on the set of the nodes. The standard method for estimating the expected outcome of the Markov Process is the TD-learning, which is used here. After a sample game has been completed, the course of the game (here as sequence of visited nodes) is used to update the evaluator of the node through TD-learning. This essentially means, that the evaluation of the given node is tuned towards the weighted average of the actual payoff and the function owns the evaluations of the intermediate nodes. The use of truncated sampling throught a function tuned by TD-learning will introduce bias in the gradient estimation, unless the function produces exact evaluations of the nodes. At the beginning of the training session, the node evaluator function will normally be initialized randomly, in which case the gradient estimation will be very poor until the function starts giving meaningful evaluations. Only the experiments can determine wether the gain in speed and the reduced randomness of the truncated sampling outweight the disadvantage of the biased gradient estimation.

5. Conclusion

At the end we can reach the point, that there are some ways of changing the traditionally known methods for two-play zero-sum games with perfect information by extending the model with the result of psychological analysis. We have seen that the gradient-search based method and a linear programming method can be extended for the usage in the lagging anchor algorithm. The application to poker showed, that for this case the developed algorithm is quite useful and gave a good approximation. Looking forward to futher questions this could be a quite good example for looking at the common techniques in an extending way.

@inproceedings(dahl,key="Dahl01", Author="Dahl, Fredrik A.", Title="A reinforcement learning algorithm applied to simplified two-player texas hold'em poker.", BookTitle="Proceedings of the 12th European Conference on Machine Learning (ECML-01)", year=2001)

@inproceedings(dahl,key="Dahl02", Author="Dahl, Fredrik A.", Title="The lagging anchor algorithm: Reinforcement learning in two-player zero-sum games with imperfect information.", BookTitle="Machine Learning", year=2002)

SVMs und Kategorisierung der Spieler

Benjamin Herbert BHERBERT@RBG.INFORMATIK.TU-DARMSTADT.DE

Abstract

1. Einleitung

Texas Hold'em ist eine sehr populäre Pokervariante, die in den letzten Jahren vermehrt Gegenstand der Forschung geworden ist. Viele Forscher aus dem Gebiet der künstlichen Intelligenz forschen an der Entwicklung von Programmen, die Poker spielen können, so genannten *Pokerbots* oder kurz *Bots*. Dabei werden verschiedene Ansätze gewählt, unter anderem regelbasierte Systeme, Simulationen und Neuronale Netzwerke. Ein Pokerbot spielt nach einer bestimmten Strategie und reagiert auf bestimmte Spielsituationen mit bestimmten Aktionen. Ein interessanter Ansatz ist es, die verfügbaren Spieldaten eines erfolgreichen Pokerspielers oder Pokerbots zu verwenden, um Anhand der in ähnlichen Spielsituationen gewählten Entscheidung die eigene Entscheidung zu treffen. Spielplattformen im Internet bieten oft die Möglichkeit Spiele zu beobachten und Spieldaten zu extrahieren. Dabei ist es von Interesse aus Spieldaten Rückschlüsse zu ziehen, wie man Spieler einordnen kann oder ob bestimmte Verhaltensweisen erfolgreiches Pokerspielen bedingen.

2. Klassifikation

Das Einordnen von Objekten anhand ihrer Merkmale in vorgegebene Klassen bezeichnet man als Klassifikation. Im Maschinellen Lernen wird mit Hilfe von Beispielen ein Klassifikator erlernt, mit dem sich die Klassen von unbekannten Beispielen vorhersagen lassen. Ein Beispiel besteht dabei aus einer Menge von Merkmalen und der zugehörigen Klasse. Oft wird die so genannte *binäre Klassifikation* mit den Klassen + und − betrachtet, diese sind jedoch symbolhaft zu verstehen und werden in der Anwendung durch andere Bezeichnungen ersetzt.

2.1 Beispiele und Klassen

Ein Beispiel wird durch Merkmale beschrieben. In Klassifikationsalgorithmen wird häufig ein sogenannter Featureraum verwendet, in dem sich Beispiele durch Punkte oder Vektoren darstellen lassen. Die Merkmale der Beispiele werden in diesen Featureraum abgebildet. Dazu ist es notwendig, dass nicht diskrete Werte in diskrete Werte überführt werden. Beispiele lassen sich also als Tupel darstellen: (**Feature**,Klasse) wobei es sich bei **Feature** um einen Vektor von Features handelt.

2.2 Anwendung im Pokerspiel

Im Pokerspiel gibt es eine Vielzahl an Merkmalen und Klassifikationsaufgaben. Es folgen eine Auswahl von Merkmalen und eine kurze Erläuterung.

2.2.1 Direkte Merkmale

Direkte Merkmale sind Merkmale, die man direkt aus dem Spiel oder einer aktuellen Hand ableiten kann. Die folgenden Merkmale sind an (Blank, 2004) angelehnt.

Handstärke Die Bewertung der Stärke einer Hand kann dazu dienen Spielentscheidungen zu treffen. So ist es bei einer sehr schwachen Hand ratsam zu passen, bei einer sehr starken Hand sollte auf Gewinnmaximierung wert gelegt werden.

Pot Odds Der Anteil zwischen aktuellem Gesamtbetrag im Spiel und dem Einsatz den man bringen muss um mitzuspielen. Wenn das Verhältnis hoch ist, so kann man auch mit schwächeren Karten mitspielen.

Positives Potential Die Fähigkeit im späteren Spielverlauf eine sehr starke Hand zu erzielen wird Positives Potential genannt. Dabei handelt es sich oft um Flush- und Straight Draws, also Möglichkeiten einen Flush oder einen Straight zu erzielen.

Für weitere Merkmale und eine tiefergehende Beschreibung sei auf (Miller, Sklansky, & Malmuth, 2004; Harrington & Robertie, 2004) verwiesen.

2.2.2 Indirekte Merkmale

Anhand von Daten, die man von vergangenen Spielen zur Verfügung hat, kann man im Nachhinein indirekte Merkmale berechnen. Einige Merkmale kann man für einzelne Spieler berechnen, diese können helfen einen Spieler zu charakterisieren.

Anteil der gespielten Hände Der Anteil der gespielten Hände eines Spielers kann ein Indiz dafür sein, dass dieser zu viele schwache Hände spielt.

Cold Calls Falls ein Spieler mit einer vorausgegangenen Erhöhung mitgeht, ohne vorher Geld investiert zu haben, nennt man das Cold-Calling. Dies sollte man normalerweise vermeiden, sofern man keine gute Hand hat, da ein aggressives Spiel meist Zeichen für eine starke Hand ist.

Diese und weitere Merkmale sind in (Johansson, Sonstrod, & Niklasson, 2006) zu finden.

2.2.3 Klassifikationsaufgaben in Poker

Eine Vielzahl an Klassifikationsaufgaben in Poker sind denkbar. Es ist denkbar den Gegner anhand seiner Spielweise als bestimmten Spielertyp zu erkennen und sein eigenes Spiel anzupassen. So ist es gegen Spieler, die ausschließlich gute Starthände spielen, notwendig sehr solide zu spielen und sich nicht auf unnötiges Risiko einzulassen. Nach Möglichkeit sollte man ein Aufeinandertreffen meiden und frühzeitig passen, sofern man keine gute Hand hält(Harrington & Robertie, 2004).

3. Support Vector Machines

Bei den Suppor Vector Machines (SVM) handelt es sich um eine Klasse von Algorithmen, die zur Klassifikation eingesetzt werden können. In der ursprünglichen Form waren SVMs lineare Klassifikatoren, die Punkte in einem Featureraum durch eine Hyperebene linear separieren.

3.1 Theorie

Im folgenden wird der Fall von linear separierbaren Klassen betrachtet, wie von Vapnik ursprünglich betrachtet. Eine Support Vector Machines beschreibt die optimal separierende Hyperebene, die die gegebenen Beispiele linear separiert. Dabei wird die Hyperebene berechnet, die den größten Abstand zu den Beispielpunkten hat. Diese Ebene kann durch folgende lineare Gleichung beschrieben werden:

$$H = \mathbf{w}\mathbf{x} + b = 0$$

wobei $\mathbf{w}$ eine Normale ist und $\frac{|b|}{|\mathbf{w}|}$ den Abstand zum Ursprung entlang der Normalen beschreibt.

Die parallelen Hyperebenen H_1 und H_2 zu H, ergeben sich zu

$$H_1 : \mathbf{w}\mathbf{x} + b = +1$$
$$H_2 : \mathbf{w}\mathbf{x} + b = -1$$

Für eine Beispielmenge $(\mathbf{x_i}, y_i)$ werden H_1 und H_2 betrachtet und deren Abstand $\frac{2}{|\mathbf{w}|}$ wird maximiert. Dazu muss $|\mathbf{w}|$ minimiert werden. Da keine Punkte in den Raum zwischen H_1 und H_2 fallen sollen, gelten folgende Randbedingungen:

$$\mathbf{w}\mathbf{x_i} + b \geq +1 \text{ if } y_i = +1$$
$$\mathbf{w}\mathbf{x_i} + b \leq -1 \text{ if } y_i = -1$$

die man kompakter als

$$y_i(\mathbf{w}\mathbf{x_i} + b) - 1 \geq 0 \text{ für alle } 1 \leq i \leq n$$

schreiben kann(Burges, 1998). Daraus lässt sich folgendes Optimierungsproblem ableiten, welches die optimal separierende Hyperebene H findet.

Wähle $(\mathbf{w}, b)$ und minimiere $|\mathbf{w}|$ mit der Randbedingung
$$c_i(\mathbf{w}\mathbf{x_i} - b) \geq 1 \text{ für alle } 1 \leq i \leq n$$

Dies kann auf ein *Quadratic Programming* Optimierungsproblem überführt werden

$$\sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i x_j$$

mit den Randbedingungen

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

Eine effiziente Lösung dieses Problems wird in (Joachims, 1999) besprochen, welche in SVMlight implementiert wurde.

3.2 Klassifikation mit SVMs

Das Klassifizieren von unbekannten Beispielen $(x_i, ?)$ erfolgt dann mittels

$$Klasse(x_i) = \begin{cases} +1 & \text{if} \mathbf{w x_i} + b > 0, \\ -1 & \text{if} \mathbf{w x_i} + b \leq 0 \end{cases}$$

Das heisst, es wird für Beispiele anhand ihrer Repräsentation im Featureraum berechnet, auf welcher Seite der Hyperebene H sie sich befinden und die jeweilige Klasse wird zugewiesen.

3.3 Nicht linear separierbare Beispiele

Um auch Klassifikationsproblemen lösen zu können, die nicht linear separierbar sind, ist es möglich, den sogenannten *Kernel-Trick* anzuwenden. Nicht linear separierbare Probleme sind in höherdimensionalen Vektorräumen lösbar. Dabei kommt eine Abbildung ϕ zum Einsatz, die die Vektoren x in diesen höherdimensionalen Featureraum F abbildet. Dies führt jedoch zu nicht mehr handhabbaren Problemen, wenn die Vektoren in diesen Räumen zu groß werden.
In oben genannter dualen Repräsentation erscheinen die Vektoren nur als Skalarprodukte. An den Featureraum angepasst muss die Gleichung also lauten

$$\sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \phi(\mathbf{x_i}) \phi(\mathbf{x_j}).$$

Da in dieser Formulierung nur das Skalarprodukt steht ist die Dimensionalität von F nebensächlich. Eine Kernfunktion, oder kurz Kern, ist eine Funktion K, die den Wert des Skalarprodukts von zwei Punkten in F zurückgibt

$$K(x_i, x_j) = \phi(\mathbf{x_i}) \phi(\mathbf{x_j})$$

Das heißt in den Formulierungen kann das Skalarprodukt durch eine geeignete Kernfunktion ersetzt werden. Im folgenden Kapitel werden einige Kernfunktionen aufgeführt.

3.4 Einige Kernfunktionen

Einige geeignete Kernfunktionen sind

lineare Kernfunktionen $K(x_i, x_j) = \langle x_i, x_j \rangle$

polynomielle Kernfunktionen $K(x_i, x_j) = \langle x_i, x_j \rangle^d$

RBF-Kernfunktionen(Radiale Basis Funktion) $K(x_i, x_j) = \exp(-\gamma |x_i - x_j|^2)$

sigmoide Kernfunktionen $K(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle + \nu)$.

Für weitere Information über Kernfunktionen sei auf (Cristianini, 2004), (Schoelkopf, 1997) und (Schoelkopf & Smola, 2002) verwiesen.

3.5 Parameterwahl

Die Auswahl einer geeigneten Kernfunktion kann experimentell geschehen. In (Blank, 2004) wurden dazu auf einem Traingsset für unterschiedliche Kernel Tests mit verschiedenen Parametern durchgeführt und mit einem Testset überprüft.

3.6 SVM Implementierungen

Es gibt zahlreiche freie SVM Implementierungen. In (Blank, 2004) wurde SVM^{light} verwendet, eine OpenSource SVM Implementierung, die für Mustererkennung, Regressionsprobleme und Ranking eingesetzt werden kann. Sie unterstützt die unter Punkt 3.4 genannten Kernel und diese können durch Parameter konfiguriert werden.

4. Entscheidungsbäume

Ein weiterer Algorithmus zur Klassifikation von Beispielen sind Entscheidungsbäume. Ein Entscheidungsbaum beschreibt ein Konzept als Disjunktion von Konjunktionen.

Dabei wird eine hierarchische Struktur von Entscheidungsknoten gelernt. Diese haben den Vorteil, dass das gelernte Modell bzw. die Entscheidungsfindung sehr gut von Menschen zu verstehen ist.

4.1 Klassifikation mit Entscheidungsbäumen

An jedem Knoten wird ein Attribut eines Beispiels untersucht und abhängig von dessen Wert wird das Beispiel mit den Entscheidungsknoten im jeweiligen Teilbaum weiter untersucht. In den Blättern eines Entscheidungsbaumes wird dann eine Klasse zugeordnet.

4.2 Lernen von Entscheidungsbäumen

Das Lernen eines Entscheidungsbaumes besteht darin, zu entscheiden welches Feature man im Wurzelknoten betrachtet. Dabei kommen verschiedene Heuristiken zum Einsatz. Ein Beispiel dafür ist die *Information Gain* Heuristik, die zum Beispiel auch in ID3 und dessen Nachfolger C4.5 zum Einsatz kommt. Im binären Fall ergeben sich somit nach der Entscheidung für ein Attribut im Wurzelknoten zwei Teilbäume. In jedem dieser Teilbäume wird

dann mit den aufgeteilten Beispielen nach dem nächsten Attribut gesucht nach welchem Unterschieden werden soll. Der Entscheidungsbaumalgorithmus von Quinlan, C4.5 ist in (Quinlan, 1993) besprochen.

4.3 Pruning

Entscheidungsbäumen ist es möglich, auf dem Trainingsset eine Genauigkeit von 100% zu erzielen. Der Entscheidungsbaum ist dann jedoch sehr verästelt und generalisiert schlecht, das heisst, dass ungesehene Beispiele wohlmöglich falsch eingeordnet werden und die Genauigkeit auf einem unbekannten Testset somit niedrig ist. Dies nennt man auch *Overfitting* zu deutsch etwa *Überanpassung*.

Eine Mittel um diese Genauigkeit auf unbekannten Daten zu erhöhen ist das sogenannte *Pruning*, bei dem Knoten aus dem Baum entfernt werden um oben genanntes Overfitting zu vermeiden. Pruning hat auch den Vorteil, die Lesbarkeit eines Entscheidungsbaums zu verbessern, da weniger Knoten im Baum vorkommen.

5. Weka

Weka ist eine Sammlung von Algorithmen für Data Mining Aufgaben. Es ist ein Open-Source Software Projekt unter der GNU General Public License. Die Software ist in Java geschrieben und kann als JAR Paket heruntergeladen werden. Es kann in eigenen Javaprogrammen eingesetzt werden um verschiedenste Data Mining Aufgaben zu bewältigen.

5.1 ARFF Dateien

Beispiele, die bestimmte Merkmale haben, lassen sich in eine sogenannte ARFF Datei schreiben. ARFF steht für Attribute-Relation File Format. In dieser Datei werden zunächst im *Header* Attribute definiert. Im *Data* Teil der Datei kommagetrennte Werte der Beispiele.

5.2 Beispiele

Beispiele werden in Weka durch die Klasse `weka.core.Instances.` repräsentiert.

5.3 J48

Weka bietet eine eigene C4.5 Implementierung an, die auch in einem der Artikel verwendet wurde. Die Zugehörige Javaklasse ist `weka.classifiers.trees.J48`.

6. Zusammenfassung: Creating an SVM to Play Strong Poker

Im Paper von Blank, Soh und Scott(Blank, 2004) ist ein Experiment beschrieben, in dem versucht wurde einen erfolgreichen Pokerspieler zu imitieren. In den folgenden Abschnitten wird das Vorgehen und das Ergebnis kurz zusammengefasst.

6.1 Ansatz

Im Experiment kam eine SVM zum Einsatz, die mit Spieldaten eines Spielers trainiert wurde. Die Spieldaten stammen vom Gewinner eines Spiels von mehreren Pokerbots, einem *Simbot* namens Hari. Es wurden die in Kapitel 2.2.1 genannten Merkmale betrachtet. Man betrachtete drei Klassen, die die Entscheidungen in bestimmten Situationen darstellen.

- Fold
- Check/Call
- Bet/Raise

Für die jeweils vorliegende Spielsituation wurde die Klasse der Entscheidung festgestellt und vermerkt. Anhand dieser Daten wurden drei SVMs mit sigmoider Kernfunktion gelernt und verwendet um die Entscheidung vorauszusagen, die Hari gewählt hätte, wenn er in einer ähnlichen Situation gewesen wäre. Diese SVMs wurden dann in einem eigenen Pokerbot implementiert und gegen weitere Pokerbots getestet.

6.2 Ergebnis

Um eine Entscheidung für eine bestimmte Spielsituation zu erhalten wurden oben genannte Features berechnet und die Ergebnisse der drei SVMs für die Spieldaten betrachtet. Die Entscheidung, die von der Mehrheit der SVMs getroffen wurde, wurde im Spiel durchgeführt. Bei eventuellem Gleichstand wurde die sicherste Entscheidung übernommen. Das Ergebnis des Tests war, dass der implementierte Pokerbot im Schnitt einen Verlust von 0,53 Small Bets pro Hand erzielte. Besonders negativ war, dass er sehr passiv spielte und keinerlei Aggression zeigte. Somit war es nicht möglich selbst mit den stärksten Händen genug Profit zu erzielen um etwaige Verluste schwächerer Hände auszugleichen. Stellt man den Anteil der Aktionen des implementierten Pokerbots und Hari gegenüber, so ist zu bemerken, dass beide Pokerbots ungefähr gleich oft passen.

7. Zusammenfassung: Explaining Winning Poker

Johansson, Sönströd und Niklasson zeigen in (Johansson et al., 2006), wie man mit Machine Learning Methoden Regeln aus Daten von Pokerspielen extrahieren kann. Die Experimente und die Ergebnisse werden nun zusammengefasst.

7.1 Ansatz

Zwischen März und Mai 2006 wurden alle Spieltische mit sechs Spielern auf Ladbroke's Onlinepokerplattform beobachtet und mittels einer Pokersoftware namens PokerOffice wurden die Spieldaten extrahiert. Dabei wurden Spieler betrachtet, die mehr als 500 Hände gespielt hatten. Aus den Daten der 105 Spieler, die genug Hände gespielt hatten, wurden verschiedene Features für die jeweiligen Spieler berechnet (siehe Kapitel 2.2.2)und die Daten wurden in einer *MySQL* Datenbank gespeichert. Es wurde mit Hilfe von Wekas J48 und dem Regellernalgorithmus G-REX untersucht, ob sich aus den so gewonnenen Daten herausfinden lässt, woran man einen gewinnenden oder einen verlierenden Spieler erkennen kann.

Dazu wurden verschiedene Sortierungen der Beispiele untersucht und jeweils die obersten 25 Spieler zu einer Klasse gezählt. Mit den so erstellten Beispielen wurde für jede Sortierung versucht das Konzept mit den erstellten Entscheidungsbäumen zu beschreiben.

7.2 Ergebnis

Die Regeln von J48 waren aufgrund eingeschaltetem Pruning (siehe Kapitel 4.3) relativ kompakt hatten aber eine weniger gute Genauigkeit als G-REX. Die mit G-REX gefundenen Regeln waren in der Lage mit relativ kompakten Regeln eine gute Genauigkeit zu erzielen. Es zeigte sich, dass es anscheinend eine Korrelation zwischen verschiedenen Merkmalen und dem Gewinn bzw. Verlust von Spielern gibt und man mit Hilfe von Entscheidungsbäumen eventuell eine grundlegende Einteilung von Spielertypen erreichen kann, wenn man genug Daten eines Spielers gesammelt hat.

8. Fazit

Der Ansatz von (Blank, 2004) die Strategie eines Pokerspielers zu imitieren ist interessant, da bei erfolgreicher Anwendung lediglich die Spieldaten eines guten Pokerspielers oder -bots benötigt würden. Das schlechte Abschneiden des implementierten Pokerbots zeigt, dass es mit den untersuchten Merkmalen allein nicht möglich ist erfolgreich zu spielen. Hier wäre es interessant weitere Merkmale zu betrachten. Die Autoren wollen in weiteren Studien auch auf die Position eines Spielers eingehen. Das Experiment von (Johansson et al., 2006) basiert auf einer relativ kleinen Datengrundlage. Die Ergebnisse zeigen, dass durch die mit G-REX gefundenen Entscheidungsbäume die jeweiligen Spielergruppen gut klassifiziert werden können. Ein denkbarer Einsatz wäre das Finden von grundlegenden Regeln für das Spielverhalten oder das Klassifizieren von Gegnern auf Grund beobachteter Daten. Hiefür müssten allerdings die Daten einer größeren Spielerzahl zur Verfügung stehen.

Literatur

Blank, T. Leen-Kiat Soh Scott, S. (2004). Creating an SVM to Play Strong Poker.. pp. 150– 155.

Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, *2*(2), 121–167.

Cristianini, J. S.-T. . N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Harrington, D., & Robertie, B. (2004). *Harrington on Hold 'Em, Volume 1: Expert Strategy for No Limit Tournaments: Strategic Play: Expert Strategy for No Limit Tournaments: 1*. Two Plus Two Publishing LLC.

Joachims, T. (1999). Making large-scale SVM learning practical. Advances in Kernel Methods - Support Vector Learning, B. Schoelkopf and C. Burges and A. Smola (ed.), MIT-Press.

Johansson, U., Sonstrod, C., & Niklasson, L. (2006). Explaining Winning Poker–A Data Mining Approach. In *ICMLA '06: Proceedings of the 5th International Conference*

on Machine Learning and Applications, pp. 129–134, Washington, DC, USA. IEEE Computer Society.

Miller, E., Sklansky, D., & Malmuth, M. (2004). *Small Stakes Hold'em: Winning Big with Expert Play* (1 edition). Two Plus Two Publishing LLC.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Schoelkopf, B. (1997). Support Vector Learning. Tech. rep..

Schoelkopf, B., & Smola, A. J. (2002). *Learning with Kernels*. The MIT Press, Cambridge, MA.

UCT: Selektive Monte-Carlo-Simulation in Spielbäumen

Michael Wächter michael_waechter@gmx.de

Abstract

Im Gegensatz zu klassischen Ansätzen wie dem Alphabeta-Algorithmus durchsuchen Monte-Carlo-Methoden in Spielen zufällig Teile des Spielbaums, um damit auf den Wert der Wurzel zu schließen und eine Zugentscheidung zu treffen.
UCT stellt einen neuen, durch (Coulom, 2006) und (Kocsis & Szepesvári, 2007) vorgestellten Monte-Carlo-Algorithmus dar: Zu Beginn durchsuchen die Monte-Carlo-Simulationen den Spielbaum zufällig. Später werden sie dann anhand der bisher berechneten Ergebnisse selektiv durch den Spielbaum geleitet, um schneller zu den gewünschten Ergebnissen zu gelangen.
Neben der Beschreibung des Algorithmus wird auf seine Eigenschaften, Relevanz für Poker, Verbesserungsmöglichkeiten und Resultate in existierenden Systemen eingegangen.

1. Einleitung

Die grundlegendste Möglichkeit, um ein Programm zu befähigen ein bestimmtes Spiel zu spielen, stellen auf Expertenwissen basierende, "handgeschnitzte" Zugentscheidungsfunktionen dar. Beispielsweise benutzte das Pokerprogramm Loki der University of Alberta zu Anfang eine solche Funktion (Billings, Papp, Lourdes, Schaeffer, & Szafron, 1999).
Dieser Ansatz leidet unter vielen Nachteilen. Zum einen ist die Spielstärke des Programms beschränkt durch das Wissen des Experten. Zum anderen ist die Integration von Wissen sehr arbeitsintensiv und fehleranfällig, da Teile des Wissens dazu neigen mit anderen zu interagieren und Programme nur schlecht fähig sind, die Wichtigkeit einzelner "Wissensteile" abzuwägen. In Go beispielsweise ist das benötigte Wissen auch oft zu abstrakt, als dass es direkt in ein Programm umgesetzt werden könnte.
Für einige deterministische Spiele mit vollständiger Information wie Schach und Dame hatte sich bereits relativ früh herausgestellt, dass dieser wissensintensive Ansatz schwierig umzusetzen und wenig von Erfolg gekrönt war. Die überwältigende Mehrheit der Programme und Forschergruppen konzentrierte sich daher auf den suchintensiven Ansatz: Die natürliche Erweiterung statischer Evaluationsfunktionen bestand im Minimax- oder Alphabeta-Algorithmus: Der Spielbaum[1] wird soweit expandiert, wie es die zur Verfügung stehenden Ressourcen zulassen. Nun werden die Blätter mit solch einer auf Expertenwissen basierenden Heuristik evaluiert und ihre Werte zur Berechnung des Werts der Wurzel verwendet. Es stellte sich heraus, dass bereits Evaluationsfunktionen, die den Wert einer Stellung nur grob annähern[2], zu hohen Spielstärken führen können, die natürlich die reine Spielstärke dieses geringen Expertenwissens bei weitem übersteigen. Die Erfolge von Chinook[3] und DeepBlue[4] bewiesen dann eindrucksvoll, dass pure Rechenkraft ausreichte, um

1. Der aktuelle Zustand bildet die Wurzel, mögliche zukünftige Zustände die Knoten und Züge die Kanten.
2. In erster Näherung reicht im Schach z.B. eine Bewertung des noch im Spiel befindlichen Materials.
3. Chinook siegte 1994 über den Weltmeister Tinsley und 2007 wurde das Spiel schließlich komplett gelöst.
4. DeepBlue besiegte 1997 den damals amtierenden Weltmeister Kasparov mit $3\frac{1}{2} - 1\frac{1}{2}$

weltmeisterliches Niveau zu spielen.
Seit den ersten Erfolgen hat sich dieser Ansatz zwar bedeutend weiterentwickelt (transposition tables, iterative deepening, null-move heuristic, killer heuristic, history heuristic, quiescence search). Aber dennoch reicht er nicht aus, um eine Vielzahl an Spielen und damit einhergehenden Problemen zu meistern. Dies kann unterschiedlichste Ursachen haben.
Für indeterministische Spiele (z.B. Poker) beispielsweise kann der Alphabeta-Algorithmus zwar erweitert werden. Allerdings explodiert dann die Größe des Spielbaums und der Algorithmus ist nicht fähig ihn in akzeptabler Zeit zu durchsuchen und brauchbare Ergebnisse zu liefern. Auch KIs für kommerzielle Spiele wie Echtzeitstrategiespiele können auf Grund von hohen Verzweigungsfaktoren und Indeterminismus nicht auf Alphabeta setzen.
Go bringt schon von sich aus einen sehr großen Verzweigungsfaktor im Spielbaum mit. Zudem ist eine Suche mit einer Tiefe von bspw. 12 Zügen schlicht nicht ausreichend, da es zahlreiche Situationen gibt, in denen Menschen fähig sind, die Konsequenzen von Zügen tiefer abzuschätzen. Der möglicherweise wichtigste Grund jedoch ist, dass Züge stark dazu neigen miteinander zu interagieren. Das Spiel ist fast immer dynamisch, selten ruhig und es gibt eine Vielzahl an Faktoren, die über den Wert einer Stellung entscheiden. Es ist daher sehr schwierig, eine Evaluationsfunktion zu schreiben, die den Wert einer Stellung an den Blättern des Alphabeta-Algorithmus adäquat bewertet (vgl. auch (Chaslot, Winands, Uiterwijk, van den Herik, & Bouzy, 2007)).
In (Brügmann, 1993) wurde dann jedoch die Möglichkeit der Anwendung von Monte-Carlo-Methoden für Go in Erwägung gezogen und in (Bouzy & Helmstetter, 2003) wieder aufgegriffen. Die Entwicklung von UCT durch (Coulom, 2006) bzw. (Kocsis & Szepesvári, 2007) führte schließlich dazu, dass die Verwendung von Monte-Carlo-Methoden für Go allgemein Beachtung fand und UCT mittlerweile von vielen der weltweit besten GO-KIs verwendet wird. Außerdem gibt es viele Forschergruppen, die an weiteren Themen zu UCT wie Parallelisierbarkeit (Cazenave & Jouandeau, 2007) oder Integration von Patterns (Gelly & Silver, 2007) arbeiten.
Auch für Poker wurde die Einsetzbarkeit von Monte-Carlo-Methoden erkannt (Billings et al., 1999), die Überlegenheit gegenüber dem zuvor verwendeten Expertenwissenansatz experimentell bestätigt und zahlreiche weitere Vorzüge des neuen Ansatzes hervorgehoben.

Kapitel 2 beschreibt die Funktionsweise von Monte-Carlo-Methoden und speziell UCT.
Kapitel 3 geht auf Eigenschaften des Algorithmus ein.
Kapitel 4 stellt dar, wie UCT für Poker eingesetzt werden kann und welche Vorzüge es gegenüber anderen Ansätzen hat.
Kapitel 5 schließlich versucht aufzuzeigen, welche Möglichkeiten zur Erweiterung von UCT bestehen oder gegenwärtig untersucht werden.
Für ein schnelles Erfassen des Inhalts dieser Ausarbeitung eignet sich am besten das Lesen der Abschnitte 2.2, 4, sowie evtl. 5.1 und 5.2.

2. Funktionsweise

2.1 Monte-Carlo-Algorithmen

Der grundlegende Monte-Carlo-Algorithmus funktioniert folgendermaßen: Zunächst wird der Spielbaum mit konstanter Tiefe 1 expandiert (also alle Folgestellungen der aktuellen

Stellung generiert). Anschließend werden alle Kindknoten der Wurzel (Folgestellungen der aktuellen Stellung) durch Monte-Carlo-Simulationen evaluiert und schließlich der Zug ausgewählt, der zur bestbewerteten Stellung führt.
Den eigentlich wichtigen Teil stellt die MC-Evaluation dar: Von der zu bewertenden Stellung ausgehend werden mehrere (i.A. mehrere 1000) Simulationen durchgeführt. Pro Simulation werden von der zu bewertenden Stellung aus so lange zufällige Züge gespielt, bis man an einer Terminalposition (eine Stellung, in der das Spiel endet) ankommt. Diese kann nun trivial bewertet werden[5]. Im Poker bspw. kann man als Bewertung einfach die gewonnene oder verlorene Geldmenge des Spielers wählen. Hat man mehrere solcher zufälligen Partien durchgeführt, ist der Wert der Stellung der Durchschnittswert aller Simulationen.

Im Vergleich zu allen zuvor in Poker oder Go verwendeten Ansätzen ist an diesem bemerkenswert, dass er mit einem Minimum an Wissen über das Spiel auskommt. Es wird abgesehen von grundlegenden Funktionen nur eine Funktion, die Terminalpositionen bewertet, benötigt.
Dieser Ansatz hatte dennoch unübersehbare Schwächen. Zum einen konnte er Go nicht auf dem selben Niveau wie klassische Ansätze spielen. Zum anderen war er zwar strategisch stark, da er seine "Aufmerksamkeit" breit fächert und stets alle Züge "in Erwägung zog", war aber taktisch schwach, da er nicht fähig war, seine "Aufmerksamkeit" auf interessante, wichtige Züge "zu konzentrieren". Es gab daher verschiedene Versuche dies zu verbessern:
In (Billings et al., 1999) wird die zufällige Zugauswahl in den Simulationen durch Expertenwissen zu den interessanten Zügen gelenkt. Die Auswahl der Züge geschieht also nicht mehr gleichverteilt sondern mit einer durch eine Expertenfunktion vorgegebenen Wahrscheinlichkeit. In Poker schien dieser Ansatz gut zu funktionieren.
In (Bouzy & Helmstetter, 2003) wurde eine weitere Expansion des Spielbaums an der Wurzel versucht. Dies erhöhte jedoch den Rechenaufwand stark und brachte keine wesentliche Spielstärkenverbesserung. In (Bouzy, 2004) wurde Iterative Deepening vorgeschlagen: In jedem Durchlauf wird der Spielbaum eine Ebene tiefer expandiert, die Blätter MC-simuliert und die daraus gewonnnen Informationen für die Beschneidung des Baums im nächsten Durchlauf mit tieferer Suchtiefe verwendet[6]. Laut (Coulom, 2006) schnitt dieser Algorithmus jedoch oft auch brauchbare Äste ab.
In (Coulom, 2006) und (Kocsis & Szepesvári, 2007) wurde schließlich UCT vorgeschlagen[7].

2.2 Upper Confidence Bounds applied to Trees (UCT)

UCT stammt vom UCB-Algorithmus ab, da UCB das Exploration-Exploitation-Dilemma löst und dies auch in Spielbäumen zu lösen ist (siehe hierzu Anhang B).

5. In einem Spiel sollte es am Ende einer Partie trivial sein, den Gewinner und wenn gewünscht die Höhe des Siegs zu ermitteln. Auf Go trifft diese Aussage allerdings nur bedingt zu. Am Ende einer Partie ist es nicht ohne weiteres möglich, den Gewinner zu ermitteln. Es ist zwar möglich die Partie weiter zu spielen bis das Ergebnis trivial ermittelbar ist, allerdings kann das Weiterspielen zu einer anderen Siegöhe und eventuell zu einem anderen Sieger führen.
6. Im 2. Durchlauf mit einem Baum der Tiefe 2 werden also die Äste, die sich im 1. Durchlauf als zu schlecht erwiesen haben, abgeschnitten.
7. (Coulom, 2006) kommt eher von der praktischen Seite der Anwendbarkeit in Spielen. (Kocsis & Szepesvári, 2007) geht vom theoretischen Multiarmed-Bandit-Problem aus und erprobt den Algorithmus auch an eher theoretischen Problemen.

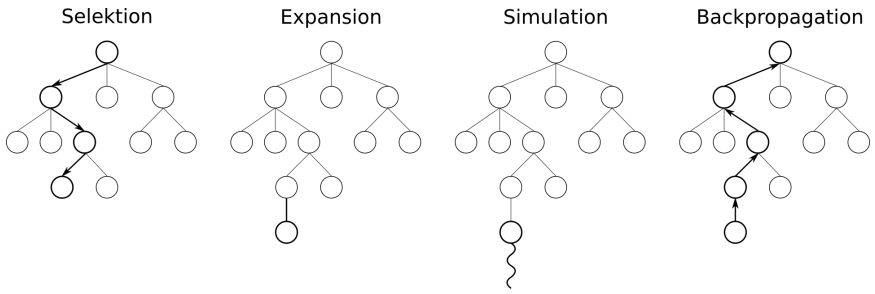

Figure 1: Die vier wesentlichen Teile von UCT. Die Zeichnung ist der Zeichnung in (Chaslot et al., 2007) nachempfunden.

Der Algorithmus expandiert den Spielbaum an der Wurzel nicht mit einer konstanten Tiefe sondern selektiv danach welche Varianten sich als erfolgversprechend herausgestellt haben oder noch herausstellen könnten. Der Baum wird komplett im Speicher gehalten[8] und abgesehen von den unbedingt notwendigen Infos, wie Zeiger auf die Nachfolgerknoten wird in jedem Knoten nur gespeichert, wie viele Simulationen schon durch diesen Knoten gelaufen sind und welchen Wert diese Simulationen summiert hatten. Somit kann später leicht bestimmt werden, welchen Wert jeder Knoten durchschnittlich hat.
Der Algorithmus gliedert sich im Wesentlichen in vier Teile, Selektion, Expansion, Simulation und Backpropagation, die in jeder Simulation nacheinander durchlaufen werden.

2.2.1 Selektion

Von der Wurzel ausgehend wird im Spielbaum ein Weg zu einem Knoten ausgewählt von dem aus der Baum weiter expandiert wird. Bei der Wahl des Knotens muss wie bereits erwähnt abgewogen werden, ob man gute (Exploitation) oder vielversprechende (Exploration) Knoten wählt.
Hierzu wechselt UCT (von der Wurzel ausgehend) an jedem Knoten p zum Kindknoten k mit der höchsten "Upper Confidence Bound": $k = argmax_{i \in I}(v_i + C\sqrt{\frac{\ln n_p}{n_i}})$
I ist die Menge der Kindknoten von p, v_i der Durchschnittswert der Simulationen, die durch i gelaufen sind, und n_p bzw. n_i die Zahl der Simulationen durch p bzw. i.
v_i ist also der bisherige Wert eines Knotens und $C\sqrt{\frac{\ln n_p}{n_i}}$ stellt eine Art Zuversichtlichkeitsbonus dar, der Knoten einen Bonus gibt, je weniger Simulationen bisher durch sie gelaufen sind, also je unsicherer sie sind. Über die Konstante C kann die "Neugierde" des Algorithmus gesteuert werden: Ist C klein, wird der Baum tiefer expandiert, da hauptsächlich die bisher beste Variante untersucht wird. Ist C groß, wird der Baum eher breiter expandiert. Laut (Chaslot et al., 2007) muss C experimentell ermittelt werden, in (Kocsis & Szepesvári, 2007) wird jedoch einfach $\sqrt{2}$ verwendet.

8. Dies führt dazu, dass der Algorithmus leider etwas speicherintensiv ist. Die Größe des Baums ist aber immernoch relativ passabel im Vergleich dazu wenn man bspw. einen kompletten Alphabeta-Baum im Speicher behalten wollte.

2.2.2 Expansion

Gelangt die Selektion an einem Knoten an, für den bisher weniger als ein Schwellwert T an Kindknoten im Baum expandiert wurden (im Spezialfall also ein Blatt), wird hier weiter expandiert, dem Knoten also weitere Kindknoten hinzugefügt.
Die einfachste denkbare Strategie hierbei ist, einen zufälligen neuen Kindknoten hinzuzufügen. Hier sind aber auch intelligentere Ansätze möglich. In (Chaslot et al., 2007) bspw. wird zu Anfang jeweils nur ein zufälliges Kind hinzugefügt. Durchläuft man schließlich den Knoten einmal wenn er bereits T Kinder hat, werden sofort alle noch fehlenden Kinder hinzugefügt. Es sind aber auch Strategien denkbar, die Expertenwissen verwenden und z.B. Schlagzüge zuerst expandieren.

2.2.3 Simulation

In der Simulation wird der Wert eines neu hinzugefügten Knotens abgeschätzt. Genauso wie beim reinen Monte-Carlo-Algorithmus wird hierzu vom Knoten aus eine simulierte Partie bis zu einem Terminalknoten gespielt. Dies geschieht im einfachsten Fall durch rein zufällige Züge, kann aber auch durch eine geeignete Simulationsstrategie (vgl. Abschnitt 5.1) gesteuert werden. Hier bieten sich bspw. Patterns (Gelly, Wang, Munos, & Teytaud, 2006; Gelly & Silver, 2007) oder die bereits bei den Monte-Carlo-Ansätzen erwähnte Strategie in (Billings et al., 1999) an.
Wird eine Simulationsstrategie verwendet, kann sich zwar das Spielniveau deutlich steigern, allerdings steigt auch die Rechenzeit des Algorithmus um evtl. mehrere Größenordnungen (vgl. (Chaslot et al., 2007)). Ein Ausweg ist, dass die Simulationsstrategie eventuell nur für wichtige (z.B. ausreichend oft besuchte) Knoten verwendet wird. Außerdem muss beachtet werden, dass die Strategie weder zu randomisiert noch zu selektiv vorgehen darf, da in beiden Fällen offensichtlich die Spielstärke leidet.
Der Wert des zu simulierenden Knotens entspricht dem Wert der durchgeführten Simulation. Ob dies die Höhe des Siegs oder einfach nur +1 für einen Sieg und -1 für eine Niederlage sein soll, muss abgewogen werden[9].

2.2.4 Backpropagation

Nach der Simulation muss das ermittelte Ergebnis im Baum nach oben bis zur Wurzel durchgereicht werden.
In der Standardversion wird einfach der Durchschnittswert aller durch den Knoten durchgelaufenen Simulationen nach oben gereicht. Wie bereits erwähnt enthält jeder Knoten Informationen, wie viele Simulationen durch ihn durchgelaufen sind und welches Ergebnis sie summiert hatten. Der Wert v_i eines Knotens in der Selektionsformel (s.o.) ist dann offensichtlich die Summe der Werte geteilt durch die Zahl der Simulationen.
(Coulom, 2006) zeigt experimentell, dass dieser "Mean"-Backupoperator dazu neigt, den wahren Wert eines Knotens zu unterschätzen. Eine Alternative, der "Max"-Backupoperator, der den Wert des Kindes mit dem höchsten Wert hochreicht, überschätzt hingegegen den

9. In Go bspw. ist die Höhe des Siegs unerheblich. Die Höhe des Siegs einfließen zu lassen, kann die KI dazu veranlassen gierig zu spielen und das Spiel damit zerbrechlich, also anfälliger gegenüber gegnerischen Atacken werden zu lassen. Liegt man vorne, reicht es aus sicher zu spielen und nicht noch die Höhe des Siegs zu vergrößern. Im Poker ist dieser Sachverhalt natürlich noch einmal ganz anders.

wahren Wert. Der "Robust Max"-Operator reicht den Wert des Kindes, durch das die meisten Simulationen gelaufen sind, hoch. Für kleine Simulationszahlen liefert er zwar schlechtere, für große Simulationszahlen hingegen deutlich bessere Vorhersagen. Der beste Operator ist jedoch "Mix": Eine Linearkombination aus "Mean" und "Robust Max" (sowie ein paar Verfeinerungen für Spezialsituationen). Er konvergiert für hohe Simulationszahlen zwar nur wenig besser als "Robust Max", liefert allerdings für niedrige Simulationszahlen wesentlich bessere Vorhersagen.
Da für "Mix" jedoch die Koeffizienten der Linearkombination optimiert werden müssen, ist für eine einfache Implementierung evtl. doch "Mean" oder "Robust Max" vorzuziehen.

2.2.5 Zugauswahl

Hat man die gewünschte Anzahl an Iterationen durchgeführt, befindet sich ein selektiv expandierter Baum im Speicher und man muss aus ihm einen Zug auswählen. Die meistverwendete Möglichkeit ist es, an der Wurzel den Zug zu wählen, durch den die meisten Simulationen gelaufen sind.

3. Eigenschaften und Ergebnisse

3.1 Konvergenz

In (Kocsis & Szepesvári, 2007) wird unter anderem gezeigt, dass der Algorithmus in seiner grundlegendsten Form (bspw. ohne Simulationsstrategie) die Fehlerwahrscheinlichkeit an der Wurzel für $t \to \infty$ polynomiell gegen 0 gehen lässt. Der Algorithmus konvergiert also gegen das korrekte Resultat, das auch eine vollständige Alphabeta-Berechnung bis hin zu den Terminalknoten geliefert hätte.

3.2 Expertenwissen

Ein bereits hervorgehobener Vorzug von UCT ist, dass in der grundlegendsten Form kein Expertenwissen ausser einer (meist trivialen) Funktion, die Terminalknoten bewertet, benötigt wird. Insbesondere wird, da immer bis zu Terminalknoten simuliert wird, nicht wie in Alphabeta eine Evaluationsheuristik für Nichtterminalknoten benötigt. In der Go-Programmierung hatte sich nämlich genau dies als Problem herausgestellt (vgl. Abschnitt 1).
In UCT dient sämtliches Wissen nur noch dem Zweck der Spielstärkensteigerung. In (Billings et al., 1999) wird auch erwähnt, dass sich bei Monte-Carlo-Ansätzen verwendetes Expertenwissen sehr gut kapseln lässt (was auf UCT auch zutreffen dürfte), was die Entwicklungszeit und Wartbarkeit verbessert.
Bei der Integration von Expertenwissen muss aber wie bereits erwähnt die Spielstärkenverbesserung mit der Rechenzeiterhöhung abgewogen werden. Ist das Wissen gut gekapselt, sollte es aber auch möglich sein an seiner Performanz zu arbeiten ohne die Lesbarkeit des Gesamtalgorithmus merklich zu verschlechtern.

3.3 "Anytime"-Eigenschaft

Der Algorithmus ist "jederzeit". Das heißt konkret, dass die Iterationen jederzeit abgebrochen werden können und das Ergebnis (der Baum samt Bewertungen) trotzdem brauch-

bar ist. Bei einem reinen Alphabeta-Algorithmus wäre das Ergebnis nach einem Abbruch höchstwahrscheinlich unbrauchbar. Zu diesem Zweck wurde für Alphabeta zwar die Technik des Iterative Deepening erfunden[10]. Nichtsdestotrotz sind die Ergebnisse der letzten Iteration unbrauchbar und da sie wegen exponentiellem Rechenzeitanstieg die meiste Rechenzeit in Anspruch genommen hat, ist der Großteil der Rechenzeit "verschwendet"[11]. Bei UCT ist dies alles nicht der Fall.
Die Möglichkeit den Algorithmus quasi jederzeit abbrechen zu können ist nicht nur generell bei Zeitdruck vorteilhaft. Es ist auch denkbar, dass der Algorithmus versucht komplizierte und einfache Stellungen zu erkennen und ihnen entsprechend eine größere oder kleinere Anzahl an Simulationen zugesteht. Außerdem ist es mit Pondering (vgl. Abschnitt 5.7) möglich "mitzudenken" solange der Gegner über seinen Zug nachdenkt und die Berechnungen abzubrechen, sobald er seinen Zug macht.

3.4 Ruhesuche

Im Alphabeta-Algorithmus besteht das sog. Horizontproblem[12]. Behoben wird es mit der sog. Rugesuche: Varianten, die dazu neigen, das Ergebnis der Partie stark zu beeinflussen (z.B. Schlagzüge), werden noch einmal tiefer als normal expandiert, bis die Stellung sich "beruhigt" hat.
In UCT besteht dieses Problem grundsätzlich nicht, da alle Varianten bis zu Terminalpositionen hin ausgespielt werden. Sollte einem auffallen, dass bestimmte wichtige Varianten von den Simulationen zu selten berücksichtigt werden, so sollte auf jeden Fall die Integration einer Simulationsstrategie (s. Abschnitt 5.1) in Erwägung gezogen werden. Diese sollte dann die Simulation der bisher zu häufig "durchs Raster gerutschten" Varianten besonders hoch gewichten.

3.5 Strategie vs. Taktik

Wie bereits in Abschnitt 2.1 erwähnt, neigen Monte-Carlo-Methoden dazu strategisch stark und taktisch schwach zu spielen. UCT gleicht dieses Problem bis zu einem gewissen Grad aus, da es selektiv die "interessanten" (die guten und die vielversprechenden) Varianten untersucht. Dennoch ist es gut möglich, dass der Algorithmus in Reinform große taktische Schwächen aufweist.
Auch dies kann durch Simulationsstrategien eliminiert werden. Außerdem kann auch der Selektionsteil (und speziell die Selektionsformel) erweitert werden. Beispiele sind Progressive Bias (Abschnitt 5.5) und Patterns (Abschnitt 5.1). Manche Ansätze (z.B. Progressive Unpruning (Abschnitt 5.4)) kommen dabei sogar ohne Expertenwissen aus.

10. Der Baum wird zunächst mit Suchtiefe 1 durchsucht, anschließend mit Tiefe 2 usw.. Die Ergebnisse einer Iteration können benutzt werden, um in der nächsten Iteration eindeutig schlechte Äste abzuschneiden und die Suche zu beschleunigen. Wird der Algorithmus abgebrochen gilt das Ergebnis der letzten vollständigen Iteration.
11. Natürlich sind die Berechnungen nicht ganz vergeblich. Bspw. wurden durch die letzte Iteration die transposition tables mit wertvollen Informationen gefüllt.
12. Der Algorithmus neigt dazu unvermeidbare aber nach hinten verschiebbare schlechte Züge (z.B. einen drohenden Damenverlust) über seinen Suchhorizont hinauszuschieben, sodass er sie nicht mehr sieht.

3.6 Ergebnisse in realen Systemen

In (Billings et al., 1999) wurde der Austausch eines wissensbasierten durch einen suchintensiven Ansatz (MC mit Simulationsstrategie aber ohne UCT) im Pokerprogramm Loki beschrieben. Das Ergebnis war eine signifikante Verbesserung (mind. 0.05 sb/hand).
UCT selbst wurde bisher auch in diversen Anwendungsfällen erprobt: Seine Entdeckung führte zu einer starken Beachtung von MC-Methoden im Computer-Go. UCT-basierte Programme, sind fähig mit wissensbasierten Programmen, in denen mehrere Mannjahre Entwicklung stecken, mitzuhalten. In 9x9-Go sind fünf der sechs besten Programme UCT-basiert, in 19x19-Go dominieren im Moment noch wissensbasiertn Programme, den UCT-Programmen wird aber ein großes Verbesserungspotenzial prognostiziert (Bouzy, 2007).
In (Kocsis & Szepesvári, 2007) wird die Leistungsfähigkeit UCTs an eher künstlichen Problemen erprobt: Das erste Experiment waren zufällige Spielbäume: Spielbäume von Zweipersonenspielen (deterministisch und mit vollständiger Information) werden zufällig erzeugt und ihr Wert muss von UCT ermittelt werden. UCT hat sich hierbei als Alphabeta überlegen herausgestellt, obwohl Alphabeta bisher der Standardansatz für deterministische Spiele mit vollständiger Information war. Auch legen die Ergebnisse nahe, dass UCT schneller als $o(B^{D/2})$ (B ist der Verzweigungsfaktor und D die Tiefe des Baums) konvergiert. Es lassen sich also (wie bei Alphabeta vs. Minimax) doppelt so tiefe Spielbäume evaluieren wie mit reinem MC-Ansatz. Das zweite Experiment war das Stochastic Shortest Path Problem: Ein Segelboot muss unter wechselnden Windbedingungen ein Ziel erreichen. Hier war UCT insbesondere bei großen Problemen (kleine Planungsraster-Größe) wesentlich erfolgreicher als die bisher auf dieses Problem speziell zugeschnittenen Algorithmen. Bei den großen Problemen waren die herkömmlichen Algorithmen zumeist garnicht fähig in akzeptabler Zeit ein Ergebnis zu liefern.

4. Relevanz für Poker

4.1 Nötige Adaptionen

Für Pokern müssen ein paar Adaptionen gemacht werden [13]: Es muss unvollständiges Wissen über den aktuellen Spielstand (verdeckte Karten etc.), Nichtdeterminismus (zufälliges Austeilen von Karten) und die Beteiligung mehrerer Spieler berücksichtigt werden.

4.1.1 Nichtdeterminismus

Die Integration von Nichtdeterminismus ist vergleichsweise einfach. Hierzu werden in den Spielbaum Ebenen zwischen den Ebenen, zwischen denen nichtdeterministische Aktionen stattfinden, eingezogen[14]. Es muss allerdings beachtet werden, dass in der Selektion und der Simulation für die Kanten, die nichtdeterministische Aktionen repräsentieren, keine

13. Diese sind z.T. in (Kocsis & Szepesvári, 2007) in der abstrakten Darstellung als Markovian Decision Problem schon berücksichtigt
14. Wenn z.B. Karten gegeben werden und anschließend eine Entscheidung eines Spielers ansteht, wird eine weitere Ebene eingezogen. Die Kanten zu dieser Ebene hin stellen das Geben der Karten und die Kanten von dieser Ebene weg die Entscheidungen des Spielers dar.

Heuristiken (vgl. Abschnitt 5.1) verwendet werden. Die Wahl der "nichtdeterministischen Kanten" muss mit der selben Wahrscheinlichkeit geschehen wie im realen Spiel[15].

4.1.2 Unvollständiges Wissen

Die einfachste Möglichkeit dies zu berücksichtigen, ist es das fehlende Wissen wie bei einer nichtdeterministischen Aktion "auszuwürfeln". Danach kann es als bekannt vorausgesetzt werden. Zum Beispiel kann direkt am Anfang des Spielbaums wie oben beschrieben eine "nichtdeterministische Ebene" eingezogen und die Karten der Gegenspieler zufällig festgelegt werden.
Kompliziertere Ansätze führen z.B. Protokoll über den bisherigen Spielverlauf (sowohl das Spielen vorheriger Hände als auch das Spielen der aktuellen Hand) und ermitteln damit ein Wahrscheinlichkeitsprofil der Karten jedes Spielers. In (Billings et al., 1999) wird dieser Ansatz verfolgt.

4.1.3 Mehrspielerspiel

Klassische Algorithmen berücksichtigen nur Zweipersonennullsummenspiele[16]. Eine Möglichkeit zur Erweiterung für mehrere Spieler ist folgende: Zunächst werden natürlich Ebenen für die Entscheidungen jedes Spielers im Spielbaum vorgesehen (zzgl. der "nichtdeterministischen Ebenen" (s.o.)). Die Funktion, die die Terminalpositionen bewertet, muss nun statt positiven/negativen Werten für gewonnene/verlorene Spiele Tupel, die die Gewinne und Verluste jedes Spielers enthalten, zurückgeben. Im Backpropagating wird dann das gesamte Tupel nach oben weitergereicht und auf das bisherige Tupel aufaddiert. In der Selektion muss nun nicht mehr der Durchschnittswert der Partie (und der Zuversichtlichkeitsbonus) maximiert werden, sondern ein anderer Ausdruck. Hier bietet sich die Differenz zwischen dem eigenen Gewinn und dem summierten oder durchschnittlichen Gewinn der Gegenspieler an. Im Spezialfall mit zwei Spielern entspricht dies wieder der ursprünglichen Maximierung.

4.2 Implizites Wissen

Einer der Hauptvorzüge von Monte-Carlo-Ansätzen allgemein (und natürlich speziell UCT) ist wie bereits erwähnt, dass grundsätzlich kein Wissen über das Spiel nötig ist.
In (Billings et al., 1999) wird bereits hervorgehoben, dass die Modellierung von Konzepten wie Handstärke, Handpotenzial und "Pot odds", sowie auch subtilere Konzepte wie "implied odds", nicht explizit geschehen muss. Sie sind bereits implizit in der Simulation enthalten. Starke Hände werden sich in der Mehrzahl der Simulationen auch als stark herausstellen und hoch bewertet werden. Für Hände mit Potenzial trifft das selbe zu.
Der Vorteil dieses impliziten Wissens ist wie bereits erwähnt leichte Kapselbarkeit des Expertenwissens, kurze Entwicklungszeiten, leichte Erweiterbarkeit, geringer Wartungsaufwand, keine Spielstärkenbeschränkung durch die Stärke des integrierten Expertenwissens uvm..

15. Es sei denn man möchte auf diese Weise Spielerprofile integrieren (vgl. Abschnitt 5.2)
16. Der Gewinn des einen Spielers ist der Verlust des anderen.

5. weitere Anwendungs- und Erweiterungsmöglichkeiten

5.1 Simulationsstrategien und Integration von Offline-Wissen

Simulationsstrategien sind scheinbar die Hauptforschungsrichtung in der momentan versucht wird UCT zu erweitern. Sie lassen die MC-Simulation an den Blättern des Baums nicht rein zufällig ablaufen, sondern steuern sie basierend auf handcodiertem Expertenwissen oder zuvor automatisch generiertem Wissen (Offlinewissen). Diese Technik ist natürlich vollständig von der jeweiligen Domäne (Poker, Go, ...) abhängig.
In Go können dies bspw. (automatisch aus Spieldatenbanken oder per Hand von Experten erzeugte) Zugpatterns, die Anzahl an gefangenen Steinen, die Nähe zum vorherigen Zug und vieles mehr sein (siehe (Chaslot et al., 2007; Gelly & Silver, 2007; Gelly et al., 2006) u.v.a.). Auch andere statische Ansätze, die in der Zeit vor UCT entstanden sind, wie gezüchtete KNNs zur automatischen Zuggenerierung (Donnelly, Corr, & Crookes, ; Richards, Moriarty, McQuesten, & Miikkulainen, 1997), sind denkbar und leicht integrierbar.
Strategien zur Simulation wurden für Poker bereits in (Billings et al., 1999) vorgestellt. Dort wird allerdings noch eine reine MC-Simulation ohne UCT durchgeführt. Die Ergebnisse sind trotz dessen allerdings schon beachtlich. Strategien können nicht nur in der Simulation sondern wie bei Progressive Bias (s.u.) auch in der Selektion benutzt werden.
Da die Verwendung von heuristischem Wissen zu einer Stellung meist sehr rechenintensiv ist, müssen Rechenzeit und Leistungsverbesserung stets vorsichtig abgewogen werden. Auch kann man (wie dies in (Chaslot et al., 2007) geschieht) die Berechnung gewisser Teile der Heuristik an- oder abschalten, je nachdem wie wichtig eine korrekte heuristische Evaluation des aktuellen Knotens ist.

5.2 Integration von Spielerprofilen

Die Integration von Spielerprofilen kann auf vielfache Weise geschehen. Beispielsweise kann in der normalen UCT-Selektionsformel (Abschnitt 2.2.1) ein weiterer Summand hinzugefügt werden, der wissensbasiert je nach Spieler unterschiedliche Züge bevorzugt. Auch ist es denkbar je nach Spieler die Werte von Knoten zu modifizieren, um Optimismus/Pessimismus oder Gier/Vorsicht zu modellieren. In der Simulationsstrategie kann ein Spielerprofil integriert werden, indem ebenfalls je nach Spieler bestimmte Züge bevorzugt werden (sowohl für "deterministische" als auch für "nichtdeterministische Kanten", vlg. Abschnitt 4.1.1).

5.3 Parallelisierbarkeit

In (Cazenave & Jouandeau, 2007) wird die Parallelisierbarkeit von UCT behandelt. Die einfachste Möglichkeit ist es, mehrere Prozessoren unabhängig an der selben Stellung rechnen zu lassen und am Ende die jeweiligen Ergebnisse an der Wurzel und ihren Kindknoten zu kombinieren und davon ausgehend den Zug zu wählen. Dieser Ansatz benötigt sehr wenig Kommunikation der Algorithmen untereinander und umgeht das Problem eines gemeinsamen Spielbaums, da jeder Prozessor einen eigenen pflegt.
Neben dieser sog. "Single-Run"-Parallelisierung werden auch noch "Multiple-Runs" und "At-the-leaves" vorgeschlagen. Sie resultieren in einer höheren Spielstärke, erfordern aber mehr Kommunikation der Prozessoren.

5.4 Progressive Widening / Unpruning

In (Chaslot et al., 2007) wird die Technik des Progressive Unpruning vorgeschlagen: Wurde ein Knoten weniger als T mal besucht ($n_p < T$), werden nacheinander alle seine Kinder erweitert und simuliert. Ist $n_p = T$, werden zunächst die meisten der Kinder temporär abgeschnitten. Es wird also angenommen, dass Zeitmangel herrscht und nur wenige Knoten werden weiter untersucht. Steht irgendwann dann doch wieder mehr Zeit zur Verfügung ($n_p > T$) werden wieder sukkzessive Knoten hinzugefügt (Unpruning) und simuliert. Ist irgendwann $n_p >> T$ sind wieder alle Knoten vorhanden.

5.5 Progressive Bias

Ebenfalls in (Chaslot et al., 2007) vorgeschlagen wird Progressive Bias: Wie in der Simulationsstrategie soll auch in der Selektion die Wahl von heuristisch als gut bewerteten Stellungen bevorzugt werden. Der Einfluss dieser Heuristik soll jedoch mit steigender Zahl an Simulationen, die durch diesen Knoten gelaufen sind, an Bedeutung verlieren, da dann eher das Ergebnis der Simulationen akkurater als die Heuristik ist. Hierzu wird der normalen UCT-Selektionsformel ein Summand $f(n_i) = \frac{H_i}{n_i+1}$ hinzugefügt. H_I stellt das heuristische Wissen dar. Mit steigender Simulationszahl n_i wird $f(n_i)$ bedeutungsloser.
Insbesondere in Kombination mit Progressive Unpruning hat diese Technik beachtliche Ergebnisse geliefert. Gegen ein Vergleichsprogramm (GnuGo 3.6) stieg die Siegrate im 19x19-Go von 0% auf 48.2%.

5.6 Transposition Tables

(Bouzy, 2007) schlägt eine Technik vor, die schon in der Schach-Programmierung benutzt wurde: Transposition Tables. Sowohl in Poker als auch in Go gibt es häufig äquivalente Knoten im Spielbaum. Bspw. kann ein und die selbe Stellung auch durch die Vertauschung der Zugabfolge erreicht werden. Man kann daher Hashtabellen benutzen, in denen die Ergebnisse vorheriger Knotenevaluationen gespeichert werden. Diese Ergebnisse werden dann an anderen Stellen im Baum wiederbenutzt. Hashtabellen sollte man allerdings vorsichtig benutzen. Bspw. sollte man nur wichtige (häufig besuchte) Knoten speichern, damit die Hashtabelle sich nicht zu sehr mit unrelevanten Daten füllt und auch ein Nachschlagen in der Tabelle sollte nicht zu häufig geschehen (z.B. nur nahe der Wurzel).

5.7 Pondering

Eine weitere Technik aus der Schachprogrammierung ist das Pondering: Damit die Zeit, in der der Gegenspieler denkt, nicht ungenutzt verstreicht, wird auch in dieser Zeit gerechnet. Auf Grund der “Anytime”-Eigenschaft (vgl. Abschnitt 3.3) kann, sobald der Gegner zieht, problemlos die Berechnung abgebrochen und mit der eigentlichen Berechnung begonnen werden.
Im einfachsten Fall benutzt man dann in der eigentlichen Berechnung nur die während des Pondering angefüllten Transposition Tables. In UCT wäre es jedoch auch denkbar, dass man den beim Pondering aufgebauten Baum wiederverwendet und die Wurzel und die durch den Zug des Gegenspielers nicht gewählten Kindknoten der Wurzel abschneidet.

5.8 Killer/History Heuristics

Ebenfalls aus Schach bekannt sind Killer/History Heuristics: Züge, die in einem Teil des Baums gut waren, sind möglicherweise auch in anderen Teilen des Baums erfolgreich. Daher kann eine Liste von erfolgreichen Zügen mitgeführt werden und diese in der Selektion oder Simulation bevorzugt werden.

5.9 "Mercy" rule

(Bouzy, 2007) schlägt die sog. "Mercy" rule vor: Ist in einem Simulationslauf das Ergebnis für einen der beiden Spieler bereits hinreichend schlecht, wird diese Variante als für den anderen Spieler gewonnen angenommen und die Simulation an diesem Punkt nicht mehr weitergespielt.
Diese Heuristik lässt sich allerdings nicht ohne weiteres auf Poker anwenden.

5.10 Weitere Anwendungsmöglichkeiten

In (Chung, Buro, & Schaeffer, 2005) wird eine sehr interessante Anwendung von MC-Methoden vorgeschlagen: Planung in Echtzeitstrategiespielen. Das Konzept von abwechselnden Zügen der Spieler wird hierzu aufgeweicht. Die Spieler können quasi gleichzeitig Züge durchführen und die Autoren schlagen vor, MC-Simulation in allen Hierarchien der Spielplanung einzusetzen (Micro Management, Schlachttaktik, Gesamtpartiestrategie). Für hohe Abstraktionsebenen (z.B. zur Planung der Gesamtstrategie) werden hierzu Elementarzüge (z.B. Vorwärtsbewegen einer Einheit um ein Feld) zu größeren Zügen zusammengefasst. Dies kann entweder vorab durch einen Programmierer oder während des Spiels automatisch (z.B. Zusammenfassung mehrerer Elementarbewegungen durch einen Wegfindungsalgorithmus) oder auch randomisiert geschehen.
Die Ergebnisse waren teilweise noch handgecodeten Programmen unterlegen, der Ansatz selbst ist aber vielversprechend und er wurde noch nicht in Kombination mit UCT erprobt. Die Brauchbarkeit von UCT scheint mit Spielen wie Go und Poker also noch lange nicht erschöpft zu sein und vermutlich wurden viele Anwendungsfelder bisher noch nicht einmal in Betracht gezogen.

Appendix A. Alpha-Beta-Algorithmus

Alphabeta stellt den Standardalgorithmus für Spiele wie z.B. Schach dar. Er wird hier nur kurz zur Vollständigkeit erwähnt. Der folgende Pseudocode entstammt dem Wikipediaartikel (Wikipedia, 2008).

```
int NegaMax(int tiefe, int alpha, int beta){
    if (tiefe == 0)
        return Bewerten();
    GeneriereMoeglicheZuege();
    while (ZuegeUebrig()){
        FuehreNaechstenZugAus();
        wert = -NegaMax(tiefe-1, -beta, -alpha);
        MacheZugRueckgaengig();
```

```
        if (wert >= beta)
            return beta;
        if (wert > alpha)
            alpha = wert;
    }
    return alpha;
}
```

Der Spielbaum wird bis zu einer festen Tiefe expandiert und die Blätter mit einer statischen Bewertungsfunktion bewertet. Anschließend werden die Bewertungen im Baum nach oben durchgereicht, indem in jeder Ebene der Wert des jeweils besten Zugs des in dieser Ebene am Zug befindlichen Spielers nach oben zurückgegeben wird (**alpha** stellt hier den Wert des besten Zugs dar). Dies an sich ist noch äquivalent zum Minimax-Algorithmus. Der Trick besteht in der Zeile **if (wert >= beta) return beta;**. Hier werden Äste des Baums abgeschnitten, die garnicht durch perfektes Spiel auf beiden Seiten zustande gekommen sein können und damit nicht betrachtet werden müssen.

Appendix B. Multiarmed-Bandit-Problem, Exploration vs. Exploitation und UCB

Das Multiarmed-Bandit-Problem sieht folgendermaßen aus: Der Spieler steht vor mehreren Kasinoautomaten, die alle eine unterschiedliche Gewinnwahrscheinlichkeit haben, die man nicht kennt und das Ziel ist es in mehreren Spielen einen möglichst hohen Gewinn zu erzielen. Das Dilemma besteht nun darin entweder am Automaten zu spielen, der bisher die höchste Gewinnwahrscheinlichkeit hatte (Exploitation) oder an anderen Automaten zu spielen, um festzustellen, dass diese eventuell doch eine höhere Gewinnwahrscheinlichkeit haben (Exploration). UCB löst dieses Problem indem es stets den Automaten mit der höchsten upper confidence bound wählt (für Details siehe (Kocsis & Szepesvári, 2007)).
In Spielbäumen besteht nun ein ähnliches Dilemma: Entweder kann der Algorithmus die bisher beste Variante im Baum genauer und tiefer evaluieren (Exploitation) oder er versucht rauszufinden, ob es nicht eventuell doch noch bessere Varianten als die bisher beste gibt (Exploration). In (Kocsis & Szepesvári, 2007) wird daher UCB auf Bäume zu UCT erweitert und bewiesen, dass die Konvergenz des Algorithmus hin zum optimalen Resultat erhalten bleibt.

References

Billings, Papp, Lourdes, Schaeffer, & Szafron (1999). Using Selective-Sampling Simulations in Poker. In *Proceedings of the AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information.*

Bouzy (2004). Associating shallow and selective global tree search with Monte Carlo for 9x9 Go. In *Proceedings Fourth International Conference on Computers and Games.*

Bouzy (2007). Old-fashioned Computer Go vs Monte-Carlo Go.. Online verfügbare Präsentation: `http://ewh.ieee.org/cmte/cis/mtsc/ieeecis/tutorial2007/Bruno_Bouzy_2007.pdf`.

Bouzy, & Helmstetter (2003). Monte Carlo Go developments. In *Proceedings of the 10th Advances in Computer Games Conference*.

Brügmann (1993). Monte Carlo Go..

Cazenave, & Jouandeau (2007). On the Parallelization of UCT. In *Proceedings CGW 2007*.

Chaslot, Winands, Uiterwijk, van den Herik, & Bouzy (2007). Progressive strategies for monte-carlo tree search. In *Proceedings of the 10th Joint Conference on Information Sciences*.

Chung, Buro, & Schaeffer (2005). Monte Carlo planning in RTS games. In *Proceedings of CIG 2005*.

Coulom (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings Computers and Games (CG-06)*. Springer.

Donnelly, Corr, & Crookes. Evolving Go playing strategy in neural networks..

Gelly, & Silver (2007). Combining online and offline knowledge in UCT. In *Proceedings ICML 2007*.

Gelly, Wang, Munos, & Teytaud (2006). Modifications of UCT with Patterns in Monte-Carlo Go..

Kocsis, & Szepesvári (2007). Bandit based Monte-Carlo Planning. In *Proceedings ECML-07*. Springer.

Richards, Moriarty, McQuesten, & Miikkulainen (1997). Evolving neural networks to play Go. In *Proceedings of the 7th International Conference on Genetic Algorithms*.

Wikipedia (2008). Alpha-Beta-Suche.. Der deutsche Wikipediaartikel zum Alphabeta-Algorithmus `http://de.wikipedia.org/wiki/Alpha-Beta-Suche`; Stand 30. März 2008, 14:15.