Abbildung eines 2-Spieler Computerspielers auf 3-Spieler-Limit-Poker

Bachelor-Thesis von Peter Glöckner aus Aschaffenburg
Mai 2013

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering

Abbildung eines 2-Spieler Computerspielers auf 3-Spieler-Limit-Poker

Vorgelegte Bachelor-Thesis von Peter Glöckner aus Aschaffenburg

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 24.Mai 2013

_____

(Peter Glöckner)

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Spiele stellen für die Informatik eine gute Möglichkeit dar, die Funktionsweise von bestimmte Verfahren anschaulich darzustellen und deren Leistung zu evaluieren. Das Studieren von Spielen ist auch deshalb interessant, da man je nach Spiel oft vor komplexe Probleme gestellt wird. Hat man es sich zum Beispiel zum Ziel gesetzt, ein möglichst gutes Schachprogramm zu entwickeln, kommt man mit einem naiven Ansatz nicht weit. Ein Programm, dass optimal Schach spielt, ist zwar schnell geschrieben, die verfügbare Rechenleistung lässt eine praktische Benutzung allerdings nicht zu. In dieser Arbeit soll es zwar nicht um Schach, sondern um die Pokervariante *Texas Hold'em* gehen, allerdings stellt auch im Computerpoker die Komplexität ein Problem dar.

Darüber hinaus besitzt Poker weitere Eigenschaften, die die Entwicklung von guten *Bots* erschweren.

- ***Multiplayer***: Obwohl mehr Spieler möglich sind, variiert die Spielerzahl zumeist zwischen zwei und zehn Personen. Wettbewerbe für Computerpoker sind momentan noch auf zwei beziehungsweise drei Spieler begrenzt.

- Im Poker erhält jeder Spieler zu Beginn einer Runde zwei nur für ihn sichtbare Karten, welche als *Handkarten* bezeichnet werden. Somit ist ein Spielzustand für einen Spieler **nicht vollständig beobachtbar**.

- Neben den *Handkarten* existieren in einer Runde bis zu fünf für alle sichtbare Karten, welche als *Tischkarten* bezeichnet werden. Da die Karten immer zufällig verteilt werden, ist Poker ein **nicht deterministisches** Spiel.

Logischerweise wird nicht nur das Entwickeln sondern auch das Evaluieren der Bots durch die genannten Eigenschaften erschwert. Ein paar wenige gespielte *Hände* reichen nicht aus, um zu entscheiden, wer der stärkere Spieler ist. Das wird deutlich, wenn man berücksichtigt, dass der Nichtdeterminismus dazu führt, dass eine richtige Entscheidung bestraft und eine falsche Entscheidung vom Spiel belohnt werden kann.

Unter anderem sollte man auch bedenken, dass auch am Ende einer *Hand*, meistens die *Handkarten* der Spieler nicht für alle sichtbarer gemacht werden, was es noch einmal schwieriger macht zu entscheiden, ob eine getätigte Aktion die Richtige war.

Wie der Titel verrät, soll in dieser Arbeit ein zwei Spieler *Bot* in einem drei Spieler Spiel verwendet werden, obwohl ein zwei Spieler *Bot* im Allgemeinen nicht mit einem drei Spieler Spielzustand umgehen kann. Ziel dieser Arbeit ist es festzustellen, ob durch eine geeignete Abbildung der Spielzuständen ein zwei Spieler *Bot* in einem drei Spieler Spiel sinnvoll verwendet werden kann. Der dazu verwendete zwei Spieler *Bot* spielt nach einer *nash-optimalen* Spielweise.

Leider ist die exakte Bestimmung einer solchen Strategie schon im zwei Spieler Spiel aufgrund der Komplexität von Poker nicht in akzeptabler Zeit möglich und nur theoretisch durchführbar. Allerdings existiert mit Counterfactual Regret Minimization (CFR) ein bereits implementiertes Verfahren, welches relativ effizient eine gute approximative Lösung für ein *Strategieprofil* liefert.

Es gibt mehrere Gründe einen *Bot* zu verwenden, der auf auf diesem Verfahren beruht. Zu erst spart man sich die Berechnungszeit, die sich bei gleicher Genauigkeit mit steigender Anzahl von Spielern stark erhöht.

Zum anderen kann man für diese Art von *Bot* leicht überprüfen, ob trotz der Abbildung seine Eigenschaften erhalten wurden. Im zwei Spieler Spiel zeichnet er sich durch eine robuste Spielweise aus. Sollte er also durch eine Abbildung in einem 3 Spieler Spiel verwendet werden und des öfteren stark ausgenutzt werden, wissen wir, dass wir unser Ziel nicht erreicht haben.

Des Weiteren wurde in [9] gezeigt, dass man mit CFR gute Strategien für ein *Multiplayer* Spiel generieren kann, allerdings die Garantie verliert, dass das Verfahren gegen eine *nash-optimale* Spielweise konvergiert.

1.2 Aufbau der Bachelorarbeit

Da *Texas Hold'em* ein sehr komplexes Spiel ist, beschränkt sich die Annual Computer Poker Competition momentan noch auf Wettbewerbe für das zwei beziehungsweise drei Spieler Spiel und zwar jeweils für *Fixed Limit* und *No Limit*. Die Spielregeln, die für das Verständnis dieser Arbeit wichtig sind, werden in Abschnitt 2.1 erklärt. Hier kann man allerdings schon festhalten, dass die Variante *Fixed Limit* weniger komplex ist, da die möglichen Aktionen eines Spieler begrenzter sind. Zu dem wird in Kapitel 2.2 auf allgemeine Probleme bei der Evaluierung eingegangen.

Das Counterfactual Regret Minimization Verfahren, welche die Strategie generiert, nach der unser 2 Spieler *Bot* spielt, wird in Kapitel 3 vorgestellt und ist eine Zusammenfassung der Beschreibung von [7]. In Abschnitt 3.2 wird verdeutlicht, warum wir mit CFR in einem drei Spieler Spiel keine nash-optimal Spielweise erreichen. Abschnitt 3.3 ist ein Überlick über einige verwandte Arbeiten, wobei sich die meisten mit Verbesserungen von CFR beschäftigen.

In Kapitel 4 werden die ersten Ansätze für Abbildungen vorgestellt. Das Kapitel ist in eine Einleitung, jeweils ein Abschnitt für die Abbildungen, eine Evaluierung und ein Fazit unterteilt.

Aufgrund der Ergebnisse wird in Kapitel 5 eine neue Abbildung vorgestellt und die bestehenden Abbildungen verbessert. In Abschnitt 5.4 sind einige kleinere umgesetzte Ideen beschrieben.

Kapitel 6 geht allgemein auf die Probleme bei der Entwicklung von Abbildung ein und Abschnitt 7 fasst die Ergebnisse der Arbeit zusammen.

2.1 Spielregeln

Dieser Abschnitt orientiert sich inhaltlich an [12].

Poker wird mit einem Kartenspiel von 52 Karten gespielt. Obwohl mehr Spieler möglich sind, variiert die Spieleranzahl zumeist zwischen 2 und 10 Personen. Jede Karte definiert sich über einen Wert und eine Farbe und ist nur einmal im Kartenstapel vorhanden.
Werte[1]: W = {2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A}
Farben[2]: F = {d, h, s, c}
Karten: K = W x F
|K| = 52

Anhand der Sitzreihenfolge am Tisch wird für jedem Spieler eine Position ermittelt. In einem 3-Spieler Spiel existiert die Position *Button* (d), *Small Blind* (sb) und *Big Blind* (bb).
Positionen: P = {d, sb, bb}

Per Zufallsentscheidung wird vor der ersten *Hand* einem Spieler die Position *Button* zugewiesen. Der Spieler links des *Buttons* erhält die Position *Small Blind*, der Spieler links des *Small Blinds* die Position *Big Blind*. Bevor die Runde beginnt müssen die Spieler in der Position *Small Blind* und *Big Blind* einen festgesetzten Einsatz setzen. Gesetzte Einsätze werden vom Guthaben eines Spielers abgezogen und dem sogenannten *Pot* hinzugefügt. Der Einsatz des *Big Blinds* entspricht dabei dem doppelten des *Small Blinds*. Nach einer *Hand* verschieben sich die Positionen im Uhrzeigersinn.[3]

Nachdem jeder Spieler zwei nur für ihn sichtbare Karten erhalten hat, beginnt die erste der vier Wettrunden. Die erste Aktion wird in der ersten Wettrunde (*Preflop*) vom Spieler links des *Big Blinds*, in den folgenden Wettrunden vom Spieler links des *Buttons*, ausgeführt. Er kann entweder

- mitgehen (*call*), also den Einsatz des vorherigen Spielers setzen,
- erhöhen (*raise*), also einen höheren Betrag als der vorherige Spieler setzen, oder
- aussteigen[4] (*fold*).

Aktionen: A = {c,r,f}

Nacheinander im Uhrzeigersinn müssen nun alle Spieler eine dieser 3 Aktionen wählen. Nachdem jeder Spieler an der Reihe war, endet die Wettrunde, wenn ein Spieler erneut an der Reihe wäre und die zuletzt getätigten Aktionen seiner Gegner entweder *fold* oder *call* waren.

Zu Beginn der zweiten Wettrunde (*Flop*) werden drei Karten aufgedeckt in die Mitte gelegt, zu Beginn der dritten (*Turn*) und vierten Wettrunde (*River*) jeweils eine. Die aufgedeckten Karten werden als *Tischkarten* bezeichnet. Ansonsten unterscheiden sich die Wettrunden nicht. Eine *Hand* endet, wenn die vierte Wettrunde abgeschlossen ist oder alle bis auf ein Spieler ausgestiegen sind.

Der Gewinner einer Runde erhält den *Pot*, der allen gesetzten Einsätzen der Spieler entspricht. Im Fall, dass alle Spieler bis auf einer ausgestiegen sind, hat der verbleibende Spieler

[1] Englische Abkürzungen für jack, queen, king, ace
[2] Englische Abkürzungen für diamonds, hearts, spades, clubs
[3] Im 2 Spieler Spiel existieren die Positionen Button und Big Blind. Der Button bezahlt den Einsatz, den der Small Blind bezahlt hätte.
[4] ein ausgestiegener Spieler kann keine Aktionen mehr tätigen und hat die aktuelle *Hand* verloren

die aktuelle *Hand* gewonnen. Sollte allerdings die vierte Wettrunde enden und mehr als zwei Spieler nicht ausgestiegen sein, kommt es zum *Showdown*. Im *Showdown* werden die Stärken der *Hände* der verbliebenen Spielern bewertet. Der Spieler mit der stärksten *Hand* gewinnt den *Pot*.

In dieser Arbeit wird *Texas Hold'em Fixed Limit* gespielt. Das bedeutet, dass bei einem *raise* die Erhöhung immer fest vorgeschrieben ist. In den ersten beiden Wettrunden beträgt diese immer ein *Big Blind*, in den anderen beiden Wettrunden zwei *Big Blinds*. Des Weiteren ist die maximale Anzahl der Aktion *raise Preflop* auf drei, in den anderen Wettrunden auf vier begrenzt.

Durch diese Spieleigenschaft und da im Computerpoker die Spieler immer über unbegrenzt viel Geld verfügen, muss hier der Fall, dass ein Spieler *all in* ist nicht betrachtet werden.

Demzufolge definiert sich ein Spielzustand über die von den Spielern gehaltenen Karten, die bisher erbrachten Aktionen und die bereits aufgedeckten *Tischkarten*.

Alle anderen Werte wie zum Beispiel Größe des *Pots* lassen sich aus diesen Informationen berechnen.

$$S = ([\{(Q,h),(Q,d)\}, \{(J,h),(J,d)\}, \{(A,s),(K,s)\}], \{(Q,s),(J,s),(10,s)\}, \{\}, \{\}, [r,c,c], [r], [], [])$$

Der hier aufgeführte Spielzustand S wird folgendermaßen interpretiert. Das erste Element des Tupels entspricht den Karten, die von den Spielern gehalten werden. Diese werden als Liste angegeben, wobei das erste Element, den Karten des *Small Blinds*, das zweite den Karten des *Big Blinds* und das dritte den Karten des *Buttons* entsprechen.

Die nächsten Elemente des Tupels entsprechen den *Tischkarten* in der Reihenfolge der Wettrunden, die darauf folgenden Elemente den Aktionen der Spieler ebenfalls unterteilt und aufgeführt in der Reihenfolge der Wettrunden.

2.2 Evaluierung

Um seine *Bots* zu testen, lässt man sie gegen andere Benchmark *Bots* in einem Turnier antreten. Das in dieser Arbeit verwendete Turnierformat entspricht dabei den offiziellen Regeln der Annual Computer Poker Competition[1] und wird in einem späteren Abschnitt genauer erläutert. In diesem Abschnitt soll nur auf die allgemeinen Probleme bei der Evaluierung eingegangen werden, wodurch später der Sinn der einzelnen Turnierregeln ersichtlich werden sollte.

Wie wahrscheinlich bekannt wird im Poker um Geld beziehungsweise im Computerpoker um Punkte gespielt. Der Gewinn eines Spieler entspricht dabei immer den Verlusten der anderen Spieler. Da sich der Gewinn in den einzelnen Runden sehr unterscheiden kann, misst man die Stärke eines Spielers nicht an der Zahl der gewonnen *Hände*, sondern am Kontostand am Ende eines Turniers. Das Konto wird im Poker als *Bankroll* bezeichnet.

Grunsätzlich kann man Pokerbots in zwei Klassen unterteilen. Ein *Bot* des ersten Typs versucht seine *Bankroll* dadurch zu maximieren, indem er die Aktionen seines Gegners beobachtet, um Fehler in seiner Strategie aufzudecken und diese auszunutzen. Der Erfolg einer solchen Strategie kann stark variieren. Trotzdem könnte solch ein *Bot* allein durch das starke Ausnutzen von sehr schwachen *Bots*, obwohl er gegen die meisten Gegner schlecht abschneidet, in einem Turnier ein hohe Platzierung erreichen. Um diesem Problem vorzubeugen wird bei der Annual Computer Poker Competition neben dem Verfahren, welches nur die akkumulierten Gewinne der Spieler in den einzelnen Partien vergleicht, ein zweites Verfahren zur Ermittlung des Gewinners verwendet. Dadurch sollen auch geheime Absprache, wie dass zwei Teilnehmer einen

Bot und einen *Bot*, der nach einer Konterstrategie zum ersten spielt, einreichen, verhindert werden.

Der andere Typ, der vor allem für diese Arbeit wichtig ist, betrachtet bei der Bestimmung seiner Strategie den Gegner nicht. In diesem Zusammenhang wird der Begriff der *nash-optimalen* Spielweise wichtig. Der Grundgedanke hierbei ist es nicht möglichst viel Gewinn zu erzielen, sondern die eigenen Verluste zu minimieren. Das Ziel ist es, eine Strategie zu entwickeln, von der man die Gewissheit hat gegen jeden beliebigen Gegner zumindest ein Unentschieden zu erreichen. Wäre man im Besitzt einer solchen Strategie, könnte man sich sicher sein nichts zu verlieren, aber es ist nicht gesagt, dass man mit einer anderen Strategie nicht mehr Gewinn erzielt hätte. Natürlich sind auch Kombination beider Ansätze denkbar.

Zu Letzt soll noch erwähnt sein, dass ein Turnier recht lange dauern kann. Zum einen liegt das daran, dass den Bots wie auch im echten Poker eine gewisse Bedenkzeit zugestanden wird, zum anderen ist wegen der Varianz im Spiel eine hohe Anzahl von gespielten *Händen* erforderlich, um aussagekräftige Ergebnisse zu erhalten.

3 CFR

3.1 CFR im zwei Spieler Spiel

Der folgende Abschnitt beschreibt die Grundidee des zwei Spieler Bots, der im drei Spieler Spiel durch eine Abbildung der Spielzustände verwendet werden soll.

Im Allgemeinen kann eine Strategie als eine Abbildung verstanden werden, die jeder Aktion in jedem Spielzustand eine Wahrscheinlichkeit zuordnet mit der sie gewählt werden soll. Unser *Bot* passt im Verlauf eines Turniers nie diese Wahrscheinlichkeiten an, da er nach einer Nash Equilibrium Strategie spielt und nicht auf seine Gegner reagiert.

Im Grunde genommen schaut unser Bot während eines Spiels nur in einer Tabelle oder ähnlichem nach und wählt dementsprechend seine Aktion. Das eigentlich Problem ist somit nicht die Implementierung des *Bots* sondern die Generierung der Strategie. Wie bereits erwähnt, ist obwohl eine solche Strategie im zwei Spieler Spiel existiert, eine exakte Berechnung aufgrund der Komplexität von *Texas Hold'em* aktuell nicht möglich.

Da wir also diese Strategie approximieren müssen, reden wir auch von einer ϵ-*Nash Gleichgewicht* Strategie. Ein Spiel befindet sich in einem ϵ-*Nash Gleichgewicht*, wenn kein Spieler durch einseitiges Abweichen von seiner Strategie seinen Gewinn um mehr als ϵ erhöhen kann.

Der Verfahren Counterfactual Regret Minimization, welches uns diese approximative Lösung liefert, ist im folgenden beschrieben und eine Zusammenfassung von [7, Kapitel 3].

Im Groben kann man sagen, dass CFR ein iteratives Verfahren ist, welches jeweils für den *Big Blind* und den *Button* eine Strategie repräsentiert als Spielbaum erstellt und in einer Iteration gegen einander spielen lässt. Nach jeder *Hand* werden die beiden Strategien ihr Spiel anpassen, um ihren Verlust zu minimieren. Eigentlich wird auch nicht eine *Hand* sondern eine *Hand* mit jedem möglichen Verlauf gespielt.

Mit einer wachsenden Anzahl von gespielten *Händen* wird das Minimieren der Verluste dazu führen, dass die beiden Halbstrategien ein *Nash Gleichgewicht* erreichen. Die beiden Strategien zusammen bilden das Ergebnis des Verfahrens.

Zum genaueren Verständnis sind folgende Definitionen notwendig:

- σ_{Button} und $\sigma_{BigBlind}$ bezeichnen die beiden Halbstrategien des *Buttons* und des *Big Blinds*
- Sei u eine Funktion, die den Nutzen einer Aktion a berechnet und a* die Aktion, die den höchsten Nutzen erzielt hätte, dann definiert sich der regret mit u(a*) - u(a).
- $\Delta_{u,i} = max_z u_i(z) - min_z u_i(z)$ bezeichnet die Differenz zwischen dem größt und kleinst möglichem Nutzen
- $\sigma_i^t \in \Sigma_i$ bezeichnet die Strategie die Spieler i in Spiel t gewählt hat
- A(I) bezeichnet die Menge der Aktionen die in einem *Information Set* möglich sind und Ψ die Menge aller *Information Sets*
- $\pi_{-i}^{\sigma}(I)$ bezeichnet die Wahrscheinlichkeit, dass I erreicht wird, wenn Spieler i das möchte.
- $\sigma|_{I \to a}$ ist das *Strategieprofil* identisch zu σ mit dem Unterschied, dass in Spieler i in I Action a wählen wird.
- Die conterfactual utility $u_i(\sigma, I)$ ist der erwartete Nutzen, wenn *Information Set* I erreicht worden ist, wobei alle Spieler nach *Strategieprofil* σ spielen mit der Ausnahme, dass Spieler i I erreichen wollte.

Die beiden Halbstrategien σ_{Button} und $\sigma_{BigBlind}$ werden durch einen Spielbaum repräsentiert und mit beliebigen Aktionswahrscheinlichkeiten initialisiert. Ein Beispiel wäre, dass die Strategie vorgibt aus jedem Information Set mit gleicher Wahrscheinlichkeit zu *folden*, zu *callen* oder zu *raisen*.

Diese beiden Strategien werden mehrere *Hände* gegen einander spielen, wobei nach jeder *Hand* für jedes *Information Set* ein regret Minimierungsagent den regret seines eigenen subtrees im Bezug auf die Strategie seines Gegners minimiert.

Wir definieren den average overall regret R_i^T von Spieler i in T Händen als der Durchschnitt der Differenz zwischen dem Nutzen der Strategie, die in jedem Spiel gewählt wurde und der Strategie, die den Nutzen über alle T Spiele maximiert hätte.

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

Des Weiteren definieren wir die durchschnittliche Strategie, die von Spieler i benutzt wurde:

$$\bar{\sigma}_i^T(I)(a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)(a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)} \tag{2}$$

Die durchschnittliche Strategie gibt die Wahrscheinlichkeit für eine Aktion $a \in A(I)$ an einem *Information Set* $I \in \Psi$ an, als die Aktionswahrscheinlichkeit von allen Strategien an diesem *Information Set* gewichtet mit der Wahrscheinlichkeit mit der die jeweilige Strategie dieses *Information Set* erreicht.

Theorem 1 *Wenn in einem Nullsummenspiel an Zeitpunkt T der average overall regret beider Spieler weniger als ϵ ist, dann ist mit $\bar{\sigma}^T$ ein 2ϵ Gleichgewicht gegeben.*

Wenn der average overall regret bei einer gegen unendlich laufender Anzahl von Spielen gegen null konvergiert, dann ist durch die durchschnittlichen Strategien der Spieler ein *Nash Gleichgewicht* gegeben. Es wird also ein regret Minimierungsalgorithmus zum Auswählen von σ_i^t benötigt.

Anstatt einen regret Wert für das gesamte Spiel zu minimieren, werden einzelne regret Werte an jedem *Information Set* betrachtet. Diese unabhängigen regret Werte werden counterfactual regret genannt und sind in der Summe größer gleich den overall regret (Theorem 2). Somit wird durch die Minimieren der counterfactual regret Werte der overall regret minimiert und wir erreichen ein *Nash equilibrium*.

In diesem Sinne definieren wir den immediate conterfactual regret $R_{i,imm}^T$ an einem *Information Set* als den Durchschnitt der Differenzen zwischen dem Nutzen, den wir mit der Aktion erreicht haben, die wir gewählt haben und dem Nutzen der Aktion, die den Nutzen für alle Spiele maximiert hätte.

$$R_{i,imm}^T = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I)) \tag{3}$$

$$R_{i,imm}^{T,+}(I) = \max(R_{i,imm}^T(I), 0) \tag{4}$$

Theorem 2 $R_i^T \leq \sum_{I \in \Delta_i} R_{i,imm}^{T,+}(I)$

Zuletzt muss noch eine Verfahren angegeben werden um den immediate counterfactual regret an jedem Information Set zu minimieren.

Analog zu $R_{i,imm}^T$ definieren wir $R_{i,imm}^T(I,a)$ als den regret den wir erhalten, wenn wir nicht *folden*, *callen* bzw. *raisen* und die von Strategie σ_i empfohlene Aktion wählen.

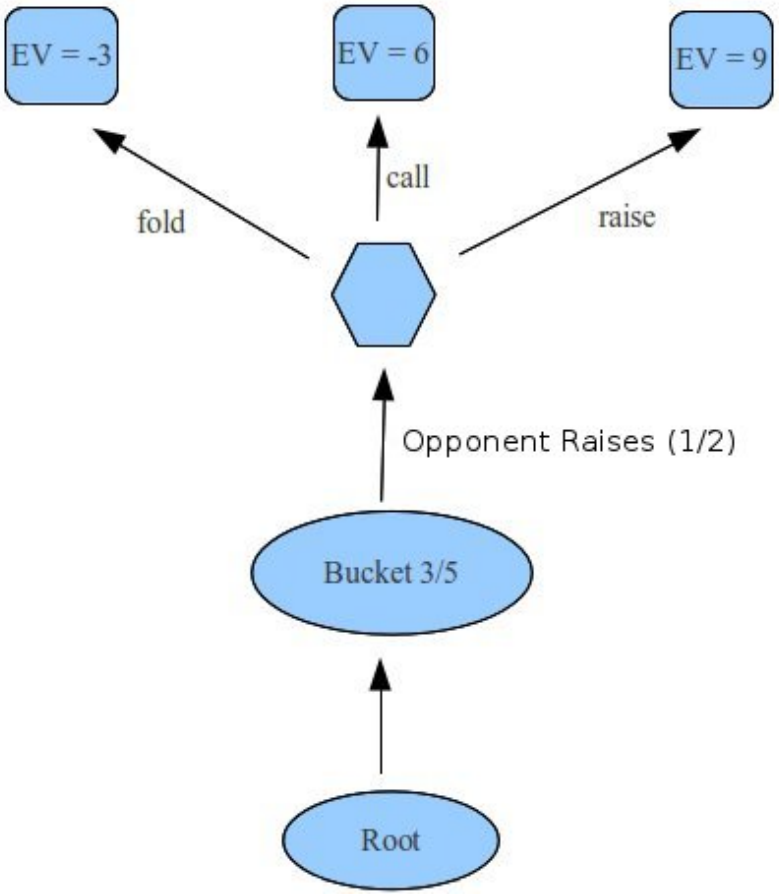
$$R_{i,imm}^T(I,a) = \frac{1}{T}\sum_{t=1}^{T} \pi_{-i}^{\sigma^t}(I)(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I)) \tag{5}$$

$$R_i^{T,+}(I,a) = \max(R_i^T(I,a), 0) \tag{6}$$

Die Aktionswahrscheinlichkeiten einer Aktion a an einem *Information Set* I, die von der Strategie, die in Spiel T + 1 benutzt wird, verwendet wird, wird folgendermaßen errechnet:

$$\sigma_i^{T+1}(I)(a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a\in A(I)} R_i^{T,+}(I,a)} & \text{if } \sum_{a\in A(I)} R_i^{T,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & \text{sonst} \end{cases} \tag{7}$$

Das folgende Beispiel (siehe hierzu *Bucketing*) zeigt wie der Algorithmus an einem Knoten im Spielbaum die Wahrscheinlichkeiten anpasst. Wichtig hierbei ist festzustellen, dass der Algorithmus vollständige Kenntnis über die beiden Strategien σ_{dealer} und $\sigma_{opponent}$ besitzt und auch beide Strategien anpasst. Ein echtes Spiel findet eigentlich gar nicht statt. Der Algorithmus kann die Erwartungswerte der Aktionen der Spieler berechnen, indem er den restlichen Teil eines Spieles mit den Strategien der Spieler und allen möglichen Ausgängen simuliert.



CFR Beispiel an einem Entscheidungsknoten [7, Figure 3.1]

Der betrachtete Entscheidungsknoten tritt zum Beginn des Spieles auf. Der Spieler besitzt eine *Hand* aus *Bucket* drei, also eine durschnittliche Hand. Der Gegenspieler *raised*, was er nach seiner Strategie in der Hälfte der Fälle getan hätte.

Wenn Aktionswahrscheinlichkeiten mit jeweils $\frac{1}{3}$ für die einzelnen Aktionen gegeben sind, errechnet sich der Erwartungswert mit $-3 * \frac{1}{3} + 6 * \frac{1}{3} + 9 * \frac{1}{3} = 4$. Die Werte $R^T_{i,imm}(I,a)$ für *folden*, *callen* und *raisen* sind mit (-3.5, 1, 2.5) , die neuen Aktionswahrscheinlichkeiten nach $\sigma_i^{T+1}(I)(a)$ mit $(0, \frac{1}{3.5}, \frac{2.5}{3.5})$ gegeben.

Theorem 3 *Wenn Spieler i seine Aktionswahrscheinlichkeiten wie in (7) vorgeschrieben anpasst, dann ist* $R^T_{i,imm} \leq \Delta_{u,i}\frac{\sqrt{|A_i|}}{\sqrt{T}}$ *und demnach ebenfalls* $R^T_{i,imm} \leq \Delta_{u,i}|I_i|\frac{\sqrt{|A_i|}}{\sqrt{T}}$ *wobei* $|A_i| = max_{h:P(h)=i}|A(h)|$.

In diesem Zusammenhang ist A(h) die Funktion, die alle Aktionen, die nach einem Spielverlauf h möglich sind, zurückliefert und P(h) die Funktion, die den Spieler i, der nach einem Spielverlauf h am Zug ist, zurückliefert.

Da $\Delta_{u,i}$, die maximale Anzahl Aktionen $\sqrt{|A_i|}$ und die Zahl der *Information Sets* $|I_i|$ konstant sind, konvergiert $R^T_{i,imm}$ gegen 0, womit Theorem 3 zeigt, dass wir auf diese Weise ein *Nash Gleichgewicht* erreichen.

3.2 CFR im drei Spieler Spiel

In [9] wird gezeigt, dass CFR generierte Strategien gut in *Multiplayerspielen* abschneiden. Des Weiteren wird beschrieben, wie mit CFR sogenannte *headsup* Experten erstellt werden können, die in zwei Spieler Situationen in einem drei Spieler Spiel eingesetzt werden.

Ebenfalls wurde in [9] an folgendem einfachen Spiel verdeutlicht, warum die theoretische Grundlage verloren geht mit CFR generierte Strategien in einem Multiplayerspiel ein Nash Gleichgewicht zu erreichen.

- Die drei teilnehmenden Spieler leisten jede Runde einen festgeschriebenen Einsatz
- Jeder Spieler besitzt eine Münze und wählt ohne das Wissen der anderen Kopf oder Zahl
- Nachdem jeder Spieler gewählt hat, werden die Münzen vorgezeigt
- Haben alle Spieler die gleiche Wahl getroffen, erhalten die Spieler ihre Einsätze zurück
- Haben genau zwei Spieler die gleiche Wahl getroffen, erhalten diese Spieler ihren Einsatz zurück und teilen sich den Einsatz des verbleibenden Spielers

Das Nash Gleichgewicht Strategieprofil würde für jedem Spieler vorschreiben, Kopf beziehungsweise Zahl mit der Wahrscheinlichkeit von $\frac{1}{2}$ zu wählen. Nehmen wir an, dass nun zwei der drei Spieler ihre Strategie wechseln und jede Runde Kopf wählen. Der Erwartungswert des verbleiben Spielers berechnet sich mit $e = \frac{1}{2} * 0 + \frac{1}{2} * (-1) = -\frac{1}{2}$. Er verliert also im Schnitt jede Runde einen halben Einsatz, womit er nach keiner Strategie spielt, die robust gegen beliebige Gegner ist.

3.3 Verwandte Arbeiten

Der folgende Abschnitt enthält einen Überblick über einige verwandte Arbeiten.

Die bereits erwähnte Masterarbeit [7] ist Grundlage für die wissenschaftliche Arbeit [13].

In der Masterarbeit [8] wurden einige CFR generierte Strategien für drei Spieler Limit Texas Hold'em Poker erstellt. Es konnte gezeigt werden, dass mit dieser Technik sehr starke *Bots* erstellt werden können. Außerdem wurde der Ansatz verfolgt, für eine Menge von zwei Spieler Situationen im drei Spieler Spiel speziell generierte Strategien anzuwenden.[9] ist die wissenschaftliche Arbeit, die aus dieser Masterarbeit hervorging.

Richard Gibson und Duane Szafron [4] beschreiben zwei Techniken, wie Expertenstrategien, die nur einen Teilbaum des gesamten Spiels abbilden, zu einer Grundstrategie hinzugefügt werden können. Der Hintergrund ist, dass man so wichtige Teile des Spiel auf einer feineren Abstraktionsebene beschreiben könnte.

Der CFR Algorithmus besucht in einer Iteration den kompletten Spielbaum und betrachtet somit alle möglichen Verläufe eines Spiels. In [6] werden neue Techniken vorgestellt, bei denen in einer Iteration Mengen von möglichen Verläufen betrachtet werden, um die Konvergenz zu beschleunigen. Der daraus resultierende Monte Carlo CFR (MCCFR) Algorithmus wird in [3] weiter verbessert.

[5] löst das Problem, dass es nicht gewährleistet ist, dass CFR mit einer feineren Abstraktion bessere Ergebnisse liefert. Der daraus resultierende Algorithmus wird CFR-BR (best response) genannt.

[11] beschäftigt sich mit dem analytischen Bestimmen von Equilibrium Strategien in drei Spieler Kuhn Poker. Kuhn Poker ist eine vereinfachte Form von *Texas Hold'em*.

4 Abbildung von drei Spieler zu zwei Spieler Zuständen

4.1 Einleitung

Wie in der Einleitung beschrieben, soll der in Abschnitt 3 vorgestellte *Bot* in einem Spiel mit drei Spielern verwendet werden. Da der *Bot* als Abbildung verstanden werden kann, die Spielzuständen ein Wahrscheinlichkeitstripel zuordnet, ist logischerweise eine Abbildung von zwei Spieler Spielzustände auf drei Spieler Spielzustände erforderlich. Würde man zum Beispiel von ihm für den Zustand

S = ([{(Q,h),(Q,d)}, {(J,h),(J,d)}, {(A,s),(K,s)}], {(Q,s),(J,s),(10,s)}, {(Q,c), {(J,c), [r,f,c], [c, r, r, r, c], [r, r, c], [])

eine Entscheidung verlangen, würde er einen Fehler ausgeben, da es im zwei Spieler Spiel nicht möglich ist, dass das Spiel nach einem *Fold* weitergeht. Offensichtlich ist das Problem die Abbildung der Aktionen. In einem zwei Spieler Spiel existieren 6378, in einem drei Spieler Spiel schon 12827520 nicht beendete Spielverläufe.

Bei den ersten Versuchen eine gültige Abbildung zu erstellen, geht es nicht darum gleich einen möglichst starken Bot zu finden. Es ist vielmehr das Ziel Daten zu sammeln, um herauszufinden welche Ansätze weiterentwickelt werden können und ob allgemein *Bots*, die in Spielen mit zwei Spielern eingesetzt werden, sinnvoll per Abbildung in Spielen mit drei Spielern verwendet werden können. Durch eine schrittweise Verbesserung der Abbildungen soll am Ende das mit diesem Ansatz bestmöglichste Ergebnis erzielt werden.

Zuerst werden die Abbildungen für die ersten drei *Bots* beschrieben, die dann in einem Turnier mit vier weiteren Benchmark *Bots* evaluiert werden. Zum Schluss folgt ein Fazit mit Ausblick auf die weiteren Verbesserungen.

4.2 Abbildung 1

Im Poker Framework wird ein Spielzustand ähnlich wie in dieser Arbeit in Abschnitt 2.1 durch folgende Eigenschaften definiert:

- Einen Index für die aktuelle *Hand*
- Eine Liste von Spielern, wobei ein Spieler sich über eine *Position*, eine Namen, seine *Handkarten* und einen *Boolean*, der angibt ob der Spieler ich selbst bin, definiert. Die Position der Spieler in der Liste muss dabei der *Position* am Tisch entsprechen.
- die *Flopkarten*, die *Turnkarte* und die *Riverkarte*
- jeweils eine Liste der getätigten Aktionen der Spieler für die vier *Wettrunden*

Der erste Ansatz kann als ein *Bot* verstanden werden, der auf möglichst einfache Weise einen zwei Spieler Zustand aus dem gegebenen drei Spieler Zustand erzeugt.

Um die Aktionslisten der *Wettrunden* anzupassen, verwendet der *Bot* die Klasse **FirstTry**, die bei der Initialisierung eine Abbildung erstellt. Das Quellcode zum Erstellen der Abbildung ist in Listing 1 angegeben.

Zum Benutzen der Abbildung bietet die Klasse die Funktion **mapStreet**, die eine *Wettrunde* und die Anzahl aktiver Spieler zu Beginn dieser *Wettrunde* erhält und die neue *Wettrunde* zurückliefert.

Hierbei ist festzustellen, dass die einzelnen *Wettrunden* getrennt voneinander angepasst werden.

Die Anforderungen an diese Abbildung können als minimale Anforderungen verstanden werden und sind wie folgt definiert.

- Es existiert keine Abbildung für *Wettrunden*, die ein Spiel abschließen, da in solchen Situationen keine Spielentscheidung mehr getroffen werden muss (zwei Spieler sind ausgestiegen).
- Die Abbildung darf keinen Zustand erzeugen, in dem ein 2 Spieler Spiel als abgeschlossen gilt.
- *Wettrunden*, in denen zu Anfangs nur 2 Spieler aktiv sind, werden nicht verändert.
- Abgeschlossene *Wettrunden* bleiben durch die Abbildung abgeschlossen, nicht abgeschlossene bleiben nicht abgeschlossen.
- Das Umformen von *Wettrunden* geschieht durch das Löschen von Einträgen, wobei die Anzahl an Löschoperationen minimal ist.

Da die Abbildung einzelne *Wettrunden* und nicht komplette Spielverläufe aufeinander abbildet, ist sie mit 370 Einträgen[5] recht übersichtlich und kann leicht beim ersten Ausführen des *Bots* in einer Textdatei gespeichert werden.

Eine Anwendung der Abbildung auf das oben genannten Beispiel könnte folgendermaßen angegeben werden:

$$\textbf{mapStreet}([r,f,c],3) -> [r,c]$$
$$\textbf{mapStreet}([c,r,r,r,c],2) -> [c,r,r,r,c]$$
$$\textbf{mapStreet}([r,r,c],2) -> [r,r,c]$$
$$\textbf{mapStreet}([],2) -> []$$

Um einen gültigen Zustand zu erhalten, muss nur noch die Liste der Spieler angepasst werden. Dazu löscht der *Bot* einen beliebigen Spieler, der nicht er selbst ist, aus dieser Liste. Des Weiteren müssen die *Positionen* der Spieler entsprechend des neuen Spielverlaufs gesetzt werden, da diese sich durch eine solche Abbildung der *Wettrunden* ändern können. Alle anderen Eigenschaften können ohne Anpassung übernommen werden.

Es ist offensichtlich, dass der erste *Bot* viel Raum für Verbesserungen bietet, aber zumindest wurde ein 3 Spieler *Bot* erstellt, der funktioniert und in jedem Spielzustand die Empfehlungen des in Kapitel 3 vorgestellten *Bots* verwendet.

4.3 Abbildung 2

Allgemein sollte das Ziel einer Abbildung sein, ein zwei Spieler Spiel zu erzeugen, das dem drei Spieler Spiel möglichst ähnlich ist, sodass der CFR-*Bot* keine falschen Annahmen über das Spiel trifft. Zum Beispiel wäre es gut, wenn sich durch die Abbildungen das Verhalten der Spieler nicht ändert. Unter anderem wird dieser Aspekt von der ersten Abbildungen nicht betrachtet.

[5] Es gibt 370 verschiedene Verläufe einer *Wettrunden*, in der zu Beginn noch 3 aktive Spieler existieren und nicht mehr als einer aussteigt. Eine Abbildung für *Wettrunden*, in denen zu Beginn zwei Spieler aktiv sind, muss nicht gespeichert werden, da dieses auf sich selbst abgebildet werden.

Um dieses Problem anzugehen, verwendet der zweite *Bot* eine Abbildung, die die Anzahl der verschiedenen Aktionen jedes Spielers nach der Abbildungen minimiert. Eine Aktion mehr beziehungsweise weniger ausgeführt zu haben, zählt dabei ebenfalls als eine unterschiedliche Aktion.

Des Weiteren soll eine Verbesserung erreicht werden, indem die *Wettrunden* nicht mehr einzeln betrachtet werden und nun komplette Spielverläufe aufeinander abgebildet werden.

Da die Anzahl der nicht beendeten Spielverläufe in einem drei Spieler Spiel zu groß ist, wurde im Gegensatz zum ersten Versuch die Abbildung nicht komplett erstellt und dann abgespeichert. Im Gegenzug wird bei Programmstart sofern vorhanden die Zielmenge der Abbildung, also 6378 nicht beendeten zwei Spieler Spielverläufe, geladen.

Des Weiteren wird besser zwischen den Zuständen, in denen zwei Spieler aktiv sind und den Zuständen, in den drei Spieler aktiv sind, differenziert.Im Fall von zwei aktiven Spielern geht der *Bot* folgendermaßen vor:

- ermittle die *Positionen* von mir und des verbleibenden Gegners im Spiel
- erstelle jeweils zwei Listen für jede *Wettrunde*, eine mit den Aktionen von mir in der jeweiligen *Wettrunde*, eine mit den Aktionen des Gegners
- iteriere über alle zwei Spieler Spielverläufe, die sich in der gleichen Wettrunde wie das aktuelle Spiel befinden und berechne eine Distanz zwischen dem drei Spieler Spiel und dem zwei Spieler Spiel
- Um die Distanz zu berechnen werden für jede *Wettrunde* und für beide Spieler die Listen des zwei Spieler Spielzustandes und des drei Spieler Spielzustandes verglichen. Es wird über die Listen iteriert und für jede unterschiedlich getätigte Aktion die Distanz um eins erhöht. Bei unterschiedlich langen Listen wird die Distanz um den Unterschied in den Längen der Listen erhöht. Die Summe dieser Distanzen ergibt die Distanz zwischen den Spielzuständen.
- forme den aktuellen drei Spieler Zustand in einen zwei Spieler Zustand um, indem der ausgestiegenen Spieler gelöscht wird, die Aktionslisten durch die Aktionslisten des ähnlichsten Spielverlaufs ersetzt werden und die *Positionen* der Spieler angepasst werden.
- übergebe dem CFR-*Bot* den erstellten Spielzustand um eine Aktion zu wählen

Im Fall das noch alle Spieler aktiv sind wird wie folgt vorgegangen:

- ermittle die *Positionen* von mir und beiden Gegnern
- erstelle jeweils drei Listen für jede *Wettrunde*, wobei eine Liste für die getätigten Aktionen eines Spielers in der jeweiligen *Wettrunde* steht
- ermittle nach dem selben Verfahren wie wenn zwei Spieler aktiv sind, für jeden Gegner einen ähnlichsten Spielverlauf. Man könnte sagen, dass zuerst der eine dann der andere Gegner ignoriert wird.
- erstelle wie zuvor aus den Spielverläufen Spielzustände, und übergebe sie dem CFR-*Bot* um Aktionswahrscheinlichkeiten zu erhalten
- mittle die Aktionswahrscheinlichkeiten und wähle dementsprechend per Zufallsentscheidung eine Aktion

Zusammenfassend könnte man sagen, dass zuerst der eine und sofern noch im Spiel dann der andere Gegner ignoriert wird und für beide Gegner nach einem zwei Spieler Spielverlauf gesucht wird, der sich in der selben *Wettrunde* wie das aktuelle Spiel befindet und in der die Aktionen der beiden betrachteten Spieler in der gleichen Reihenfolge möglichst dieselben sind.

4.4 Abbildung 3

Der dritte *Bot* kann als kleine Abwandlung des zweiten *Bots* gesehen werden. Der Unterschied besteht darin, dass die zwei Spieler Spielzustände, in denen nach dem ähnlichsten gesucht wird, zuvor nicht nur nach dem Index der aktuellen Wettrunde, sondern auch nach der Größe des *Pots* gefiltert werden.

Es werden also nur Spielzustände zurückgeliefert, deren *Potgröße* am nächsten an der wirklichen *Potgröße* des aktuellen Spiels liegen. Die Klasse **PotSizeFilter** besitzt eine Funktion **computeHistories**, die zu einem gegeben Index, der für die aktuelle Wettrunde steht und einer gegeben *Potgröße* eine Liste von Spielzuständen zurückliefert, die sich in der gleichen *Wettrunde* befinden und von allen Spielzuständen die ähnlichste *Potgröße* besitzen. Ein Teil des Sourcecodes ist in Listing 2 angegeben.

Bei der Initialisierung werden alle nicht beendeten zwei Spieler Spielzustände und möglichen *Potgrößen* berechnet oder sofern vorhanden aus einer Textdatei geladen, wobei alle Spielverläufe, die sich in der gleichen *Wettrunde* befinden und die gleiche *Potgröße* besitzen, in die gleiche Liste einsortiert werden.

Dass Spielverläufe noch an weiteren Eigenschaften, als die, die dieser Filter benutzt, unterschieden werden sollten, erkennt man daran, dass die 6378 verschiedenen zwei Spieler Spielverläufe sonst in nur 54 Klassen aufgeteilt werden würden. Zum Beispiel existieren 536 Spielverläufe die sich in *Wettrunde* drei befinden und eine *Potgröße* von 52 besitzen.

4.5 Evaluierung

Die Stärke der *Bots* wird in einem Turnier mit folgenden Benchmark-*Bots* getestet.

- **RaiseBot**: Der **RaiseBot** führt in jedem Spielzustand die Aktion raise aus. Ein raise in einem Zustand, in dem kein raise möglich ist, wird vom Spiel als call interpretiert.
- **CallBot**: Analog zum **RaiseBot** führt dieser *Bot* in jedem Spielzustand die Aktion call aus.
- **akuma**: Monte-carlo Simulation mit *opponent-modelling* durch *hand-ranges* und Aktionshäufigkeiten (Zweiter der TUD PokerChallenge 2010 und 2011)
- **ownbot**: Sieger der TUD PokerChallenge 2011

Das Turnierformat welches im Folgenden beschrieben ist entspricht den offiziellen Regeln der Annual Computer Poker Competition [1].

Jeder Spieler muss ein Match gegen jede mögliche Kombination von Gegnern absolvieren. Dem zufolge werden mindestens $\binom{AnzahlBots}{3}$ Matches gespielt, welche als duplicated Matches bezeichnet werden. In einem duplicated Match ist die Anzahl der gespielten *Hände* immer gleich und wird je nach verfügbaren Ressourcen vor dem Turnier festgesetzt. Ansonsten ist ein duplicated Match fogerndermaßen definiert:

- (1) sei N die Anzahl der gespielten *Hände* pro duplicated Match

- (2) platziere die Spieler in einer beliebigen Reihenfolge am Tisch, sei in diesem Fall *Bot1* der *Small Blind*, *Bot2* der *Big Blind* und *Bot3* der *Button*
- (3) spiele $\frac{N}{6}$ *Hände* nach Standard Poker Regeln: nach jeder Hand rotieren die *Positionen* nach links
- (4) setze den Speicher der *Bots* zurück
- (5) setze die Spieler auf die *Positionen*, die sie vor den gespielten *Händen* hatten und rotiere diese nach links, sodass in diesem Beispiel *Bot1* der *Button*, *Bot2* der *Small Blind*, *Bot3* der *Big Blind* ist
- (6) spiele $\frac{N}{6}$ *Hände*, wobei dieselben Karten auf die selben *Positionen* wie zuvor verteilt werden (die erste *Hand* von *Bot1* ist jetzt die erste *Hand* von *Bot3*)
- (7) setze den Speicher der *Bots* zurück
- (8) setze die *Positionen* auf die *Positionen* vor den letzten $\frac{N}{6}$ *Händen* zurück und rotiere nach dem gleichen Prinzip nochmal
- (9) spiele dieselben $\frac{N}{6}$ *Hände* nochmal
- (10) platziere die Spieler folgendermaßen - *Bot1 Small Blind*, *Bot3 Big Blind*, *Bot2 Button*
- (11) wiederhole Schritt 3 bis 9

In unserem Fall wurde 6000 *Hände* also 1000 *Hände* zwischen der Speicherresets pro duplicated Match gespielt.

Um den Sieger zu ermitteln werden 2 Verfahren angewendet. Das Total Bankroll Verfahren addiert für jeden *Bot* die Gewinne beziehungsweise Verluste in den einzelnen Matches auf und platziert die *Bots* dementsprechend. In diesem Format soll der *Bot* ermittelt werden, der es am besten versteht, seine Gewinne gegenüber allen Gegnern zu maximieren.

Allerdings ist es mit diesem Verfahren möglich, sich sehr weit vorne zu platzieren, indem man einen speziellen Gegner sehr stark ausnutzt, obwohl man gegen die anderen Spieler nicht gut abschneidet. Deshalb wurde das Bankroll Instant Run-off Verfahren eingeführt, welches wie folgt definiert ist:

- (1) initialisierte S als Menge mit der verbleibenden *Bots* mit allen teilnehmenden *Bots*
- (2) Spiele alle möglichen $\binom{m}{3}$ Matches zwischen den *Bots* aus S, wobei $m = |S|$ und berechne die *Bankrolls* der *Bots*
- (3) definiere T als die Menge der *Bots* mit der niedrigsten *Bankroll*. Diese erhalten eine gemeinsame Platzierung unterhalb der anderen *Bots* aus S
- (4) entferne T aus S
- wiederhole Schritt zwei bis vier bis $|S|=3$
- spiele ein Match zwischen den drei verbleibenden *Bots* und berechne die *Bankrolls*
- die letzten drei *Bots* werden erster, zweiter und dritter basierend auf den *Bankrolls* dieses Matches

-	HBB	HBB2	HBB3	RaiseBot	CallBot	Akuma	ownbot
HBB, HBB2	-	-	-17046	23266	-5785	4216	8532
HBB, HBB3	-	10674	-	51224	1344	5272	18755
HBB, RaiseBot	-	-11943	-51968	-	-6416	9210	7632
HBB, CallBot	-	1737	-7618	-2729	-	6810	16244
HBB, Akuma	-	-2153	-7408	-4755	-10651	-	-1272
HBB, ownbot	-	-3869	-20936	11471	-16176	2673	-
HBB2, HBB3	6372	-	-	61140	-419	1700	17378
HBB2, RaiseBot	-11323	-	-55657	-	2903	11892	9465
HBB2, CallBot	4048	-	-10618	2645	-	8248	14526
HBB2, Akuma	-2063	-	-5502	-9583	-10702	-	255
HBB2, ownbot	-4663	-	-20110	5851	-13641	1518	-
HBB3, RaiseBot	744	-5483	-	-	23944	11600	8994
HBB3, CallBot	6274	11037	-	25899	-	12187	23233
HBB3, Akuma	2136	3802	-	3357	-9057	-	1138
HBB3, ownbot	2208	2732	-	21985	-9920	2229	-
RaiseBot, CallBot	9145	-5548	-49843	-	-	17369	47640
RaiseBot, Akuma	-4455	-2309	-14957	-	-9874	-	671
RaiseBot, ownbot	-19103	-15316	-30979	-	-23592	6559	-
CallBot, Akuma	3841	2454	-3130	-7495	-	-	5829
CallBot, ownbot	-68	-885	-13313	-24048	-	8911	-
Akuma, ownbot	-1401	-1773	-3367	-7230	-14740	-	-

Tabelle 1: Ergebnis des Turniers (10 = 1 *Big Blind*, 6000 *Hände* pro Match)

ownbot	RaiseBot	akuma	HBB	HBB2	CallBot	HBB3
179020	150998	110394	-8308	-16843	-102782	-312479
109522	-12607	77406	-26042	-39605	-108674	-
25283	19020	**36068**	-43008	-37363	-	-
10391	-10962	**19969**	-	-19398	-	-
671	-7230	**6559**	-	-	-	-

Tabelle 2: Bankroll Instant Runoff (10 = 1 *Big Blind*)

Das Ergebnis des Turniers ist in Tabelle 1 angegeben, wobei in einer Spalte die Ergebnisse eines *Bots* gegen die in der Zeile angegeben Gegner steht. Die *Bots* HBB, HBB2 und HBB3 entsprechen dabei den Abbildungen 1, 2 und 3. Die Platzierungen nach dem Bankroll Instant Run-off Verfahren sind in Tabelle 2 enthalten. Hier kann die erste Zeile als Ergebnis des Total Bankroll Verfahrens angesehen werden. Neben dem schlechten Abschneiden der eigenen *Bots* ist vor allem das Ergebnis des RaiseBots überraschend, der seine besten Resultate in Partien gegen unsere *Bots* erzielt hat.

4.6 Fazit

Der folgende Abschnitt versucht die schlechten Ergebnis zu erklären und enthält einen Ausblick auf die weiteren Verbesserungen.

Eine Grundregel im Poker sagt, dass man je mehr Spieler am Tisch sind desto weniger *Hände* spielen sollte oder anders gesagt mehr *Hände Preflop folden* sollte.

Dies lässt sich dadurch begründen, dass die Gewinnwahrscheinlichkeit einer *Hand* bei mehr Spielern am Tisch sinkt, da man mehr Spieler zum Aufgeben bringen muss oder eine stärkere *Hand* als mehr Spieler halten muss, um den *Pot* zu gewinnen.
Dieser Aspekt wird von allen *Bots* nicht betrachtet, da sie aus dem gegeben Spielzustand, einen oder mehrere Spielzustände mit 2 Spielern erzeugen, die dem CFR-*Bot* übergeben werden. Bei

der Ermittlungen der Aktionswahrscheinlichkeiten fließt die erhöhte Anzahl der Spieler also nicht ein.

Um dieses Problem in den Griff zu kriegen, soll ein BIAS eingeführt werden, der je nach Spielsituation, die Wahrscheinlichkeit zu *folden* erhöht oder erniedrigt.

Des Weiteren soll implementiert werden, dass die *Bots* die Möglichkeit besitzen, nicht nur den nach ihrer Abbildung nächsten Zustand, sondern eine Menge von x nächsten Zuständen in ihre Entscheidung mit einzubeziehen. Es ist zwar fraglich, ob dadurch eine Verbesserung erreicht wird, allerdings werden dadurch bessere Analysen der Abbildungen möglich. Zum Beispiel könnte die Varianz in den Aktionswahrscheinlichkeiten unter den x nächsten Zuständen der jeweiligen Abbildungen berechnet werden. Hier würde dann der Schluss nahe liegen, dass eine Abbildung mit geringerer Varianz sinnvoller ist.

Ein anderer Punkt, der von den vorherigen Implementierung nicht betrachtet wird, ist die Verschiebung der *Positionen* durch die Abbildungen. Unter anderem spricht man im Poker von Position auf einen Gegner haben, wenn man nach dem Gegner seine Aktion wählen muss. Dies wird allgemein als großer Vorteil angesehen.

Des Weiteren liegt es nahe, dass Zustände, in denen Aufgrund der Begrenzung der Anzahl der *raises* in einer *Wettrunde* weitere *raises* möglich beziehungsweise nicht mehr möglich sind, nur auf Zustände abgebildet werden, in denen dies ebenfalls möglich beziehungsweise nicht möglich ist.

Zum Schluss sollen überprüft werden, ob durch die Betrachtung der *potodds* eine Verbesserung erzielt werden kann.

5.1 Implementierung

Aufgrund der Ergebnisse aus Abschnitt 4 wurde ein *Bot* Framework implementiert, dessen Schlüsselklassen in diesem Abschnitt vorgestellt werden sollen.

Die Klasse **Bot** ist dabei die Oberklasse von der alle *Bots* erben müssen. Die Implementierung ist in Listing 3 angedeutet.

Die Methode **getAction** liefert dabei die Aktion zurück, die im aktuellen Spielzustand gewählt werden soll. Dabei wird zuerst ein **Filter** angewendet, der die Menge an möglichen zwei Spieler Zuständen eingrenzt. Dieser **Filter** wird von jedem *Bot* selbst definiert, wobei jeder das Interface **Filter** implementieren muss. Diese Interface schreibt die Implementierung der Methode **computeHistories** vor, die einen Spielzustand erhält und eine Liste von 2 Spieler Spielverläufen zurückliefert.

Wenn zwei Spieler aktiv sind, wird mit der Methode **computeProbabilities** aus dem nach der Abbildung nächsten Zuständen Aktionswahrscheinlichkeiten berechnet. Sind drei Spieler aktiv werden im Bezug auf jeden Gegner Aktionswahrscheinlichkeiten berechnet und diese Wahrscheinlichkeiten gemittelt. Hierbei hat der Bot die Möglichkeit über die Variable **bias** zu bestimmen, ob der in Abschnitt 4.6 erwähnte und in 5.3 weiter erläuterte Bias angewandt werden soll.

Des Weiteren existiert für jeden verbleibenden Gegner ein **Comparator**, der vom *Bot* selbst definiert wird. Diese werden von der **computeWeightedProbabilities** und der **computeUnweigthedProbabilities** Methode benutzt, um die verfügbaren zwei Spieler Spielzustände nach Distanz zum aktuellen drei Spieler Spielzustand zu sortieren, um die x nächsten Zustände zu bestimmen. Dazu sind durch das Interface **Comparator** die Methoden **compare** und **computeDistanzList** vorgeschrieben.

Der *Bot* hat ebenfalls die Möglichkeit die Zahl der nächsten Zustände, die betrachtete werden sollen, zu bestimmen. Dieser Wert wird in der Klasse mit **numClosestHistories** angegeben.

Jeder *Bot* hat die Möglichkeit über die Variable **weightProbabilities** zu bestimmen, ob die Aktionswahrscheinlichkeiten der nächsten Zustände speziell gewichtet werden sollen. Was das bedeutet wird noch im Späteren genau erläutert. Sollte die Variable allerdings auf false sein, wird das Mittel der Aktionswahrscheinlichkeiten der nächsten Zustände gebildet.

Um unseren zweiten *Bot* (HBB2) ins Framework zu integrieren, muss also ein **Filter** und ein **Comparator** implementiert werden. Der **Filter**, der von HBB2 benutzt wird, ist in der Klasse **Streetfilter** implementiert. Dieser liefert beim Aufruf der Methode **computeHistories** alle Spielzustände zurück, die sich in der gleichen *Wettrunde* wie der übergebene Spielzustand befinden.

Der **Comparator** von HBB2 benutzt die selben Techniken, wie sie in Abschnitt 4.3 beschrieben sind, um die Distanz zwischen dem aktuellen Zustand und einem beliebigem zwei Spieler Spielverlauf zu berechnen. Die *compare* Methode, die zwei zwei Spieler Spielverläufe erhält, muss dann nur noch im Fall von gleichen Distanzen 0, im Fall, dass Spielverlauf eins eine geringere Distanz hat -1 und ansonsten 1 zurückliefern.

Der dritte *Bot* (HBB3) benutzt den gleichen **Comparator** wie HBB2 und behält weiterhin den bereits in Abschnitt 4.4 beschriebenen **PotSizeFilter**.

Des Weiteren wurde ein vierter *Bot* (HBB4) implementiert, der weiter Ideen aus dem vorangegangen Fazit umsetzt.

x	HBB2	HBB3	HBB4
1	940.0	-2247.0	-2180.0
2	-2821.0	-4462.0	-3624.0
5	-4421.0	-6014.0	-3911.0
10	-5312	-4945	-5956
15	-6188.0	-7319.0	-6166.0

Tabelle 3: Ergebnisse eines duplicated Matches mit akuma und ownbot bei steigender Anzahl von nächsten Zuständen (10 = 1 *Big Blind*, 6000 *Hände* pro Match)

Hierzu verwendet dieser *Bot* als **Filter** die Klasse **PotOddsFilter**, die analog zum **PotSizeFilter** beim Aufruf der Methode **computeHistories** alle zwei Spieler Spielverläufe zurückliefert, die die nächsten *potodds* zum aktuellen Spielzustand besitzen und sich in der gleichen *Wettrunde* befinden. Somit teilen sich die 6378 zwei Spieler Spielverläufe auf 76 Klassen auf.

Der **Comparator**, der von HBB4 verwendet wird, erweitert den **Comparator** von HBB2 und HHB3 und ist in der Klasse **AdvancedComparator** umgesetzt. Die Implementierung ist in Listing 4 angedeutet. Für das Sortieren sind folgende Eigenschaften in der angegebenen Reihenfolge von Bedeutung:

- Wenn sich zwei Spielverläufe im Wert, der angibt, ob ein *raise* möglich ist, unterscheiden, ist der näher zum aktuellen Spielzustand, der in diesem Wert mit dem aktuellen Spielzustand übereinstimmt.
- Jeder **Comparator** ermittelt ein bevorzugte *Position*, die unser *Bot* in einem zwei Spieler Spiel haben sollte. Wenn diese *Position* existiert, haben wir genau wie in der aktuellen *Hand* im erstellten zwei Spieler Spiel Position auf unseren betrachteten Gegner oder nicht.[6] Wer Position auf wen hat kann sich zu Beginn des *Flops* ändern. Ein zwei Spieler Spiel ist näher zum aktuellen Spielzustand, wenn es die bevorzugte Positionierung besitzt und der andere Spielzustand nicht.
- Ansonsten ist der Spielzustand näher am aktuellen Spielzustand, der eine geringer Distanz besitzt, die wie bei den *Bots* HBB2 und HBB3 ermittelt wird.

5.2 Betrachtung mehrerer Zustände

Tabelle 3 enthält Ergebnisse von duplicated Matches zwischen unseren *Bots* und den beiden stärksten Spielern aus dem vorangegangenen Test. Die ausgeteilten Karten unterscheiden sich zwischen den duplicate Matches nicht.

Dabei wurde nicht mehr nur der nächste Spielverlauf betrachtet sondern die x nächsten. Des Weiteren sind unsere *Bots* so eingestellt, dass die Aktionswahrscheinlichkeiten, die aus den nächsten Zuständen hervorgehen, gemittelt werden. Benutzen die *Bots* für x den Wert 1 sind sie äquivalent zu den *Bots* aus dem ersten Test.

Es sind mehrere Gründe denkbar, warum die Ergebnisse trotzdem unterschiedlich sind. Logischerweise führt die Varianz im Spiel zu unterschiedlichen Ergebnissen zwischen den Partien. Eine andere Möglichkeit wären absichtlich beziehungsweise unabsichtlich behobene Bugs durch die neue Implementierung.

Zu aller erst fällt auf, dass sich die Ergebnisse mit steigendem x stark verschlechtern. Es liegt nahe Tests zu starten, in dem die Spielverläufe in Abhängigkeit zu ihrer Distanz zum aktuellen

[6] Für jeden verbliebenen Gegner im Spiel existiert genau ein **Comparator**.

raise möglich	t	t	f	f
Position	SB	BB	BB	BB
Distanz in den Aktionen	4	2	3	4
Distanz	4	6	9	10

Tabelle 4: Die Zustände sind bereits nach Nähe zum aktuellen Zustand sortiert. Die numerische Distanz eines Zustandes wird berechnet, indem seine Distanz in den Aktionen auf die Distanz des Zustandes addiert wird, mit dem er zuletzt nicht in den Eigenschaften *raise* möglich und Position übereingestimmt hat.

x	HBB2	HBB3	HBB4
1	-1884	-789	-504
2	-4300	-2558	-2731
5	-4975	-4121	-3065
10	-6276	-4122	-3733
15	-5714	-3864	-3153

Tabelle 5: Ergebnisse beim unterschiedlichen Gewichten der nächsten Spielverläufe gegen zwei Instanzen des *Bots* Dpp (10 = 1 *Big Blind*, 3000 *Hände* pro Match)

Spielzustand zu gewichten werden. Wenn wir mit a_i die Aktionswahrscheinlichkeit und mit d_i die Distanz des Zustand i bezeichnen, ist die von der Abbildung berechneten Aktionswahrscheinlichkeit a gegeben mit:

$$a = \frac{\sum \frac{1}{1+d_i} a_i}{\sum \frac{1}{1+d_i}} \tag{8}$$

Allerdings kann nicht jeder *Bot* diese Gewichtung ohne weiteres verwenden, da das Sortieren der nächsten Zustände nicht immer durch die Berechnung eines numerischen Wertes geschieht. Damit jeder *Bot* diese Variante benutzen kann, muss sein **Comparator** die Methode **computeDistanzList** implementieren. Diese Methode fügt der bereits sortierten Liste der nächsten Zuständen einen numerischen Wert, der für die Distanz zum aktuellen Zustand steht, hinzu.

Die Implementierung dieser Methode für den **Comparator**, den die *Bots* HBB2 und HBB3 verwenden, ist somit trivial. Zur Erläuterung von HBB4 betrachten wir das in Tabelle 4 angegebene Beispiel. Es wird angenommen, dass die Abbildung vier Zustände zurückgeliefert hat. Des Weiteren wird durch den aktuellen Spielzustand vorgeschrieben, das Spielverläufe in denen ein *raise* möglich ist bevorzugt werden. Ist diese Merkmal gleich werden Spielverläufe in denen wir die Position *Small Blind* hätten bevorzugt, ansonsten entscheidet die Distanz in den Aktionen.

Die Ergebnisse dieses Ansatzes sind in Tabelle 5 angegeben. Hier wurden duplicated Matches gegen zwei Instanzen des mathematisch fairen *Bots* dpp durchgeführt. Dabei wurden die duplicated Matches nicht mit allen möglichen Positionierung der Spieler durchgeführt. Von den sechs Positionierungen wurden nur drei gewählt, sodass unsere *Bots* zumindest einmal auf jedem Platz waren. Dies ist möglich, da die Gegner identisch sind und es aus unsere Sicht egal ist, welcher der Gegner Platz eins und zwei hat, wenn wir auf Platz drei sitzen.

Bis auf bei 15 nächsten Spielverläufen steigen die Verluste unsere *Bots* mit steigendem x. Allerdings sind die Ergebnisse nicht ausreichend gut, um diesen Ansatz weiter zu verfolgen.

Trotz dieser schlechten Ergebnisse ermöglicht uns das Selektieren von mehreren Spielverläufen das Durchführen von mehreren Analysen. Unter anderem wurde eine Varianzanalyse durchgeführt, in der duplicated Matches zwischen den *Bots* und sogenannten Random-*Bots*, die ihre

Aktion unabhängig vom Spielzustand zufällig wählen, gespielt wurden. Dabei fiel zum ersten Mal auf, dass der CFR-*Bot*, nicht wie erwartet für jeden Spielzustand ein Entscheidungstripel[7] zurückliefert, sondern für manche Spielzustände die Aktionswahrscheinlichkeiten jeweils mit NAN[8] angibt.

Eine Erklärung könnte sein, dass der CFR-*Bot*, da er in einem normalen zwei Spieler Spiel nicht in alle Spielzustände gelangt, weil gewisse Aktionen eine Aktionswahrscheinlichkeit von 0 besitzen, nicht für alle Spielzustände Aktionswahrscheinlichkeiten anbietet. Beim Erstellen der zwei Spieler Spielzustände beachten wir diese Tatsache nicht. Das Programm wurde danach dahingehend erweitert, dass bei der Auswahl der x nächsten Spielverläufe darauf geachtet wird, dass keine Spielverläufe zurückgeliefert werden, die ein NAN Entscheidungstripel erzeugen würden. Die Tests aus Tabelle 5 sind dabei mit dem verbesserten Framework durchgeführt worden.

Für die Analyse spielte jeder *Bot* duplicated Matches gegen zwei Random-*Bots*, wobei zuerst die maximale Anzahl von nächsten Spielverläufen auf 10 und dann auf 15 begrenzt worden ist. In einer Logdatei wurden für jede durchgeführte Abbildung folgende Werte abgespeichert:

- die Anzahl der verfügbaren Spielverläufe nach dem Filtern
- die Zahl der Spielzustände, die aufgrund der NAN Entscheidungstripeln nicht zu gebrauchen waren
- die Zahl der nächsten Spielverläufe
- die Zahl der aktiven Spieler
- die gemittelten Aktionswahrscheinlichkeiten
- die Varianz der Aktionswahrscheinlichkeiten

Im Fall dass drei Spieler aktiv sind, werden zwei Abbildung pro Entscheidung in einem Spielzustand benötigt, im Fall von zwei Spielern nur eine.

In diesem Zusammenhang wäre es interessant zu wissen, ob es aufgrund der NAN Entscheidungstripeln vorkommt, dass bei manchen Abbildungen kein gültiger Spielzustand erstellt werden konnte. Bei den Tests mit 15 nächsten Zuständen war dies für HBB2 in 0 von 57759, für HBB3 in 51 von 55790 und für HBB4 in 75 von 58180 Abbildungen der Fall. Insgesamt produzieren zwischen $\frac{1}{3}$ und $\frac{1}{4}$ der Spielverläufe, die nach dem Filtern zu Verfügung stehen, ein NAN Entscheidungstripel.

[7] (0.3, 0.3, 0.4) wäre ein Entscheidungstripel, dass für einen Spielzuständ angeben würde mit einer Wahrscheinlichkeit von 0.3 zu *raisen*, 0.3 zu *callen* beziehungsweise 0.4 zu *folden*.

[8] not a number

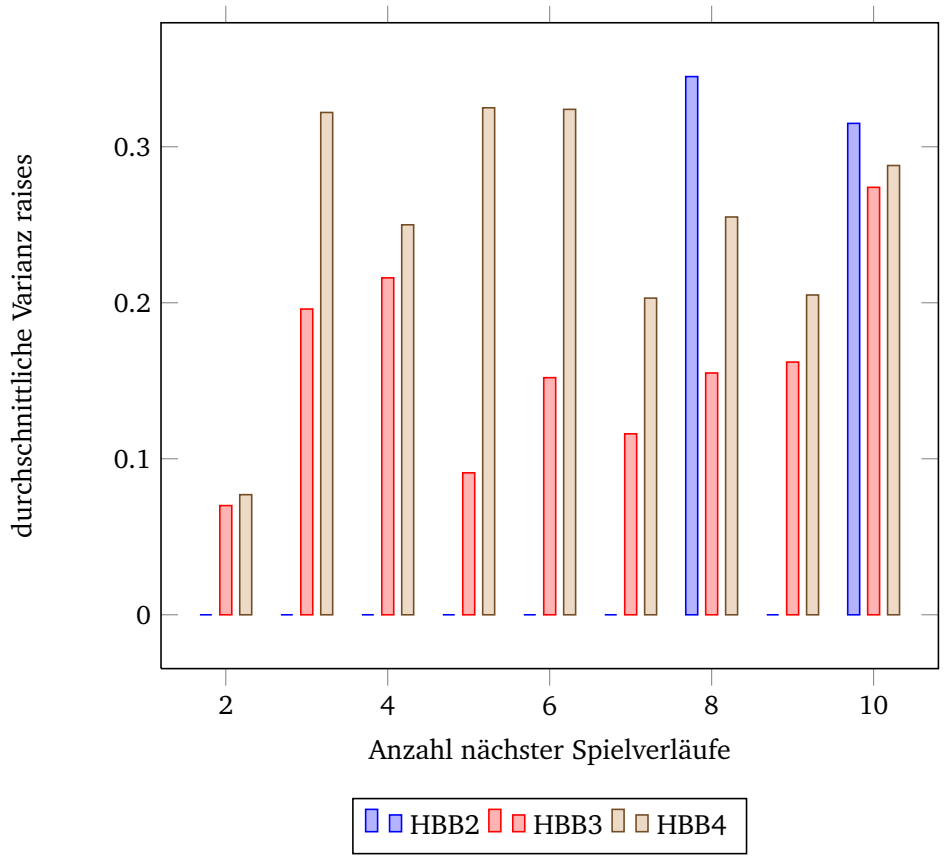


Diagramm 1: Varianzanalyse

Aus diesen Logdateien wurden mehrere Diagramme gewonnen, von den eins hier und einige im Anhang C angegeben sind. Das Diagramm hier und die ersten beiden aus dem Anhang sind aus den Tests mit maximal 10, das Letzte aus dem Anhang aus den Tests mit maximal 15 nächsten Spielverläufen. Die Abbildung wurden in Klassen unterteilt, die angegeben wie viele nächste Spielverläufe verfügbar waren. Auf der y-Achse ist die durchschnittliche Varianz der Aktionswahrscheinlichkeiten einer bestimmten *Aktion* angegeben ist.

Zu aller erst fällt auf das für HBB2 nur Werte für 8, 10 oder 15 nächste Spielverläufe angegeben ist. Das Liegt daran, dass dieser *Bot* nur Spielverläufe ausschließt, die sich nicht in der gleichen *Wettrunde* wie der aktuelle Spielzustand befinden. Dadurch sind immer die maximale Anzahl an Spielverläufen vorhanden, wobei *Preflop* maximal nur 8 nicht beendete zwei Spieler Spielverläufe existieren.

Bis auf die Erkenntnis, dass HBB3 in diesem Test am besten abschneidet, lässt sich zumindest momentan nichts weiteres ableiten. Ebenfalls lässt sich kein Trend, wie dass mit mehr verfügbaren Spielverläufen die Varianz steigt, feststellen. Die Ergebnisse dieses Tests wurden auch nicht weiter in dieser Arbeit berücksichtigt.

Im Folgenden wird eine weitere Analyse vorgestellt, die mit Distanzanalyse bezeichnet wird. Hierbei wurden wie zuvor duplicated Matches mit maximal 10 beziehungsweise 15 nächsten Spielverläufen gegen Random-*Bots* durchgeführt, wobei eine Logdatei erstellt wurde, die folgende Informationen für jeden nächsten Spielverlauf speichert:

- die Durchschnittswerte für die Aktionswahrscheinlichkeiten für die Abbildung, zu der der nächste Zustand gehört
- die Distanz des Spielverlaufs zu dem aktuellen Spielzustand

- die Aktionswahrscheinlichkeiten, die aus dem aktuellen Spielverlauf hervorgehen

Wenn im Spiel eine Entscheidung getroffen werden muss, wird also für jeden verbleiben Gegner eine Abbildung erstellt, die maximal 10 beziehungsweise 15 Einträge in der Logdatei speichert.

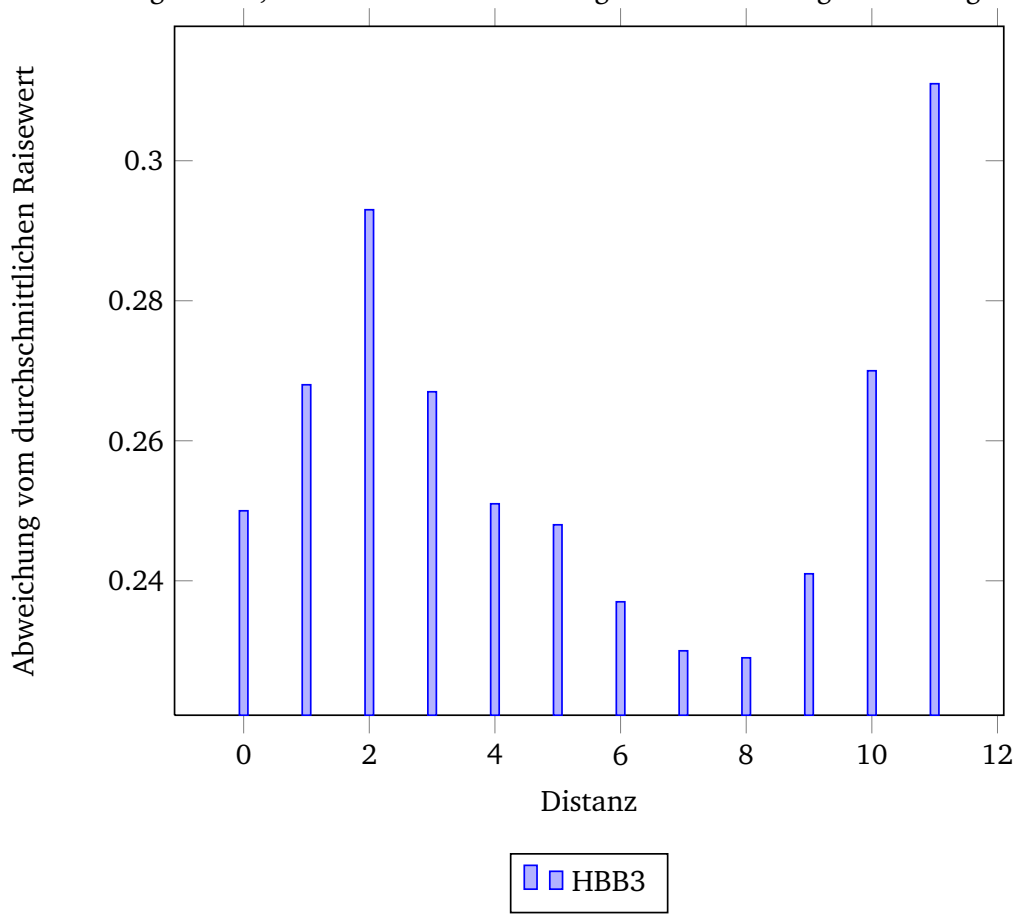


Diagramm 2: Distanzanalyse

Von den Diagrammen, die aus dieser Analyse hervorgingen, ist nur eins angeben, welches die Abbildung von HBB3 analysiert. Die Einträge wurden in Klassen aufgeteilt, wobei alle Einträge, die den gleichen Distanzwert haben, dieselben Klasse besitzen. Auf der y-Achse ist die durchschnittliche Abweichung der Klassen vom Durchschnittswert der jeweiligen Abbildung im Bezug auf eine *Aktion* angegeben. Der erste Balken im Diagramm bedeutet also, dass Zustände mit Distanz 0 durchschnittliche um 0.25 vom Durchschnittswert der Aktionswahrscheinlichkeit abweichen.

Der Hintergrund dieser Analyse ist, dass man davon ausgeht, dass bei guten Abbildung Spielverläufe mit hohen Distanzen nicht repräsentativ sind, also eine geringe Abweichung vom Durchschnitt besitzen.

Um die Abbildungen, die in einem Diagramm berücksichtigt werden, vergleichbar zu halten, sind für jedes Diagramm eine Aktionswahrscheinlichkeit und eine Abweichung gegeben. In unserem Diagramm werden nur Abbildungen berücksichtigt, wenn sie mit ihrer durchschnittlichen *Raisewahrscheinlichkeit* um nicht mehr als 0.1 von 0.7 abweichen.

In dieser Arbeit ist nur eins dieser Diagramme angegeben, da es repräsentative für die Ergebnisse dieser Analyse steht. Der erhoffte Zusammenhang konnte für keine Abbildung nachgewiesen werden.

Distanz	0	1	2	3	4	5	6	7	8	9	10	11
Anzahl Zustände	520	1593	2228	2964	3506	2859	2166	1263	397	83	12	2

Tabelle 6: Anzahl der Zustände, die für eine gewisse Distanz bei der Distanzanalyse erstellt wurden

Wichtig bei diesem Diagrammen ist es, die Größe der Klassen zu berücksichtigen, welche in Tabelle 6 angegeben sind. Hier kann man sehen, dass die Abbildung nur zwei nächste Zustände mit Distanz 11 gefunden hat, womit die berechnete Abweichung vom Durchschnitt für diese Distanz sicherlich kein aussagekräftiger Wert ist.

5.3 Bias

In Abschnitt 4.6 wurde bereits begründet, warum wir einen Bias auf die Aktionswahrscheinlichkeiten benötigen. Der folgende Abschnitt beschreibt mögliche Umsetzungen.

Dabei wurde bereits erwähnt, dass unseres *Bot* möglicherweise zu wenig *Hände Preflop* aufgibt. Um dies zu überprüfen vergleichen wir unsere *Bots* mit hyperborean, dem Sieger der Annual Poker Competition 2012. Dem zufolge wurde die Partie zwischen ihm und dem zweit und dritt platzierten der Annual Poker Competition 2012 analysiert. In der Tabelle 7 sind für jede mögliche *Wettrunden* und jede mögliche Anzahl von aktiven Spielern seine prozentualen Anteile der getätigten Aktionen angegeben.

Die Tabellen 8, 9 und 10 hingegen enthalten die Werte unserer *Bots* und deren Differenz zu den Werten von hyperborean.

Dabei lässt sich ein deutlichen Unterschied nur bei drei aktiven Spielern *Preflop* feststellen. Der logische erste Schritt wäre es, die Wahrscheinlichkeit *Preflop* zu *folden* zu erhöhen. Allerdings muss dies in Abhängigkeit der gehaltenen Karten geschehen. Ansonsten würden unsere *Bots* sogar zwei Asse hin und wieder vor dem *Flop* weglegen.

Der Anhang B enthält Tabellen mit den Aktionswahrscheinlichkeiten der *Bots Preflop* im Bezug auf die gehaltenen Karten. Hierbei soll noch erwähnt werden, dass bei der Erstellung der Tabellen für unsere *Bots*, eine erhöhte Anzahl von gespielten Händen sinnvoll gewesen wäre.

Ein möglicher Bias für einen *Bot* könnte sich aus den Differenzen zwischen seiner Tabelle und der Tabelle von hyperborean berechnen. Hierbei wurden Situationen, in denen kein Einsatz zu leisten ist, in der Statistik nicht berücksichtigt, da logischerweise hier ein *fold* keinen Sinn macht.[9] Zustätzlich wurden Situationen mit zwei aktiven Spielern ignoriert.

Als Beispiel betrachten wir die *Hand* Q2s. HBB2 hat mit dieser *Hand* zu 0 Prozent *gefoldet*, hyperborean hingegen zu 0.481 Prozent. Sollte sich das Spiel in der *Wettrunde Preflop* befinden und HBB2 diese *Hand* halten, wird er seine Wahrscheinlichkeit zu *folden* um 48.1 Prozent erhöhen. In diesem Sinne wurde für jede mögliche Kombination von gehaltenen Karten ein Biaswert bestimmt, der beim Programmstart aus einer Textdatei geladen wird.

Tabelle 11 enthält die Ergebnisse von duplicated Matches zwischen unseren *Bots* und zwei Instanzen des *Bots* dpp einmal mit und einmal ohne Bias. Wie in Abschnitt 5.2 wurden die duplicated Matches nur mit drei Positionierungen durchgeführt

Die ausgeteilten Karten unterscheiden sich zwischen den duplicated Matches nicht. Logischerweise sind es aber andere Karten als die, die bei der Erstellung der Biaswerte verwendet wurden.

Zu aller erst kann man feststellen, dass die Spiele mit Bias erfolgreicher waren. Allerdings erscheinen die Ergebnisse schlechter, wenn man weiß, dass in einem zwei Spieler Spiel eine

[9] Dies kommt *preflop* nur vor, wenn der Spieler *Big Blind* ist und vorher kein Spieler *geraised* hat.

CFR genierte Strategie einem mathematisch fairen *Bot* weit überlegen ist. Trotzdem sollte man festhalten, dass weiter Verbesserungen in diese Richtung gehen sollten.

5.4 Verschiedenes

Neben den in dieser Arbeit vorgestellten Ansätzen wurden noch weitere Ideen getestet, von denen hier nur zwei beschrieben werden sollen, da auch durch diese keine starke Verbesserung erreicht werden konnte.

HBB5 ist ein *Bot*, der den gleichen Filter wie HBB2 verwendet, also bei einer Abbildung alle Spielverläufe, die sich in der gleichen *Wettrunde* wie der aktuelle Spielzustand befinden, zurückliefert. Bei der Sortierung der nächsten Spielverläufe ist zu aller Erst entscheidend, welche *potodds* der Spielverlauf besitzt. Haben die Spielverläufe die gleiche Distanz in den *potodds* zum aktuellen Spielzustand entscheidet die Größe des Pots. Ist auch diese gleich wird die Vergleichsmethode der **AdvancedComparator** Klasse, deren Implementierung in 4 angedeutet ist, aufgerufen.

Dieser *Bot* wurde aufgrund der Beobachtung in Abschnitt 5.2 implementiert, in der festgestellt wurde, dass durch das Filtern in machen Fällen sogar keine Spielverläufe mehr zu Verfügung stehen, die einen gültigen Spielzustand produzieren. Die Ergebnisse eines Matches von ihm gegen zwei Instanzen des *Bots* dpp sind in der Tabelle 12 angegeben.

Eine andere Idee die getestet wurde, modifiziert die Methode die aufgrund der Aktionswahrscheinlichkeiten eine Aktion wählt, indem immer die Aktion mit der größten Wahrscheinlichkeit ausgewählt wird. Diese Modifikation wurde mit HBB5 getestet, wobei die Ergebnisse in Tabelle 12 angegeben sind. Die Karten und die Gegner unterscheiden sich zwischen den beiden Partien aus der Tabelle nicht.

Die Ergebnisse mit und ohne Modifikation sind zwar ähnlich, allerdings ist zu beachten, dass wenn man in einem Spielzustand immer die gleiche Aktion wählt, nur gegen *Bots* ein Problem bekommt, die versuchen ihren Gegner auszunutzen. Die mathematisch fairen *Bots*, die in diesem Test verwendet wurden, gehört nicht zu dieser Klasse von Gegnern.

hyperborean	preflop	flop	turn	river
2	(0.139,0.511,0.349)	(0.085,0.511,0.403)	(0.084,0.565,0.349)	(0.083,0.556,0.359)
3	(0.545,0.107,0.347)	(0.15,0.543,0.305)	(0.156,0.516,0.327)	(0.19,0.529,0.28)

Tabelle 7: Aktionen von hyperborean bei 2 bzw. 3 aktiven Spielern in den jeweiligen *Wettrunden* im Format (*fold*, *call*, *raise*)

HBB2	preflop	flop	turn	river
2	(0.076,0.229,0.694)	(0.065,0.487,0.447)	(0.086,0.508,0.404)	(0.086,0.514,0.398)
	(0.063, 0.282, -0.345)	(0.02, 0.024, -0.043)	(-0.001, 0.056, -0.055)	(-0.002, 0.042, -0.039)
3	(0.122,0.124,0.753)	(0.017,0.347,0.634)	(0.043,0.47,0.486)	(0.054,0.463,0.482)
	(0.423, -0.017, -0.406)	(0.133, 0.196, -0.329)	(0.113, 0.046, -0.158)	(0.136, 0.066, -0.201)

Tabelle 8: Aktionen von HBB2 bei 2 bzw. 3 aktiven Spielern in den jeweiligen *Wettrunden* im Format (*fold*, *call*, *raise*)

HBB3	preflop	flop	turn	river
2	(0.076,0.229,0.694)	(0.072,0.379,0.548)	(0.075,0.509,0.415)	(0.081,0.519,0.399)
	(0.063, 0.282, -0.345)	(0.013, 0.132, -0.145)	(0.009, 0.055, -0.066)	(0.002, 0.037, -0.04)
3	(0.1,0.23,0.669)	(0.121,0.421,0.456)	(0.123,0.505,0.371)	(0.13,0.455,0.413)
	(0.445, -0.123, -0.322)	(0.028, 0.122, -0.151)	(0.033, 0.011, -0.043)	(0.06, 0.074, -0.132)

Tabelle 9: Aktionen von HBB3 bei 2 bzw. 3 aktiven Spielern in den jeweiligen *Wettrunden* im Format (*fold*, *call*, *raise*)

HBB4	preflop	flop	turn	river
2	(0.076,0.227,0.696)	(0.075,0.375,0.549)	(0.075,0.516,0.408)	(0.094,0.515,0.39)
	(0.063, 0.284, -0.347)	(0.01, 0.136, -0.146)	(0.009, 0.048, -0.059)	(-0.01, 0.041, -0.031)
3	(0.1,0.286,0.612)	(0.029,0.534,0.436)	(0.05,0.505,0.444)	(0.111,0.455,0.433)
	(0.445, -0.179, -0.265)	(0.121, 0.009, -0.131)	(0.106, 0.011, -0.117)	(0.079, 0.074, -0.152)

Tabelle 10: Aktionen von HBB4 bei 2 bzw. 3 aktiven Spielern in den jeweiligen *Wettrunden* im Format (*fold*, *call*, *raise*)

-	ohne Bias	mit Bias
HBB2	-1698	-1245
HBB3	-1378	-716
HBB4	-1318	-737

Tabelle 11: Ergebnisse in duplicated Matches gegen zwei Instanzen des *Bots* dpp mit und ohne bias (10 = 1 *Big Blind*, 3000 *Hände* pro Match)

HBB5	HBB5 mit Mod
-1966	-2073

Tabelle 12: Ergebnisse von HBB5 in duplicated Matches gegen zwei Instanzen des *Bots* dpp einmal mit einmal ohne Modifikation (10 = 1 *Big Blind*, 6000 *Hände* pro Match)

Bei der Abbildung von drei auf zwei Spieler Zuständen ergeben sich mehrere Probleme. Aufgrund der Regel, dass im zwei Spieler Spiel der *Big Blind* ab dem *Flop* als erster an der Reihe ist, sind nicht einmal der drei Spieler Verlauf, in dem der *Button foldet* und der zwei Spieler Verlauf, in dem noch keine Aktion ausgeführt wurde, identisch. Auf dieses Problem wurde in Kapitel 5 eingegangen.

Dass bei einer Abbildung viel Information verloren geht, kann man allein dadurch erkennen, dass wir einer Menge von 12827520 drei Spieler Spielverläufen einer Menge von 6378 zwei Spieler Spielverläufen zuordnen. Deshalb erscheint es sinnvoll, dass wir für jeden Spieler einen eignen Spielverlauf erstellt haben.

Die folgenden Beispiele sollen verdeutlichen, dass durch unsere Abbildungen falsche Entscheidungen getroffen werden. Für die Gegner sind keine *Handkarten* angegeben, da sie unserem *Bot* im Spiel logischerweise nicht bekannt sind.

$$S_1 = ([\{(\_,\_),(\_,\_)\}, \{(\_,\_),(\_,\_)\}, \{(A,h),(K,d)\}], \{(7,s),(8,s),(9,s)\}, \{\}, \{\}, [r, c, c], [r, r], [], [])$$

In diesem Spielzustand sind die Größe des *Pots* mit 90 und die *potodds* mit 90:20 gegeben. Hierbei muss erwähnt werden, dass ein *Big Blind* einen Wert von 10, ein *Small Blind* einen Wert von 5 besitzt.

Trotz schlechtem *Flop* und *raises* der Gegner würde HBB2 zu 99% *callen*. Wenn man bedenkt, dass unsere *Bots* nicht auf die Gegner reagieren und versuchen optimal zu spielen, sind diese Aktionswahrscheinlichkeiten nicht verständlich.

In Tabelle 13 sind einige zwei Spieler Zustände angegeben, die von einer Abbildung erzeugt werden könnte. Der erste Eintrag in der Tabelle wäre zum Beispiel ein Zustand, der nach Abbildung 2 ein Distanz von 0 hätte.

Bei einer Abbildung durch HBB2 würde dieser *Bot*, da beide Gegner sich gleich verhalten, für jeden diesen Spielzustand erstellen und den Schnitt der Aktionswahrscheinlichkeiten bilden, wodurch sich die 99% Chance zu *callen* erklärt.

Der zweite Einträg der Tabelle enthält einen Eintrag, der die gleiche Größe des *Pots* und die gleichen *potodds* wie der ursprüngliche Zustand besitzt. Dies soll verdeutlichen, dass wenn wir uns an die bisherigen Auswahlmethoden für zwei Spieler Spielverläufe halten, uns nicht von 99% Chance zu *callen* entfernen.

Da wie beschrieben eine Erhöhung der *Foldwahrscheinlichkeit* sinnvoll wäre, macht es Sinn, dass unsere bisherigen Ansätze noch nicht funktioniert haben beziehungsweise der Ansatz eines Bias eine Verbesserung erzielen konnte.

Als zweites Beispiel betrachten wir folgenden Spielzustand:

$$S_2 = ([\{(\_,\_),(\_,\_)\}, \{(\_,\_),(\_,\_)\}, \{(K,s),(9,c)\}], \{(J,c),(3,d),(6,d)\}, \{\}, \{\}, [c, c, c], [c, c, c], [r,r], [])$$

Die Situation ist dem Spielzustand S_1 ähnlich. Die Gegner zeigen durch ihre Erhöhungen Stärke und wir selbst halten keine starke *Hand*. Die Größe des *Pots* ist mit 90 und die *potodds* sind mit 90:40 gegeben. Tabelle 14 enthält möglich zwei Spieler Spielverläufe.

Der erste Eintrag der Tabelle, ist ein zwei Spieler Zustand, der so von HBB2 für jeden Gegner erstellt werden könnte, da er eine Distanz von 0 besitzen würde. Somit erklärt dies, die von HBB2 berechnete Wahrscheinlichkeit von 73% zu *callen* und 26% zu *folden*.

Hero Position	Preflop	Flop	Pot	Potodds	CFR-Bot
SB	r,c	r	50	50:10	99% call
SB	r,c	r,r,r	90	90:20	99% call
BB	c,r,c	c,r	50	50:10	99% call

Tabelle 13: mögliche zwei Spieler Zustände, die von einer Abbildung für den Zustand S_1 erzeugt werden könnten

Hero Position	Preflop	Flop	Turn	Pot	Potodds	CFR-Bot
SB	cc	cc	r	40	40:20	73% call 26% fold
SB	cc	rc	r	60	60:20	13% call 86% fold

Tabelle 14: mögliche zwei Spieler Zustände, die von einer Abbildung für den Zustand S_2 erzeugt werden könnten

Wie zuvor setzen wir uns zum Ziel die Wahrscheinlichkeit zu *folden* zu erhöhen. Dies würden wir erreichen, wenn wir unseren drei Spieler Zustand auf den zweiten Eintrag in der Tabelle abbilden. Bei Betrachtung dieses Spielverlaufs stellen wir fest, dass sich die Größe des *Pots*, die *potodds* und die Aktion der Spieler geändert haben.

Leider wissen wir nicht, ob ein drei Spieler CFR-*Bot* eine erhöhte Foldwahrscheinlichkeit besitzt. Da dies aber anzunehmen ist, können wir festhalten, dass keine unserer Abbildungen, sofern sie nicht einen Bias verwendet, unser Ziel erreicht.

In diesem Zusammenhang kann man sich Gedanken machen, ob es Sinn machen würde, die Abbildungen mit einem anderen *Bot* weiter zu verbessern, für den wir sowohl eine drei Spieler als auch zwei Spieler Variante besitzen. Dass dies nicht so einfach möglich ist, sieht man daran, dass nicht alle *Bots* für einen Spielzustand ein Entscheidungstripel zurückliefern.

So können die Aktionswahrscheinlichkeiten, die aus den für jeden Gegner erstellten Zuständen hervorgehen, nicht gemittelt werden. Zum Beispiel liefert ein mathematisch fairer *Bot* für einen Spielzustand eine Aktion oder eine Gewinnwahrscheinlichkeit gegen eine zufällige *Hand* des Gegners zurück.

Des Weiteren soll untersucht werden, ob unsere *Bots* zumindest in zwei Spieler Situationen sinnvoll eingesetzt werden können.

Dass das Verwenden eines zwei Spieler CFR-*Bots* in zwei Spieler Situationen sinnvoll möglich ist, zeigt der bereits zuvor verwendete OwnBot [2]. Hier wurde ebenfalls durch das Ignorieren der Aktionen des ausgestiegenen Gegners und einer Umformung des restlichen Spielverlaufes ein zwei Spieler Zustand generiert, der dem CFR-*Bot* übergeben wurde.

Sartre3P [10] ist ein anderer *Bot*, der über eine Abbildung der Spielverläufe einen zwei Spieler *Bot* in einem drei Spieler Spiel verwendet. Die Abbildung wird dabei nur angewendet, wenn ein Spieler *Preflop* aufgegeben hat. Bei der Abbildung wird unter anderem die Position der Spieler und die Anzahl der *raises* bedacht. Sartre3P erreichte bei der ACPC 2011 nach dem total bankroll Verfahren im *limit* drei Spieler Wettbewerb den ersten Platz.

In Tabelle 15 sind die Ergebnisse eines Tests angegeben, in dem unsere *Bots* in einen mathematische fairen *Bot* eingebunden worden sind und nur in zwei Spieler Situationen beziehungsweise falls ein Spieler bereits *Preflop* aufgeben hat, angewendet wurden. Die Gegner waren die gleichen mathematisch fairen Spieler, die in keiner Situation unsere *Bots* verwendet haben, wobei die Partien wie in Kapitel 5 nicht mit allen möglichen Positionierungen durchgeführt wurden. Des Weiteren sind zum Vergleich Ergebnisse von Partien angegeben, in denen unsere *Bots* in allen Situationen (also auch in drei Spieler Situationen) angewendet worden sind.

-	HBB1	HBB2	HBB3	HBB4	HBB5
keine Anwendung der *Bots*	-41.0				
Bot wird immer angewendet	-2103.0	-863.0	-1796.0	-1095.0	-1196.0
ein Spieler hat *Preflop* aufgegeben	-543.0	469.0	-97.0	232.0	423.0
ein Spieler hat aufgegeben	-650.0	601.0	-717.0	344.0	525.0

Tabelle 15: Ergebnisse eines mathematisch fairen *Bots* mit Anwendung unserer *Bots* in verschiedenen Situationen gegen zwei Instanzen des selben mathematisch fairen *Bots* (10 = 1 *Big Blind*, 3000 *Hände* pro Match)

Hierbei wird offensichtlich, dass vor allem das Spiel mit drei Spielern ein Problem ist. Werden unsere *Bots* immer angewendet, verlieren sie sehr stark. Werden sie nur in zwei Spieler Situationen angewendet, können sie sehr oft die Leistung steigern. Ob es sinnvoll ist, einen *Bot* nur anzuwenden, wenn ein Gegner *Preflop* aufgegeben hat, lässt sich aus dieser Tabelle nicht ableiten.

Die zu Anfangs in Kapitel 4 eingeführten Abbildungen können als erste Versuche gesehen werden und brachten logischerweise noch nicht die gewünschten Ergebnisse. Die erste Abbildung formt die Spielverläufe durch das Löschen von Aktionen um, die zweite sucht nach Spielverläufen, in denen die Spieler eine ähnliche Aktionsfolge haben und die dritte entspricht der zweiten mit dem Unterschied, dass nur Spielverläufe mit einem möglichst ähnlich großem *Pot* betrachtet werden.

Des Weiteren konnte weder durch die in Kapitel 5 neu hinzugekommene Abbildung, noch durch die Betrachtung von mehreren nächsten Spielzuständen in Abschnitt 5.2 eine Verbesserung erzielt werden. Die in diesem Zusammenhang erstellte Distanzanalyse zeigte auch, dass kein Zusammenhang zwischen Abweichung vom Durchschnitt und der von unseren Abbildung berechneten Distanz existiert.

Ein erster Fortschritt konnte in Kapitel 5.3 mit einem Bias erreicht werden. Der logische nächste Schritt wäre es, die Aktionswahrscheinlichkeiten differenzierter, also nicht nur an den gehalten Karten, zu betrachten. Ein Beispiel wäre ein Bias für jede Kartenkombination und jeden Spielverlauf *Preflop*. Man sollte aber auch nicht vergessen, dass die Ergebnisse mit Bias zwar besser, allerdings alles andere als gut waren.

Dass ein Bias nicht nur *Preflop* nötig ist, wird in Abschnitt 6 erklärt. Kurz kann man das dadurch begründen, dass wenn man bei drei aktiven Spielern Abbildungen verwendet, die für jeden Gegner nach einem ähnlichen Spielzustand suchen, keine Anpassung an das drei Spieler statt findet.

Verwendet man einen grundsätzlich anderen Ansatz für eine Abbildung, ist logischerweise auch eine Lösung ohne Bias vorstellbar. Dass das eigentliche Problem das Spiel mit drei aktiven Spielern ist, wird in Kapitel 6 deutlich. Hier konnte die Leistung eines *Bots* durch die Anwendung unserer *Bots* in gewissen zwei Spieler Situationen verbessert werden.

Abschließend kann man sagen, dass keiner der Ansätze sehr gut funktioniert hat. Allerdings bieten diese Ansätze viel Raum für Verbesserung und auch viele weitere Ansätze sind denkbar. Angesichts der Ergebnisse scheint es aber fraglich, ob ein zwei Spieler *Bot* mit einfachen Abbildungen der Spielzustände auf ein drei Spieler Spiel angepasst werden kann.

Glossar

Aktionen in Texas Hold'em
: Ist ein Spieler an der Reihe kann er maximal zwischen den drei Aktionen fold, call und raise wählen. Ein Spieler der foldet steigt aus der aktuellen Runde aus, ein Spieler der called geht den vorherigen Einsatz seines Gegners mit. Bei einem Raise erhöht der Spieler seine Einsatz um den durch das Limit vorgeschriebenen Wert. 5, 6, 9–11, 13, 16, 19, 21–26, 28–30

all in
: Ein Spieler ist all in, wenn er durch seinen Einsatz sein gesamtes Geld dem Pot hinzugefügt hat. Dadurch dass die Computer Spieler im Fixed Limit unbegrenzt Geld besitzen können, muss dieser Sonderfall nicht betrachtet werden. 6

Bankroll
: Mit Bankroll bezeichnet man das gesamte Kapital eines Spielers. 6, 17

Bot
: Ein von einem Computer gesteuerter Spieler. 3, 4, 6–8, 12–24, 26–32

Bucketing
: Im CFR Algorithmus werden die Hände in Klassen unterteilt. Dieser Vorgang wird als Bucketing bezeichnet und ist nötig um die Größe des Spiel in den Griff zu bekommen. Die erste Idee wäre es, die Hände anhand ihrer Stärke zu unterscheiden. Ein spezielle Hand kann zwischen den einzelnen Wettrunden die Buckets wechseln. Somit besitzen alle Hände mit der gleichen Bucket Reihenfolge dieselben Aktionswahrscheinlichkeiten[7, Kapitel 2.5.3]. 10, 11

Hand
: Der Begriff Hand besitzt zwei Bedeutungen. Zum einen bezeichnet man damit seine gehalten Karten, zum anderen eine komplette Spielrunde. 3, 5–9, 11, 13, 16–18, 21, 22, 26, 28–31

hand-range
: Mit hand-range bezeichnet man die möglichen Karten, die ein Spieler in einem bestimmten Spielzustand erwartungsgemäß hält. 16

Handkarten
: Die Karten, die einem Spieler gehören und nur für ihn sichtbar sind, werden als Handkarten bezeichnet. 3, 13, 29

headsup
: Ein Poker Spiel mit zwei Spielern wird als headsup bezeichnet. 11

Information Set
: Ein Information Set ist eine Menge von Spielzuständen, zwischen denen ein Spieler aus seiner Sicht nicht unterscheiden kann. 8–11

Limits im Texas Hold'em

Texas Hold'em wird in verschieden Limits gespielt. In den jeweiligen Limits sind unterschiedliche Einsatzgrößen erlaubt. In dem für diese Arbeit wichtigen Fixed Limit sind die Einsätze für die einzelnen Wettrunden fest vorgeschrieben. Eine Erhöhung in den ersten beiden Wettrunden beträgt immer ein, in den anderen beiden Wettrunden immer zwei Big Bilnds. Zu dem darf die Aktion raise in der ersten Wettrunde 3, in den anderen Wettrunden nur 4 mal ausgeführt werden. 4, 6, 30

Multiplayer

Spiel mit mehr als zwei Spielern. 3, 4, 11

nash-optimal

Ein Spiel befindet sich in einem Nash-Gleichgewicht, wenn es für die beiden Teilnehmer nicht von Vorteil ist, wenn nur sie ihre Strategie ändern. Diese Strategien werden auch nash-optimal Strategien genannt. 3, 4, 7–9, 11

Nullsummenspiel

In einem Nullsummenspiel entsprechen die Gewinne der Spieler immer den Verlusten der anderer Spieler. 9

opponent-modelling

Ein Spieler betreibt opponent-modelling, wenn er die Aktionen seiner Gegenspieler beobachtet, um Fehler in ihren Strategien zu entdecken, damit er diese auszunutzen kann. 16

Positionen in Texas Hold'em

Die Position der Spieler bestimmt die Handlungsreihenfolge. Im drei Spieler Spiel existieren die Positionen Small Blind, Big Blind und Button. Mit Small und Big Blind werden auch die Einsätze der Spieler in den Positionen Small und Big Blind bezeichnet, welche sie vor dem Beginn einer Hand erbringen müssen. Dabei ist gewöhnlich der Big Blind doppelt so groß wie der Small Blind. 5, 6, 8, 13–15, 17–19, 21, 22, 26, 28, 29, 31

Pot

Im Pot werden die Einsätze aller Spieler gesammelt. Der Gewinner einer Runde erhählt den Pot. 5, 6, 16, 18, 29, 30, 32

potodds

Potodds ist das Verhältnis von Größe des Pots zu Höhe des Einsatzes, der bei einem call erbracht werden muss. 19, 21, 27, 29, 30

Showdown

Ist eine Spielrunde vorbei und mehr als zwei Spieler haben nicht aufgegeben, kommt es zum Showdown. Jeder Spieler wählt aus seinen Handkarten und den öffentlichen Tischkarten fünf Karten aus, sodass die für ihn beste Kombination entsteht. Der Spieler mit der besten Kombination erhählt den Pot.. 6

Strategieprofil

Im Poker besitzt jeder Spieler eine Position, die die Handlungsreihenfolge der Spieler

bestimmt. Offensichtlich ist eine Hand aus der Sicht eines Spielers mit geänderter Position unterschiedlich. Demnach bezeichnet ein Strategieprofil eine Menge von Strategien, wobei jede Strategie für genau eine Position vorgesehen ist. Der Begriff Halbstrategie bezeichnet in einem Spiel mit zwei Spielern eine Strategie für eine der beiden Positionen. 3, 8, 9

Texas Hold'em

Texas Hold'em ist ein Pokervariante, in der die Anzahl der Tischkarten auf fünf und die Anzahl der Handkarten für jeden Spieler auf zwei begrenzt ist. 3, 4, 6, 8, 12

Tischkarten

Alle Karten, die während einer Runde für alle Spieler sichtbar sind, werden Tischkarten genannt. 3, 5, 6, 13

Wettrunden in Texas Hold'em

Texas Hold'em ist in vier Wettrunden unterteilt. Eine Wettrunde endet wenn alle Spieler bis auf einer ausgestiegen sind oder ein Spieler zum zweiten mal an der Reihe wäre und die vorangegangen Aktionen seiner Gegner entweder fold oder call waren. Die Wettrunden werden Preflop, Flop, Turn und River genannt. Die Tischkarten, die zu Beginn der letzten drei Wettrunden aufgedeckt werden, tragen den selben Namen wie die jeweilige Wettrunde. 5, 6, 13–16, 18–21, 24, 26–32

[1] Annual Computer Poker Competition. Website. http://www.computerpokercompetition.org/, Zugriff am 27.2.2013.

[2] Denny Dittmar, Nikolaus Korfhage, Christian Olczak. Ownbot pokerchallenge praktikum ki. 2011. Praktikumsbericht.

[3] Richard Gibson, Marc Lanctot, Neil Burch, Duane Szafron, Michael Bowling. Generalized sampling and variance in counterfactual regret minimization. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI)*, Seiten 1355–1361, 2012.

[4] Richard Gibson, Duane Szafron. On strategy stitching in large extensive form multiplayer games. *Advances in Neural Information Processing Systems*, 24:100–108, 2011.

[5] Michael Johanson, Nolan Bard, Neil Burch, Michael Bowling. Finding optimal abstract strategies in extensive-form games. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI)*, 2012.

[6] Michael Johanson, Nolan Bard, Marc Lanctot, Richard Gibson, Michael Bowling. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, Seiten 837–846. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[7] Michael Bradley Johanson. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta, 2007.

[8] Nicholas Abou Risk. Using counterfactual regret minimization to create a competitive multiplayer poker agent. Master's thesis, University of Alberta, 2009.

[9] Nick Abou Risk, Duane Szafron. Using counterfactual regret minimization to create competitive multiplayer poker agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, Seiten 159–166. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[10] Jonathan Rubin, Ian Watson. Case-based strategies in computer poker. *AI Communications*, 25(1):19–48, 2012.

[11] Duane Szafron, Richard Gibson, Nathan Sturtevant. A parameterized family of equilibrium profiles for three-player kuhn poker. 2013. Praktikumsbericht.

[12] Wikipedia. Website. http://de.wikipedia.org/wiki/Texas_Hold'em, Zugriff am 27.3.2013.

[13] Martin Zinkevich, Michael Johanson, Michael Bowling, Carmelo Piccione. Regret minimization in games with incomplete information. *Advances in neural information processing systems*, 20:1729–1736, 2008.

Anhang

A Sourcecode

Listing 1: FirstTry

```
1  public class FirstTry {
2          Map<Integer, List<ActionList>> histories_2player;       //alle moeglichen 2 Spieler Wettrunden
3          int max_Length_2player_histories;                       //die maximale Laenge einer 2 Spieler Wettrunde
4          List<ActionList> histories_3player;                     //alle mÃűglichen 3 Spieler Wettrunden
5          Map<String, List<Action>> mapping;                      //die Abbildung
6
7          private void computeMapping(){
8                  computeAllPossible2PlayerHistories(new ActionList());
9                  computeAllPossible3PlayerHistories(new ActionList());
10                 for(ActionList al : histories_3player){
11                         computeMapping(al);
12                 }
13                 saveMapping();
14         }
15         private void computeMapping(ActionList al){
16                 int i=0;
17                 if(al.size() > max_Length_2player_histories){
18                         i = al.size() - max_Length_2player_histories;   //i == minimale Anzahl an Aktionen
19                 }                                                       //die entfernt werden muessen
20                 for(; i<=al.size(); i++){
21                         if(computeMappingByRemoving(i, al)){
22                                 break;
23                         }
24                 }
25         }
26         private boolean computeMappingByRemoving(int deleteCount, ActionList al){
27                 return computeMappingByRemoving(deleteCount, 0, al, new ArrayList<Integer>());
28         }
29         private boolean computeMappingByRemoving(int x, int start, ActionList al, ArrayList<Integer> removeList){
30                 if(x == 0){
31                         return computeMappingByRemoving(al, removeList);
32                 }else{
33                         for(int i = start; i<al.size(); i++){
34                                 int index = removeList.size();
35                                 removeList.add(i);
36                                 if(computeMappingByRemoving(x-1, i+1, al, removeList)){
37                                         return true;
38                                 }
39                                 removeList.remove(index);
40                         }
41                         return false;
42                 }
43         }
44         private boolean computeMappingByRemoving(ActionList al, ArrayList<Integer> removeList){
45                 List<Action> alCopy = new ArrayList<Action>(al);
46                 for(int i=0; i<removeList.size(); i++){
47                         alCopy.remove(removeList.get(i) - i);
48                 }
49                 List<ActionList> histories = histories_2player.get(alCopy.size());
50                 if(histories != null){
51                         for(int i=0; i<histories.size(); i++){
52                                 //check testet ob beide Wettrunden abgeschlossen oder beide nicht abgeschlossen sind
53                                 if(histories.get(i).equals(alCopy) && check(histories.get(i), al)){
54                                         mapping.put(al.toString(), alCopy);
55                                         return true;
56                                 }
57                         }
58                 }
59                 return false;
60         }
61 }
```

Listing 2: PotSizeFilter

```
1  public class PotSizeFilter implements Filter{
2
3          private Map<Integer, List<Integer>> sortedKeys;
4          private Map<Integer, Map<Integer, List<History2Player>>> mapping;
5
6          private int[] computeClosestPotSizes(int street, int potSize){
7                  List<Integer> keys = sortedKeys.get(street);
8                  int actualKey = keys.get(0);
9                  for(int i=1; i<keys.size(); i++){
10                         if(Math.abs(keys.get(i) - potSize) < Math.abs(actualKey - potSize)){
11                                 actualKey = keys.get(i);
12                         }else if(Math.abs(keys.get(i) - potSize) == Math.abs(actualKey - potSize)){
13                                 int[] result = {actualKey, keys.get(i)};
14                                 return result;
15                         }else{
16                                 break; //da die keys sortiert sind, kann hier abgebrochen werden
17                         }
18                 }
19
20                 int[] result = {actualKey};
21                 return result;
22         }
23
24         @Override
25         public List<History2Player> computeHistories(GameState gameState) {
26                 int street = gameState.getCurrentStreet().ordinal();
27                 int potSize = (int) (gameState.getPotsize() / FixValues.SMALL_BLIND_VALUE);
28                 int[] closestPotSizes = computeClosestPotSizes(street, potSize);
29
30                 List<History2Player> result = new ArrayList<History2Player>();
31                 for(int i=0; i<closestPotSizes.length; i++){
32                         result.addAll(mapping.get(street).get(closestPotSizes[i]));
33                 }
34
35                 return result;
36         }
37
38 }
```

Listing 3: Bot

```
public abstract class Bot extends PokerClient{
        /* ...  */
        protected boolean weightProbabilities;
        protected boolean bias;
        private int numClosestHistories;
        /* ... */
        protected Action getAction(GameState gameState, Comparator comparatorFirstOpponent,
                        Comparator comparatorSecondOpponent, Filter filter){

                List<History2Player> histories = filter.computeHistories(gameState);

                if(gameState.getActivePlayerCount() == 2){
                        double[] probabilities = computeProbabilities(histories, gameState,
                                                                2, comparatorFirstOpponent);
                        if(bias){
                                double[] biasedDecision = computeBiasedDecision(
                                                                probabilities[0], probabilities[1], gameStat
                                return computeDecision(biasedDecision[0], biasedDecision[1]);
                        }
                        return computeDecision(probabilities[0], probabilities[1]);
                }else{ //assumption playercount == 3

                        double[] probabilities = computeProbabilities(histories, gameState,

                        double[] probabilities2 = computeProbabilities(histories, gameState,

                        double raiseProb = (probabilities[0] + probabilities2[0]) / 2.0;
                        double callProb = (probabilities[1] + probabilities2[1]) / 2.0;
                        if(bias){
                                double[] biasedDecision = computeBiasedDecision(
                                                        raiseProb, callProb, gameState);
                                return computeDecision(biasedDecision[0], biasedDecision[1]);
                        }
                        return computeDecision(raiseProb, callProb);
                }
        }
        protected double[] computeProbabilities(List<History2Player> histories,
                                GameState gameState, int numPlayers, Comparator comparator){
                if(weightProbabilities){
                        return computeWeightedProbabilities(/* ... */);
                }
                return computeUnweightedProbabilities(/* ... */);
        }
        private double[] computeUnweightedProbabilities(List<History2Player> histories,
                                GameState gameState, int numPlayers, Comparator comparator){
                /* ...  */
                List<GameState> closestHistories = Mapping.computeXClosestGameStates(
                        histories, numClosestHistories, comparator, cfrUtils, gameState, nanCount);
                /* ...  */
                return result;
        }
}
```

Listing 4: AdvancedComparator

```
1  public class AdvancedComparator extends StandardComparator{
2          public boolean isRaisePossible; //gibt an, ob im aktuellen Spiel ein raise moeglich ist
3          public Position preferredPosition;
4          public AdvancedComparator(/* ... */) {
5                  /* ... */
6                  this.preferredPosition = computePrefferedPosition(heroPosition, opponentPosition);
7          }
8          @Override
9          public int compare(History2Player h1, History2Player h2) {
10                 if(h1.isRaisePossible() != h2.isRaisePossible()){
11                         if(h1.isRaisePossible() == isRaisePossible){
12                                 return -1;
13                         }else{
14                                 return 1;
15                         }
16                 }
17                 //positionToAct entspricht unserer Position im zwei Spieler Spiel
18                 if(!preferredPosition.equals(Position.INVALID)){
19                         if(h1.getPositionToAct() != h2.getPositionToAct()){
20                                 if(h1.getPositionToAct() == preferredPosition){
21                                         return -1;
22                                 }else{
23                                         return 1;
24                                 }
25                         }
26                 }
27                 int dif1 = computeDif(h1);       //Distanz nach dem Verfahren von
28                 int dif2 = computeDif(h2);  //HBB2 und HBB3 berechnen
29                 if(dif1 == dif2){
30                         return 0;
31                 }else if(dif1 < dif2){
32                         return -1;
33                 }
34                 return 1;
35         }
36
37         protected Position computePrefferedPosition(Position heroPosition,
38                                         Position opponentPosition){
39                 if(heroPosition.equals(Position.SB) && opponentPosition.equals(Position.BU)){
40                         return Position.BB;      //unsere gewuenschte Position im zwei Spieler Spiel
41                 }else if(heroPosition.equals(Position.BB) && opponentPosition.equals(Position.BU)){
42                         return Position.BB;
43                 }else if(heroPosition.equals(Position.BU) && opponentPosition.equals(Position.SB)){
44                         return Position.SB;
45                 }else if(heroPosition.equals(Position.BU) && opponentPosition.equals(Position.BB)){
46                         return Position.SB;
47                 }
48                 return Position.INVALID; //es gibt kein zwei Spieler Spiel, indem wir wie
49                                 // im drei Spieler Spiel Position auf unseren Gegner haben
50         }
51
52 }
```

B Bias

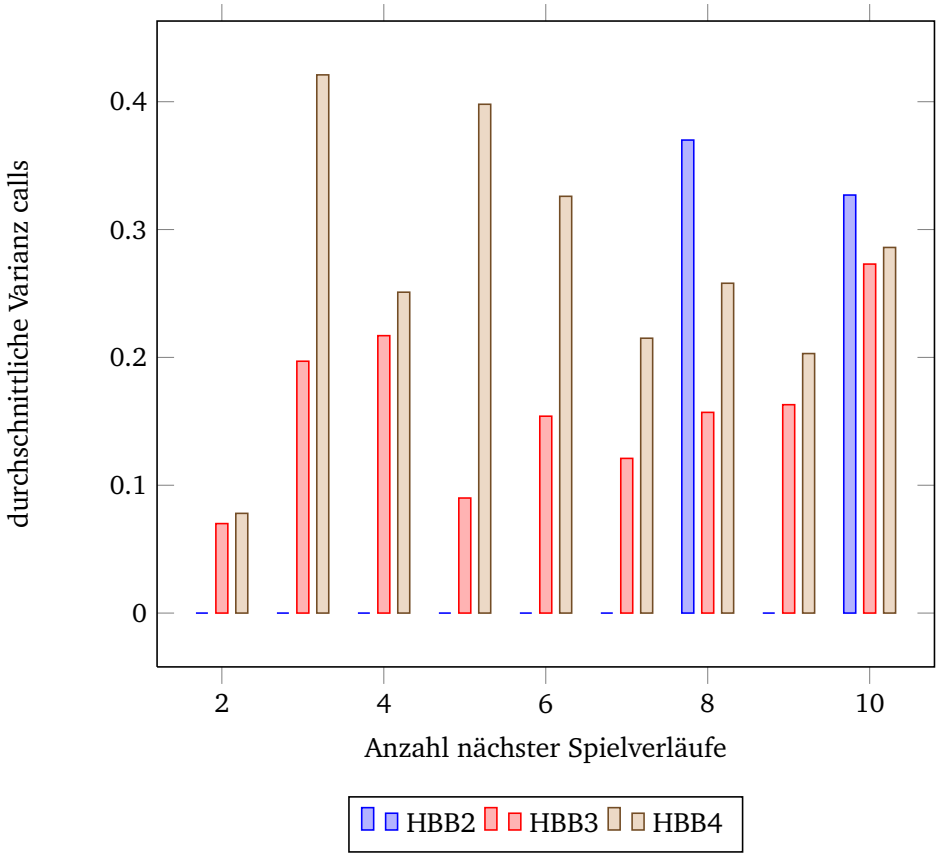
Die folgenden Tabellen enthalten die prozentualen Häufigkeiten der getätigten Aktionen im Bezug auf die gehaltenen Karten im Format $\begin{pmatrix} fold \\ call \\ raise \end{pmatrix}$. Ein Eintrag oberhalb der Diagonalen beschreibt Hände, in denen beide Karten dieselbe Farbe besitzen, ein Eintrag unterhalb der Diagonalen Hände mit Karten unterschiedlicher Farbe.

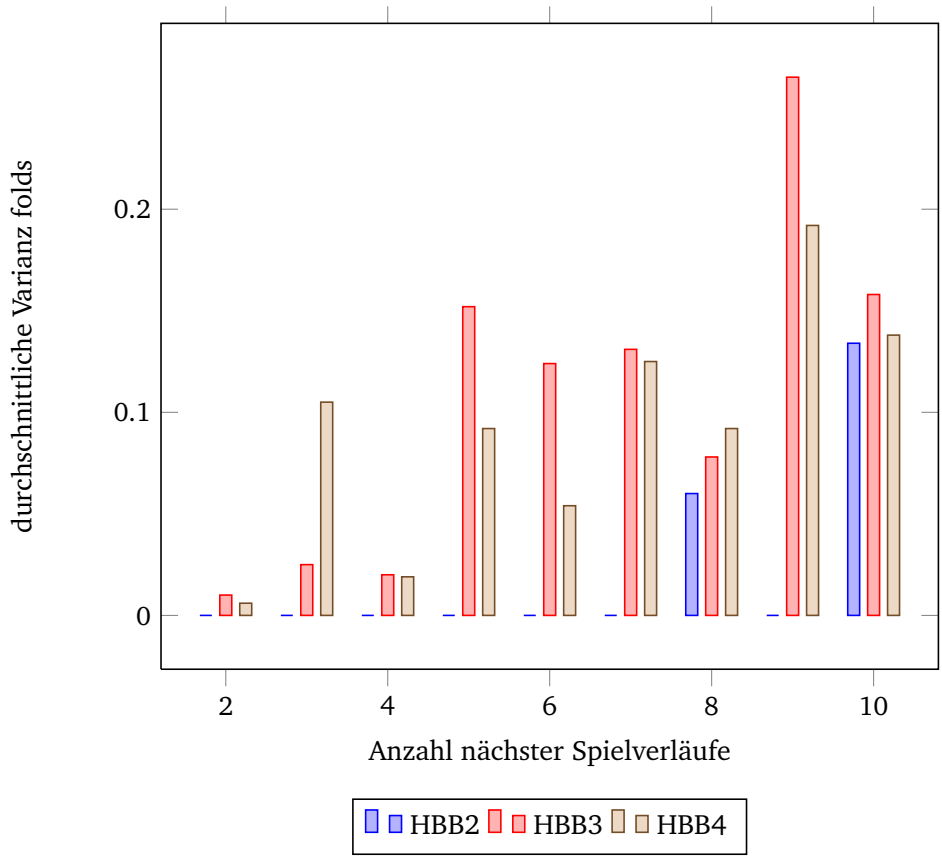
hyperborean	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.047	0.067	0.051	0.072	0.088	0.068	0.112	0.093	0.104	0.098	0.105	0.374	0.406
	0.952	0.932	0.948	0.927	0.911	0.931	0.887	0.906	0.895	0.901	0.894	0.625	0.593
K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.002	0.0	0.011
	0.088	0.067	0.07	0.069	0.09	0.108	0.409	0.295	0.256	0.303	0.381	0.437	0.506
	0.911	0.932	0.929	0.93	0.909	0.891	0.59	0.704	0.743	0.696	0.616	0.562	0.481
Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.001	0.034	0.285	0.95
	0.073	0.106	0.077	0.143	0.07	0.085	0.314	0.407	0.288	0.372	0.462	0.106	0.047
	0.926	0.893	0.922	0.856	0.929	0.914	0.685	0.592	0.711	0.625	0.504	0.607	0.002
J	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.196	0.325	0.59	0.955	0.982
	0.12	0.19	0.245	0.086	0.088	0.093	0.233	0.38	0.269	0.128	0.066	0.044	0.017
	0.879	0.809	0.754	0.913	0.911	0.906	0.766	0.619	0.533	0.545	0.343	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.002	0.272	0.956	0.941	0.958	0.95
	0.102	0.238	0.298	0.322	0.077	0.112	0.289	0.299	0.132	0.043	0.058	0.041	0.05
	0.897	0.761	0.7	0.677	0.922	0.887	0.71	0.698	0.595	0.0	0.0	0.0	0.0
9	0.0	0.054	0.049	0.085	0.06	0.0	0.0	0.0	0.005	0.928	0.963	0.961	0.959
	0.156	0.285	0.333	0.321	0.304	0.079	0.225	0.236	0.368	0.071	0.036	0.038	0.04
	0.843	0.66	0.617	0.593	0.635	0.92	0.774	0.763	0.625	0.0	0.0	0.0	0.0
8	0.009	0.296	0.928	0.891	0.392	0.323	0.0	0.0	0.0	0.297	0.953	0.959	0.959
	0.178	0.102	0.052	0.052	0.092	0.082	0.086	0.213	0.315	0.154	0.046	0.04	0.04
	0.811	0.6	0.019	0.056	0.515	0.593	0.913	0.786	0.684	0.548	0.0	0.0	0.0
7	0.048	0.473	0.959	0.964	0.962	0.948	0.841	0.0	0.0	0.0	0.96	0.965	0.954
	0.223	0.096	0.04	0.035	0.037	0.051	0.052	0.121	0.171	0.343	0.037	0.034	0.045
	0.727	0.43	0.0	0.0	0.0	0.0	0.106	0.878	0.828	0.656	0.002	0.0	0.0
6	0.061	0.956	0.961	0.975	0.974	0.97	0.974	0.961	0.0	0.0	0.001	0.95	0.978
	0.281	0.043	0.038	0.024	0.025	0.029	0.025	0.038	0.1	0.223	0.44	0.049	0.021
	0.657	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.899	0.776	0.557	0.0	0.0
5	0.052	0.953	0.964	0.976	1.0	1.0	0.974	0.969	0.964	0.0	0.0	0.865	0.971
	0.252	0.046	0.035	0.023	0.0	0.0	0.025	0.03	0.035	0.118	0.301	0.126	0.028
	0.694	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.881	0.698	0.007	0.0
4	0.075	0.971	0.965	0.982	1.0	1.0	1.0	0.961	0.967	0.965	0.0	0.931	0.947
	0.29	0.028	0.034	0.017	0.0	0.0	0.0	0.038	0.032	0.034	0.167	0.068	0.052
	0.634	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.832	0.0	0.0
3	0.295	0.977	0.967	0.978	1.0	1.0	1.0	1.0	0.965	0.961	0.967	0.0	0.97
	0.118	0.022	0.032	0.021	0.0	0.0	0.0	0.0	0.034	0.038	0.032	0.247	0.029
	0.585	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.752	0.0
2	0.967	0.957	0.974	1.0	1.0	1.0	1.0	1.0	1.0	0.964	1.0	1.0	0.009
	0.032	0.042	0.025	0.0	0.0	0.0	0.0	0.0	0.0	0.035	0.0	0.0	0.402
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.587

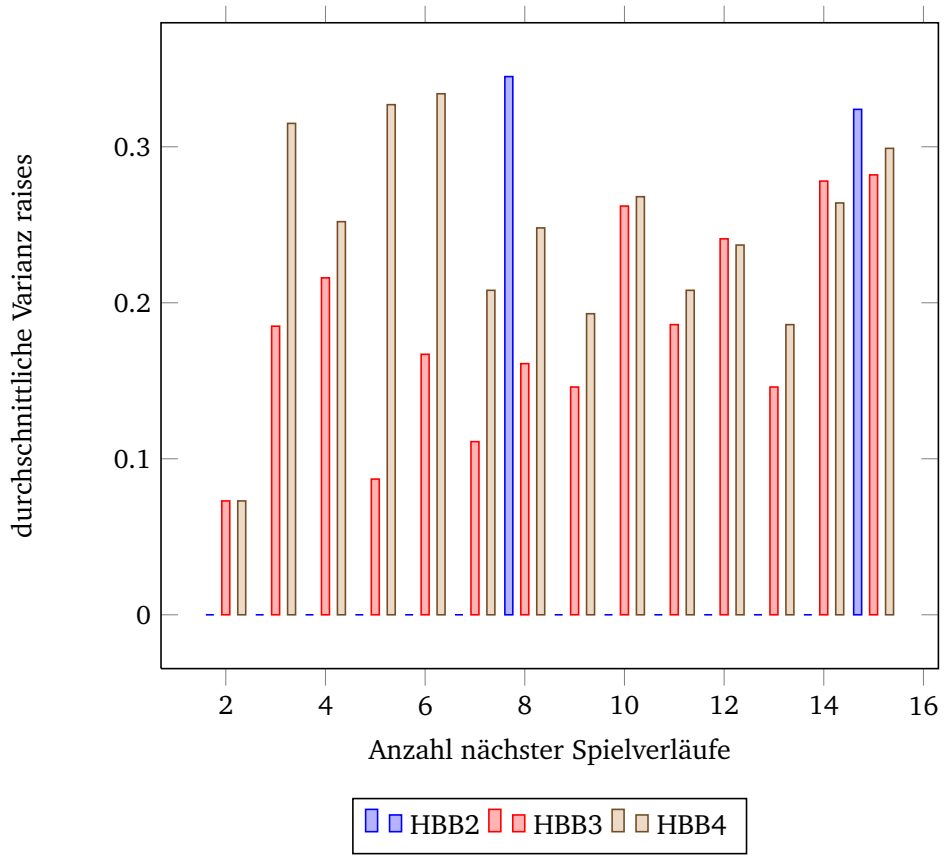
HBB2	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.125
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.875
J	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.166
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.833
T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.071	0.0	0.0	0.0	0.0	0.0	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	0.928	1.0	1.0	1.0	1.0	1.0	1.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.05
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.95
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.181	0.0	0.0	0.166	0.0	0.0	0.166	0.071
	1.0	1.0	1.0	1.0	1.0	0.818	1.0	1.0	0.833	1.0	1.0	0.833	0.928
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
	0.0	0.0	0.166	0.074	0.12	0.3	0.066	0.0	0.142	0.0	0.166	0.333	0.0
	1.0	1.0	0.833	0.925	0.88	0.7	0.933	1.0	0.857	1.0	0.833	0.666	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.75
	0.0	0.0	0.0	0.0	0.038	0.1	0.029	0.181	0.0	0.1	0.0	0.0	0.25
	1.0	1.0	1.0	1.0	0.961	0.9	0.97	0.818	1.0	0.9	1.0	1.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.038	0.0	0.02	0.0	0.187	0.066	0.055	0.0	0.125	0.083	0.166
	1.0	1.0	0.961	1.0	0.98	1.0	0.812	0.933	0.944	1.0	0.875	0.916	0.833
4	0.0	0.0	0.0	0.0	0.0	0.95	0.916	0.85	0.833	0.0	0.0	0.0	1.0
	0.0	0.136	0.25	0.083	0.038	0.05	0.083	0.15	0.166	0.166	0.0	0.0	0.0
	1.0	0.863	0.75	0.916	0.961	0.0	0.0	0.0	0.0	0.833	1.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.939	0.923	1.0	1.0	0.961	0.766	0.0	0.666
	0.0	0.04	0.222	0.058	0.041	0.06	0.076	0.0	0.0	0.038	0.233	0.0	0.333
	1.0	0.96	0.777	0.941	0.958	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.966	0.958	0.863	0.848	0.909	0.866	0.869	0.918	0.0
	0.0	0.121	0.136	0.0	0.033	0.041	0.136	0.151	0.09	0.133	0.13	0.081	0.25
	1.0	0.878	0.863	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.75

HBB3	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.5
J	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.25	0.666	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.75	0.333	1.0
T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.111	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.888	1.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.125	0.176	0.125	0.25	0.0
	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.875	0.823	0.875	0.75	1.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.333	0.0	0.0	0.333	0.0	0.062	0.083	0.076
	1.0	1.0	1.0	1.0	1.0	0.666	1.0	1.0	0.666	1.0	0.937	0.916	0.923
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
	0.0	0.0	0.416	0.153	0.115	0.4	0.066	0.0	0.142	0.0	0.25	0.0	0.0
	1.0	1.0	0.583	0.846	0.884	0.6	0.933	1.0	0.857	1.0	0.75	1.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
	0.0	0.0	0.156	0.187	0.192	0.25	0.09	0.238	0.0	0.5	0.0	0.0	0.5
	1.0	1.0	0.843	0.812	0.807	0.75	0.909	0.761	1.0	0.5	1.0	1.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.038	0.05	0.16	0.043	0.062	0.066	0.157	0.0	0.25	0.083	0.166
	1.0	1.0	0.961	0.95	0.84	0.956	0.937	0.933	0.842	1.0	0.75	0.916	0.833
4	0.0	0.0	0.0	0.0	0.0	0.9	0.833	0.6	0.666	0.0	0.0	0.0	0.0
	0.0	0.227	0.305	0.194	0.153	0.1	0.166	0.4	0.333	0.194	0.0	0.0	1.0
	1.0	0.772	0.694	0.805	0.846	0.0	0.0	0.0	0.0	0.805	1.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.666	0.769	0.909	0.857	0.846	0.6	0.0	0.0
	0.0	0.2	0.277	0.117	0.125	0.333	0.23	0.09	0.142	0.153	0.4	0.0	1.0
	1.0	0.8	0.722	0.882	0.875	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.666	0.72	0.545	0.727	0.795	0.857	0.75	0.888	0.0
	0.0	0.272	0.09	0.0	0.333	0.28	0.454	0.272	0.204	0.142	0.25	0.111	0.5
	1.0	0.727	0.909	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5

HBB4	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.0	0.0	0.083	0.0	0.333	0.1	0.2	0.25	0.0	0.666	0.25	0.25	0.333
	1.0	1.0	0.916	1.0	0.666	0.9	0.8	0.75	1.0	0.333	0.75	0.75	0.666
K	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.1	0.125	0.0	0.0	0.0	0.0	0.2	0.285	0.333	0.142	0.0	0.333	0.142
	0.9	0.875	1.0	1.0	1.0	1.0	0.8	0.714	0.666	0.857	1.0	0.666	0.857
Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.062	0.095	0.055	0.3	0.0	0.2	0.333	0.0	0.0	0.0	0.0	0.125	0.5
	0.937	0.904	0.944	0.7	1.0	0.8	0.666	1.0	0.0	1.0	0.0	0.875	0.5
J	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.093	0.0	0.25	0.5	0.2	0.0	0.5	0.333	0.083	0.0	0.5	0.666	0.666
	0.906	1.0	0.75	0.5	0.8	1.0	0.5	0.666	0.916	1.0	0.5	0.333	0.333
T	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.156	0.25	0.111	0.083	0.125	0.384	0.285	1.0	0.0	0.0	0.25	0.4	0.2
	0.843	0.75	0.888	0.916	0.875	0.615	0.714	0.0	1.0	1.0	0.75	0.6	0.8
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.178	0.277	0.233	0.448	0.184	0.083	0.0	0.25	0.125	0.294	0.25	0.333	0.157
	0.821	0.722	0.766	0.551	0.815	0.916	1.0	0.75	0.875	0.705	0.75	0.666	0.842
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.071	0.153	0.25	0.304	0.172	0.454	0.0	0.25	0.333	0.0	0.25	0.333	0.428
	0.928	0.846	0.75	0.695	0.827	0.545	1.0	0.75	0.666	1.0	0.75	0.666	0.571
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
	0.214	0.2	0.416	0.153	0.23	0.4	0.333	0.35	0.142	0.0	0.2	0.333	0.0
	0.785	0.8	0.583	0.846	0.769	0.6	0.666	0.65	0.857	1.0	0.8	0.666	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
	0.214	0.333	0.187	0.375	0.307	0.4	0.352	0.545	0.111	0.5	0.0	0.0	0.5
	0.785	0.666	0.812	0.625	0.692	0.6	0.647	0.454	0.888	0.5	1.0	1.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	0.166	0.285	0.076	0.2	0.44	0.416	0.25	0.333	0.222	0.285	0.5	0.333	0.5
	0.833	0.714	0.923	0.8	0.56	0.583	0.75	0.666	0.777	0.714	0.5	0.666	0.5
4	0.0	0.0	0.0	0.0	0.0	0.9	0.833	0.6	0.666	0.0	0.0	0.0	0.0
	0.307	0.333	0.388	0.388	0.307	0.1	0.166	0.4	0.333	0.388	0.166	0.0	1.0
	0.692	0.666	0.611	0.611	0.692	0.0	0.0	0.0	0.0	0.611	0.833	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.647	0.769	0.909	0.857	0.846	0.6	0.0	0.0
	0.4	0.23	0.5	0.294	0.333	0.352	0.23	0.09	0.142	0.153	0.4	0.571	1.0
	0.6	0.769	0.5	0.705	0.666	0.0	0.0	0.0	0.0	0.0	0.0	0.428	0.0
2	0.0	0.0	0.0	0.0	0.666	0.75	0.545	0.705	0.818	0.8	0.818	0.888	0.0
	0.2	0.294	0.272	0.0	0.333	0.25	0.454	0.294	0.181	0.2	0.181	0.111	0.5
	0.8	0.705	0.727	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5


durchschnittliche Varianz calls
0.4
0.3
0.2
0.1
0
2
4
6
8
10
Anzahl nächster Spielverläufe
HBB2
HBB3
HBB4


durchschnittliche Varianz folds
0.2
0.1
0
2
4
6
8
10
Anzahl nächster Spielverläufe
HBB2
HBB3
HBB4


durchschnittliche Varianz raises
0.3
0.2
0.1
0
2
4
6
8
10
12
14
16
Anzahl nächster Spielverläufe
HBB2
HBB3
HBB4