

Implementierung von Counterfactual Regret Minimization in zunehmend komplexen Pokervarianten

vorgelegt von

Friedemann Doll

EDV.Nr.: 924315

dem Fachbereich VI – Informatik und Medien
der Berliner Hochschule für Technik Berlin
vorgelegte Bachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Engineering (B.Eng.)

im Studiengang

Technische Informatik

Tag der Abgabe 2. Februar 2026



Gutachter

Prof. Dr.-Ing. Stefan Edlich

Prof. Dr.-Ing. habil. Alexander Löser

Berliner Hochschule für Technik

Berliner Hochschule für Technik

Studiere Zukunft

Abstract Counterfactual Regret Minimization (CFR) ist ein etablierter Algorithmus zur Berechnung von Nash-Gleichgewichten in Spielen mit imperfekter Information, der insbesondere in der Pokerspielforschung eingesetzt wird. Diese Arbeit untersucht die Leistungsfähigkeit verschiedener CFR-Varianten und Implementierungsansätze bei steigender Spielkomplexität.

Es wurden die Algorithmen Vanilla CFR, CFR+ und Discounted CFR (DCFR) in Python implementiert und auf vier Pokervarianten unterschiedlicher Komplexität angewendet: Kuhn Poker, Leduc Hold'em, Twelve Card Poker und Small Island Hold'em. Die Implementierungen umfassen einen dynamischen Ansatz, einen rekursiven Tree-basierten Ansatz sowie eine Layer-basierte Architektur. Zur Bewertung der Strategiequalität wurde die Exploitability mittels eines Best-Response-Agents auf Basis eines Public-State-Trees berechnet.

Die Ergebnisse zeigen, dass DCFR und CFR+ deutlich schneller konvergieren als Vanilla CFR, wobei DCFR generell besser abschneidet als CFR+, jedoch die Performance abhängig vom Spieltyp ist. Bei kleinen Spielen wie Kuhn Poker sind die Unterschiede zwischen den Implementierungsansätzen vernachlässigbar, während bei größer werdenden Spielen die Unterschiede signifikant werden. Bei Small Island Hold'em erweist sich Vanilla CFR mit der vorliegenden Implementierung als unpraktikabel, da selbst nach 45 Stunden Training keine hinreichend niedrige Exploitability erreicht werden konnte.

Die Arbeit demonstriert die Notwendigkeit algorithmischer und implementierungstechnischer Optimierungen für die Skalierung von CFR auf komplexere Pokervarianten und identifiziert die Problemstellen.

0.1 KI-Disclaimer

In dieser Arbeit wurden KI-basierte Tools, genauer gesagt Cursor verwendet.

Zu den Anwendungen in der Dokumentation gehörten Korrekturen von Rechtschreibung, Grammatik, sowie L^AT_EX-Formatierung.

Im Rahmen der Implementierung wurde Agentic Coding zum Schreiben, Dokumentieren und Debuggen von Code verwendet.

Darüber hinaus wurde in vielen Situationen KI als Diskussionspartner genutzt.

Inhaltsverzeichnis

0.1 KI-Disclaimer	3
1 Einleitung	11
1.1 Motivation und Relevanz	11
1.2 Zielsetzung und Fragestellungen	12
1.3 Aufbau der Arbeit	13
2 Theoretische Grundlagen	15
2.1 Grundlagen der Spieltheorie	15
2.1.1 Definition und Konzepte	15
2.1.2 Extensive Games with Imperfect Information	17
2.2 Counterfactual Regret Minimization	19
2.2.1 Definitionen	19
2.2.2 Regret-Minimierung	20
2.2.3 Counterfactual Regret	21
2.2.4 Regret Matching	22
2.2.5 CFR-Algorithmus	23
2.3 Monte Carlo CFR	23
2.3.1 Allgemeine Definition	23
2.3.2 Chance Sampling CFR	24
2.3.3 External Sampling MCCFR	24
2.3.4 Outcome Sampling MCCFR	25
2.3.5 Theoretische Eigenschaften	26
2.4 CFR+	26
2.4.1 Unterschiede zu Vanilla CFR	27
2.4.2 Regret Matching+	27

2.4.3	Tracking Regret	28
2.4.4	Linear Weighted Average	28
2.5	Discounted CFR	29
2.5.1	Linear CFR	29
2.5.2	Discounting	30
2.5.3	Gewichtung der Durchschnittsstrategie	30
2.5.4	Standard-Parameter	30
2.5.5	Optimistisches Regret Matching	30
2.6	Exploitability	31
2.6.1	Konventionelle Best Response Berechnung	31
2.6.2	Public State Tree	33
2.6.3	Effiziente Evaluation der Terminalknoten	34
2.6.4	Spielspezifische Isomorphismen	35
2.6.5	Parallele Berechnung	35
2.7	Hold'em-Spiele	35
3	Analyse	37
3.1	Spielvarianten	37
3.2	CFR-Solver	37
3.3	Evaluationsmethoden	38
4	Architektur und Design	39
4.1	Game Environment	40
4.2	CFR-Solver	40
4.2.1	Dynamisch erzeugter Spielbaum	41
4.2.2	Statischer Spielbaum	47
4.3	Best Response Agent	49
4.3.1	Gesamtlauf der Best-Response-Berechnung	49
4.3.2	Details der rekursiven Traversierung	50
5	Entwicklungsvergang	53
5.1	Chronologischer Ablauf	53
5.2	Erkenntnisse und Reflexion	55

6 Evaluation	57
6.1 Versuchsaufbau	57
6.2 Ergebnisse	59
6.2.1 Kuhn Poker	59
6.2.2 Leduc Hold'em	62
6.2.3 Twelve Card Poker	64
6.2.4 Small Island Hold'em	66
6.3 Diskussion	67
7 Fazit	69
7.1 Zielerreichung und Limitationen	69
7.2 Ausblick	70
7.2.1 Implementierungsoptimierungen	70
7.2.2 Algorithmische Optimierungen	71
A Pokerhände-Übersicht	73
B Spieldefinitionen	75
B.0.1 Kuhn Poker	75
B.0.2 Leduc Hold'em	75
B.0.3 Zwölf-Karten-Poker	76
B.0.4 Small Island Hold'em	76
B.0.5 Rhode Island Hold'em	76
B.0.6 Limit Texas Hold'em	77
C M-Wert	79
Literaturverzeichnis	81

Abbildungsverzeichnis

2.1	Game Tree für Kuhn Poker	32
2.2	Information Set Trees für beide Spieler in Kuhn Poker	32
2.3	Public State Tree für Kuhn Poker	34
4.1	Klassendiagramm der Game Environment für Kuhn Poker und Leduc Hold'em	40
4.2	Klassendiagramm der dynamischen CFR Solver Architektur	41
4.3	Aktivitätsdiagramm des Trainingsprozesses der dynamischen CFR Solver Implementierung	42
4.4	Aktivitätsdiagramm der Spielbaum-Traversierung der rekursiven Implementierung	46
4.5	Klassendiagramm der Layer-by-Layer CFR Solver Architektur	47
4.6	Aktivitätsdiagramm der Layer-by-Layer Traversierung	48
4.7	Aktivitätsdiagramm der Best-Response-Berechnung	49
4.8	Aktivitätsdiagramm der Public-State-Tree-Traversierung	50
6.1	Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Kuhn Poker	59
6.2	Vergleich der Implementierungsansätze und Gesamtvergleich aller Kombinationen bei Kuhn Poker	61
6.3	Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Leduc Hold'em	62
6.4	Vergleich der Implementierungsansätze und Gesamtvergleich aller Kombinationen bei Leduc Hold'em	63
6.5	Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Twelve Card Poker (Layer-basierter Ansatz)	64
6.6	Vergleich der Implementierungsansätze und Gesamtvergleich aller Kombinationen bei Twelve Card Poker	65
6.7	Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Small Island Hold'em	66

6.8	Trainingszeit der verschiedenen CFR-Algorithmen bei Small Island Hold'em	66
A.1	Übersicht der Pokerhände und deren Rangordnung, https://en.wikipedia.org/wiki/List_of_poker_hands	73

Kapitel 1

Einleitung

1.1 Motivation und Relevanz

Im Alltag werden ständig Entscheidungen getroffen, ohne dass Entscheidungsträger stets vollständig über alle relevanten Umstände informiert sind. Um dennoch handlungsfähig zu bleiben, treffen Menschen bewusst oder unterbewusst Annahmen über unbekannte Aspekte einer Situation. Das Verhalten von Akteuren in solchen strategischen Entscheidungssituationen lässt sich durch spieltheoretische Modelle beschreiben. Die Spieltheorie umfasst sowohl Modelle für Situationen bzw. Spiele mit vollständiger als auch mit unvollständiger Information. Ein prominentes Beispiel für ein Spiel mit unvollständiger Information ist Poker, insbesondere Hold'em-Varianten wie Texas Hold'em oder Omaha Hold'em.

Ich habe im letzten Jahr ein ausgeprägtes Interesse an dem Spiel No-Limit Texas Hold'em entwickelt. Wenn man sich mit Strategien dieses Spiels auseinandersetzt, stößt man schnell auf ein Konzept, das „Game Theory Optimized“ (GTO) genannt wird. Dieses Konzept hat innerhalb der letzten Jahre immer mehr an Aufmerksamkeit und Relevanz gewonnen und ist heutzutage kaum noch aus der Pokerwelt wegzudenken.

Eine GTO-Strategie ist im Grunde eine spieltheoretisch optimale Lösung, die durch den Einsatz von Algorithmen approximiert wird. Diese Strategien sind für Spiele mit einem enorm großen Zustandsraum, wie No-Limit Texas Hold'em, so komplex, dass sie von Menschen nicht vollständig erlernt und angewendet werden können. Pokerprofis sind allerdings in der Lage, diese Strategien in abstrakter Form zu adaptieren, indem sie grundlegende strategische Prinzipien und Verhaltensmuster erkennen und anschließend auf ihre Spielsituationen anwenden. Die Auseinandersetzung mit GTO-Strategien in meiner Freizeit weckte mein Interesse an den algorithmischen Grundlagen, die zur Berechnung solcher Strategien verwendet werden.

Ein zentraler Algorithmus zum Finden solcher Strategien ist Counterfactual Regret Minimization (CFR). Seit seiner Einführung in „Regret Minimization in Games with Incomplete Information“ [Zinkevich et al., 2007] hat sich CFR als Standardansatz etabliert und bildet die Grundlage für die erfolgreichsten Poker-KIs.

Zu den größten Erfolgen gehören:

- Cepheus [Bowling et al., 2015] löste erstmals das Spiel Heads-Up Limit Texas Hold'em.
- Libratus [Brown and Sandholm, 2018, Brown and Sandholm, 2017a] erweiterte diesen Erfolg auf die Heads-Up No-Limit-Variante.
- Durch Pluribus [Brown and Sandholm, 2019b] wurde gezeigt, dass diese Erfolge auch auf Mehrspielerzenarien übertragen werden können.

1.2 Zielsetzung und Fragestellungen

Im Rahmen dieser Arbeit wird Counterfactual Regret Minimization in Python implementiert und auf verschiedenen Pokervarianten unterschiedlicher Komplexität angewendet. Das Projekt ist unter der URL

<https://github.com/Miedefran/CounterfactualRegretMinimizationPoker> zu finden. Dabei wird untersucht, wie sich die Leistungsfähigkeit des Algorithmus bei steigender Spielkomplexität verhält. Wichtige Parameter, um die Leistungsfähigkeit einzuschätzen, sind die Laufzeit, der Speicherverbrauch und die Qualität der trainierten Strategie. Durch Betrachtung verschiedener Komplexitätsstufen soll untersucht werden, ab welchem Komplexitätsgrad algorithmische und implementierungstechnische Optimierungen vorgenommen werden müssen.

Im Exposé dieser Arbeit waren drei Spiele geplant, Kuhn Poker als simpelstes existierendes Pokerspiel, Leduc Hold'em als etabliertes Benchmark-Spiel und eine selbst entwickelte Variante mit 12 Karten als nächste Stufe auf der Komplexitätsleiter.

Zusätzlich sollten optimierte CFR-Verfahren wie CFR+ und Monte Carlo CFR implementiert werden und auf ihr Konvergenzverhalten untersucht werden.

Um die Qualität einer trainierten Strategie zu bewerten, wird berechnet, wie ausnutzbar diese ist. Dies geschieht mithilfe eines Best Response Agents, der die optimale Antwort gegen eine Gegnerstrategie spielt. Im Verlauf des Projekts wurden diese Ziele grundsätzlich beibehalten, jedoch wurde Monte Carlo CFR durch Discounted CFR ersetzt.

Im Folgenden werden drei Forschungsfragen formuliert, die im Verlauf des Projekts beantwortet werden sollen.

1. **Algorithmusvergleich:** Wie unterscheiden sich die verschiedenen CFR-Algorithmen in ihrem Konvergenzverhalten bei steigender Spielgröße, und ab welcher Spielgröße ist Vanilla CFR nicht mehr praktikabel?
2. **Implementierungsoptimierungen:** Welche Implementierungsoptimierungen sind erforderlich, damit das Training bei wachsender Spielgröße praktikabel bleibt?
3. **Limitationen:** Durch welche Komponenten wird die geplante Implementierung limitiert?

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in acht Kapitel, die den Weg von den theoretischen Grundlagen über die Architektur und den Entwicklungsverlauf bis hin zur Evaluation der trainierten Strategien beschreiben.

- **Kapitel 2** stellt die theoretischen Grundlagen vor. Zu Beginn werden grundlegende spieltheoretische Begriffe eingeführt und eine formale Definition eines „Extensiven Spiels mit imperfekter Information“ gegeben.
Anschließend wird die Theorie des CFR-Algorithmus detailliert erläutert, gefolgt von den optimierten Varianten CFR+, MCCFR und DCFR.
Darauf folgt die Theorie des Accelerated Best Response Agents, der zur Evaluation der Strategiequalität verwendet wird. Abschließend wird definiert, wodurch sich ein Hold’em Spiel auszeichnet. Die Definition einzelner Pokervarianten erfolgt im Anhang.
- **Kapitel 3** nennt die zu lösenden Pokerspiele, die dazu verwendeten Algorithmen und begründet die Auswahl.
- **Kapitel 4** beschreibt zunächst die generelle Systemarchitektur und geht anschließend auf einzelne Komponenten ein.
- **Kapitel 5** liefert einen Überblick über den Entwicklungsverlauf und hebt dabei Probleme sowie die daraus gewonnenen Erkenntnisse hervor.
- **Kapitel 6** veranschaulicht den Konvergenzverlauf der trainierten Strategien und diskutiert die Ergebnisse.
- **Kapitel 7** fasst das Gelernte hinsichtlich der zu Beginn definierten Fragestellungen zusammen, diskutiert Limitationen und gibt einen Ausblick auf mögliche zukünftige Erweiterungen.
- **Kapitel 8** ist der Anhang und enthält alles, was nicht in den Hauptteil der Arbeit aufgenommen wurde.

Kapitel 2

Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen der zentralen Konzepte dieser Arbeit erklärt. Zunächst werden spieltheoretische Begriffe definiert. Anschließend werden die verschiedenen CFR-Varianten anhand der jeweiligen Konferenzpapiere erläutert. Ausgehend von der Vanilla-Version [Zinkevich et al., 2007], über das samplingbasierte Monte-Carlo-Framework [Lanctot et al., 2009], bis hin zu CFR+ [Bowling et al., 2015] und Discounted CFR [Brown and Sandholm, 2019a]. Danach wird die Theorie hinter der Hauptevaluationsmetrik Exploitability und dem zugehörigen Best-Response-Agent erläutert [Johanson et al., 2011]. Abschließend wird eine grundlegende Definition von Heads-Up-Hold'em-Spielen gegeben, mit welcher jedes beliebige Spiel dieser Kategorie dargestellt werden kann.

2.1 Grundlagen der Spieltheorie

In diesem Abschnitt wird ein kurzer Überblick darüber gegeben, was Spieltheorie ist und wo sie angewendet wird. Dabei wird definiert, was im spieltheoretischen Kontext unter einem *Spiel* und einer *Lösung* verstanden wird. Anschließend werden die wichtigsten Typen von Spielen umrissen und der Typ extensiver Spiele mit imperfekter Information näher betrachtet. Als Grundlage für dieses Kapitel dient das Buch „A Course in Game Theory“ [Osborne and Rubinstein, 1994].

2.1.1 Definition und Konzepte

Die Spieltheorie beschäftigt sich mit der Analyse strategischer Interaktionen zwischen Entscheidungsträgern und bietet hierfür eine Vielzahl analytischer Methoden. Diese basieren auf zwei zentralen Annahmen [Osborne and Rubinstein, 1994, Kap. 1.1].

Die erste Annahme ist, dass Entscheidungsträger rational handeln. Ein rationaler Entscheidungsträger kennt seine Handlungsmöglichkeiten, kann Erwartungen über unbekannte Situationen bilden und hat klare Vorstellungen darüber, welche Ergebnisse er bevorzugt. Er wählt seine Aktion bewusst nach einem Prozess der Optimierung. Liegt keine Unsicherheit vor, wählt er die Aktion, die zu den für ihn besten Konsequenzen führt. Diese Präferenzen lassen sich durch eine Nutzenfunktion beschreiben, die jedem

möglichen Ergebnis einen Wert zuordnet. Die beste Aktion ist die, die den höchsten Nutzen erzielt. Sind die Konsequenzen von Aktionen unsicher, maximiert ein rationaler Entscheidungsträger den erwarteten Nutzen. Hierfür muss er in der Lage sein, Wahrscheinlichkeiten für unsichere Ereignisse abzuschätzen [Osborne and Rubinstein, 1994, Kap. 1.4].

Die zweite Annahme ist, dass Entscheidungsträger ihr Wissen oder ihre Erwartungen über das Verhalten anderer Spieler bei ihrer Entscheidungsfindung berücksichtigen. Dies nennt man auch strategisches Handeln. Spieltheoretisches Denken berücksichtigt, dass jeder Entscheidungsträger vor seiner Entscheidung versucht, Informationen über das Verhalten der anderen Spieler zu erhalten. Im Gegensatz dazu nimmt die Theorie des Wettbewerbsgleichgewichts an, dass jeder Akteur sich nur für Umgebungsparameter wie Preise interessiert, auch wenn diese durch die Aktionen aller Akteure bestimmt werden [Osborne and Rubinstein, 1994, Kap. 1.3].

Ein anschauliches Beispiel hierfür ist das Gefangenendilemma. Zwei Verdächtige werden getrennt verhört und müssen jeweils entscheiden, ob sie schweigen oder den anderen belasten. Schweigen beide, erhalten beide zwei Jahre Haft. Wenn beide gestehen, erhalten sie fünf Jahre Haft. Gesteht jedoch nur einer, wird dieser freigelassen, während der andere zu zehn Jahren Haft verurteilt wird. Die optimale Entscheidung ist abhängig von den Tendenzen anderer Entscheidungsträger. Strategisches Handeln bedeutet, solche Erwartungen über das Verhalten anderer Spieler in die Entscheidungsfindung mit einzubeziehen [Osborne and Rubinstein, 1994, Kap. 1.1].

Spieltheoretische Modelle befinden sich auf einer hohen Abstraktionsebene und können daher verwendet werden, um ein breites Spektrum an Phänomenen zu betrachten. Zur formalen Darstellung dieser Modelle werden mathematische Definitionen genutzt. Die zugrunde liegenden Konzepte sind jedoch nicht primär mathematischer Natur und lassen sich auch ohne formale Notation erläutern.

Die Spieltheorie findet Anwendung in unzähligen Bereichen. Zu den klassischen Anwendungsgebieten zählen die Analyse wirtschaftlicher und politischer Wettbewerbssituationen. Darüber hinaus wird sie auch in der Evolutionsbiologie, der Soziologie und anderen Disziplinen genutzt, um strategische Interaktionen zu modellieren [Osborne and Rubinstein, 1994, Kap. 1.1].

Ein Spiel beschreibt eine strategische Interaktion zwischen Spielern mit festgelegten Rahmenbedingungen. Es definiert, welche Aktionen den Spielern zur Verfügung stehen und welche Ergebnisse in ihrem Interesse sind. Das Spiel legt damit die Struktur und die Regeln der Interaktion fest, bestimmt aber nicht, wie sich die Spieler tatsächlich verhalten.

Die Frage, welche Aktionen die Spieler tatsächlich ausführen, wird nicht durch das Spiel selbst beantwortet, sondern durch die Lösung des Spiels. Eine Lösung ist ein Konzept, das systematisch beschreibt, welche Ergebnisse in einer Klasse von Spielen unter bestimmten Verhaltensannahmen zu erwarten sind. Sie liefert damit eine Vorhersage über das tatsächliche Verhalten der Spieler unter den getroffenen Annahmen über Rationalität und strategisches Denken [Osborne and Rubinstein, 1994, Kap. 1.2].

Spiele können anhand verschiedener Kriterien systematisch kategorisiert werden. Grundlegend für alle spieltheoretischen Modelle ist das Konzept des Spielers. Ein Spieler kann dabei sowohl eine einzelne Person als auch eine Gruppe von Personen darstellen, die gemeinsam eine Entscheidung treffen.

Die erste grundlegende Unterscheidung betrifft kooperative und nicht-kooperative Spiele. Nicht-kooperative Spiele modellieren die Aktionen einzelner Spieler. Kooperative Spiele hingegen konzentrieren sich auf Gruppen von Spielern. Dabei wird nicht betrachtet, wie diese Gruppen intern funktionieren [Osborne and Rubinstein, 1994, Kap. 1.2].

Eine weitere wichtige Kategorisierung betrifft die Darstellungsform des Spiels. Hierbei werden strategische Spiele, auch als Spiele in Normalform bezeichnet, von extensiven Spielen unterschieden. Bei strategischen Spielen wählen die Spieler ihre Strategie zu Beginn des Spiels und passen diese während des Spielverlaufs nicht mehr an. Die Entscheidungen werden zudem simultan getroffen, sodass keine Informationen über die Aktionen der Gegenspieler verfügbar sind. Extensive Spiele hingegen spezifizieren die möglichen Reihenfolgen von Ereignissen und erlauben es jedem Spieler, seinen Aktionsplan nicht nur zu Beginn, sondern auch bei jeder weiteren Entscheidung zu überdenken [Osborne and Rubinstein, 1994, Kap. 1.2].

Die letzte hier zu nennende Unterscheidung betrifft den Informationsstand der Spieler. Bei Spielen mit perfekter Information sind alle Teilnehmer vollständig über die Züge der anderen Spieler informiert. Bei Spielen mit imperfekter Information hingegen können die Spieler unvollständig über die Aktionen der anderen Spieler oder über Ausgänge von Zufallsereignissen informiert sein [Osborne and Rubinstein, 1994, Kap. 1.2].

Ein *strikt kompetitives Spiel* (auch *Nullsummenspiel*) liegt vor, wenn der Gewinn des einen Spielers genau dem Verlust des anderen entspricht. Das bedeutet, dass für alle möglichen Spielausgänge die Summe der Nutzenwerte beider Spieler null ergibt. Formal: Ist die Nutzenfunktion von Spieler 1 durch u_1 gegeben, so gilt für die Nutzenfunktion u_2 von Spieler 2 stets $u_1 + u_2 = 0$ [Osborne and Rubinstein, 1994, Def. 21.1].

2.1.2 Extensive Games with Imperfect Information

Nachdem die grundlegenden Konzepte der Spieltheorie eingeführt wurden, widmet sich dieser Abschnitt den extensiven Spielen mit imperfekter Information. Diese Spielklasse ist für die vorliegende Arbeit von zentraler Bedeutung, da die meisten Pokerspiele, insbesondere Hold'em-Varianten, zu dieser Kategorie gehören.

Definition

Ein extensives Spiel mit imperfekter Information modelliert eine sequenzielle Interaktion zwischen Spielern, bei der nicht alle Spieler stets vollständig über alle vergangenen Aktionen informiert sind. Dabei werden die möglichen Spielverläufe als Historien erfasst, wobei eine Historie eine Abfolge von Aktionen darstellt. Zu jedem Zeitpunkt ist entweder ein Spieler oder der Zufall am Zug.

Um die imperfekte Information darzustellen, werden Informationsets verwendet. Diese fassen Zustände zusammen, die ein Spieler nicht voneinander unterscheiden kann. Hierbei können sowohl vergangene Aktionen anderer Spieler als auch Ausgänge von Zufallsereignissen betroffen sein.

Ein extensives Spiel mit imperfekter Information lässt sich formal durch folgende Komponenten definieren:

- Eine Menge von Historien H . Eine Historie ist eine endliche oder unendliche Sequenz von Aktionen, getätigt von Spielern oder dem Zufall.

Die Menge H erfüllt folgende Eigenschaften:

- Die leere Sequenz \emptyset ist ein Element von H .
 - Ist $(a^k)_{k=1,\dots,K} \in H$ (wobei K unendlich sein kann) und $L < K$, dann gilt $(a^k)_{k=1,\dots,L} \in H$. Das bedeutet, wenn eine Sequenz ein Teil von H ist, dann ist auch jeder Präfix dieser Sequenz ein Teil von H .
 - Wenn für eine unendliche Folge $(a^k)_{k=1}^\infty$ die Bedingung $(a^k)_{k=1,\dots,L} \in H$ für jede positive ganze Zahl L erfüllt ist, dann gilt $(a^k)_{k=1}^\infty \in H$.
 - Eine Historie $(a^k)_{k=1,\dots,K} \in H$ ist terminal, wenn sie unendlich ist oder wenn kein a^{K+1} existiert, so dass $(a^k)_{k=1,\dots,K+1} \in H$ gilt. Man bezeichnet die Menge aller terminalen Historien mit Z .
- Die Menge der Aktionen, die nach einer nicht terminalen Historie h verfügbar sind, wird mit $A(h) = \{a : (h, a) \in H\}$ bezeichnet.
- Die Funktion P ordnet jeder nicht terminalen Historie $h \in H \setminus Z$ einen Spieler aus N oder c zu. $P(h)$ bezeichnet den Spieler, der nach h am Zug ist. Wenn $P(h) = c$ ist, bestimmt der Zufall die nächste Aktion.
 - Die Funktion f_c ordnet jeder Historie h mit $P(h) = c$ eine Zufallsverteilung $f_c(\cdot | h)$ auf $A(h)$ zu. $f_c(a | h)$ ist die Wahrscheinlichkeit, dass nach h die Aktion a folgt. Diese Verteilungen sind unabhängig voneinander.
 - Für jeden Spieler $i \in N$ wird die Menge der Historien h mit $P(h) = i$ in Informati- onsets unterteilt. Diese Informationspartition wird bezeichnet mit \mathcal{I}_i . Ein Element $I_i \in \mathcal{I}_i$ heißt Informationset von Spieler i . Historien h und h' , die zum selben Infor- mationset I_i gehören, sind für Spieler i nicht unterscheidbar. Wesentlich ist dabei, dass alle Historien in einem Informationset I_i die gleichen verfügbaren Aktionen haben, also $A(h) = A(h')$ für alle $h, h' \in I_i$. Für ein Informationset I_i bezeichnet $A(I_i)$ die verfügbaren Aktionen und $P(I_i)$ den Spieler, der am Zug ist.
 - Für jeden Spieler $i \in N$ ist eine Präferenzrelation \succeq_i über Lotterien auf Z gegeben. Eine Lotterie ist hierbei eine Wahrscheinlichkeitsverteilung über terminale Historien. Diese Präferenzen lassen sich durch eine Nutzenfunktion $u_i : Z \rightarrow \mathbb{R}$ repräsentieren, sodass für zwei Lotterien L_1, L_2 gilt: $L_1 \succeq_i L_2 \iff \mathbb{E}_{z \sim L_1}[u_i(z)] \geq \mathbb{E}_{z \sim L_2}[u_i(z)]$.

Formale Definition entspricht [Osborne and Rubinstein, 1994, Ch. 11, Sec. 11.1, Def. 200.1].

Strategien

In extensiven Spielen mit imperfekter Information gibt es verschiedene Möglichkeiten, wie die Strategien von Spielern definiert werden können.

Eine reine Strategie von Spieler $i \in N$ ordnet jedem Informationset $I \in \mathcal{I}_i$ genau eine zulässige Aktion $s_i(I) \in A(I)$ zu. Sie legt somit einen vollständigen Handlungsplan für alle Informationslagen von i fest.

Eine gemischte Strategie α_i ist eine Wahrscheinlichkeitsverteilung über der Menge der reinen Strategien von i .

Eine behavioral Strategie β_i hingegen ist eine Sammlung unabhängiger Wahrscheinlichkeitsverteilungen $(\beta_i(I))_{I \in \mathcal{I}_i}$, wobei $\beta_i(I)$ eine Wahrscheinlichkeitsverteilung über die Aktionen $A(I)$ am Informationset I ist [Osborne and Rubinstein, 1994, Ch. 11, Sec. 11.4, Def. 212.1].

Der Unterschied liegt darin, dass bei einer gemischten Strategie vor Spielbeginn zufällig eine komplette reine Strategie ausgewählt wird, die dann während des gesamten Spiels deterministisch befolgt wird.

Im Gegensatz dazu wird bei einer behavioral Strategie in jedem Informationset unabhängig eine Wahrscheinlichkeitsverteilung über die verfügbaren Aktionen verwendet, sodass die zufällige Auswahl der Aktionen während des Spiels erfolgt [Osborne and Rubinstein, 1994, Ch. 11, Sec. 11.4].

Ein extensives Spiel hat „Perfect Recall“, wenn sich jeder Spieler zu jedem Zeitpunkt an alles erinnert, was er in der Vergangenheit wusste und welche Aktionen er selbst getroffen hat [Osborne and Rubinstein, 1994, Ch. 11, Sec. 11.1]. Unter dieser Annahme erzeugen beide Repräsentationen outcome-äquivalente Wahrscheinlichkeitsverteilungen über die terminalen Historien [Osborne and Rubinstein, 1994, Ch. 11, Sec. 11.4, Prop. 214.1].

Ein Profil von behavioral Strategien $\beta = (\beta_i)_{i \in N}$ zusammen mit den Zufallszügen f_c bestimmt eine Wahrscheinlichkeitsverteilung über die terminalen Historien Z . Daraus lässt sich für jede Auszahlungsfunktion u_i der Erwartungswert $\mathbb{E}_{z \sim O(\beta, f_c)}[u_i(z)]$ berechnen.

Diese spieltheoretischen Grundlagen bilden die Basis für das Verständnis der Algorithmen, die in den kommenden Abschnitten vorgestellt werden.

2.2 Counterfactual Regret Minimization

Dieser Abschnitt stellt die theoretischen Grundlagen von Counterfactual Regret Minimization (CFR), dem Kernstück dieser Arbeit, vor. Als fachliche Grundlage dient das Paper „Regret Minimization in Games with Incomplete Information“ [Zinkevich et al., 2007], in welchem CFR erstmals vorgestellt wurde.

Zuerst werden einige wichtige Werte und Zusammenhänge definiert, die essenziell für das Verständnis der darauffolgenden Konzepte sind. Anschließend wird erklärt was Regret-Minimierung ist. Darauf aufbauend wird das von Zinkevich et al. eingeführte Konzept des Counterfactual Regret sowie Regret Matching erläutert. Abschließend fügen sich die einzelnen Komponenten zu einem Algorithmus zusammen.

2.2.1 Definitionen

Wahrscheinlichkeiten und Werte

Für ein Strategieprofil σ werden folgende Wahrscheinlichkeiten und Werte definiert:

Die Wahrscheinlichkeit, dass eine Historie h auftritt, wenn alle Spieler gemäß σ handeln, wird mit $\pi^\sigma(h)$ bezeichnet. Diese Wahrscheinlichkeit lässt sich in ein Produkt zerlegen:

$$\pi^\sigma(h) = \prod_{i \in N \cup \{c\}} \pi_i^\sigma(h), \quad (2.1)$$

wobei $\pi_i^\sigma(h)$ die Wahrscheinlichkeit bezeichnet, dass Spieler i in all seinen Entscheidungsknoten auf dem Pfad zu h gemäß σ die Aktionen gewählt hat, die zu h führen. Das Produkt der Wahrscheinlichkeiten aller anderen Spieler (inklusive Zufallsereignisse) wird mit $\pi_{-i}^\sigma(h)$ bezeichnet.

Für ein Informationset I wird die Erreichbarkeitswahrscheinlichkeit definiert als:

$$\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h). \quad (2.2)$$

Analog sind $\pi_i^\sigma(I)$ und $\pi_{-i}^\sigma(I)$ als die Wahrscheinlichkeiten definiert, dass Spieler i beziehungsweise die anderen Spieler das Informationset I erreichen.

Der erwartete Nutzen für Spieler i unter Strategieprofil σ , auch *Overall Value* genannt, wird wie folgt berechnet:

$$u_i(\sigma) = \sum_{h \in Z} u_i(h) \pi^\sigma(h), \quad (2.3)$$

Nash-Gleichgewicht

Ein Nash-Gleichgewicht beschreibt ein Strategienprofil, bei dem kein Spieler durch eine einseitige Änderung seiner Strategie seinen erwarteten Nutzen erhöhen kann [Osborne and Rubinstein, 1994, Ch. 11, Sec. 11.5].

Ein Nash-Gleichgewicht für ein Zweispielernullsummenspiel ist ein Strategieprofil $\sigma = (\sigma_1, \sigma_2)$, das folgende Ungleichungen erfüllt:

$$u_1(\sigma) \geq \max_{\sigma'_1 \in \Sigma_1} u_1(\sigma'_1, \sigma_2), \quad u_2(\sigma) \geq \max_{\sigma'_2 \in \Sigma_2} u_2(\sigma_1, \sigma'_2) \quad (2.4)$$

([Zinkevich et al., 2007, Eq. 1]).

Eine Approximation eines Nash-Gleichgewichts, auch ε -Nash-Gleichgewicht genannt, ist ein Strategieprofil σ , das folgende Ungleichungen erfüllt:

$$u_1(\sigma) + \varepsilon \geq \max_{\sigma'_1 \in \Sigma_1} u_1(\sigma'_1, \sigma_2), \quad u_2(\sigma) + \varepsilon \geq \max_{\sigma'_2 \in \Sigma_2} u_2(\sigma_1, \sigma'_2) \quad (2.5)$$

([Zinkevich et al., 2007, Eq. 2]).

2.2.2 Regret-Minimierung

Regret misst den Unterschied zwischen dem maximal erreichbaren Nutzen unter Verwendung der zu diesem Zeitpunkt bestmöglichen Strategie und dem tatsächlich erzielten Nutzen.

Um Regret-Minimierung zu definieren, wird das wiederholte Spielen eines extensiven

Spiels betrachtet. Sei σ_i^t die von Spieler i in Runde t verwendete Strategie. Der durchschnittliche *Overall Regret* von Spieler i zum Zeitpunkt T ist:

$$R_i^T = \frac{1}{T} \max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^T (u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma^t)). \quad (2.6)$$

([Zinkevich et al., 2007, Eq. 3])

Die Durchschnittsstrategie $\bar{\sigma}_i^T$ für Spieler i von Zeit 1 bis T wird für jedes Informationset $I \in \mathcal{I}_i$ und jede Aktion $a \in A(I)$ definiert als:

$$\bar{\sigma}_i^T(I)(a) = \frac{\sum_{t=1}^T \pi_i^{\sigma^t}(I) \sigma_i^t(I)(a)}{\sum_{t=1}^T \pi_i^{\sigma^t}(I)}. \quad (2.7)$$

([Zinkevich et al., 2007, Eq. 4])

Die Durchschnittsstrategie gewichtet Aktionen proportional zur Erreichbarkeitswahrscheinlichkeit $\pi_i^{\sigma^t}(I)$ des Informationsets I in Runde t .

Es besteht ein wichtiger Zusammenhang zwischen Regret und Nash-Gleichgewichten. In einem Nullsummenspiel gilt: Ist der durchschnittliche Overall Regret beider Spieler zum Zeitpunkt T kleiner als ε , dann ist $\bar{\sigma}^T$ ein 2ε -Nash-Gleichgewicht ([Zinkevich et al., 2007, Thm. 2]).

Ein Algorithmus zur Bestimmung von σ_i^t ist Regret-minimierend, wenn der durchschnittliche Overall Regret von Spieler i unabhängig von der Strategiefolge σ_{-i}^t des Gegners gegen Null konvergiert, während T gegen Unendlich läuft. Daher können Regret-minimierende Algorithmen im Self-Play verwendet werden, um ein approximatives Nash-Gleichgewicht zu berechnen.

2.2.3 Counterfactual Regret

Die zentrale Idee, die Zinkevich et al. in ihrem Paper "Regret Minimization in Games with Incomplete Information" präsentieren, ist, den gesamten Regret-Wert in eine Menge aus additiven Termen zu zerlegen. Diese Terme können dann unabhängig voneinander minimiert werden. Insbesondere wird ein völlig neues Regret-Konzept mit dem Namen *Counterfactual Regret* eingeführt, das für jedes Informationset definiert ist.

Um das Konzept zu erläutern, wird ein Informationset $I \in \mathcal{I}_i$ betrachtet.

Die *Counterfactual Utility* oder auch *Counterfactual Value* $u_i(\sigma, I)$ ist der erwartete Nutzen für Spieler i unter der Annahme, dass das Informationset I erreicht wird. Dabei spielen alle Spieler gemäß dem Strategieprofil σ , wobei Spieler i so handelt, dass I tatsächlich erreicht wird. Mit $\pi^\sigma(h, h')$ als Wahrscheinlichkeit, von h nach h' zu gelangen, gilt:

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi_{-i}^\sigma(h) \pi^\sigma(h, h') u_i(h')}{\pi_{-i}^\sigma(I)}. \quad (2.8)$$

([Zinkevich et al., 2007, Eq. 5])

Für alle $a \in A(I)$ ist $\sigma|_{I \rightarrow a}$ ein Strategieprofil, das identisch zu σ ist, außer dass Spieler i immer die Aktion a ausführt, wenn er das Informationset I erreicht. Der *Immediate Counterfactual Regret* ist

$$R_{i,\text{imm}}^T(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^T \pi_{-i}^{\sigma^t}(I) (u_i(\sigma^t|_{I \rightarrow a}, I) - u_i(\sigma^t, I)). \quad (2.9)$$

([Zinkevich et al., 2007, Eq. 6])

Dieser Term misst den Regret im Informationset I , gewichtet mit der *Counterfactual Probability*, dass I erreicht werden würde, wenn Spieler i es erzwingen wollte. Die positiven Immediate Counterfactual Regret-Werte werden definiert als:

$$R_{i,\text{imm}}^{T,+}(I) = \max(R_{i,\text{imm}}^T(I), 0). \quad (2.10)$$

Theorem 3 beschreibt die erste große Erkenntnis des Papers.

$$R_i^T \leq \sum_{I \in \mathcal{I}_i} R_{i,\text{imm}}^{T,+}(I) \quad (2.11)$$

([Zinkevich et al., 2007, Thm. 3])

Das heißt, wenn der Immediate Counterfactual Regret minimiert wird, dann wird auch der Average Overall Regret minimiert.

2.2.4 Regret Matching

Mit folgender Formel wird der Counterfactual Regret für eine konkrete Aktion a in einem Informationset I berechnet.

$$R_i^T(I, a) = \frac{1}{T} \sum_{t=1}^T \pi_{-i}^{\sigma^t}(I) (u_i(\sigma^t|_{I \rightarrow a}, I) - u_i(\sigma^t, I)) \quad (2.12)$$

([Zinkevich et al., 2007, Eq. 7])

Dabei sei $R_i^{T,+}(I, a) = \max(R_i^T(I, a), 0)$ als positiver Anteil des Immediate Counterfactual Regret für Aktion a definiert.

Die Strategie für Iteration $T + 1$ wird dann durch Regret Matching bestimmt:

$$\sigma_i^{T+1}(I)(a) = \begin{cases} \frac{R_i^{T,+}(I, a)}{\sum_{a' \in A(I)} R_i^{T,+}(I, a')} & \text{falls } \sum_{a' \in A(I)} R_i^{T,+}(I, a') > 0 \\ \frac{1}{|A(I)|} & \text{in jedem anderen Fall.} \end{cases} \quad (2.13)$$

([Zinkevich et al., 2007, Eq. 8])

Diese Formel weist Aktionen Wahrscheinlichkeiten proportional zu ihrem positiven Regret aus dem Nichtspielen dieser Aktion zu. Gibt es keine positiven Regretwerte, werden alle Aktionen gleichverteilt gewählt.

Theorem 4 zeigt, dass bei Verwendung von Formel 2.13 der Immediate Counterfactual Regret und der Overall Regret folgendermaßen beschränkt sind:

$$R_{i,\text{imm}}^T(I) \leq \frac{\Delta_{u,i} \sqrt{|A_i|}}{\sqrt{T}}, \quad (2.14)$$

$$R_i^T \leq \frac{\Delta_{u,i} |\mathcal{I}_i| \sqrt{|A_i|}}{\sqrt{T}}, \quad (2.15)$$

wobei $\Delta_{u,i} = \max_z u_i(z) - \min_z u_i(z)$ der Nutzenbereich und $|A_i| = \max_{h:P(h)=i} |A(h)|$ die maximale Anzahl verfügbarer Aktionen für Spieler i ist ([Zinkevich et al., 2007, Thm. 4]). Die Schranke für den Average Overall Regret ist linear in der Anzahl der Informati-onsets $|\mathcal{I}_i|$ und fällt asymptotisch mit $O(1/\sqrt{T})$. Zusammen mit Theorem 2 folgt, dass die Durchschnittsstrategie $\bar{\sigma}^T$ im Selbstspiel gegen ein Nash-Gleichgewicht konvergiert ([Zinkevich et al., 2007, Thm. 2]).

2.2.5 CFR-Algorithmus

Das CFR-Verfahren verbindet die Komponenten aus den vorherigen Abschnitten zu einer wiederholten Selbstspielprozedur. In jeder Iteration wird die Strategie gemäß Formel 2.13 aktualisiert. Für jedes Informationset I und jede Aktion a werden die kumulierten Regretwerte $R_i^t(I, a)$ gespeichert und nach jeder Iteration aktualisiert. Parallel dazu wird die Durchschnittsstrategie gemäß Formel 2.7 aktualisiert. Nach T Iterationen gibt der Algorithmus $(\bar{\sigma}_1^T, \bar{\sigma}_2^T)$ als approximatives Nash-Gleichgewicht zurück.

In diesem Abschnitt wurde die Basisversion des Counterfactual Regret Minimization Algorithmus vorgestellt. In den kommenden Abschnitten folgen die optimierten Varianten, beginnend mit MCCFR.

2.3 Monte Carlo CFR

Der CFR-Algorithmus benötigt in jeder Iteration eine vollständige Traversierung des gesamten Spielbaums. Monte Carlo CFR ist ein im Paper „Monte Carlo Sampling for Regret Minimization in Extensive Games“ [Lanctot et al., 2009] veröffentlichtes Framework von Algorithmen, die diesen Aufwand durch verschiedene Sampling-Varianten reduzieren. In diesem Abschnitt wird zuerst die Grundidee des Frameworks erklärt und anschließend genauer auf die einzelnen Sampling-Verfahren eingegangen. Abschließend werden die theoretischen Eigenschaften besprochen. Hierfür wird ein neues Maß zur Abschätzung der Komplexität eines Spiels eingeführt, das im Anhang C erklärt wird.

2.3.1 Allgemeine Definition

MCCFR fasst die Menge aller terminalen Historien Z in Teilmengen zusammen, die als Blöcke bezeichnet werden. Die Menge dieser Teilmengen ist definiert als $Q = \{Q_1, \dots, Q_r\}$. In jeder Iteration wird zufällig ein Block ausgewählt und nur die terminalen Historien dieses Blocks betrachtet.

Die Definition eines Blocks hängt von der verwendeten Samplingstrategie ab. In dem Fall, dass Chance Sampling verwendet wird, enthält ein Block Q_j alle terminalen Historien mit derselben Sequenz von Zufallsereignissen.

Beim External Sampling hingegen wird ein Block durch eine Kombination aus Zufallsereignissen und Aktionen des Gegners bestimmt.

Die extremste hier genannte Form des Samplings ist das Outcome Sampling, bei dem Zufallsereignisse, Aktionen des Gegners sowie private Aktionen zufällig ausgewählt werden.

Sei $q_j > 0$ die Wahrscheinlichkeit, dass Block Q_j in der aktuellen Iteration ausgewählt wird. Für die Summe aller Blockwahrscheinlichkeiten gilt $\sum_{j=1}^r q_j = 1$.

Die Wahrscheinlichkeit, dass eine terminale Historie z in der aktuellen Iteration betrachtet wird, beträgt $q(z) = \sum_{j:z \in Q_j} q_j$.

Als nächstes wird im Paper eine gewichtete Form des Counterfactual Value eingeführt, um zu berücksichtigen, dass eine terminale Historie nur mit Wahrscheinlichkeit $q(z)$ betrachtet wird. Folgende Formel beschreibt den Sampled Counterfactual Value, wenn Block Q_j aktualisiert wird:

$$\tilde{v}_i(\sigma, I|j) = \sum_{z \in Q_j \cap Z_I} \frac{1}{q(z)} u_i(z) \pi_{-i}^\sigma(z[I]) \pi^\sigma(z[I], z). \quad (2.16)$$

([Lanctot et al., 2009, Eq. 6])

Die Gewichtung mit $\frac{1}{q(z)}$ sorgt dafür, dass der Erwartungswert des Sampled Counterfactual Value dem tatsächlichen Counterfactual Value entspricht. Im Gegensatz zu [Zinkevich et al., 2007] wird der Counterfactual Value hier mit $v_i(\sigma, I)$ anstelle von $u_i(\sigma, I)$ beschrieben.

$$\mathbb{E}_{j \sim q_j} [\tilde{v}_i(\sigma, I|j)] = v_i(\sigma, I). \quad (2.17)$$

([Lanctot et al., 2009, Lemma 1])

Durch die Gewichtung wird garantiert, dass MCCFR im Erwartungswert die gleichen Regret-Updates wie CFR durchführt, obwohl nur ein Teil des Spielbaums in jeder Iteration betrachtet wird. Der MCCFR-Algorithmus sampelt in jeder Iteration einen Block und berechnet für jedes von diesem Block betroffene Informationset die Sampled Immediate Counterfactual Regrets. Diese Regretwerte werden akkumuliert und die Strategie wird in der nächsten Iteration mittels Regret Matching aktualisiert.

2.3.2 Chance Sampling CFR

Chance Sampling CFR ist ein Spezialfall des MCCFR-Frameworks, bei dem nur die Ausgänge von Zufallsereignissen gesampelt werden. Dies betrifft sowohl die Ausgänge von privaten als auch öffentlichen Zufallsereignissen. Nach dem Sampling wird der gesamte Subtree für diese Kombination traversiert, das heißt, alle möglichen Aktionen beider Spieler werden betrachtet.

2.3.3 External Sampling MCCFR

External Sampling MCCFR sampelt nur die Aktionen des Gegners sowie die Ausgänge von Zufallsereignissen. Für jede deterministische Strategie τ von Gegner und Zufall wird

ein Block $Q_\tau \in Q$ definiert. Dabei ist τ eine deterministische Abbildung von Informati-onsets $I \in \mathcal{I}_c \cup \mathcal{I}_{N \setminus \{i\}}$ auf Aktionen $A(I)$.

Die Blockwahrscheinlichkeiten sind:

$$q_\tau = \prod_{I \in \mathcal{I}_c} f_c(\tau(I)|I) \prod_{I \in \mathcal{I}_{N \setminus \{i\}}} \sigma_{-i}(\tau(I)|I). \quad (2.18)$$

Der Block Q_τ enthält alle terminalen Historien z , die mit τ konsistent sind (d. h. entlang von z wählt τ an allen $I \in \mathcal{I}_c \cup \mathcal{I}_{N \setminus \{i\}}$ jeweils die in z auftretende Aktion). Es gilt $q(z) = \pi_{-i}^\sigma(z)$.

Für jeden Spieler $i \in N$ werden an jeder History h mit $P(h) \neq i$ Aktionen gesampelt. Die gesampelten Aktionen werden pro Informationset festgelegt, sodass innerhalb einer Iteration an allen Knoten desselben Informationsets dieselbe Aktion verwendet wird. In Spielen mit „Perfect Recall“ wird ein Informationset auf einem einzelnen Pfad höchstens einmal erreicht.

Für jedes besuchte Informationset werden die Sampled Immediate Counterfactual Re-grets $\tilde{r}(I, a)$ berechnet:

$$\tilde{r}(I, a) = (1 - \sigma(a|I)) \sum_{z \in Q_\tau \cap Z_I} u_i(z) \pi_i^\sigma(z[I]a, z). \quad (2.19)$$

([Lanctot et al., 2009, Eq. 11])

2.3.4 Outcome Sampling MCCFR

Outcome Sampling MCCFR sampelt pro Iteration genau eine terminale Historie z und aktualisiert nur die Informationsets entlang dieses Pfads. Damit ist der Aufwand pro Iteration minimal, da ein Block die Größe $|Q_j| = 1$ besitzt.

Welche Historie z gesampelt wird, legt eine *Sampling Policy* σ' fest: $q(z) = \pi^{\sigma'}(z)$. Die Sampling Policy σ' kann von der aktuellen Strategie σ abweichen. Um diese Abweichung zu korrigieren, wird der Beitrag der Stichprobe mit einem Gewicht w_I versehen (Importance Sampling). Damit w_I wohldefiniert bleibt, muss $q(z) \geq \delta > 0$ gelten, beispielsweise durch $\sigma'_i(a|I) \geq \epsilon > 0$ für alle Aktionen.

In jeder Iteration wird eine terminale Historie z gemäß σ' gesampelt. Anschließend erfolgt eine Rückwärtstraversierung entlang z . An jedem Informationset I auf diesem Pfad werden die Sampled Immediate Counterfactual Regrets $\tilde{r}(I, a)$ berechnet und akkumuliert:

$$\tilde{r}(I, a) = \begin{cases} w_I \cdot (1 - \sigma(a|z[I])) & \text{falls } (z[I]a) \sqsubset z \\ -w_I \cdot \sigma(a|z[I]) & \text{sonst} \end{cases}, \quad (2.20)$$

mit

$$w_I = \frac{u_i(z) \pi_{-i}^\sigma(z) \pi_i^\sigma(z[I]a, z)}{\pi^{\sigma'}(z)}. \quad (2.21)$$

([Lanctot et al., 2009, Eq. 10])

Die Notation $(z[I]a) \sqsubset z$ bedeutet, dass auf dem gesampelten Pfad z am Informationset I die Aktion a gewählt wurde (Fall 1); andernfalls wird Fall 2 verwendet.

2.3.5 Theoretische Eigenschaften

Lanctot et al. [Lanctot et al., 2009] verwenden zur genaueren Angabe von Regretschranken die Größe M_i . Der M-Wert M_i beschreibt, wie sich die Informationsets von Spieler i über seine Aktionssequenzen verteilen, und liegt zwischen $\sqrt{|\mathcal{I}_i|}$ und $|\mathcal{I}_i|$. Dieses Maß ist abhängig von der Struktur des spezifischen Spiels und schätzt die Komplexität genauer ab, als die reine Anzahl der Informationsets. Die genaue Definition sowie die Erklärung anhand eines Beispiels ist im Anhang C zu finden.

Für Vanilla CFR gilt:

$$R_i^T \leq \Delta_{u,i} M_i \sqrt{|A_i|} / \sqrt{T}, \quad (2.22)$$

([Lanctot et al., 2009, Thm. 3]), wobei $\Delta_{u,i}$ und $|A_i|$ wie in Abschnitt 2.2.4 definiert sind.

Für External Sampling MCCFR gilt für jedes p mit $0 < p \leq 1$ mit Wahrscheinlichkeit mindestens $1 - p$:

$$R_i^T \leq \left(1 + \sqrt{\frac{2}{p}}\right) \Delta_{u,i} M_i \sqrt{|A_i|} / \sqrt{T}. \quad (2.23)$$

([Lanctot et al., 2009, Thm. 4]).

Obwohl External Sampling nur einen konstanten Faktor mehr Iterationen benötigt als Vanilla CFR, wird pro Iteration nur ein Bruchteil des Spielbaums traversiert. Für ausgewogene Spiele, in denen die Spieler ungefähr gleich viele Entscheidungen treffen, liegen die Iterationskosten (Anzahl besuchter Historien) bei External Sampling bei $O(\sqrt{|H|})$, während Vanilla CFR $O(|H|)$ benötigt. Damit ergibt sich asymptotisch eine geringere Gesamtzeit zur Berechnung eines approximativen Gleichgewichts ([Lanctot et al., 2009, Sec. 4]).

Für Outcome Sampling MCCFR gilt für jedes p mit $0 < p \leq 1$ mit Wahrscheinlichkeit mindestens $1 - p$:

$$R_i^T \leq \left(1 + \sqrt{\frac{2}{p}} \cdot \frac{1}{\sqrt{\delta}}\right) \Delta_{u,i} M_i \sqrt{|A_i|} / \sqrt{T}, \quad (2.24)$$

wenn $q(z) \geq \delta > 0$ für alle relevanten terminalen Historien z gilt ([Lanctot et al., 2009, Thm. 5]). Der Faktor $1/\sqrt{\delta}$ hängt von der Sampling Policy ab.

In diesem Abschnitt wurde gezeigt, wie die Iterationszeit des CFR-Algorithmus durch verschiedene Sampling-Techniken erheblich reduziert werden kann, ohne das Konvergenzverhalten dabei zu stark zu beeinträchtigen. Im nächsten Abschnitt wird CFR+ vor gestellt, eine Optimierung, die die Konvergenzgeschwindigkeit verbessert und damit schneller zu einer Strategie hoher Qualität führt.

2.4 CFR+

CFR+ bildet die Grundlage für Cepheus, das erste Programm, das Heads-up Limit Texas Hold'em gelöst hat [Bowling et al., 2015]. Der Algorithmus erreicht durch drei Hauptänderungen gegenüber Vanilla CFR eine deutlich schnellere Konvergenz.

In diesem Abschnitt werden die algorithmischen Unterschiede zu CFR beschrieben: Regret Matching+ als zentrale Änderung, alternierende Updates und eine linear gewichtete Durchschnittsstrategie. Zudem werden die Tracking-Regret-Eigenschaften erläutert, die die praktische Überlegenheit von CFR+ erklären.

2.4.1 Unterschiede zu Vanilla CFR

CFR+ unterscheidet sich von Vanilla CFR durch drei Hauptänderungen.

Erstens verwendet CFR+ eine linear gewichtete Durchschnittsstrategie statt der uniformen Durchschnittsstrategie. Die gewichtete Durchschnittsstrategie ist definiert als $\bar{\sigma}_p^T = 2/(T^2 + T) \sum_{t=1}^T t\sigma_p^t$, wobei spätere Iterationen stärker gewichtet werden.

Zweitens führt CFR+ alternierende Updates durch. Im Gegensatz zu Vanilla CFR, der die Regretwerte beider Spieler simultan aktualisiert, aktualisiert CFR+ die Regretwerte der Spieler abwechselnd.

Drittens verwendet CFR+ Regret Matching+ anstelle von Regret Matching. Regret Matching+ setzt negative Regretwerte auf null zurück, statt sie zu ignorieren, was zu besseren Tracking-Regret-Eigenschaften führt.

2.4.2 Regret Matching+

Regret Matching+ ist ein Regret minimierender Algorithmus, der ähnlich zu Regret Matching operiert. Der entscheidende Unterschied liegt im Umgang mit negativen Regretwerten. Während Regret Matching Aktionen mit akkumuliertem negativen Regret ignoriert, werden diese von Regret Matching+ auf null zurückgesetzt.

Formal speichert Regret Matching+ nicht die Regretwerte $R^t(a)$, sondern verwendet einen Regret-ähnlichen Wert:

$$Q^t(a) = (Q^{t-1}(a) + \Delta R^t(a))^+, \quad (2.25)$$

wobei $x^+ = \max(x, 0)$ den positiven Anteil bezeichnet. Dabei bezeichnet $\Delta R^t(a)$ den in Iteration t hinzukommenden Regretanteil (Zuwachs des kumulierten Regrets) für Aktion a . Die Strategie basiert auf folgenden Werten:

$$\sigma^t(a) = \frac{Q^{t-1}(a)}{\sum_{b \in A} Q^{t-1}(b)}. \quad (2.26)$$

Für Regret Matching+ gilt folgende Regret-Schranke [Bowling et al., 2015, Thm. 1]: Sei A eine Menge von Aktionen und $v^t : A \rightarrow \mathbb{R}$ eine Sequenz von T Wertfunktionen. Falls $|v^t(a) - v^t(b)| \leq L$ für alle t und $a, b \in A$ gilt, dann hat ein Agent, der nach Regret Matching+ handelt, einen Regret von höchstens:

$$R^T \leq L\sqrt{|A|T}. \quad (2.27)$$

([Bowling et al., 2015, Thm. 1])

Diese Schranke ist identisch mit der Schranke von Regret Matching. CFR+ hat daher die gleiche theoretische Regret-Schranke wie CFR, konvergiert in der Praxis jedoch deutlich schneller. Dies lässt sich durch die besseren Tracking-Regret-Eigenschaften von Regret Matching+ erklären.

Die empirische Überlegenheit von Regret Matching+ zeigt sich insbesondere dann, wenn sich die beste Aktion plötzlich ändert. Bei Regret Matching muss der Algorithmus warten, bis eine zuvor schlechte Aktion sich beweist, und muss dabei ihren gesamten akkumulierten negativen Regret überwinden. Regret Matching+ kann dagegen sofort die neue beste Aktion spielen, da negative Regrets auf null zurückgesetzt werden und nicht erst überwunden werden müssen.

2.4.3 Tracking Regret

Tracking Regret ist ein von [Herbster and Warmuth, 1998] eingeführtes Konzept. Externes Regret vergleicht nur gegen statische Strategien, die immer die gleiche Aktion wählen. Tracking Regret erlaubt dagegen den Vergleich gegen Strategien, die ihre Aktion höchstens $(k - 1)$ Mal ändern können.

Regret Matching hat sehr schlechte Tracking-Regret-Eigenschaften. Der Algorithmus kann lineares Regret aufweisen, das proportional zur Anzahl der Iterationen T wächst, selbst wenn die Vergleichsstrategie nur einmal ihre Aktion ändert ($k = 2$). Im Gegensatz dazu ist Regret Matching+ der erste Regret-Matching-basierte Algorithmus mit sublinearem Tracking-Regret.

Für alternative Strategien, die bis zu $(k - 1)$ Mal wechseln können, hat Regret Matching+ eine Regret-Schranke von

$$R^T \leq kL\sqrt{|A|T}. \quad (2.28)$$

([Bowling et al., 2015, Thm. 2])

Diese Schranke zeigt, dass der Regret linear mit der Anzahl der Partitionen k skaliert, aber sublinear mit der Anzahl der Iterationen T bleibt.

2.4.4 Linear Weighted Average

CFR+ verwendet eine linear gewichtete Durchschnittsstrategie anstelle der uniformen Durchschnittsstrategie von CFR. Dabei wird jede Iteration t mit Gewicht t versehen, wodurch spätere Iterationen stärker gewichtet werden. Die linear gewichtete Durchschnittsstrategie ist definiert als:

$$\bar{\sigma}_p^T = \frac{2}{T^2 + T} \sum_{t=1}^T t \sigma_p^t. \quad (2.29)$$

Durch [Bowling et al., 2015, Thm. 3] wird folgendes ausgesagt: Wenn CFR+ für T Iterationen in einem extensiven Spiel läuft, dann ist die linear gewichtete Durchschnittsstrategie ein approximatives Nash-Gleichgewicht. Voraussetzung ist, dass die maximale Differenz zwischen den Auszahlungen für jeden Spieler durch eine Konstante L beschränkt

ist. Die Qualität der Approximation wird durch die Schranke $2(|\mathcal{I}_1| + |\mathcal{I}_2|)L\sqrt{k}/\sqrt{T}$ beschrieben. Dabei bezeichnet $k = \max_{I \in \mathcal{I}} |A(I)|$ die maximale Anzahl von Aktionen pro Informationset. Die Schranke impliziert eine Konvergenzrate von $O(1/\sqrt{T})$, die asymptotisch identisch mit der Rate der uniformen Gewichtung ist.

Theorem 3 gilt nicht für CFR. Im Gegensatz zu CFR+, wo lineare Gewichtung die Performance verbessert, verschlechtert lineare Gewichtung die Performance von Vanilla CFR.

In der Praxis erreicht die aktuelle Strategie in CFR+ oft bereits eine gute Approximation zum Nash Equilibrium, sodass die aggressive Gewichtung die Konvergenz weiter beschleunigen kann.

Empirisch wurde beobachtet, dass eine quadratische Gewichtung mit t^2 statt t die Konvergenz von CFR+ weiter beschleunigen kann [Brown and Sandholm, 2019a].

2.5 Discounted CFR

Discounted CFR (DCFR) wurde im Paper „Solving Imperfect-Information Games via Discounted Regret Minimization“ [Brown and Sandholm, 2019a] als Optimierung von CFR vorgestellt. Die Motivation für DCFR liegt darin, dass CFR+ in Spielen mit stark suboptimalen Aktionen relativ schlecht abschneidet. Ein anschauliches Beispiel hierfür ist der Fall, dass ein Spieler zwischen drei Aktionen mit den Payoffs 0, 1 und -1.000.000 wählen muss. CFR+ benötigt in diesem Fall 471.407 Iterationen, um die beste Aktion mit 100% Wahrscheinlichkeit zu wählen, da der frühe Fehler der ersten Iteration lange im kumulierten Regretwert erhalten bleibt. Durch die Abwertung früherer Iterationen können solche Fehler schneller aus dem Regretwert verschwinden, wodurch die Konvergenz beschleunigt wird.

Der Algorithmus unterscheidet sich in drei Punkten von CFR+:

- Frühere Iterationen werden bei der Regret-Akkumulation abgewertet (Discounting), wobei positive und negative Regretwerte unterschiedlich behandelt werden.
- Iterationen werden bei der Berechnung der Durchschnittsstrategie unterschiedlich gewichtet.
- Optimistische Regret-Matching-Algorithmen können verwendet werden.

2.5.1 Linear CFR

Linear CFR (LCFR) ist ein Vorläufer von DCFR, der das Konzept des Regret Discountings einführt. In LCFR werden die Regretwerte nach jedem Update mit dem Faktor $\frac{t}{t+1}$ multipliziert, wodurch frühere Iterationen weniger stark gewichtet werden. Zusätzlich wird die Durchschnittsstrategie mit Gewicht t in Iteration t gewichtet. Im oben genannten Beispiel benötigt LCFR nur 970 Iterationen im Gegensatz zu den 471.407 Iterationen bei CFR+, um mit einhundert Prozent Wahrscheinlichkeit die richtige Aktion zu wählen. DCFR übernimmt dieses Konzept und erweitert es um differenziertes Discounting für positive und negative Regretwerte.

2.5.2 Discounting

Discounting der Regretwerte bedeutet, dass frühere Iterationen weniger stark gewichtet werden. Dies beschleunigt die Konvergenz insbesondere in Spielen mit stark suboptimalen Aktionen, da frühe Fehler schneller aus dem kumulierten Regretwert verschwinden. Positive und negative Regretwerte erfahren hierbei eine gesonderte Behandlung. Nach jedem Update werden positive Regretwerte mit Faktor $\frac{t^\alpha}{t^\alpha+1}$ multipliziert, negative Regretwerte mit $\frac{t^\beta}{t^\beta+1}$. Formal werden die Regretwerte in Iteration t wie folgt aktualisiert:

$$R^t(I, a) = \begin{cases} (R^{t-1}(I, a) + r^t(I, a)) \cdot \frac{t^\alpha}{t^\alpha+1} & \text{falls } R^{t-1}(I, a) + r^t(I, a) > 0 \\ (R^{t-1}(I, a) + r^t(I, a)) \cdot \frac{t^\beta}{t^\beta+1} & \text{sonst} \end{cases} \quad (2.30)$$

Die Parameter α und β sind frei wählbar, das Paper schlägt allerdings $\alpha = 3/2$ und $\beta = 0$ vor.

DCFR hat eine Konvergenzschranke von $O(1/\sqrt{T})$, die sich von CFR nur durch einen konstanten Faktor unterscheidet.

2.5.3 Gewichtung der Durchschnittsstrategie

DCFR verwendet wie CFR+ eine gewichtete Durchschnittsstrategie, wobei die Gewichtung durch den Parameter γ gesteuert wird. Iteration t trägt mit Gewicht t^γ bei. Dies ist unabhängig vom Regret Discounting, das die Regretwerte während des Trainings beeinflusst. Die Gewichtung der Durchschnittsstrategie bestimmt, wie stark jede Iteration zur finalen Output-Strategie beiträgt.

2.5.4 Standard-Parameter

DCFR wird durch drei Parameter charakterisiert: α für das Discounting positiver Regretwerte, β für das Discounting negativer Regretwerte und γ für die Gewichtung der durchschnittlichen Strategie.

Die empfohlene Standard-Parameterkonfiguration ist DCFR_{3/2,0,2} mit $\alpha = 3/2$, $\beta = 0$ und $\gamma = 2$.

Experimentelle Ergebnisse zeigen, dass diese Konfiguration CFR+ in allen im Paper getesteten Spielen übertrifft.

Bei $\beta = 0$ gehen negative Regretwerte nicht gegen $-\infty$, sondern nähern sich einem konstanten Wert an. Dies macht DCFR_{3/2,0,2} nicht kompatibel mit Pruning-Algorithmen, die auf negativen Regretwerten basieren [Brown and Sandholm, 2015]. Für Anwendungen, die Pruning verwenden, wird DCFR_{3/2,1/2,2} mit $\beta = 0.5$ empfohlen, da diese Konfiguration suboptimale Aktionen erlaubt, deren Regret gegen $-\infty$ geht, und damit Pruning ermöglicht.

2.5.5 Optimistisches Regret Matching

Optimistisches Regret Matching ist eine optionale Erweiterung, bei der die letzte Iteration doppelt gezählt wird, wenn die Strategie für die nächste Iteration berechnet wird.

Formal wird dabei ein modifizierter Regretwert $R_{\text{mod}}^T(I, a) = \sum_{t=1}^{T-1} r^t(I, a) + 2r^T(I, a)$ verwendet, der die letzte Iteration zweifach berücksichtigt. Experimentelle Ergebnisse zeigen, dass Optimistic DCFR_{3/2,0,2} in allen getesteten HUNL-Subgames schlechter abschneidet als normales DCFR_{3/2,0,2}. Optimistic LCFR kann hingegen in Spielen mit besonders großen suboptimalen Aktionen zu schnellerer Konvergenz führen als LCFR.

2.6 Exploitability

Zur Evaluation der Strategiequalität wird die Metrik Exploitability verwendet. Als theoretische Grundlage für dieses Kapitel dient [Johanson et al., 2011]. Zur Berechnung der Exploitability wird die Best-Response-Methode genutzt. Eine Best Response ist die optimale Strategie eines Spielers gegen eine gegebene Gegnerstrategie. Formal ist die Best Response von Spieler i gegen die Gegnerstrategie σ_{-i} definiert als:

$$b_i(\sigma_{-i}) = \arg \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i}), \quad (2.31)$$

wobei Σ_i die Menge aller möglichen Strategien von Spieler i bezeichnet.

Der Wert $u_i(\sigma_i, b_{-i}(\sigma_i))$ ist eine untere Schranke für den erwarteten Nutzen von Spieler i . Die Exploitability einer Strategie misst, wie viel Nutzen gegen einen Gegner verloren geht, der die Best-Response-Strategie spielt, verglichen mit dem Wert, der von einer optimalen Strategie erreicht werden würde.

In einem Zweipersonen-Nullsummenspiel ist die Exploitability einer Strategie σ_i definiert als:

$$\varepsilon_i(\sigma_i) = v_i - u_i(\sigma_i, b_{-i}(\sigma_i)), \quad (2.32)$$

wobei v_i der Spielwert für Spieler i ist (die untere Schranke für den Nutzen eines optimalen Spielers in Position i) und $b_{-i}(\sigma_i)$ die Best Response des Gegners gegen σ_i darstellt. Eine Strategie ist optimal, wenn ihre Exploitability null ist.

Für Strategien, die in beiden Positionen gespielt werden, wird die Exploitability als Durchschnitt der Best-Response-Werte aus beiden Positionen berechnet:

$$\varepsilon(\sigma) = \frac{u_2(\sigma_1, b_2(\sigma_1)) + u_1(b_1(\sigma_2), \sigma_2)}{2}. \quad (2.33)$$

In Nullsummenspielen ist die Exploitability eng mit dem Nash-Gleichgewicht verbunden. Ein Nash-Gleichgewicht ist unexploitable, da es gegen jeden Gegner mindestens den Spielwert erreicht. Daher ist die Exploitability eine zentrale Evaluationsmetrik.

2.6.1 Konventionelle Best Response Berechnung

Die konventionelle Berechnung einer Best Response erfolgt durch eine rekursive Traversierung des Informationset Trees des betrachteten Spielers. Der Algorithmus traversiert den Baum von der Wurzel zu den Terminalknoten und berechnet dabei für jedes Informationset den erwarteten Nutzen.

An Terminalknoten muss der Algorithmus alle möglichen Spielzustände berücksichtigen, die für den Spieler nicht unterscheidbar sind. Da der Spieler nicht weiß, in welchem konkreten Spielzustand er sich befindet, berechnet er die Erreichbarkeitswahrscheinlichkeiten des Gegners für die verschiedenen Informationsets. Der Nutzen wird als gewichtete Summe über alle möglichen Spielzustände berechnet, wobei jeder Nutzen mit der entsprechenden Erreichbarkeitswahrscheinlichkeit gewichtet wird. Dieser Wert wird zurückgegeben.

Während der Rücktraversierung durch den Baum werden an den verschiedenen Knotentypen unterschiedliche Operationen durchgeführt. An Entscheidungsknoten des betrachteten Spielers wählt der Algorithmus die Aktion mit dem höchsten erwarteten Nutzen und speichert diese Wahl als Teil der Best-Response-Strategie. Der Wert dieser gewählten Aktion wird zurückgegeben. An Entscheidungsknoten des Gegners und an Zufallsknoten wird der erwartete Nutzen als Summe der Werte der Kindknoten berechnet und zurückgegeben. Wenn der Algorithmus zur Wurzel zurückkehrt, ist der zurückgegebene Wert der Best-Response-Wert gegen die gegebene Gegnerstrategie.

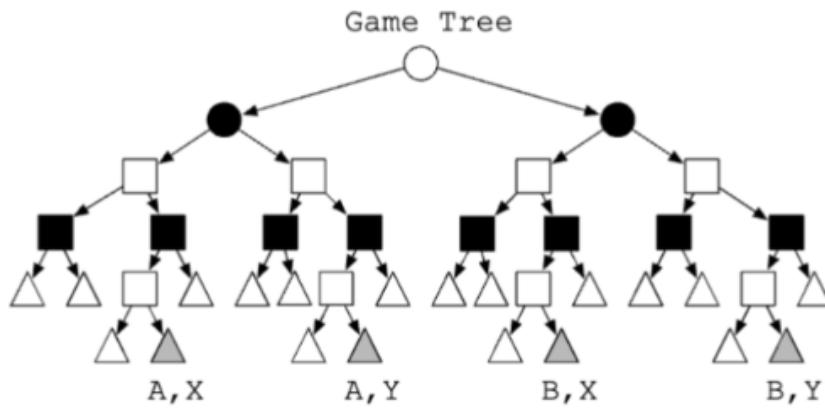


Abbildung 2.1: Game Tree für Kuhn Poker

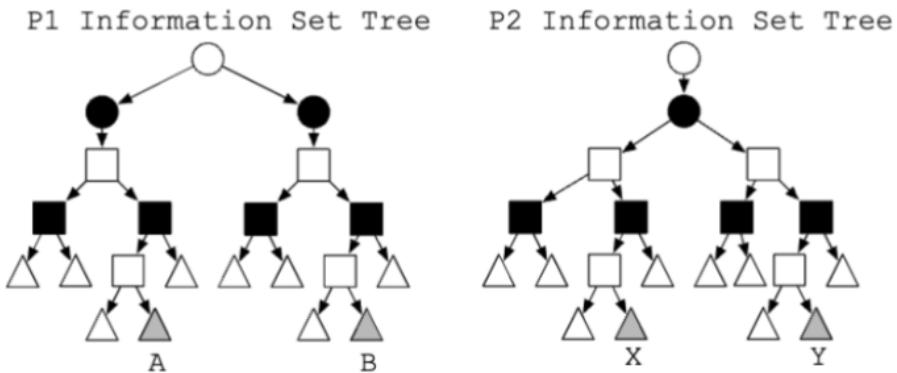


Abbildung 2.2: Information Set Trees für beide Spieler in Kuhn Poker

Die Abbildungen zeigen das Beispiel Kuhn Poker zur Illustration des Algorithmus. Im Game Tree (Abbildung 2.1) repräsentieren die Knoten A,X , A,Y , B,X und B,Y die terminalen Spielzustände, wobei A und B die Aktionen von Spieler 1 und X und Y die Aktionen

von Spieler 2 bezeichnen. Im Information Set Tree von Spieler 2 (Abbildung 2.2) kann dieser am Terminalknoten X nicht unterscheiden, ob er sich im Game Tree Zustand A, X oder B, X befindet, da diese nur durch die private Information von Spieler 1 unterschieden werden. Daher muss der Algorithmus die Erreichbarkeitswahrscheinlichkeiten für beide möglichen Zustände berechnen.

[Johanson et al., 2011] beschreiben eine beschleunigte Version des Algorithmus, die vier Optimierungen umfasst und im Folgenden erläutert wird.

2.6.2 Public State Tree

Der zentrale Ansatz der beschleunigten Best-Response-Berechnung besteht darin, statt des Informationset Trees einen *Public State Tree* zu traversieren. Ein *Public State* ist definiert als eine Partition der Historien, die folgende Eigenschaften erfüllt: Keine zwei Historien aus demselben Informationset befinden sich in unterschiedlichen Public States. Zwei Historien aus unterschiedlichen Public States haben keine Nachfahren im selben Public State, wodurch eine Baumstruktur entsteht. Kein Public State enthält sowohl terminale als auch nicht-terminale Historien.

Informell ist ein Public State durch alle Informationen definiert, die beiden Spielern bekannt sind, also wie das Spiel für einen Beobachter ohne private Informationen aussieht. Der Public State Tree ist deutlich kleiner als der Informationset Tree, da er nur die öffentlich sichtbaren Informationen enthält.

Der entscheidende Vorteil bei der Traversierung des Public State Trees liegt in der Wiederverwendung von Erreichbarkeitswahrscheinlichkeiten. In der konventionellen Methode werden die Erreichbarkeitswahrscheinlichkeiten des Gegners an jedem Terminalknoten neu berechnet. Im Public State Tree werden diese Wahrscheinlichkeiten jedoch einmal berechnet und für alle Informationsets des betrachteten Spielers in diesem Public State wiederverwendet. Dies ist möglich, weil die Erreichbarkeitswahrscheinlichkeiten für alle eigenen Informationsets identisch sind, die sich im selben Public State befinden.

Statt wie in der konventionellen Methode einen einzelnen Wert für ein Informationset zurückzugeben, gibt der Algorithmus beim Public State Tree einen Vektor von Werten zurück. Einen für jedes Informationset des betrachteten Spielers im Public State. An Terminalknoten werden die Werte für alle Informationsets berechnet, wobei die Erreichbarkeitswahrscheinlichkeiten des Gegners einmal berechnet und für alle eigenen Informationsets verwendet werden. An Entscheidungsknoten des betrachteten Spielers wird für jedes Informationset die beste Aktion gewählt, und es wird ein Vektor zurückgegeben, dessen Einträge die maximalen Aktionswerte darstellen. An Entscheidungsknoten des Gegners und an Zufallsknoten werden die Wertevektoren der Kindknoten summiert und zurückgegeben. Diese Umstrukturierung führt zu denselben Berechnungen wie die konventionelle Methode, ändert jedoch die Reihenfolge, sodass Strategieabfragen beim Gegner effizienter wiederverwendet werden können. In Spielen wie Texas Hold'em, in denen jeder Spieler bis zu 1326 Informationsets pro Public State haben kann, ermöglicht dies eine erhebliche Reduzierung der Strategieabfragen.

Abbildung 2.3 zeigt den Public State Tree für das Beispiel Kuhn Poker. Im Vergleich zum Game Tree (Abbildung 2.1) und den Information Set Trees (Abbildung 2.2) ist der Public

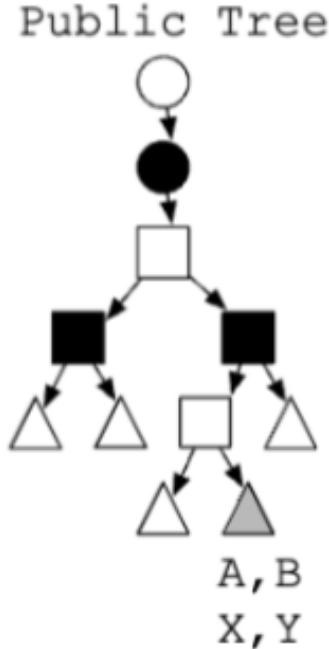


Abbildung 2.3: Public State Tree für Kuhn Poker

State Tree deutlich kompakter, da er nur die öffentlich sichtbaren Informationen enthält. Der Terminalknoten A, B, X, Y repräsentiert alle vier möglichen terminalen Spielzustände A, X , A, Y , B, X und B, Y aus dem Game Tree. An diesem Terminalknoten können die Erreichbarkeitswahrscheinlichkeiten des Gegners einmal berechnet und für alle Informationsets beider Spieler in diesem Public State wiederverwendet werden.

2.6.3 Effiziente Evaluation der Terminalknoten

Ein weiterer Beschleunigungsschritt betrifft die Evaluierung von Terminalknoten. Eine naive Methode hätte $O(n^2)$ Komplexität, wobei n die Anzahl der Informationsets pro Spieler ist. In Pokerspielen kann diese auf $O(n)$ reduziert werden, indem die Informationsets nach Handstärke sortiert werden. Da der Nutzen nur von der relativen Ordnung der Handstärken abhängt, können Indizes in der sortierten Liste des Gegners verwendet werden, die markieren, wo sich die Relation ändert (schwächer, gleich, stärker). Um ein Informationset zu evaluieren, benötigt man nur die Gesamtwahrscheinlichkeit der Gegner-Informationsets in diesen drei Abschnitten. Beim Übergang zum nächsten stärkeren Informationset werden die Indizes nur um einen Schritt verschoben, anstatt alle Gegner-Informationsets erneut zu betrachten.

Da die möglichen Hände der Spieler abhängig voneinander sind (eine Karte kann nicht von beiden Spielern gehalten werden), wird das Inklusions-Exklusions-Prinzip verwendet. Beim Berechnen der Wahrscheinlichkeit von besseren oder schlechteren Gegnerhänden müssen unmögliche Hände ausgeschlossen werden, da der Gegner keine der eigenen Karten halten kann. Hände, die eine der eigenen Karten enthalten, werden subtrahiert. Hände, die beide eigene Karten enthalten, werden wieder addiert, da sie sonst doppelt subtrahiert würden (einmal für jede Karte).

2.6.4 Spielspezifische Isomorphismen

Ein weiterer Beschleunigungsschritt nutzt strategisch äquivalente Aktionen oder Ereignisse. In vielen Kartenspielen sind Karten mit gleichem Rang, aber unterschiedlicher Farbe strategisch äquivalent. In Poker ist das zumindest solange der Fall, bis weitere Karten aufgedeckt werden. Bevor der Flop ausgeteilt wird, ist eine Herz zwei genauso gut wie eine Kreuz zwei. Solche Mengen äquivalenter Zufallsereignisse werden als *isomorph* bezeichnet, und eine wird willkürlich als Repräsentant gewählt.

Wenn die zu evaluierende Strategie für jedes Mitglied jeder Menge isomorpher Historien gleich ist, kann die Größe des Public State Trees erheblich reduziert werden, indem nur die Repräsentanten betrachtet werden. Beim Zurückkehren durch einen Zufallsknoten während der Traversierung muss der Nutzen eines Repräsentanten mit der Anzahl der isomorphen States gewichtet werden, die er repräsentiert. In Texas Hold'em führt diese Reduktion zu einem Public-State-Tree, der 21,5-mal kleiner ist als das vollständige Spiel.

2.6.5 Parallele Berechnung

Ein vierter Beschleunigungsschritt nutzt die Möglichkeit, unabhängige Teilbäume des Public State Trees parallel zu lösen. Public States können parallel gelöst werden, wenn keiner ein Nachfahre des anderen ist, da sie keine gemeinsamen Berechnungen haben. Beispielsweise können beim Erreichen eines öffentlichen Zufallsereignisses alle Kinder parallel gelöst werden.

In diesem Abschnitt wurde die Hauptmetrik beschrieben, die verwendet wird, um die Qualität einer Strategie zu bewerten, die Exploitability. Darüber hinaus wurde beschrieben, wie diese mithilfe eines Best Response Agents berechnet wird und was für Optimierungen in der Berechnung vorgenommen werden können.

2.7 Hold'em-Spiele

In diesem Abschnitt wird erklärt, wodurch sich Hold'em-Spiele als Untergruppe von Pokerspielen auszeichnen. Ist die Grundstruktur von Hold'em-Spielen einmal definiert, lässt sie sich einfach erweitern. Die Definition spezifischer Pokervarianten erfolgt im Anhang B.

Diese Arbeit beschäftigt sich ausschließlich mit Heads-Up-Limit-Versionen von Hold'em-Spielen. Heads-Up bedeutet, dass genau zwei Spieler teilnehmen. Die Klassifikation Limit bezieht sich darauf, dass die Setzgrößen der Spieler auf vorher festgelegte Beträge beschränkt sind. Diese beiden Einschränkungen reduzieren die Komplexität erheblich. Die folgende Definition gilt für alle Heads-Up-Limit-Hold'em-Spiele [Southey et al., 2005].

Eine Partie nennt man eine *Hand*.

Eine Hand wiederum besteht aus mehreren Runden.

In der ersten Runde erhalten beide Spieler eine feste Anzahl privater Karten (Hole Cards). In *jeder* Runde können zusätzlich öffentliche Karten (Board Cards) aufgedeckt werden (auch null).

Nach dem Austeiln bzw. Aufdecken der Karten folgt jeweils eine Setzrunde, in der die

Spieler abwechselnd handeln. Zur Verfügung stehen drei Aktionen: „Fold, Call und Raise“.

Wählt ein Spieler die Aktion *Fold*, endet die Hand sofort und der andere Spieler gewinnt den Pot. Der Pot ist die Ansammlung aller Einsätze, die im Verlauf der Hand gesetzt wurden.

Bei *Call* gleicht der Spieler den Einsatz des Gegners (der Beitrag kann null sein; dann spricht man von *Check*).

Bei *Raise* gleicht der Spieler den Gegnerzug und setzt zusätzlich den vorgeschriebenen Aufschlag.

Die Setzrunde endet, sobald ein Spieler *foldet* (dann endet die Hand sofort) oder *callt* (dann endet nur die Runde).

Um endlose Raise-Ketten zu vermeiden, ist die Zahl der Raises pro Runde begrenzt. Sobald das Limit erreicht ist, sind nur noch die Aktionen Call und Fold erlaubt.

Läuft die letzte Setzrunde ohne Fold aus, folgt ein *Showdown*: Beide Spieler zeigen ihre Hole Cards und bilden aus Hole und Board Cards die bestmögliche Hand. Die stärkere Hand gewinnt den Pot.

Die zulässigen Pokerhände und deren Rangfolge hängen von der Art des Spiels ab. Eine Übersicht über die möglichen Pokerhände in Texas Hold’em findet sich im Anhang A. Die Rangfolge anderer Spiele ist in der Definition der einzelnen Spiele enthalten B.

Im Folgenden werden die zentralen Bestandteile genannt, die relevant sind, um Heads-Up-Limit-Hold’em-Varianten zu beschreiben.

- die Anzahl der Hole Cards und Board Cards
- die Anzahl der Setzrunden
- das Setzlimit pro Runde
- die im Deck enthaltenen Karten
- die möglichen Pokerhände
- die festgelegten Setzgrößen

In diesem Kapitel wurden die theoretischen Grundlagen der zentralen Komponenten dieser Arbeit dargestellt. Das folgende Kapitel legt fest, welche Algorithmen, Evaluationsmethoden und Pokerspiele in der Implementierung tatsächlich umgesetzt werden.

Kapitel 3

Analyse

In diesem Kapitel wird der Rahmen für den Implementierungsteil dieser Arbeit gesetzt. Dies umfasst die Spielvarianten, die gelöst werden sollen, die zur Lösung verwendeten CFR-Algorithmen sowie die Evaluationsmethoden, die zur Bewertung der Strategiequalität genutzt werden. Zudem wird kurz auf die ursprünglich im Exposé gestellten Anforderungen an die Implementierung und die Unterschiede zur letztendlichen Umsetzung eingegangen.

3.1 Spielvarianten

Im Exposé wurde eine grobe Einschätzung darüber gegeben, welche Pokervarianten gelöst werden sollen. Dazu gehören zunächst das wohl einfachste Pokerspiel Kuhn Poker sowie die häufig genutzte Form Leduc Hold'em, die beide ausschließlich zu Forschungszwecken verwendet werden. Das dritte Spiel wurde eigenständig entwickelt und als „Twelve Card Poker“ bezeichnet. Dies erfolgte, da zu diesem Zeitpunkt noch keine weiteren Forschungsvarianten bekannt waren und eine weitere Komplexitätsstufe benötigt wurde. Im Laufe der Arbeit wurde eine weitere Pokervariante mit dem Namen „Small Island Hold'em“ als nächste Komplexitätsstufe unterhalb von Rhode Island Hold'em entwickelt.

3.2 CFR-Solver

In dieser Arbeit werden vier Varianten von CFR-Algorithmen implementiert: Vanilla CFR, CFR+, DCFR und MCCFR. Die Auswahl der Varianten CFR, CFR+ und MCCFR erfolgte zu Beginn der Arbeit auf Basis einer ersten Literaturrecherche. Diese Varianten wurden in der zu diesem Zeitpunkt bekannten Literatur am häufigsten diskutiert und bilden die Grundlage für viele erfolgreiche Anwendungen.

Vanilla CFR dient als Baseline für den Vergleich. MCCFR wurde ausgewählt, um eine Variante zu untersuchen, die nicht den gesamten Spielbaum explizit durchläuft, sondern auf Sampling basiert. Als drittes wird CFR+ implementiert, da es die Grundlage für erfolgreiche Poker-KIs wie Cepheus oder Libratus bildet.

Aufgrund von Problemen bei der Evaluation der MCCFR Ergebnisse wurde im Verlauf der Arbeit DCFR als weitere Optimierung hinzugefügt, um genügend Daten für eine sinnvolle Evaluation zu haben.

3.3 Evaluationsmethoden

Die Evaluation erfolgt mittels Exploitability-Tests mit einem Best Response Agent. Diese Methode ermöglicht es, die Qualität der trainierten Strategien zu bewerten, indem ein optimaler Gegner simuliert wird, der die Strategie maximal ausnutzt. Um das Konvergenzverhalten eines Algorithmus einschätzen zu können, wird während des Trainings in regelmäßigen Abständen die Exploitability bestimmt. Nach dem Training wird diese dann in Abhängigkeit von der Anzahl der Iterationen oder der vergangenen Zeit in einem Diagramm dargestellt.

Eine weitere Metrik, die ursprünglich geplant war, ist der Erwartungswert einer Strategie im Self-Play. Hierbei spielt eine Strategie wiederholt gegen sich selbst, wodurch im Falle eines Nash-Equilibriums der Erwartungswert des Spiels berechnet werden kann. Bei Forschungsvarianten wie Kuhn Poker und Leduc Hold'em ist dieser Wert bekannt und kann daher zur Verifikation der Implementierung genutzt werden. Im Verlauf der Arbeit hat sich jedoch herausgestellt, dass diese Metrik nicht aussagekräftig genug ist und außerdem implizit in der Best Response Berechnung enthalten ist.

In diesem Kapitel wurden die Komponenten dieser Arbeit vorgestellt und die Anforderungen an diese definiert. Im nächsten Kapitel wird die Architektur der Software beschrieben. Dabei werden Klassen und Aktivitätsdiagramme genutzt, um die Funktionsweise verschiedener Implementierungsarten darzustellen. Zusätzlich werden zu ausgewählten Themen Code-Abschnitte präsentiert.

Kapitel 4

Architektur und Design

In diesem Kapitel wird die Architektur des Softwareprojekts beschrieben. Hierbei wird zuerst ein Überblick darüber gegeben, wie die einzelnen Bestandteile miteinander interagieren. Anschließend wird auf die Struktur der einzelnen Komponenten eingegangen und die wichtigsten von ihnen mit UML-Diagrammen dargestellt. Kritische Designentscheidungen werden erklärt und Probleme, die durch diese entstanden sind, werden erwähnt. Die Probleme und deren Konsequenzen werden anschließend in Kapitel 5 genauer erläutert.

Eine zentrale Designentscheidung, die schon zu Beginn getroffen wurde, ist eine explizite Trennung der Pokerspiellogik vom CFR-Algorithmus. Die Logik der Pokerspiele, ab jetzt Game Environment genannt, wird als separate Entität implementiert, die das Spielverhalten und die Zustandsverwaltung übernimmt. Ziel dieser Unterteilung war es, das Projekt später einfach erweiterbar für neue Spiele und Algorithmen zu halten.

Dieses Kapitel gliedert sich in drei Hauptkomponenten: Game Environment, CFR-Solver und Evaluationsmethoden. Außerdem werden zwei verschiedene Arten der Implementierung des CFR-Solvers besprochen.

Eigener und verwendeter Code Die logische Struktur des Game Environment (Aufteilung in die Klassen Dealer, Player, Judger, Game und Round) wurde von der Bibliothek RL Card (<https://rlcard.org>) übernommen. Die CFR-Solver wurden initial anhand der zugrunde liegenden Paper implementiert. Zu diesem Entwicklungszeitpunkt wurde noch kaum auf Agentic Coding zurückgegriffen. OpenSpiel [Lanctot et al., 2019] diente später als Referenz zum Auffinden von Fehlern und für strukturelle Optimierungen. Ein Beispiel hierfür ist die Realisierung der CFR+ und DCFR Operationen mit Hooks nach jeder Traversierung. In späteren Entwicklungsphasen, vor allem bei dem Layer-basierten Ansatz wurde Agentic Coding deutlich mehr verwendet. Die TUI (Poker CFR Manager) verwendet die Bibliothek Memray von Bloomberg <https://github.com/bloomberg/memray>. Das GUI stammt aus dem Modul Advanced Python und wurde nicht im Rahmen dieser Arbeit implementiert.

4.1 Game Environment

In diesem Abschnitt wird der Aufbau der Spielumgebung mithilfe eines Klassendiagramms beschrieben. Die Spielumgebung ist für die Logik des Pokerspiels sowie die Verwaltung des Spielzustands zuständig. Eine Spielumgebung besteht aus den folgenden Komponenten: Player (repräsentiert die beiden Spieler), Dealer (verwaltet das Kartendeck), Judge (bestimmt den Gewinner) und Game (koordiniert die anderen Komponenten). Bei allen Varianten außer Kuhn Poker wird diese Architektur zusätzlich durch eine Round-Klasse erweitert, die den Übergang zwischen Setzrunden steuert. Diese Unterteilung der Funktionalitäten der Spielumgebung wurde von RLCARD übernommen (<https://rlcard.org>).

Abbildung 4.1 zeigt das Klassendiagramm der Game Environment für die Pokervarianten Kuhn Poker und Leduc Hold'em. Alle weiteren Spielvarianten basieren auf Leduc Hold'em und erweitern diese um zusätzliche Funktionalitäten.

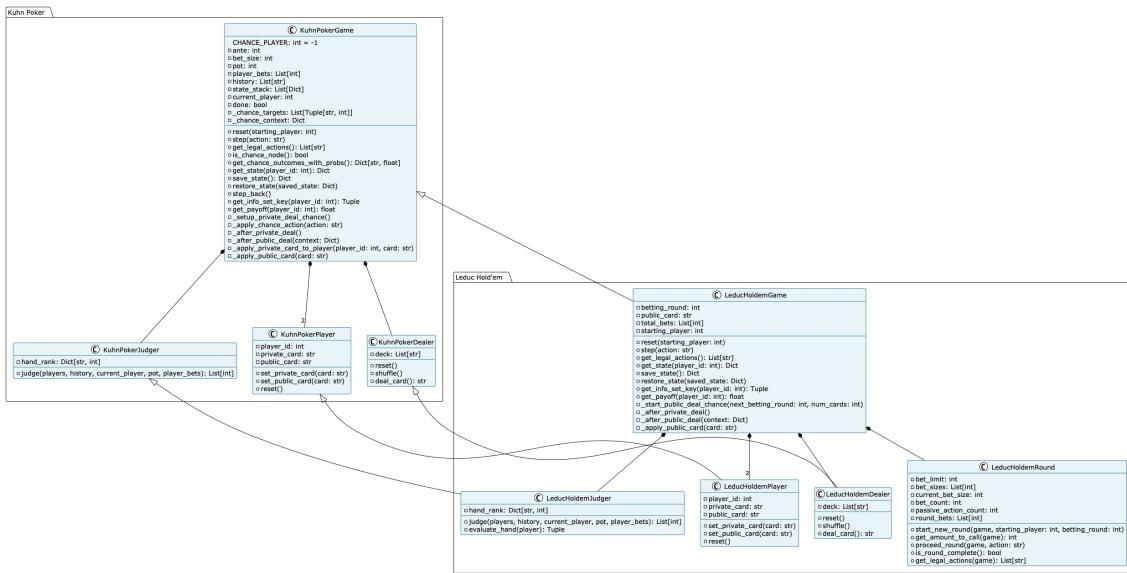


Abbildung 4.1: Klassendiagramm der Game Environment für Kuhn Poker und Leduc Hold'em

4.2 CFR-Solver

Dieser Abschnitt beschreibt verschiedene Designs, die zur Implementierung eines CFR-Algorithmus genutzt wurden. Das erste Design nutzt die öffentlichen Methoden der Game Environment, um während des Trainings den Spielbaum dynamisch aufzubauen und zu durchlaufen. Das zweite Design verwendet einen statischen Spielbaum, der im Voraus mithilfe der Game Environment gebaut und gespeichert wurde. Diese Variante speichert den Spielbaum als Python-Objekte in Dictionary-Strukturen und durchläuft ihn rekursiv. Die dritte Variante nutzt ebenfalls einen statischen Spielbaum, jedoch in einer Array-basierten Repräsentation mit NumPy-Arrays statt Python-Objekten. Anstelle einer rekursiven Traversierung wird hier eine schichtweise Berechnung verwendet, bei der Knoten nach ihrer Tiefe gruppiert und verarbeitet werden.

Erwähnenswert ist, dass ursprünglich keine expliziten Zufallsknoten implementiert wurden, sondern alle Zufallskombinationen vorab mit einer Klasse namens Combination Generator erzeugt wurden. Dies hat an vielen Stellen Probleme bereitet und wurde erst zu einem späten Zeitpunkt aus dem Design entfernt. Deshalb sind an einigen Stellen noch Rückstände dieses Codes enthalten.

4.2.1 Dynamisch erzeugter Spielbaum

Die dynamische Implementierung des CFR Solvers verzichtet auf die Speicherung eines expliziten Spielbaums. Stattdessen wird der Spielbaum bei jeder Iteration neu durch wiederholte Aufrufe von `game.step()` und `game.step_back()` traversiert. Dieser Ansatz benötigt weniger Arbeitsspeicher als die statische Variante, da nicht der gesamte Baum im Speicher gehalten werden muss. Allerdings ist er durch wiederholtes Aufrufen der Spielzustandsfunktionen deutlich langsamer.

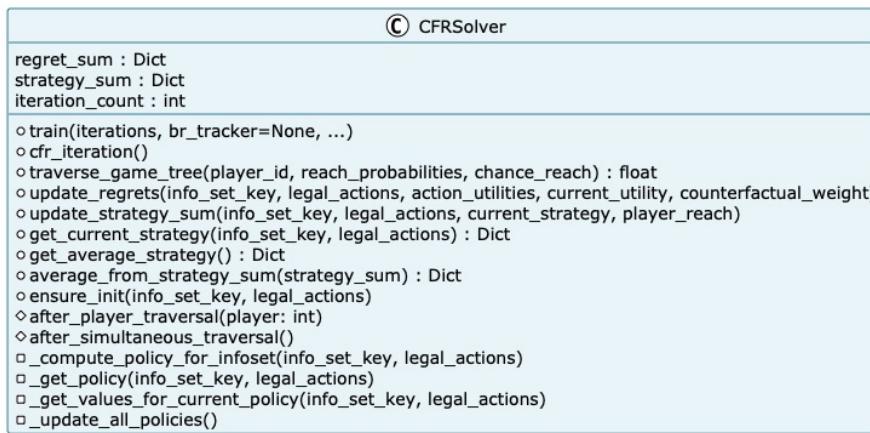


Abbildung 4.2: Klassendiagramm der dynamischen CFR Solver Architektur

Der Trainingsprozess beginnt mit der Initialisierung der Objekte Game und Solver. Eine Iteration des CFR-Algorithmus enthält dann folgende Schritte:

Der Spielbaum wird für beide Spieler durchlaufen, wobei entweder alternierende oder simultane Updates durchgeführt werden können. Bei alternierenden Updates wird zunächst der Spielbaum für Spieler 0 durchlaufen und dessen Strategie aktualisiert. Danach für Spieler 1 mit der bereits aktualisierten Strategie von Spieler 0. Bei simultanen Updates wird der Spielbaum für beide Spieler mit derselben Strategie durchlaufen, anschließend werden die Strategien beider Spieler aktualisiert. Nachdem der Spielbaum traversiert wurde, wird die Strategie für alle Informationsets mittels Regret Matching basierend auf den kumulierten Counterfactual Regrets aktualisiert. Hiermit endet eine Iteration.

Optional kann während des Trainings eine Exploitability-Berechnung durchgeführt werden. Hierfür wird die durchschnittliche Strategie berechnet und anschließend eine Best-Response-Evaluation durchgeführt.

Dieser Prozess wird für die gewünschte Anzahl an Iterationen wiederholt und anschließend wird das Modell gespeichert. Wichtig zu erwähnen ist, dass die Best-Response-Evaluation nicht bei jeder Iteration durchgeführt wird, sondern in regelmäßigen Abständen gemäß einer konfigurierbaren Evaluationsfrequenz. Diese Abstände werden so

gewählt, dass aus den gesammelten Werten anschließend ein aussagekräftiger Konvergenzplot erstellt werden kann.

Auf der folgenden Seite wird der eben wörtlich beschriebene Ablauf mithilfe eines UML-Aktivitätsdiagramms veranschaulicht.

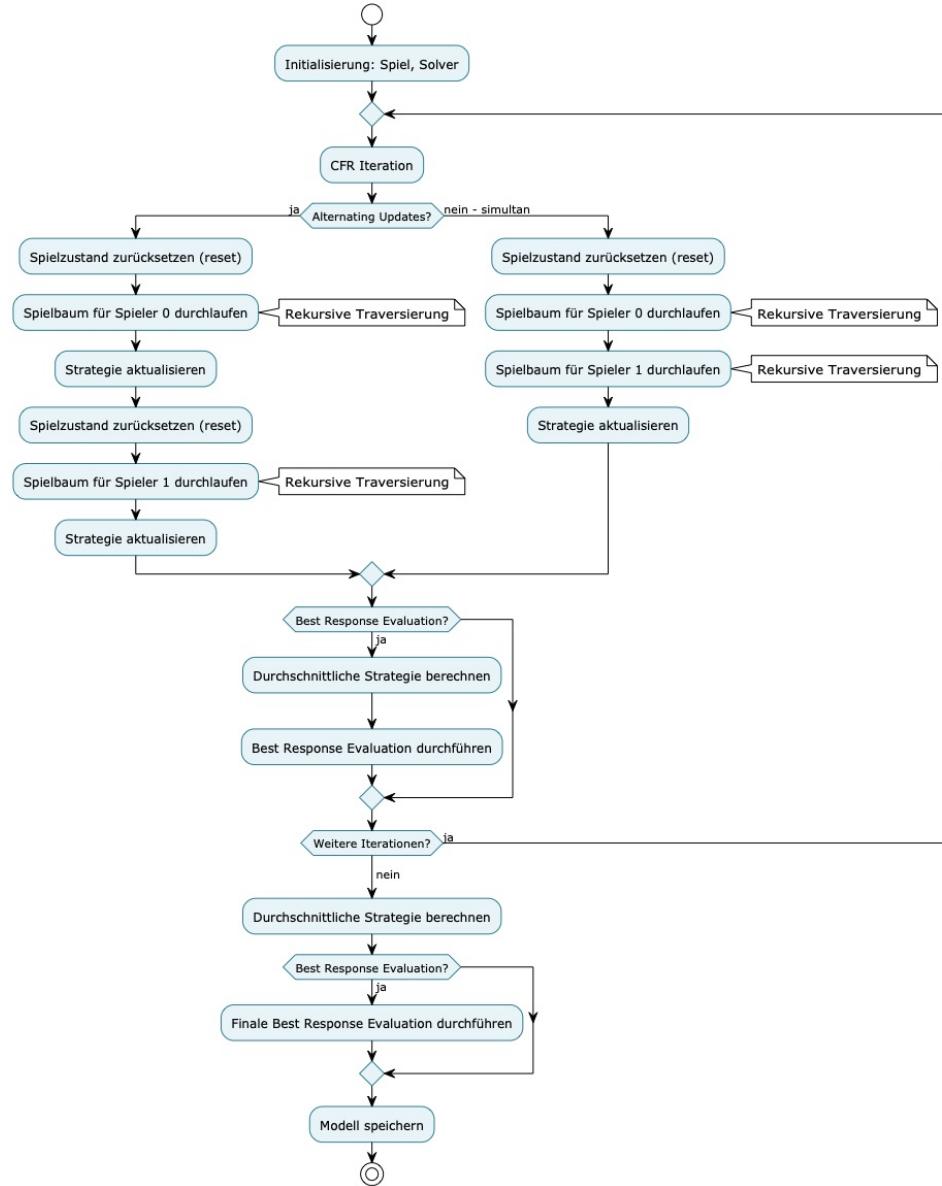


Abbildung 4.3: Aktivitätsdiagramm des Trainingsprozesses der dynamischen CFR Solver Implementierung

Auf dem UML Diagramm ist eine rekursive Traversierungsfunktion zu sehen. Diese wird hier genauer erklärt und mit einem weiteren Aktivitätsdiagramm veranschaulicht. Die rekursive Traversierungsfunktion berechnet den erwarteten Nutzen für einen gegebenen Spieler und aktualisiert dabei dessen Counterfactual Regrets. Als Parameter erhält sie die Spieler-ID, die Erreichbarkeitswahrscheinlichkeiten beider Spieler sowie die Zufallswahrscheinlichkeit. Die Erreichbarkeitswahrscheinlichkeiten beschreiben die Wahrscheinlichkeit, den aktuellen Knoten zu erreichen, während die Zufallswahrscheinlichkeit die kumulierte Wahrscheinlichkeit aller Zufallsereignisse entlang des Pfades beschreibt. Beim initialen Aufruf starten beide Wahrscheinlichkeiten bei 1.0.

Die Traversierung unterscheidet zwischen vier Knotentypen: Bei einem **Terminalknoten** wird der reale Nutzen für den betrachteten Spieler vom Game-Objekt abgerufen und zurückgegeben.

Bei einem **Zufallsknoten** werden alle möglichen Outcomes mit ihren Wahrscheinlichkeiten abgerufen. Für jedes Outcome wird die Zufallswahrscheinlichkeit mit der Outcome-Wahrscheinlichkeit multipliziert und rekursiv traversiert. Der erwartete Nutzen wird über alle Outcomes gewichtet berechnet. Es werden keine Regrets aktualisiert, die Zufallswahrscheinlichkeit wird jedoch weitergegeben, da sie für die Gewichtung der Regret-Updates benötigt wird.

Bei einem **Gegner-Knoten** wird die aktuelle Strategie für das Informationset bestimmt. Jede mögliche Aktion wird ausgeführt, die Erreichbarkeitswahrscheinlichkeiten werden aktualisiert und der Spielbaum rekursiv durchlaufen, anschließend wird die Aktion rückgängig gemacht. Der erwartete Nutzen wird als gewichtete Summe über alle Aktionen berechnet, wobei die Gewichtung durch die Strategie des Gegners erfolgt.

Bei einem **Spieler-Knoten** wird analog vorgegangen, jedoch wird der Nutzen jeder Aktion gespeichert. Nach der Berechnung des erwarteten Nutzens wird für jede Aktion der Counterfactual Regret berechnet und mit der Erreichbarkeitswahrscheinlichkeit des Gegners sowie der Zufallswahrscheinlichkeit multipliziert. Die kumulierten Regrets und die kumulierte Strategie werden für das Informationset aktualisiert.

Die Methode `traverse_game_tree` realisiert die rekursive CFR-Traversierung für einen Spieler. Sie berechnet den erwarteten Nutzen entlang des Baums und aktualisiert an jedem *Spieler-Knoten* die kumulierten Regrets sowie die kumulierte Strategie für das zugehörige Informationsset. An Zufalls- und Gegner-Knoten wird nur der Nutzen propagiert; `reach_probabilities` und `chance_reach` entsprechen den Erreichbarkeitswahrscheinlichkeiten $\pi_i^o(h)$ bzw. der kumulierten Zufallswahrscheinlichkeit und werden für die Gewichtung in `update_regrets` und `update_strategy_sum` benötigt.

Es folgt Quellcode der rekursiven Traversierungsfunktion des dynamischen CFR Solvers:

Listing 4.1: `traverse_game_tree`

```

def traverse_game_tree(self, player_id, reach_probabilities, chance_reach: float = 1.0):
    """
    Traverse game tree and compute counterfactual regret for one player.
    """
    if self.game.done:
        return self.game.get_payoff(player_id)

    # Chance node: expectation over chance outcomes (no regret updates)
    if hasattr(self.game, 'is_chance_node') and self.game.is_chance_node():
        outcomes_with_probs = self.game.get_chance_outcomes_with_probs()
        if not outcomes_with_probs:
            return 0.0
        value = 0.0
        for outcome, prob in outcomes_with_probs.items():
            if prob == 0:
                continue
            self.game.step(outcome)
            value += prob * self.traverse_game_tree(
                player_id, reach_probabilities, chance_reach * prob)
            self.game.step_back()
    return value

    current_player = self.game.current_player

    # Opponent's node, don't update regrets
    if current_player != player_id:
        legal_actions = self.game.get_legal_actions()
        opponent = 1 - player_id
        opponent_info_set = KeyGenerator.get_info_set_key(self.game, opponent)
        self.ensure_init(opponent_info_set, legal_actions)
        opponent_strategy = self.get_current_strategy(
            opponent_info_set, legal_actions)
        state_value = 0.0
        for action in legal_actions:
            action_prob = opponent_strategy[action]
            self.game.step(action)
            new_reach_probs = reach_probabilities.copy()
            new_reach_probs[opponent] *= action_prob
            state_value += action_prob * self.traverse_game_tree(
                player_id, new_reach_probs, chance_reach)
            self.game.step_back()
    return state_value

    # Player's node, update regrets
    info_set_key = KeyGenerator.get_info_set_key(self.game, player_id)
    legal_actions = self.game.get_legal_actions()
    self.ensure_init(info_set_key, legal_actions)
    current_strategy = self.get_current_strategy(info_set_key, legal_actions)
    action_utilities = {}
    for action in legal_actions:
        self.game.step(action)
        new_reach_probs = reach_probabilities.copy()
        new_reach_probs[player_id] *= current_strategy[action]

```

```

        action_utilities[action] = self.traverse_game_tree(
            player_id, new_reach_probs, chance_reach)
        self.game.step_back()
    current_utility = sum(
        current_strategy[action] * action_utilities[action]
        for action in legal_actions)
    counterfactual_weight = reach_probabilities[1 - player_id] * chance_reach
    player_reach = reach_probabilities[player_id] * chance_reach
    self.update_regrets(info_set_key, legal_actions, action_utilities,
                        current_utility, counterfactual_weight)
    self.update_strategy_sum(info_set_key, legal_actions, current_strategy,
                            player_reach)
    return current_utility

```

An Spielerknoten werden die Regretwerte sowie die Strategie akkumuliert.

Listing 4.2: update_regrets und update_strategy_sum

```

def update_regrets(self, info_set_key, legal_actions, action_utilities,
                   current_utility, counterfactual_weight):
    for action in legal_actions:
        instantaneous_regret = counterfactual_weight * (
            action_utilities[action] - current_utility)
        self.regret_sum[info_set_key][action] += instantaneous_regret

def update_strategy_sum(self, info_set_key, legal_actions,
                       current_strategy, player_reach):
    weight = self._get_strategy_sum_weight()
    for action in legal_actions:
        self.strategy_sum[info_set_key][action] += (
            weight * player_reach * current_strategy[action])

```

Die aktuelle Strategie wird für jedes Informationset mittels Regret Matching bestimmt.

Listing 4.3: Regret Matching: _compute_policy_for_infoset

```

def _get_values_for_current_policy(self, info_set_key, legal_actions):
    regrets = self.regret_sum.get(info_set_key, {})
    return {a: max(0.0, regrets.get(a, 0.0)) for a in legal_actions}

def _compute_policy_for_infoset(self, info_set_key, legal_actions):
    values = self._get_values_for_current_policy(
        info_set_key, legal_actions)
    total = sum(values.values())
    if total > 0:
        return {a: values[a] / total for a in legal_actions}
    return {a: 1.0 / len(legal_actions) for a in legal_actions}

```

Hier wird der Ablauf der rekursiven Traversierung mit einem UML-Aktivitätsdiagramm beschrieben.

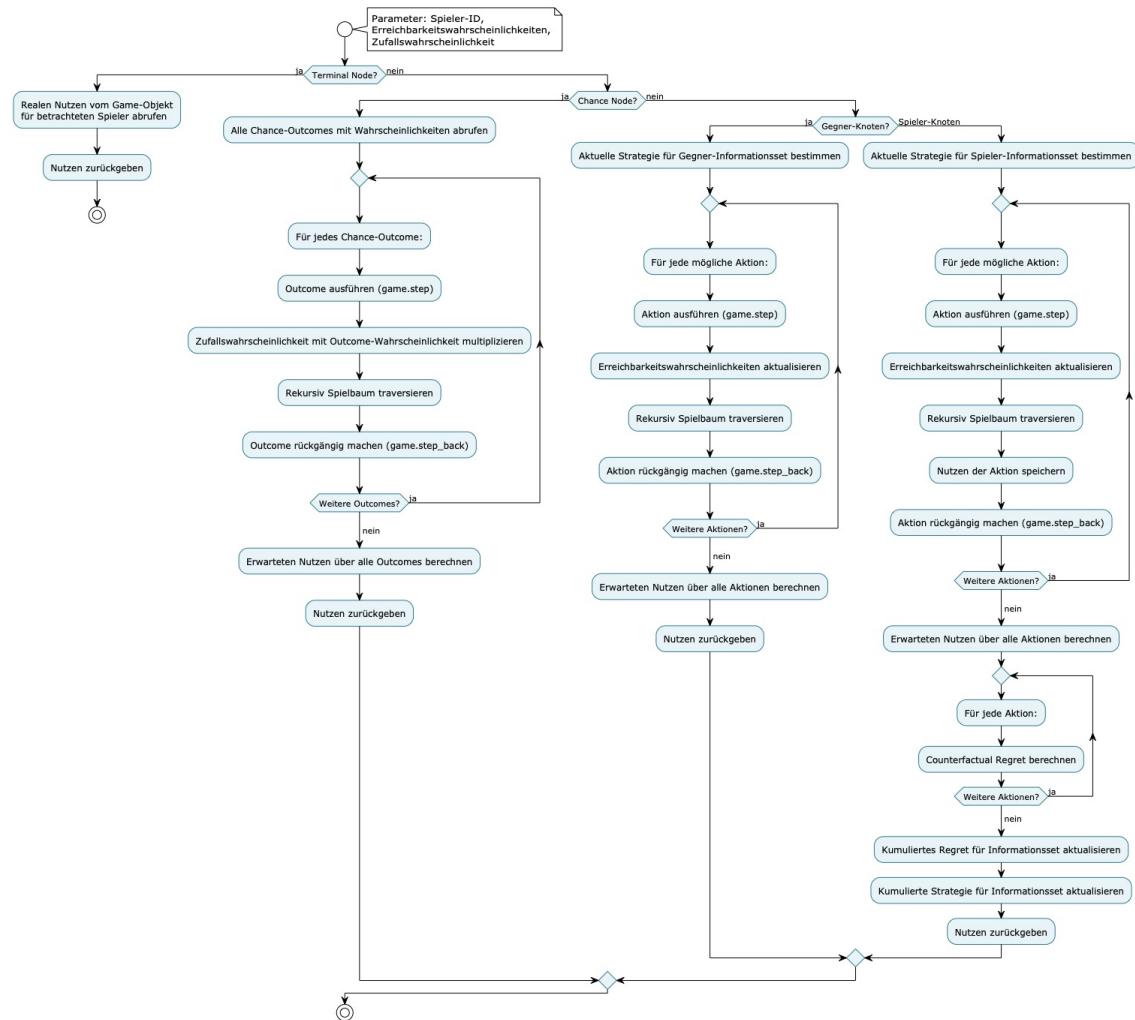


Abbildung 4.4: Aktivitätsdiagramm der Spielbaum-Traversierung der rekursiven Implementierung

4.2.2 Statischer Spielbaum

Es gibt zwei verschiedene Varianten eines statischen Spielbaums, die in dieser Arbeit umgesetzt wurden. Die erste Variante durchläuft einen Spielbaum aus Python-Objekten rekursiv, die zweite nutzt eine schichtweise Berechnung auf einem Spielbaum aus NumPy-Arrays. In diesem Kapitel wird ausschließlich der zweite Ansatz beschrieben, da sich die Traversierung des ersten Ansatzes kaum von der des dynamischen unterscheidet. Die Idee für den Layer-by-Layer Aufbau kommt aus den folgenden zwei Quellen [Kim, 2024, Reis, 2015].

Die `FlatTree` Datenstruktur speichert den Spielbaum mit NumPy-Arrays statt Python-Objekten und benötigt deshalb deutlich weniger Speicherplatz. Jeder Knoten wird über eine Integer-ID identifiziert. Die Struktur des Baums wird durch verschiedene Arrays repräsentiert. `node_type` speichert den Typ jedes Knotens (Terminal, Entscheidung oder Zufall). Für Entscheidungsknoten speichert `children` die Verbindungen zu ihren Kindknoten. Für Zufallsknoten werden die möglichen Outcomes und deren Wahrscheinlichkeiten in separaten Arrays gespeichert. Zusätzlich werden die Knoten nach ihrer Tiefe in Layern gruppiert (`layer_terminal_nodes`, `layer_chance_nodes`, `layer_decision_nodes`). Auf der folgenden Seite ist die Layer-basierte Traversierung in einem UML-Aktivitätsdiagramm dargestellt.

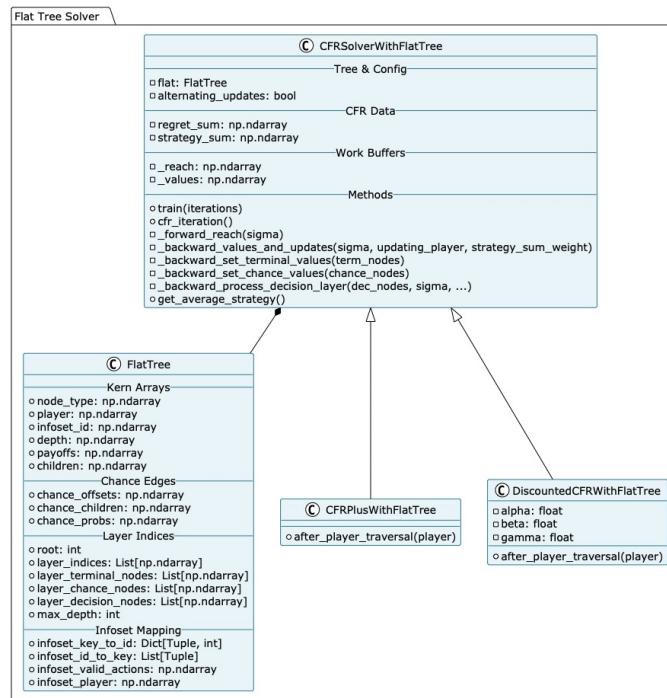


Abbildung 4.5: Klassendiagramm der Layer-by-Layer CFR Solver Architektur

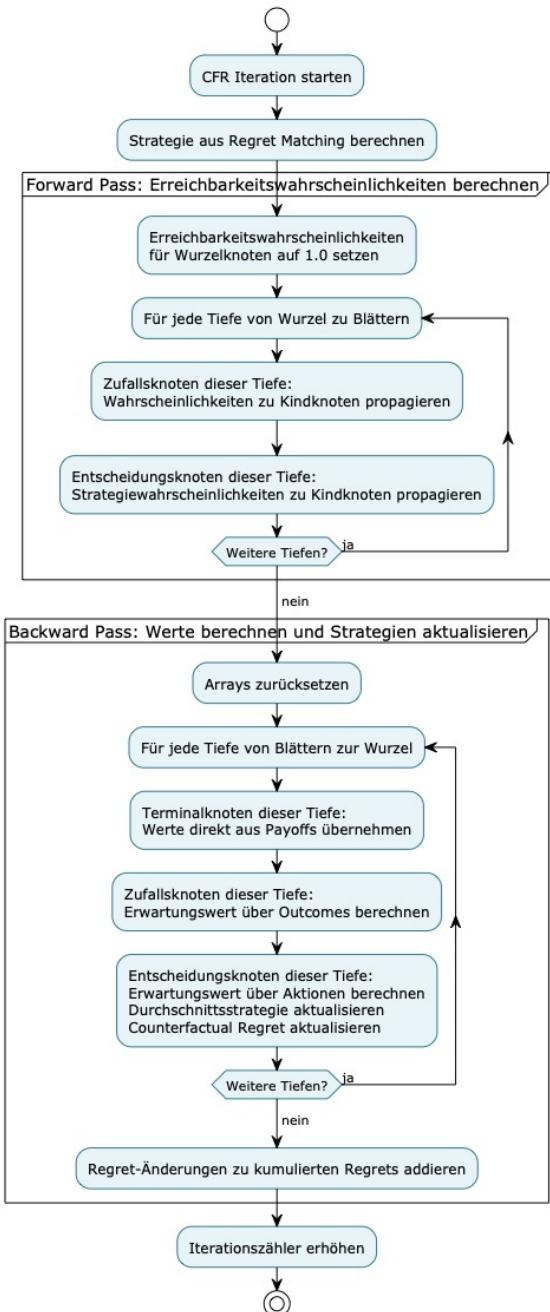


Abbildung 4.6: Aktivitätsdiagramm der Layer-by-Layer Traversierung

4.3 Best Response Agent

Der Best Response Agent berechnet die Exploitability einer Strategie durch Traversierung des Public State Trees. In diesem Abschnitt wird dieser Ablauf mittels UML Diagrammen beschrieben.

Der Public State Tree wird einmalig gebaut und anschließend gespeichert.

4.3.1 Gesamtablauf der Best-Response-Berechnung

Die Berechnung beginnt mit dem Laden des Public State Trees und der durchschnittlichen Strategie. Für jedes Informationset des betrachteten Spielers werden die gültigen Gegner-Informationsets bestimmt (ausschließlich solche mit unterschiedlichen Karten). Die Erreichbarkeitswahrscheinlichkeiten für den Gegner werden berechnet, und anschließend wird der Public State Tree rekursiv traversiert. Der Gesamtnutzen wird als gewichtete Summe über alle Informationsets berechnet und als Exploitability zurückgegeben.

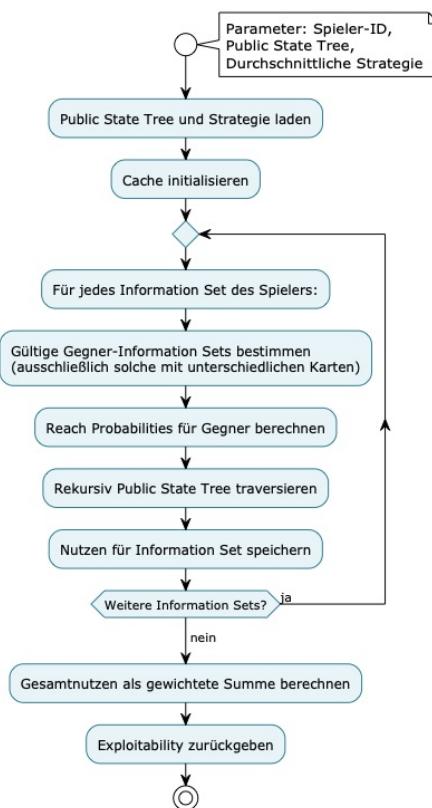


Abbildung 4.7: Aktivitätsdiagramm der Best-Response-Berechnung

4.3.2 Details der rekursiven Traversierung

Die rekursive Traversierung des Public-State-Trees unterscheidet zwischen vier Knotentypen. Handelt es sich um einen **Terminalknoten**, wird die Hand-Strength für alle Informationsets berechnet (mit Caching zur Performance-Optimierung), und die Payoffs werden als gewichtete Summe über die Erreichbarkeitswahrscheinlichkeiten des Gegners berechnet.

Bei einem **Zufallsknoten** wird für jedes Zufallsereignis der neue Public State bestimmt und rekursiv weiter traversiert. Der erwartete Nutzen wird als gewichtete Summe über alle Zufallsereignisse berechnet.

Bei einem **Entscheidungsknoten** wird unterschieden, ob es sich um einen Knoten des betrachteten Spielers oder des Gegners handelt. Handelt es sich um einen Knoten des **betrachteten Spielers**, wird für jede mögliche Aktion rekursiv traversiert und der Nutzen gespeichert. Für jedes Informationset wird anschließend die Best Response berechnet, indem das Maximum über alle Aktionen gewählt wird.

Handelt es sich um einen **Gegner-Knoten**, wird die Strategie des Gegners für die entsprechenden Informationsets abgerufen. Für jede mögliche Aktion werden die Erreichbarkeitswahrscheinlichkeiten aktualisiert und rekursiv weiter traversiert. Der erwartete Nutzen wird als gewichtete Summe über alle Aktionen berechnet, wobei die Gewichte die Strategie-Wahrscheinlichkeiten des Gegners sind.

Die Terminalknoten-Evaluierung nutzt Hand-Strength-Sortierung für $O(n)$ statt $O(n^2)$ Komplexität, und Hand-Strength-Evaluierungen werden gecacht, um wiederholte Berechnungen zu vermeiden.

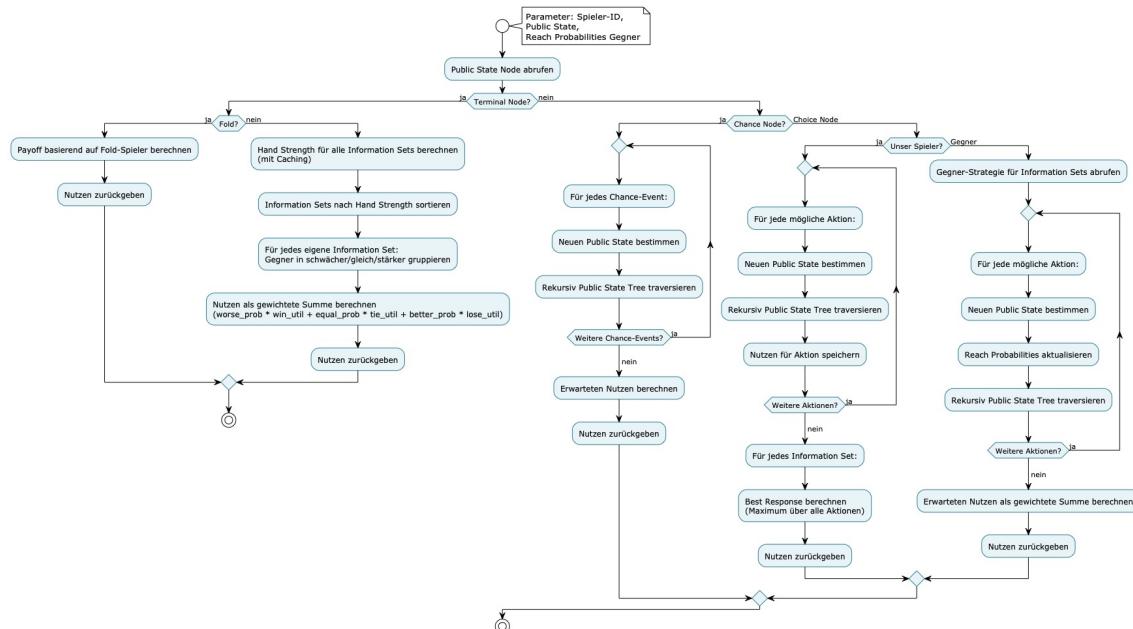


Abbildung 4.8: Aktivitätsdiagramm der Public-State-Tree-Traversierung

Text-based User Interface Die Anwendung bietet eine textbasierte Benutzeroberfläche (TUI), den *Poker CFR Manager*, zum Verwalten von Trainingsläufen, Anzeigen des Speicherverbrauchs sowie starten des GUI. Das Feature wurde gegen Ende des Projekts umgesetzt und ist daher nicht in allen Teilen ausgereift. Es erleichtert jedoch Nutzern des Projekts den Trainingsvorgang deutlich. Während der Entwicklung erfolgte das Training über die Kommandozeile mit vielen Parametern als Flags, was durch die TUI vereinfacht wird. Die TUI gliedert sich in vier Tabs.

Im Tab *Training Queue* lässt sich das Training von Modellen über Buttons konfigurieren. Im Tab *Current Task* wird der Verlauf des aktuellen Trainings inklusive der Best-Response-Werte angezeigt.

Im Tab *Models* finden sich die trainierten Strategien. Von dort aus kann der Konvergenzverlauf mehrerer Strategien verglichen sowie das GUI gestartet werden, das im nächsten Abschnitt beschrieben wird.

Der Tab *Memory Profiler* zeigt den Speicherverbrauch zur Laufzeit. Dafür wird die Open-Source-Bibliothek Memray von Bloomberg eingesetzt <https://github.com/bloomberg/memray>.

Graphical User Interface Das GUI ermöglicht es, gegen eine trainierte Strategie zu spielen. Es wird aus der TUI im Tab *Models* gestartet. Die Implementierung des GUI ist nicht Gegenstand dieser Bachelorarbeit, sondern gehört zum Modul Advanced Python. Diese Arbeit nutzt es lediglich dazu, eine greifbare Evaluation einer Strategie zu ermöglichen.

In diesem Kapitel wurde die Unterteilung der Software in die Komponenten Spielumgebung, Solver und Evaluation mit UML Diagrammen beschrieben. Hierbei wurde genauer auf die Funktionsweise der rekursiven und Layer-basierten Traversierung eingegangen. Außerdem wurde der Ablauf des Best Response Agents und die rekursive Traversierung näher beschrieben. Im nächsten Kapitel wird der Entwicklungsvorgang beschrieben und anschließend auf die dabei entstandenen Probleme eingegangen.

Kapitel 5

Entwicklungsverlauf

Dieses Kapitel schildert zunächst den chronologischen Entwicklungsverlauf. Dabei werden die aufgetretenen Probleme sowie die damit einhergehenden Konsequenzen dargestellt. Abschließend werden die gewonnenen Erkenntnisse präsentiert und es wird beschrieben, was beim nächsten Schreiben einer wissenschaftlichen Arbeit verbessert werden kann.

5.1 Chronologischer Ablauf

Dieser Abschnitt beschreibt den praktischen Entwicklungsweg von der ersten lauffähigen Implementierung bis zu dem Stand, der in Kapitel 6 evaluiert wird. Hierbei wird erklärt, welche Entscheidungen im Projektverlauf getroffen wurden und welche Auswirkungen diese hatten.

Die Entwicklung begann mit der Implementierung der Game-Environment. Hierbei wurden zuerst nur die Pokervarianten Kuhn Poker und Leduc Hold'em umgesetzt, um sich in das Thema einzuarbeiten. Es wurde noch kein Wert auf Schnelligkeit des Codes oder Speicheranforderungen gelegt, sondern es ging lediglich darum, Verständnis über den Algorithmus zu erlangen.

Anschließend wurde die erste Version eines CFR Solvers umgesetzt. Bei der Implementierung wurde sich an der Beschreibung des Papers [Zinkevich et al., 2007] orientiert. Daraus resultierte ein Algorithmus, der den Spielbaum dynamisch mithilfe der Zustandsfunktionen der Game-Environment aufbaut und rekursiv traversiert.

Als initiale Evaluation wurde ein Nash-Equilibrium Test geschrieben, welcher nur für Kuhn Poker funktioniert, indem er spezifische Strategiewahrscheinlichkeiten extrahiert und diese gegen die bekannten Nash-Equilibrium-Bedingungen prüft [Kuhn, 1950].

Als nächste Evaluation wurde ein Skript erstellt, das den Erwartungswert einer Strategie im Self-Play berechnet. Es stellte sich jedoch schnell heraus, dass dies keine aussagekräftige Metrik ist. Dieser Wert ist nur von Nutzen, falls der Erwartungswert des betrachteten Spiels bekannt ist. Dies ist für Spiele wie Kuhn Poker [Kuhn, 1950] und Leduc Hold'em [Berns, 2024] der Fall, jedoch sind solche Werte für größere Pokervarianten schwer zu finden.

Im nächsten Schritt wurde zur Verifizierung der Strategiequalität mit der Implementierung des Accelerated-Best-Response-Agents begonnen. Dies funktionierte für Kuhn Poker sofort, jedoch gab es für Spiele, die öffentliche Zufallsereignisse beinhalten, Probleme. Diese Probleme sind rückblickend auf simple Fehlentscheidungen in der Struktur der Game-Environment zurückzuführen.

Die initiale Implementierung hatte zwei große Schwächen. Zunächst gibt es keinen expliziten Knoten für öffentliche Zufallsereignisse. Das heißt, es gibt keinen Zustand, in dem die erste Setzrunde beendet ist, aber noch keine öffentliche Board-Karte ausgeteilt wurde. Die Game-Environment teilt zum Rundenende sofort eine öffentliche Karte aus. Dies stellt ein Problem dar, da der Public-State-Tree, über den der Best-Response-Agent traversiert, explizite Zufallsknoten benötigt, um zuverlässig funktionieren zu können.

Das zweite Problem ist direkt mit dem ersten verknüpft. Da es keine expliziten Zufallsknoten gibt, wird die Verteilung der Karten bereits vor dem Spiel festgelegt. Dies geschah durch die Klasse „Combination Generator“, die deterministisch alle möglichen Kartenkombinationen eines Spiels findet und dann vor dem Spiel das Deck in eine spezifische Reihenfolge mischt, damit die Game-Environment nach Rundenende die gewünschte Karte austeilt. Hierdurch wurde für jede mögliche Kombination also ein eigener Subtree aufgebaut. Dies ist problematisch, da dadurch bei jeder Iteration unnötig viele Pfade durchlaufen werden, die bereits durchlaufen wurden. Dadurch ist die Anzahl an Knoten der Spiele fehlerhaft, wodurch die Komplexität nicht richtig eingeschätzt werden kann. Außerdem steigert sich die Laufzeit pro Iteration enorm.

Diese Fehler waren zu diesem Zeitpunkt noch nicht erkannt, weshalb zunächst die Dokumentation der Arbeit in den Vordergrund gerückt ist. Anschließend wurde im Best-Response-Agent und Public State Tree sehr viel Workaround-Code geschrieben, um die vorher genannten Probleme zu umgehen. Durch diese Workarounds wurde die Evaluation funktionsfähig, war jedoch extrem langsam und schwer auf neue Situationen anzupassen.

Nachdem es dann möglich war, mittels des Best-Response-Agents die Exploitability einer Strategie zu berechnen, fielen einige Unstimmigkeiten im Konvergenzverhalten der Solver auf. Die Probleme an den Solvern CFR und CFR+ konnten relativ schnell behoben werden. Jedoch konnte für alle Sampling-basierten Ansätze kein erwartetes Konvergenzverhalten festgestellt werden. Zu diesem Zeitpunkt wurde vermutet, dass dies an der impliziten Zufallsknoten-Implementierung läge. Später stellte sich jedoch heraus, dass dies andere Gründe hat.

Da die Solver relativ langsam waren, fiel der Fokus in der folgenden Zeit nicht darauf, die Sampling-Solver zum Laufen zu bringen, sondern die bestehenden CFR- und CFR+ Implementierungen zu beschleunigen. Deshalb wurde ein Ansatz implementiert, der einen vorher gebauten und gespeicherten Tree verwendet und damit die teuren Spielzustandsfunktionsaufrufe der Game-Environment vermeidet. Jedoch brachte diese Implementierung bei größeren Spielen erhebliche Speicherprobleme mit sich. Ab einer gewissen Größe passt der Spielbaum nicht mehr in den Speicher.

Gegen Ende des Projekts wurde die Game-Environment überarbeitet, sodass explizite Zufallsknoten existieren. Dadurch wurde der gesamte Workaround-Code im Best-Response-Agent obsolet.

Da die Sampling-Solver weiterhin nicht wie erwartet funktionierten, wurde zu diesem Zeitpunkt der DCFR Algorithmus hinzugezogen.

Zusätzlich wurde ein neuer Ansatz mit einem statischen Spielbaum implementiert, der eine Layer-basierte Traversierung beinhaltet. Die Idee hierfür wurde aus den folgenden Paper [Kim, 2024] [Reis, 2015] entnommen. Dieser Ansatz reduziert die Iterationszeit gegenüber dem rekursiven Verfahren enorm.

Schließlich wurde am Ende des Projekts erkannt, warum die Sampling-Solver sich nicht wie erwartet verhalten. Sampling-basierte Solver traversieren nur einen zufällig ausgewählten Teil des Spielbaums. Dadurch ist die Strategie nach einer Iteration nur für die Pfade definiert, die besucht wurden. Der Best-Response-Agent evaluiert alle möglichen Informationsets eines Spielers. Wenn für ein Informationset keine Strategie vorhanden ist, fällt er auf eine uniforme Strategie zurück. Dadurch wird das Ergebnis stark verfälscht. Wenn man die Exploitability eines Sampling-basierten Solvers messen will, braucht man einen anderen Ansatz. Da diese Erkenntnis jedoch zu spät im Projektverlauf gewonnen wurde, werden die Sampling-basierten Ansätze nicht in der Evaluation betrachtet.

Es wurden im Verlauf des Projekts auch Dinge wie Partial Pruning getestet. Dies ist eine Pruning-Technik, bei der alle Pfade, für die die Erreichbarkeitswahrscheinlichkeit des Gegners null ist, nicht weiter betrachtet werden. Dies kann erhebliche Laufzeitverbesserungen mit sich bringen, hat aber aufgrund des gleichen Problems wie die Sampling-Solver nicht zu den erwünschten Konvergenzverhalten geführt.

Themenbereiche wie Abstraktion wurden ebenfalls betrachtet, jedoch stellte sich heraus, dass dieser Bereich einen so großen Umfang hat, dass er im Rahmen dieser Arbeit aufgrund von Zeitmangel nicht weiter verfolgt wurde.

5.2 Erkenntnisse und Reflexion

Dieser Abschnitt präsentiert die während des Entwicklungsvorgangs gewonnenen Erkenntnisse und reflektiert mögliche Verbesserungen.

Im Projektverlauf hat sich gezeigt, dass eine funktionsfähige Evaluation das wichtigste Werkzeug zur Sicherstellung der Korrektheit der Algorithmen ist. Ohne eine verlässliche Exploitability-Berechnung können Implementierungsfehler lange unentdeckt bleiben. Darüber hinaus hat sich gezeigt, dass Architekturentscheidungen, die zunächst wie reine Implementierungsdetails wirken, schwerwiegende Folgen haben können. Das beste Beispiel hierfür ist die Modellierung der Zufallsknoten. Solche Problemstellen sollten möglichst schnell behoben werden, da der Aufwand für Workaround-Code schnell wächst.

Im Verlauf der Arbeit wurden viele Erkenntnisse in Bezug auf die Strukturierung einer wissenschaftlichen Arbeitsweise gewonnen. Zunächst hätte die Literaturrecherche breiter angelegt werden sollen, statt sich früh auf wenige Standardvarianten wie CFR, CFR+ und MCCFR zu versteifen. Dadurch hätte früher erkannt werden können, dass es durchaus noch andere interessante CFR-Varianten wie DCFR gibt. Ein guter Einstieg wäre der

Kurs *Computational Game Solving* gewesen [Sandholm and Anagnostides, 2025]. Die Folien sind didaktisch deutlich besser zu verarbeiten als die zugrunde liegenden Paper und hätten in den Anfangsphasen einen breiten Überblick über das Thema geboten.

Die größte Schwierigkeit war das Setzen des Rahmens der Arbeit. Dem lag die Arbeitsreihenfolge von Recherche, Implementierung und Dokumentation zugrunde.

Nach einer ersten Literaturrecherche wurden die Spielumgebung und die ersten Solver implementiert. Als der Best Response Agent nicht funktionierte, wurde sich der Dokumentation gewidmet. Dies war jedoch ein Fehler. Große Teile der zu diesem Zeitpunkt verfassten Dokumentation erwiesen sich später als nicht mehr brauchbar. Außerdem wurde der Best Response Agent erst in der Mitte der Bearbeitungszeit erfolgreich umgesetzt, sodass Fehler erst spät festgestellt wurden und Optimierungen warten mussten. Gegen Ende der Arbeit wurden fortlaufend neue interessante Bereiche entdeckt, die man hätte untersuchen können. Jedoch reichte die Zeit nicht mehr aus, diese erfolgreich umzusetzen und anschließend zu dokumentieren. Mit einer besseren zeitlichen Planung hätte ein breiteres und besser ausgewähltes Spektrum an Themen rund um Counterfactual Regret Minimization gewählt werden können. Außerdem hätte die Entdeckung eines Kurses wie *Computational Game Solving* zu einem frühen Zeitpunkt massiv beim Setzen des Rahmens geholfen.

Wenn diese Arbeit erneut geschrieben würde, wäre ein sinnvoller Ablauf: intensive Literaturrecherche, Rahmen festlegen, Implementierung, anschließend Dokumentation. Die Vermischung der Phasen hat nur Probleme bereitet.

In diesem Kapitel wurden der Entwicklungsverlauf und die aufgetretenen Probleme dargestellt. Außerdem wurden Verbesserungsmöglichkeiten der Arbeitsweise für künftige wissenschaftliche Arbeiten festgehalten.

Kapitel 6

Evaluation

In diesem Kapitel werden zunächst die Rahmenbedingungen für die erreichten Ergebnisse vorgestellt. Anschließend werden die Ergebnisse mittels Exploitability/Iteration und Exploitability/Zeit Graphen dargestellt. Hierbei werden die Unterschiede zwischen den Algorithmen sowie den Implementierungsansätzen hervorgehoben.

6.1 Versuchsaufbau

Die Evaluation umfasst die in Kapitel 3 beschriebenen Solver-Varianten CFR, CFR+ und Discounted CFR, jeweils mit den Implementierungsansätzen Dynamic, Rekursiver Tree-Ansatz und Layer-basiert. Zusätzlich wird CFR mit alternierenden Updates betrachtet. Discounted CFR wird dabei mit der Standardparameterkonfiguration DCFR_{3/2,0,2} ($\alpha = 3/2$, $\beta = 0$, $\gamma = 2$) verwendet. Diese verschiedenen Algorithmen in Kombination mit den Implementierungsansätzen werden auf die in Kapitel 3 vorgestellten Pokervarianten angewendet: Kuhn Poker, Leduc Hold'em, Twelve Card Poker und Small Island Hold'em.

Als primäre Evaluationsmetrik wird die Exploitability zur Bewertung der Strategiequalität verwendet. Als sekundäre Metrik wird die Trainingszeit betrachtet, um die Effizienz der verschiedenen Implementierungsansätze zu vergleichen. In der Literatur gilt eine Strategie als hinreichend gelöst, wenn die Exploitability unter 1 mb/g (Milli-Blinds pro Spiel) liegt [Bowling et al., 2015]. Das Training wurde mit einem Abbruchkriterium von 1 mb/g durchgeführt. Während des Trainings wurde die Strategie regelmäßig evaluiert, um den Konvergenzverlauf zu dokumentieren. Die Evaluationsfrequenz ist an die Spielgröße angepasst, da die Best-Response-Berechnung bei großen Spielen einen erheblichen Zeitaufwand erfordert. Es ist zu beachten, dass die finalen gemessenen Exploitability-Werte unter dem Abbruchkriterium liegen können, da die Strategie nicht in jeder Iteration, sondern nur in regelmäßigen Abständen evaluiert wird.

In den früheren Iterationen wurde häufiger evaluiert, da hier die stärksten Änderungen zu sehen sind. In späteren Iterationen waren die Unterschiede minimal, weshalb größere Abstände zwischen den Evaluationen gewählt wurden. Die Ergebnisse werden in logarithmischer Darstellung auf beiden Achsen dargestellt, um sowohl das frühe als auch das späte Konvergenzverhalten gut sichtbar zu machen. Für jedes Spiel werden verschiedene

Graphen angefertigt. Diese verwenden grundsätzlich zwei Darstellungsformen. Exploitability über Iterationen ermöglicht den Vergleich der Algorithmen untereinander. Exploitability über Zeit veranschaulicht, wie sich die Implementierungsansätze verhalten. Zusätzlich wird für jedes Spiel ein Gesamtplot präsentiert, der alle Kombinationen aus Algorithmus und Implementierung zeigt.

Determinismus-Problem Alle evaluierten CFR-Algorithmen sind von Grund auf deterministisch und verwenden eine uniforme Initialisierung der Strategie bei Regretwerten von 0. Während der Evaluation wurde jedoch ein gravierender Fehler festgestellt, der die Reproduzierbarkeit der Ergebnisse beeinträchtigt. Da dieser Fehler zu spät festgestellt wurde, um alle Modelle neu zu trainieren, wird er im Folgenden erläutert.

Solange keine expliziten Zufallsknoten verwendet wurden, waren die Solver-Ergebnisse vollständig reproduzierbar. Mit der Einführung expliziter Zufallsknoten ergab sich folgendes Problem: In den verwendeten Spielumgebungen wird bei jedem Aufruf von `reset()` das Deck neu gemischt. Dadurch ändert sich die Reihenfolge, in der die Zufallsknoten verarbeitet werden, und damit auch die Reihenfolge, in der Informationssets besucht werden. In exakter Arithmetik wäre die Reihenfolge der Additionen unerheblich (Assoziativgesetz). Bei Gleitkommaarithmetik gilt dies jedoch nicht. Die Reihenfolge der Additionen beeinflusst die Rundung bei jedem Schritt. Eine unterschiedliche Update-Reihenfolge führt daher zu leicht unterschiedlich akkumulierten Strategie- und Regretwerten. Über viele Iterationen können diese Abweichungen messbar werden.

Dieser Fehler betrifft am stärksten die dynamischen Implementierungen, da hier nach jeder CFR-Iteration die `reset`-Funktion des Game-Objekts aufgerufen wird. Dadurch ist jeder einzelne Trainingslauf unterschiedlich.

Bei den beiden Varianten mit statischem Spielbaum tritt die Abweichung lediglich beim Konstruieren des Spielbaums auf. Das heißt, jeder einzelne Build eines Spielbaums liefert andere Ergebnisse in der Evaluation. Nach Feststellung des Fehlers wurde ein fester Seed gesetzt, um fortan vollständig reproduzierbare Ergebnisse zu gewährleisten.

Alle Experimente wurden auf einem Apple M1 Pro mit 16 GB RAM durchgeführt.

6.2 Ergebnisse

In diesem Abschnitt folgen die Evaluationsdiagramme, welche vorher im Versuchsaufbau besprochen wurden.

6.2.1 Kuhn Poker

Abbildung 6.1 zeigt den Vergleich der verschiedenen CFR-Algorithmen hinsichtlich ihrer Konvergenzgeschwindigkeit über die Anzahl der Iterationen. Um eine Exploitability von unter 0.9 mb/g zu erreichen, benötigt CFR+ 91 Iterationen (finale Exploitability: 0.640 mb/g), Discounted CFR benötigt 211 Iterationen (finale Exploitability: 0.876 mb/g), während Vanilla CFR mit alternierenden Updates 911 Iterationen benötigt (finale Exploitability: 0.783 mb/g). Vanilla CFR mit simultanen Updates erreicht auch nach 50.000 Iterationen keine Exploitability von unter 0.9 mb/g (finale Exploitability: 0.999 mb/g).

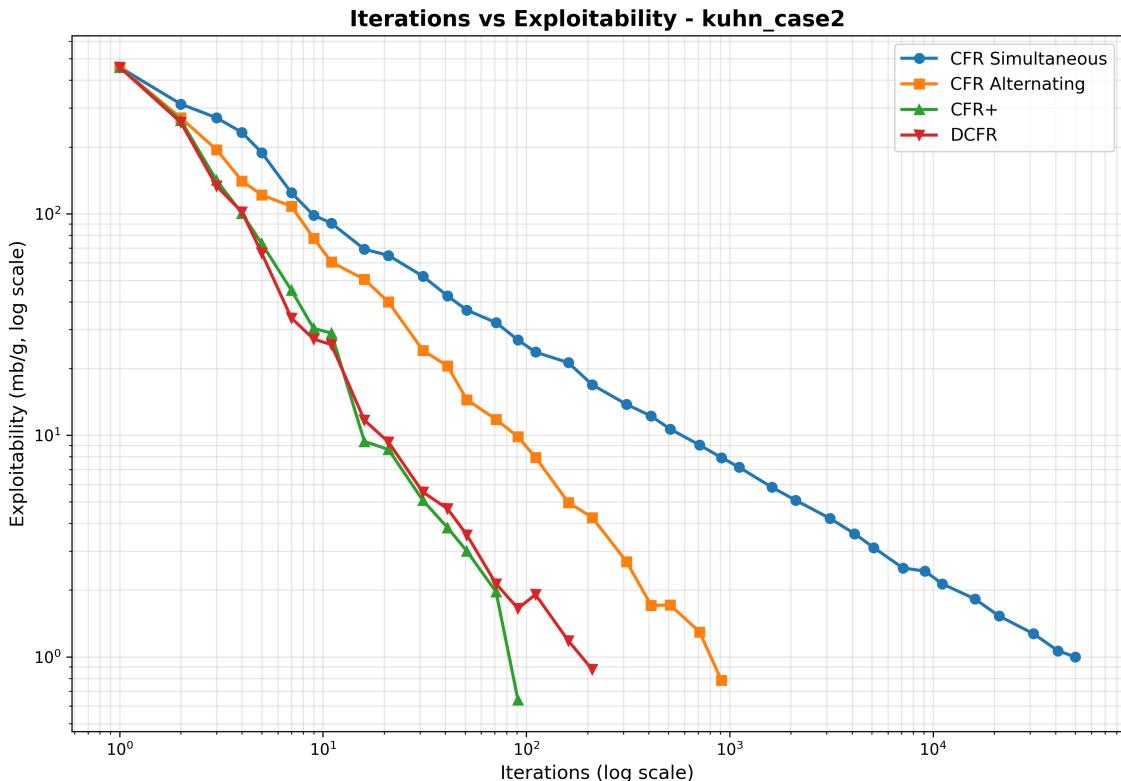


Abbildung 6.1: Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Kuhn Poker

Abbildung 6.2a vergleicht die drei Implementierungsansätze hinsichtlich ihrer Zeiteffizienz. Um eine Exploitability von unter 0.9 mb/g zu erreichen, benötigt der Rekursive Tree-Ansatz 0.09 Sekunden, der Dynamic Ansatz 0.26 Sekunden und der Layer-basierte Ansatz 0.44 Sekunden. Bei dieser kleinen Spielgröße ist der Rekursive Tree-Ansatz am schnellsten, während der Layer-basierte Ansatz aufgrund des Overheads der Layer-basierten Traversierung am langsamsten ist.

Abbildung 6.2b zeigt einen Gesamtvergleich aller evaluierten Kombinationen aus Algorithmus und Implementierung für Kuhn Poker. Die Trainingszeiten reichen von 0.01 Sekunden bis zu 13.34 Sekunden, wobei die Unterschiede zwischen den Algorithmen deutlich größer sind als zwischen den Implementierungen.

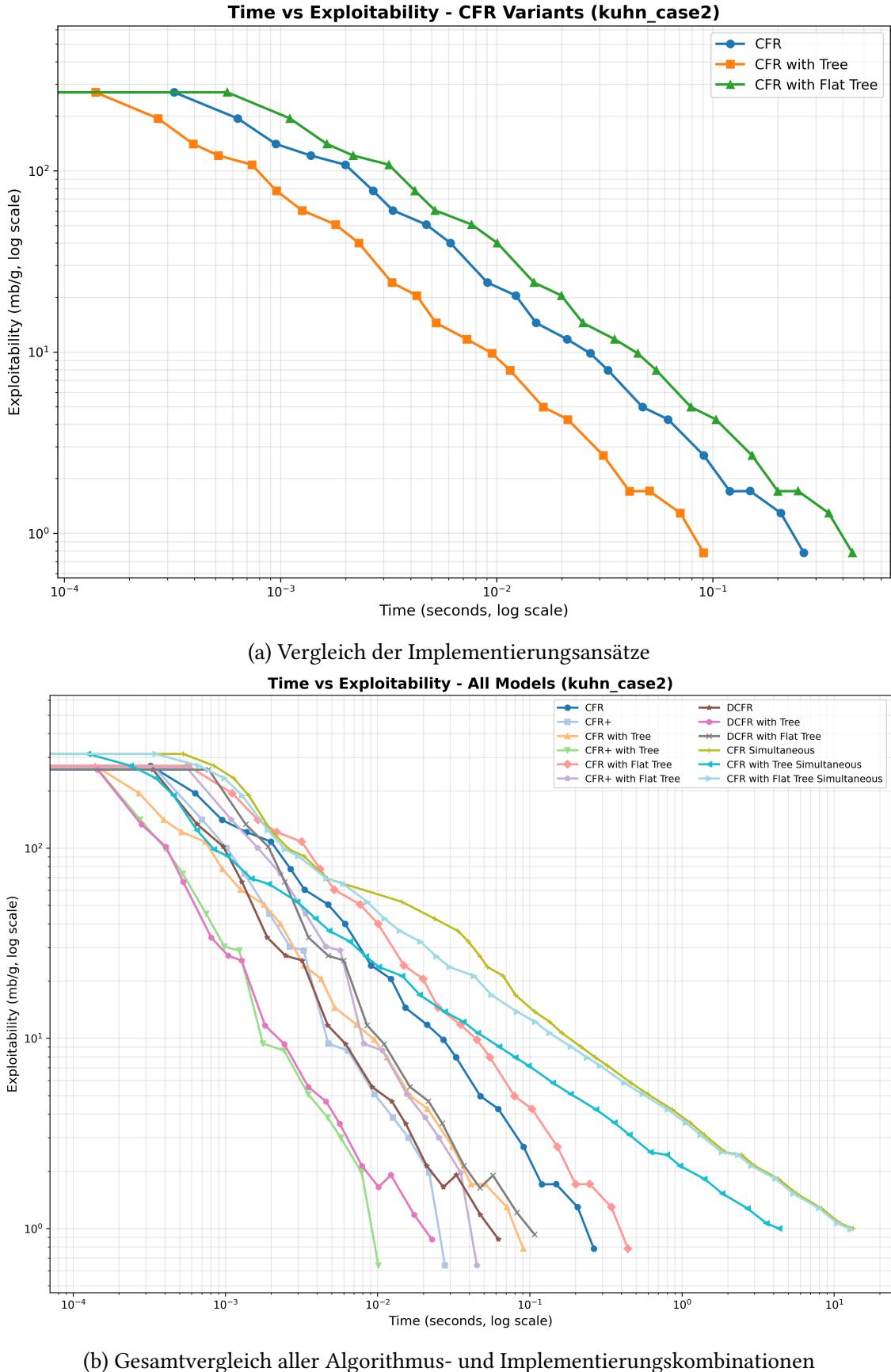


Abbildung 6.2: Vergleich der Implementierungsansätze und Gesamtvergleich aller Kombinationen bei Kuhn Poker

6.2.2 Leduc Hold'em

Abbildung 6.3 zeigt den Vergleich der verschiedenen CFR-Algorithmen hinsichtlich ihrer Konvergenzgeschwindigkeit über die Anzahl der Iterationen. Um eine Exploitability von unter 1 mb/g zu erreichen, benötigt Discounted CFR 411 Iterationen (finale Exploitability: 0.836 mb/g), CFR+ benötigt 511 Iterationen (finale Exploitability: 0.813 mb/g), während Vanilla CFR mit alternierenden Updates 31.111 Iterationen benötigt (finale Exploitability: 0.820 mb/g). Vanilla CFR mit simultanen Updates benötigt 911.111 Iterationen (finale Exploitability: 0.967 mb/g). Hier sieht man deutlich, welchen enormen Unterschied alternierende Updates bei einem relativ kleinen Spiel wie Leduc schon machen können. Aus diesem Grund werden simultane Updates in den nächstgrößeren Varianten nicht mehr evaluiert.

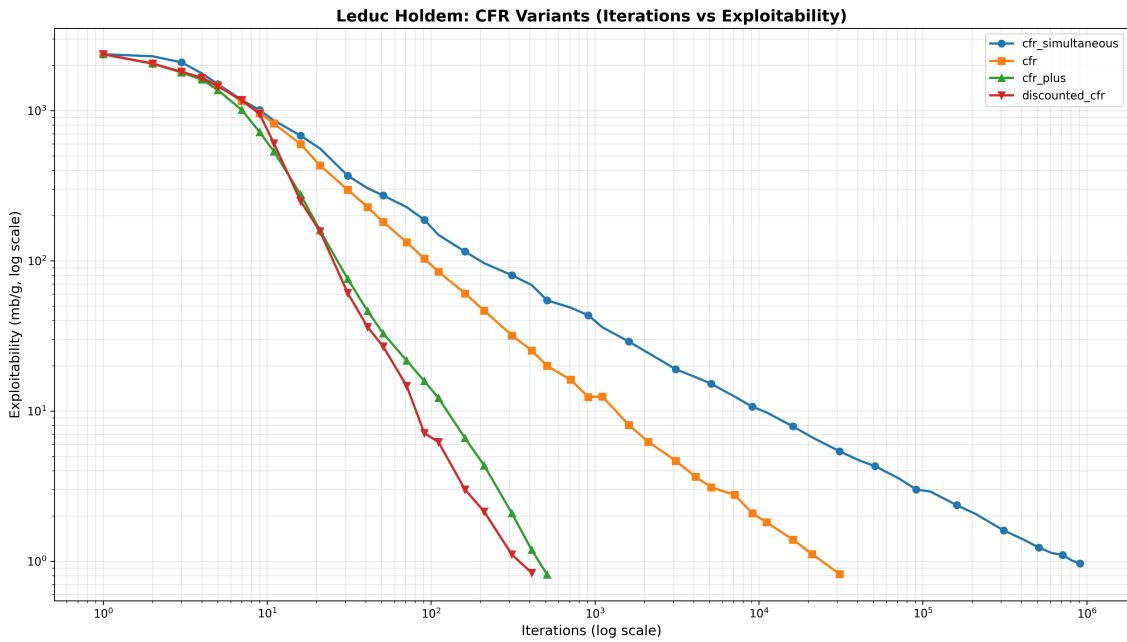


Abbildung 6.3: Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Leduc Hold'em

Abbildung 6.4a vergleicht die drei Implementierungsansätze hinsichtlich ihrer Zeiteffizienz. Um eine Exploitability von unter 1 mb/g zu erreichen, benötigt der Layer-basierte Ansatz 1.42 Minuten, der Rekursive Tree-Ansatz 1.85 Minuten und der Dynamic Ansatz 6.94 Minuten. Bei dieser Spielgröße zeigt sich, dass die Wahl des Algorithmus einen größeren Einfluss auf die Trainingszeit hat als der Implementierungsansatz.

Abbildung 6.4b zeigt einen Gesamtvergleich aller evaluierten Kombinationen aus Algorithmus und Implementierung für Leduc Hold'em. Die Trainingszeiten reichen von wenigen Sekunden bis zu mehreren Stunden, wobei die Unterschiede zwischen den Algorithmen deutlich größer sind als zwischen den Implementierungen.

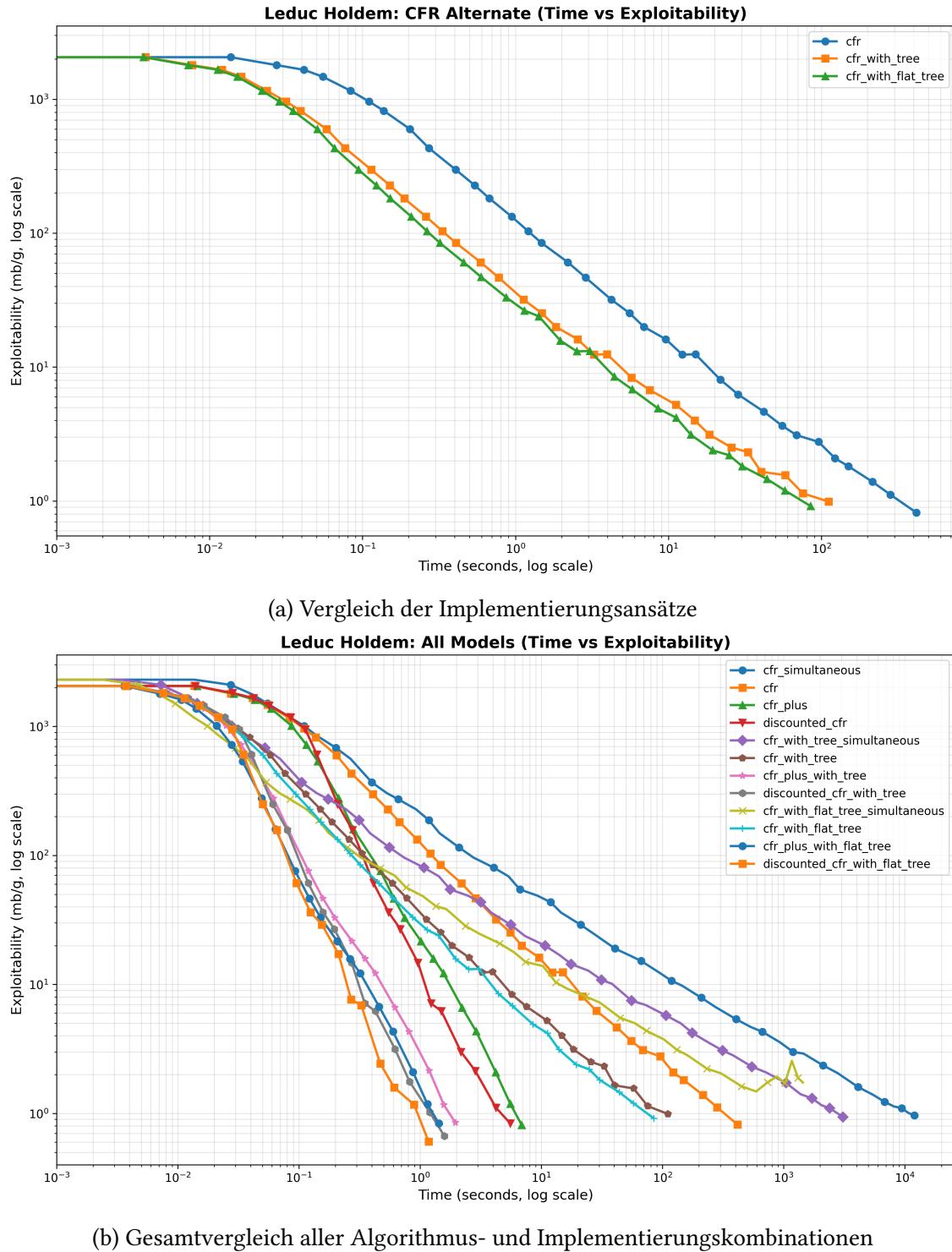


Abbildung 6.4: Vergleich der Implementierungsansätze und Gesamtvergleich aller Kombinationen bei Leduc Hold'em

6.2.3 Twelve Card Poker

Bei Twelve Card Poker wurden nur der Rekursive Tree-Ansatz und der Layer-basierte Ansatz evaluiert. Abbildung 6.5 zeigt den Vergleich der verschiedenen CFR-Algorithmen hinsichtlich ihrer Konvergenzgeschwindigkeit über die Anzahl der Iterationen für den Layer-basierten Ansatz. Um eine Exploitability von unter 1 mb/g zu erreichen, benötigt Discounted CFR 911 Iterationen (finale Exploitability: 0.839 mb/g), CFR+ benötigt 1611 Iterationen (finale Exploitability: 0.941 mb/g), während Vanilla CFR 161.111 Iterationen benötigt (finale Exploitability: 0.818 mb/g).

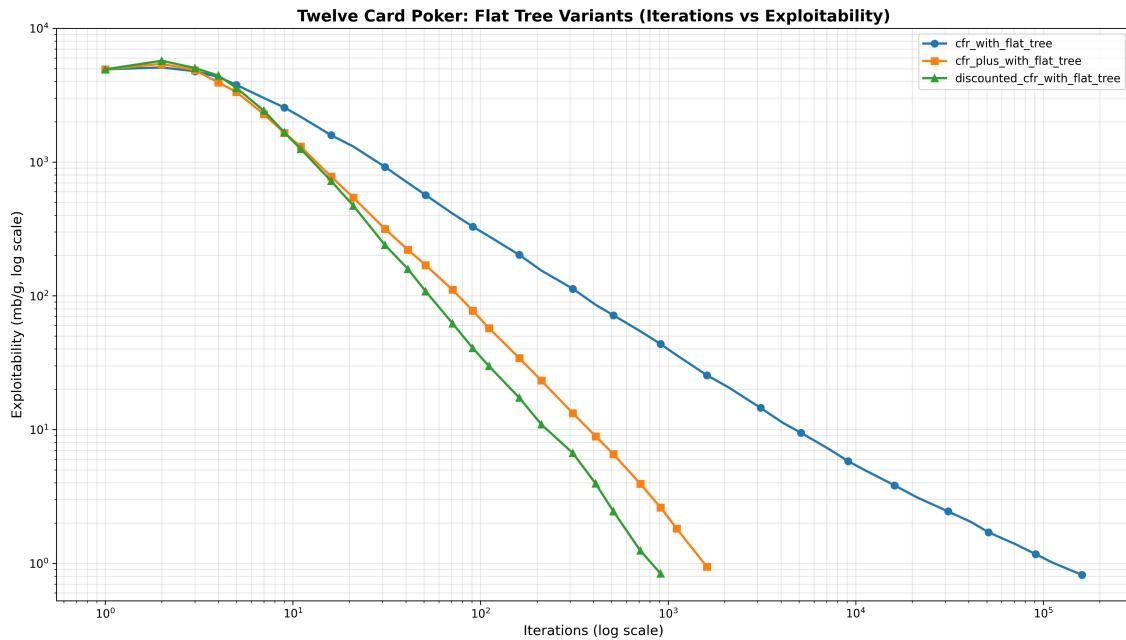


Abbildung 6.5: Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Twelve Card Poker (Layer-basierter Ansatz)

Abbildung 6.6a vergleicht die beiden Implementierungsansätze hinsichtlich ihrer Zeiteffizienz für Vanilla CFR. Um eine Exploitability von unter 1 mb/g zu erreichen, benötigt der Layer-basierte Ansatz 2.46 Stunden, der Rekursive Tree-Ansatz 9.33 Stunden. Bei dieser Spielgröße zeigt sich, dass der Layer-basierte Ansatz deutlich schneller ist als der Rekursive Tree-Ansatz.

Abbildung 6.6b zeigt einen Gesamtvergleich aller evaluierten Kombinationen aus Algorithmus und Implementierung für Twelve Card Poker. Die Trainingszeiten reichen von Sekunden bis zu mehreren Stunden, wobei die Unterschiede zwischen den Algorithmen deutlich größer sind als zwischen den Implementierungen.

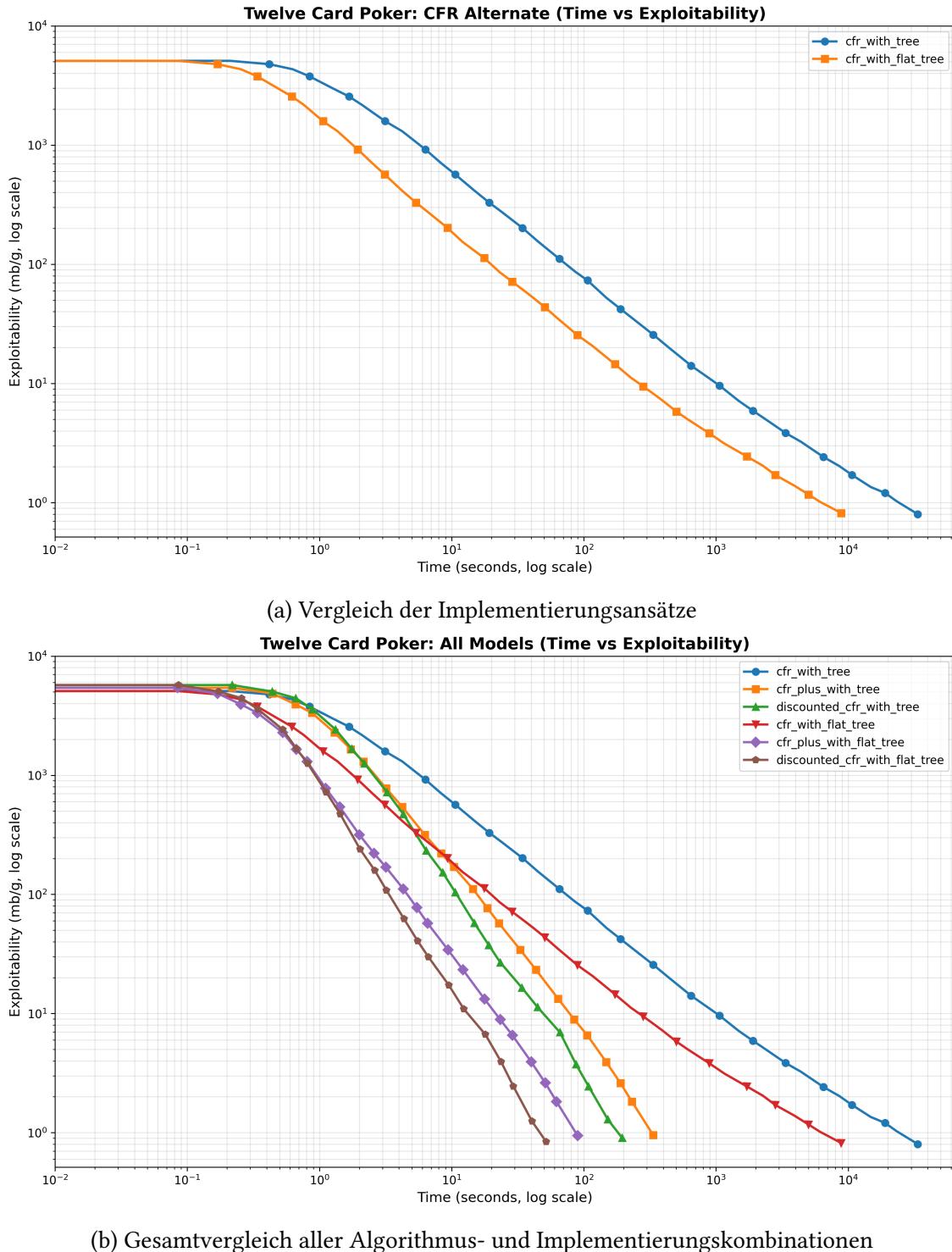


Abbildung 6.6: Vergleich der Implementierungsansätze und Gesamtvergleich aller Kombinationen bei Twelve Card Poker

6.2.4 Small Island Hold'em

Bei Small Island Hold'em wurden nur die Layer-basierten Implementierungen evaluiert, da die Rekursive Tree Implementierung aufgrund von Speicherproblemen nicht mehr praktikabel ist. Abbildung 6.7 zeigt den Algorithmusvergleich. Um eine Exploitability von unter 1 mb/g zu erreichen, benötigt Discounted CFR 381 Iterationen in 1.75 Stunden (finale Exploitability: 0.609 mb/g), CFR+ benötigt 781 Iterationen in 3.58 Stunden (finale Exploitability: 0.891 mb/g). Vanilla CFR mit alternierenden Updates erreicht auch nach 10.000 Iterationen beziehungsweise 45.66 Stunden noch keine Exploitability von unter 1 mb/g und verbleibt bei 1.94 mb/g.

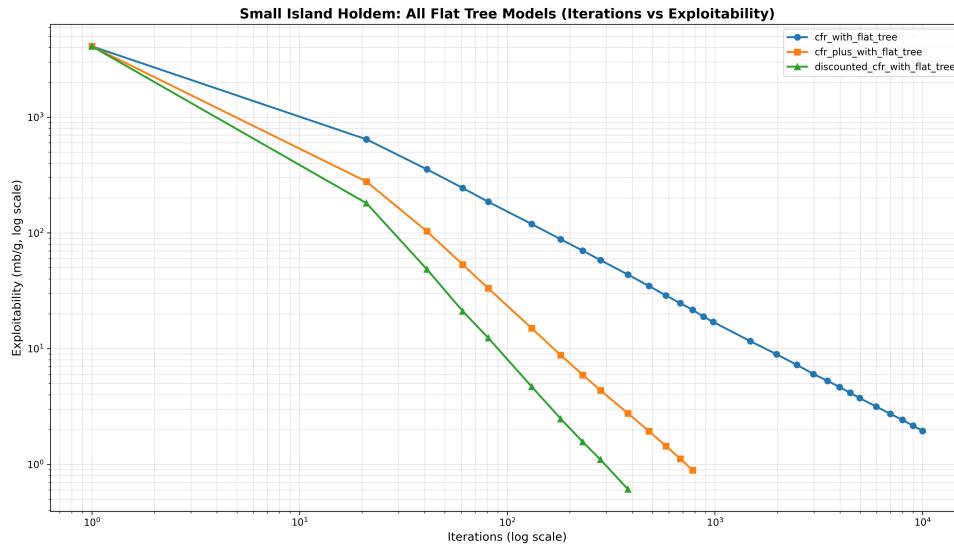


Abbildung 6.7: Konvergenzverhalten der verschiedenen CFR-Algorithmen bei Small Island Hold'em

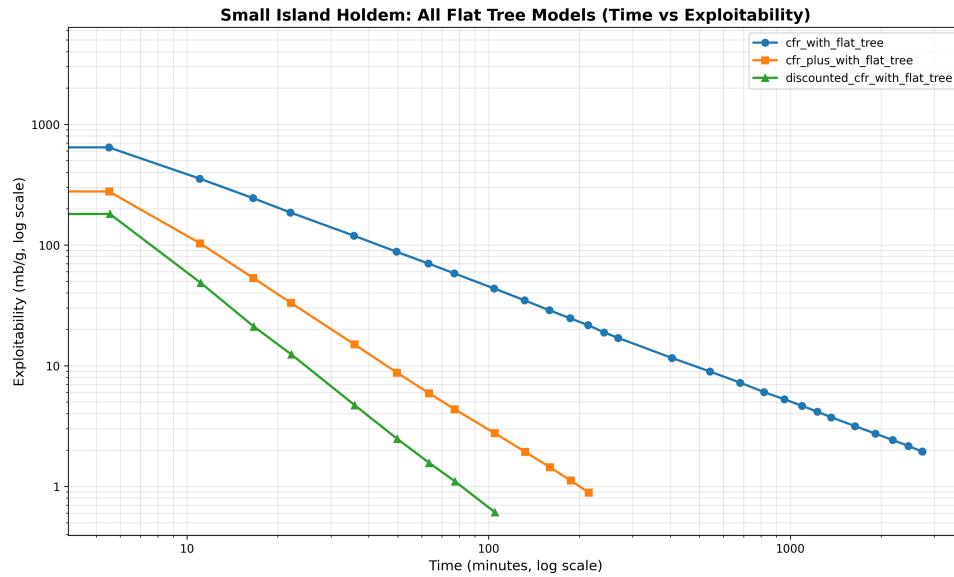


Abbildung 6.8: Trainingszeit der verschiedenen CFR-Algorithmen bei Small Island Hold'em

6.3 Diskussion

Bei kleinen Spielen wie Kuhn Poker sind die Unterschiede zwischen den Algorithmen noch nicht so klar sichtbar wie bei größeren Varianten. Der Unterschied zwischen CFR mit simultanen und alternierenden Updates zeigt sich jedoch schon stark. Der Layer-basierte Ansatz schneidet wegen Programmier-Overhead sogar schlechter ab als die anderen Ansätze.

Bei Leduc Hold'em werden die Unterschiede zwischen DCFR/CFR+ und Vanilla CFR deutlich sichtbar. Außerdem hält der Layer-basierte Ansatz ab hier Schritt mit dem rekursiven Tree.

Ab Twelve Card Poker zeigt sich der Vorteil der Layer-basierten Traversierung deutlich. Der dynamische Ansatz wird hier aus Laufzeitgründen bereits nicht mehr evaluiert, genauso wie CFR mit simultanen Updates.

Ab Small Island Hold'em wurde aufgrund von Speicherproblemen nur noch der Flat Tree-Ansatz evaluiert. Standard CFR mit alternierenden Updates ist auch hier noch in der Lage, eine hinreichende Strategie zu finden, jedoch ist fraglich, ob 45 Stunden bis zu einer Exploitability von knapp 2 mb/g noch als praktikabel gelten kann.

Die Verhältnisse der benötigten Iterationen relativ zu Vanilla CFR mit alternierenden Updates (1,0) variieren über die Spielgrößen:

Kuhn Poker: Discounted CFR 23% (211 zu 911), CFR+ 10% (91 zu 911).

Leduc Hold'em: Discounted CFR 1,3% (411 zu 31.111), CFR+ 1,6% (511 zu 31.111).

Twelve Card Poker: Discounted CFR 0,6% (911 zu 161.111), CFR+ 1,0% (1611 zu 161.111).

Small Island Hold'em: Discounted CFR 3,81% (381 zu mindestens 10.000), CFR+ 7,81% (781 zu mindestens 10.000). Die Small Island Hold'em Werte sind allerdings nicht repräsentativ, da Vanilla CFR nach 10.000 Iterationen und einer Exploitability von knapp 2 mb/g abgebrochen wurde. Dies geschah unabsichtlich und es gab keine Zeit, das Modell neu zu trainieren. Es ist jedoch zu erwarten, dass die Reduktion von knapp 2 mb/g auf unter 1 mb/g noch eine erhebliche Zeitspanne erfordert hätte.

Über die größer werdenden Pokerspiele bleibt das Verhältnis von CFR mit alternierenden Updates und CFR+ nahezu identisch. Die Performance von DCFR schwankt mehr und ist abhängiger vom Spieltyp.

Ein weiterer Weg, die Qualität einer Strategie zu evaluieren, ist persönlich dagegen zu spielen. Dies ist mit dem in Abschnitt 4.3.2 beschriebenen GUI möglich. Eine genaue Anleitung hierfür findet sich im README des GitHub-Repositories.

In diesem Kapitel wurden die erreichten Ergebnisse mithilfe von Exploitability-Diagrammen dargestellt, welche das Konvergenzverhalten der einzelnen Algorithmen und Implementierungsansätze zeigen. Im nächsten Kapitel werden diese Ergebnisse im Hinblick auf die zu Beginn formulierten Forschungsfragen bewertet.

Kapitel 7

Fazit

7.1 Zielerreichung und Limitationen

In diesem Abschnitt werden die Ergebnisse der Arbeit zusammengefasst und im Hinblick auf die Zielsetzung und Fragestellungen aus Kapitel 1.2 eingeordnet.

Es wurden mehrere Pokervarianten in einer einheitlichen Struktur implementiert. Darauf aufbauend wurden mehrere CFR-Varianten implementiert und in verschiedenen Formen realisiert. Zur Bewertung der Strategien wurde die Exploitability über einen Best-Response-Agent auf Basis eines Public State Trees berechnet. Es wurde eine vollständige Pipeline von der Spiellösung bis hin zur Evaluation umgesetzt.

Ein geplantes Ziel wurde nicht erreicht: Im Exposé war vorgesehen, neben CFR+ auch Sampling-basierte Verfahren (MCCFR) praktisch zu vergleichen. Diese Algorithmen konnten nicht evaluiert werden, hierfür wurde aber der Algorithmus DCFR als Vergleich hinzugezogen. Die Gründe hierfür wurden bereits in Kapitel 5 erläutert.

Die drei Forschungsfragen aus Kapitel 1.2 können anhand der erreichten Ergebnisse beantwortet werden:

1. **Algorithmusvergleich:** Wie unterscheiden sich die verschiedenen CFR-Algorithmen in ihrem Konvergenzverhalten bei steigender Spielgröße, und ab welcher Spielgröße ist Vanilla CFR nicht mehr praktikabel?

Die Unterschiede im Konvergenzverhalten der untersuchten Algorithmen bleiben bei steigender Spielgröße auf einer logarithmischen Skala ähnlich. Bei Small Island Hold'em ist Vanilla CFR mit alternierenden Updates nicht mehr praktikabel, da selbst nach 45 Stunden Training keine hinreichend niedrige Exploitability erreicht werden konnte.

2. **Implementierungsoptimierungen:** Welche Implementierungsoptimierungen sind erforderlich, damit das Training bei wachsender Spielgröße praktikabel bleibt?

Um das Training bei wachsender Spielgröße praktikabel zu halten, sind verschiedene Implementierungsoptimierungen erforderlich. Ein wichtiger Aspekt ist der Umstieg auf speichereffiziente Datenstrukturen, wie beispielsweise die Verwendung von NumPy-Arrays statt Python-Objekten. Zudem wird irgendeine Form der

parallelen Berechnung erforderlich. In dieser Arbeit wurde die Layer-basierte Traversierung gewählt, welche eine gesamte Schicht gleichzeitig durch Vektoroperationen parallel verarbeitet. Diese Optimierungen ermöglichen es, größere Spiele zu lösen, jedoch wird jede Methode, die einen statischen Spielbaum benutzt, bei wachsender Spielgröße irgendwann durch den Speicherverbrauch beschränkt.

3. **Limitationen:** Durch welche Komponenten wird die geplante Implementierung limitiert?

Die Skalierungsgrenzen sind je nach Implementierungsansatz unterschiedlich. Die dynamische Implementierung ist durch ihre Laufzeit begrenzt. Die Layer-basierte Tree-Implementierung ist durch den Speicherverbrauch begrenzt. Ein weiterer Bottleneck, der bei weiterer Skalierung relevant wird, ist der Best Response Agent. Dies betrifft zum einen den Speicherbedarf des Public State Trees sowie die Laufzeit.

Diese Arbeit bietet einen fundierten Einstieg in das Thema „Computational Game Solving“ mittels Counterfactual Regret Minimization. Es wurde gezeigt, wie man extensive Spiele mit imperfekter Information mit spieltheoretischen Algorithmen löst. Darüber hinaus wurden die limitierenden Faktoren für eine weitere Skalierung der Spielgröße herausgearbeitet. Im folgenden Ausblick wird besprochen, welche Maßnahmen getroffen werden müssen, um die Implementierung weiter zu skalieren.

7.2 Ausblick

In diesem Abschnitt werden konkrete nächste Schritte skizziert, um die Implementierung weiter zu skalieren. Hierbei wird zwischen Implementierungsansätzen und Algorithmischen Ansätzen unterschieden.

7.2.1 Implementierungsoptimierungen

Auf der Implementierungsseite ist ein Ansatz, der einen statischen Spielbaum aufbaut, bei Spielen höherer Komplexität nicht mehr praktikabel. Daher ist ab einem bestimmten Komplexitätsgrad ein Umstieg auf eine dynamische Implementierung erforderlich. Um dies praktikabel zu machen, muss die Spielumgebung grundlegend überarbeitet und auf Laufzeiteffizienz optimiert werden. Eine weitere Optimierung wäre eine parallelisierte Berechnung auf einem dynamischen Baum. Zusätzlich wäre es sinnvoll, einen Sampling-basierten Ansatz zu verwenden, sobald ein dynamischer Ansatz verfolgt wird. Hierfür müsste eine passende Evaluation implementiert werden.

Ein weiterer interessanter Ansatz wäre die Verwendung anderer Programmiersprachen wie C++ oder Rust. Da diese Sprachen näher an der Maschinensprache sind, könnte damit eine erhebliche Laufzeitverbesserung erreicht werden. Auf die Speicherprobleme der Ansätze mit statischem Spielbaum hätte dies jedoch voraussichtlich keine Auswirkung.

Die Evaluation mittels des Best Response Agents kann weiter optimiert werden, indem die noch nicht umgesetzten Optimierungen aus Kapitel 2.5 implementiert werden. Außerdem sollte der Best Response Agent genauso wie die Game-Environment auf effizientere Datenstrukturen umgestellt werden.

Bei einer Layer-basierten Implementierung bietet sich die GPU-Optimierung an, die in [Kim, 2024, Reis, 2015] beschrieben wird. Die bereits verwendeten NumPy-Arrays könnten dabei durch GPU-Tensoren ersetzt werden, was eine erhebliche Beschleunigung der Berechnungen ermöglichen würde.

7.2.2 Algorithmische Optimierungen

Bei den algorithmischen Optimierungen gibt es zahlreiche Möglichkeiten.

Eine erste Maßnahme wäre die Verwendung von Abstraktionstechniken, um die Größe des Spielbaums zu reduzieren. Bei der Abstraktion wird eine strategische Approximation des vollständigen Spiels erstellt, indem ähnliche Informationssets zu abstrakten Informationssets zusammengefasst werden. Dadurch wird die Anzahl der zu betrachtenden Zustände erheblich reduziert. Dies hilft sowohl allen Tree-basierten Ansätzen, da dadurch der abstrahierte Baum des nächstgrößeren Spiels in den Speicher passen könnte, als auch der Laufzeit pro Iteration, da nicht so viele Zustandsknoten besucht werden müssen. Relevante Literatur hierzu ist [Gilpin and Sandholm, 2007, Billings et al., 2003, Brown et al., 2015, Kroer and Sandholm, 2018].

Eine zweite Optimierung wäre das Einbauen verschiedener Arten von Pruning. Beim Pruning werden bestimmte Teilbäume nicht weiter besucht, sobald bestimmte Umstände eingetroffen sind. Bei Partial Pruning werden Updates für einen Spieler übersprungen, wenn der Gegner den entsprechenden Knoten mit Wahrscheinlichkeit null erreicht. Eine weiterentwickelte Variante ist Regret-Based Pruning, bei der Pfade übersprungen werden können, wenn einer der Spieler Aktionen mit Wahrscheinlichkeit null wählt, unabhängig davon, welcher Spieler am Zug ist. Die Anzahl der Iterationen, während derer eine Aktion gepruned werden kann, hängt dabei vom negativen Regret dieser Aktion ab. Relevante Literatur hierzu ist [Brown and Sandholm, 2015, Brown and Sandholm, 2017b].

Eine weitere Optimierung wäre die Verwendung von Deep CFR [Brown et al., 2019]. Deep CFR verwendet Funktionenapproximation mit tiefen neuronalen Netzen, um das Verhalten von tabularem CFR zu approximieren. Anstatt die Strategie für jedes Informationsset explizit zu speichern, wird sie durch ein neuronales Netz dargestellt. Dies ermöglicht es, auch für sehr große Spiele Strategien zu speichern, die nicht mehr vollständig im Speicher gehalten werden können.

Eine weitere Optimierung wäre Subgame Solving [Brown and Sandholm, 2018] [Brown and Sandholm, 2019b]. Im Gegensatz zu perfekt-information games kann bei imperfect-information games ein Teilspiel nicht isoliert gelöst werden, da die optimale Strategie davon abhängt, wie in anderen Teilen des Spiels gespielt wird. Beim Subgame Solving wird daher zuerst eine grobe Strategie für das gesamte Spiel berechnet. Während des Spiels wird dann für jedes erreichte Teilspiel eine detailliertere Strategie berechnet, die mit der groben Gesamtstrategie kompatibel ist.

Anhang A

Pokerhände-Übersicht

Royal flush*	A♦	K♦	Q♦	J♦	10♦
Straight flush*	J♣	10♣	9♣	8♣	7♣
Four of a kind	5♣	5♦	5♥	5♠	2♦
Full house	6♠	6♥	6♦	K♣	K♦
Flush*	J♦	9♦	8♦	4♦	3♦
Straight*	10♦	9♠	8♥	7♦	6♣
Three of a kind	Q♦	Q♣	Q♥	9♥	2♠
Two pair	J♦	J♣	3♣	3♠	2♥
One pair	10♠	10♥	8♠	7♥	4♣
High card	K♦	Q♦	7♠	4♠	3♦

Abbildung A.1: Übersicht der Pokerhände und deren Rangordnung, https://en.wikipedia.org/wiki/List_of_poker_hands

Anhang B

Spieldefinitionen

B.0.1 Kuhn Poker

Die einfachste hier betrachtete Pokervariante ist One-Card-Poker, auch Kuhn Poker genannt. Der Name geht auf Harald Kuhn zurück, der das Spiel 1951 in "Simplified Two-Person Poker „beschrieb [Kuhn, 1950]. Es wird mit drei Karten gespielt. Kuhn notiert sie als 1, 2 und 3 (1 schlechteste, 3 beste Karte); in dieser Arbeit wird J, Q, K verwendet (entsprechend den Rängen eines Standard-Kartenspiels).

Zu Beginn zahlen beide einen Ante von einem Chip; jeder erhält eine Hole Card, keine Board-Karten. Es folgt eine einzige Setzrunde mit Setzlimit 1 Chip pro Raise. Im Showdown gewinnt die höhere Karte ($J < Q < K$).

Kuhn beschreibt weitere Versionen über das Verhältnis von Ante und Setzgröße. Der hier zugrunde gelegte Fall (Ante = Setzgröße) entspricht Kuhns Fall 2; in Fall 1 ist die Setzgröße kleiner als der Ante, in Fall 3 und 4 größer (bis zum Doppelten des Antes).

Kuhn Poker hat 58 Knoten und 12 Informationsets.

B.0.2 Leduc Hold'em

Leduc Hold'em [Southey et al., 2005] ist eine vereinfachte Hold'em-Variante für zwei Spieler.

Es wird mit sechs Karten gespielt, die drei Ränge (J, Q, K) umfassen, wobei jeder Rang doppelt vorhanden ist. Das Spiel besteht aus zwei Runden: In der ersten Runde erhält jeder Spieler eine Hole Card, in der zweiten Runde wird eine Board-Karte aufgedeckt.

Zu Beginn zahlen beide Spieler einen Ante von 1 Chip. Pro Runde sind maximal zwei Bets erlaubt. Die Setzgrößen betragen 2 Chips in der ersten Runde und 4 Chips in der zweiten Runde.

Mögliche Hände sind Paar oder High Card, wobei ein Paar eine High Card schlägt.

Leduc Hold'em hat 1939 Knoten und 288 Informationsets.

B.0.3 Zwölf-Karten-Poker

Diese Variante wurde eigens als Zwischenstufe zwischen Leduc Hold'em und Rhode Island Hold'em entwickelt.

Es wird ein Deck aus zwölf Karten verwendet (J, Q, K, A jeweils dreimal). In Runde 1 erhält jeder eine Hole Card; in Runde 2 und 3 wird je eine Board-Karte aufgedeckt. Zu Beginn zahlen beide einen Ante von 1 Chip; pro Runde sind maximal zwei Bets erlaubt. Setzgrößen: 2 Chips (Runde 1), 4 Chips (Runde 2), 8 Chips (Runde 3).

Das Spiel endet nach Runde 3 mit Showdown. Eine Hand bildet sich aus der Hole Card und den beiden Board-Karten. Hand-Rankings: Three of a Kind schlägt Paar, Paar schlägt High Card.

Zwölf-Karten-Poker hat 99 545 Knoten und 10 104 Informationsets.

B.0.4 Small Island Hold'em

Small Island Hold'em ist eine für diese Arbeit entwickelte Vereinfachung von Rhode Island Hold'em.

Es wird ein reduziertes Deck mit 20 Karten verwendet (Ränge T, J, Q, K, A; in vier verschiedenen Suits). Zu Beginn zahlen beide einen Ante von 5 Chips und jeder erhält eine Hole Card. Das Spiel hat drei Runden mit Flop (1 Board-Karte nach Runde 1) und Turn (1 Board-Karte nach Runde 2). Pro Runde sind maximal drei Raises erlaubt. Die Setzgrößen sind 10 Chips (Runde 1) bzw. 20 Chips (Runde 2 und 3).

Beim Showdown bildet jeder die beste 3-Card-Poker-Hand aus Hole Card und den beiden Board-Karten. Die Hand-Rankings entsprechen Rhode Island Hold'em: Straight Flush > Three of a Kind > Straight > Flush > Pair > High Card. Bei Gleichstand wird der Pot geteilt.

Small Island Hold'em hat 44 123 721 Knoten und 1 037 520 Informationsets.

B.0.5 Rhode Island Hold'em

Rhode Island Hold'em [Shi and Littman, 2001] wurde von Shi und Littman (2001) als Testbed für KI-Forschung entwickelt. Das Spiel ist auch hier [Sandholm and Gilpin, 2005] definiert.

Es wird ein Standarddeck mit 52 Karten verwendet. Zu Beginn zahlen beide einen Ante von 5 Chips in den Pot und jeder erhält eine Hole Card. Das Spiel hat drei Runden mit Flop (1 Board-Karte nach Runde 1) und Turn (1 Board-Karte nach Runde 2). Pro Runde sind maximal drei Raises erlaubt. Die Setzgrößen sind 10 Chips (Runde 1) bzw. 20 Chips (Runde 2 und 3).

Beim Showdown bildet jeder die beste 3-Card-Poker-Hand aus Hole Card und den beiden Board-Karten. Die Hand-Rankings weichen von 5-Card-Poker ab: Straight Flush > Three of a Kind > Straight > Flush > Pair > High Card. Bei Gleichstand wird der Pot geteilt.

Rhode Island Hold'em hat ungefähr 3.1 Milliarden Knoten [Sandholm and Gilpin, 2005].

B.0.6 Limit Texas Hold'em

Texas Hold'em ist das verbreiteste Poker Format. Die hier gegebene Heads-Up Limit Hold'em Definition stammt aus [Southey et al., 2005].

Es wird ein Standarddeck mit 52 Karten (2 bis Ass in vier Farben) verwendet. Statt eines Ante gibt es Blinds: 5 Chips (Small Blind) und 10 Chips (Big Blind). Jeder erhält zwei Hole Cards; es folgen vier Setzrunden mit Asteilung von Flop (3 Board-Karten), Turn (1), River (1). Pro Runde sind maximal vier Bets erlaubt; Setzgrößen 10 Chips (Runde 1 und 2) bzw. 20 Chips (Runde 3 und 4).

Zulässige Hände entsprechen der üblichen 5-Card-Poker-Rangfolge (Royal Flush bis High Card).

Anhang C

M-Wert

Lanctot et al. [Lanctot et al., 2009] führen zur Formulierung der Regret-Bounds ein strukturabhängiges Maß der Spielkomplexität ein: den M-Wert. Dieser Wert hängt von der spezifischen Struktur des extensiven Spiels ab und ermöglicht eine präzisere Abschätzung der Konvergenzrate als die einfache Anzahl der Information Sets.

Sei \vec{a}_i eine Teilsequenz einer Historie, die nur die Aktionen von Spieler i in dieser Historie enthält. Sei \vec{A}_i die Menge aller solcher Aktionssequenzen von Spieler i . Für eine Aktionssequenz \vec{a}_i sei $\mathcal{I}_i(\vec{a}_i)$ die Menge aller Information Sets, bei denen die Aktionssequenz von Spieler i bis zu diesem Information Set genau \vec{a}_i ist. Der M-Wert für Spieler i ist dann definiert als

$$M_i = \sum_{\vec{a}_i \in \vec{A}_i} \sqrt{|\mathcal{I}_i(\vec{a}_i)|}. \quad (\text{C.1})$$

Der M-Wert summiert also über alle möglichen Aktionssequenzen von Spieler i die Wurzel der Anzahl der Information Sets, die zu dieser Sequenz gehören. Dies erfasst, wie sich die Information Sets über die verschiedenen Aktionssequenzen verteilen.

Für den M-Wert gilt die folgende Ungleichung:

$$\sqrt{|\mathcal{I}_i|} \leq M_i \leq |\mathcal{I}_i|, \quad (\text{C.2})$$

wobei beide Seiten dieser Schranke von bestimmten Spielen erreicht werden können ([Lanctot et al., 2009]).

Die untere Schranke $\sqrt{|\mathcal{I}_i|}$ wird erreicht, wenn alle Information Sets zu derselben Aktionssequenz gehören. Die obere Schranke $|\mathcal{I}_i|$ wird erreicht, wenn jede Aktionssequenz zu genau einem Information Set führt.

Als Beispiel dient Kuhn Poker. Beide Spieler haben jeweils sechs Information Sets: $|\mathcal{I}_0| = |\mathcal{I}_1| = 6$.

Für Spieler 0 gibt es zwei Aktionssequenzen mit Information Sets: die leere Sequenz [] mit drei Information Sets ($J, (), 0$), ($Q, (), 0$), ($K, (), 0$) am Spielbeginn, sowie die Sequenz ['check'] mit drei Information Sets ($J, ('check', 'bet'), 0$), ($Q, ('check', 'bet'), 0$), ($K, ('check', 'bet'), 0$) nach einem Check von Spieler 0 und einem Bet von Spieler 1. Somit gilt $M_0 = \sqrt{3} + \sqrt{3} = 2\sqrt{3} \approx 3.46$.

Für Spieler 1 gibt es nur eine Aktionssequenz mit Information Sets: die leere Sequenz $[]$ mit sechs Information Sets, nämlich drei nach einem Check von Spieler 0 und drei nach einem Bet von Spieler 0. Somit gilt $M_1 = \sqrt{6} \approx 2.45$.

Die Verifikation der Schranken ergibt: $\sqrt{6} \approx 2.45 \leq M_0 = 3.46 \leq 6$ und $\sqrt{6} \approx 2.45 \leq M_1 = 2.45 \leq 6$. Für Spieler 1 wird die untere Schranke erreicht, da alle sechs Information Sets zur selben Aktionssequenz gehören. Für Spieler 0 liegt M_0 zwischen den Schranken, da die Information Sets auf zwei verschiedene Aktionssequenzen verteilt sind.

Der M-Wert wird in den Regret-Bounds für Vanilla CFR und MCCFR verwendet.

Literaturverzeichnis

- [Berns, 2024] Berns, B. (2024). Nash equilibrium details for leduc hold'em. Computer Science Stack Exchange, <https://cs.stackexchange.com/questions/169593/nash-equilibrium-details-for-leduc-holdem>. Zugriff: 19.01.2026.
- [Billings et al., 2003] Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenseberg, T., and Szafron, D. (2003). Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*. PDF: <https://poker.cs.ualberta.ca/publications/IJCAI03.pdf> (Zugriff: 19.01.2026).
- [Bowling et al., 2015] Bowling, M., Burch, N., Johanson, M., and Tammelin, O. (2015). Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149. PDF: <https://www.ijcai.org/Proceedings/15/Papers/097.pdf> (Zugriff: 19.01.2026).
- [Brown et al., 2015] Brown, N., Ganzfried, S., and Sandholm, T. (2015). Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit texas hold'em agent. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*. PDF: <https://www.cs.cmu.edu/~sandholm/hierarchical.aamas15.pdf> (Zugriff: 19.01.2026).
- [Brown et al., 2019] Brown, N., Lerer, A., Gross, S., and Sandholm, T. (2019). Deep counterfactual regret minimization. *arXiv preprint arXiv:1811.00164*. arXiv:1811.00164, <https://arxiv.org/abs/1811.00164> (Zugriff: 19.01.2026).
- [Brown and Sandholm, 2015] Brown, N. and Sandholm, T. (2015). Regret-based pruning in extensive-form games. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 1972–1980. PDF: <https://www.cs.cmu.edu/~sandholm/regret-basedPruning.nips15.withAppendix.pdf> (Zugriff: 19.01.2026).
- [Brown and Sandholm, 2017a] Brown, N. and Sandholm, T. (2017a). Libratus: The superhuman ai for no-limit poker (demonstration). In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 5226–5228. PDF: <https://www.ijcai.org/proceedings/2017/0772.pdf> (Zugriff: 19.01.2026).
- [Brown and Sandholm, 2017b] Brown, N. and Sandholm, T. (2017b). Reduced space and faster convergence in imperfect-information games via pruning. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*. PDF: <https://www.cs.cmu.edu/~sandholm/cs15-888F21/pruning.icml17.pdf> (Zugriff: 19.01.2026).

- [Brown and Sandholm, 2018] Brown, N. and Sandholm, T. (2018). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424. PDF: <https://www.science.org/doi/10.1126/science.aao1733> (Zugriff: 19.01.2026).
- [Brown and Sandholm, 2019a] Brown, N. and Sandholm, T. (2019a). Solving imperfect-information games via discounted regret minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1829–1836. PDF: <https://arxiv.org/pdf/1809.04040.pdf> (Zugriff: 19.01.2026).
- [Brown and Sandholm, 2019b] Brown, N. and Sandholm, T. (2019b). Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890. PDF: <https://www.science.org/doi/10.1126/science.aay2400> (Zugriff: 19.01.2026).
- [Gilpin and Sandholm, 2007] Gilpin, A. and Sandholm, T. (2007). Lossless abstraction for imperfect information games. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 343. doi: <https://doi.org/10.1145/1284320.1284324>.
- [Herbster and Warmuth, 1998] Herbster, M. and Warmuth, M. K. (1998). Tracking the best expert. *Machine Learning*, 32(2):151–178. doi: <https://doi.org/10.1023/A:1007424614876>.
- [Johanson et al., 2011] Johanson, M., Waugh, K., Bowling, M., and Zinkevich, M. (2011). Accelerating best response calculation in large extensive games. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 258–265. PDF: <https://webdocs.cs.ualberta.ca/~bowling/papers/11ijcai-rgbr.pdf> (Zugriff: 19.01.2026).
- [Kim, 2024] Kim, J. (2024). Gpu-accelerated counterfactual regret minimization. arXiv:2408.14778, PDF: <https://arxiv.org/pdf/2408.14778.pdf> (Zugriff: 19.01.2026).
- [Kroer and Sandholm, 2018] Kroer, C. and Sandholm, T. (2018). A unified framework for extensive-form game abstraction with bounds. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. PDF: <https://proceedings.neurips.cc/paper/2018/file/aa942ab2bfa6ebda4840e7360ce6e7ef-Paper.pdf> (Zugriff: 19.01.2026).
- [Kuhn, 1950] Kuhn, H. W. (1950). A simplified two-person poker. In Kuhn, H. W. and Tucker, A. W., editors, *Contributions to the Theory of Games, Volume I*, pages 97–103. Princeton University Press. PDF: <https://sites.math.rutgers.edu/~zeilberg/akherim/PokerPapers/Kuhn1951.pdf> (Zugriff: 19.01.2026).
- [Lanctot et al., 2019] Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., et al. (2019). Openspiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453. arXiv:1908.09453, <https://arxiv.org/pdf/1908.09453.pdf> (Zugriff: 19.01.2026); Code: <https://github.com/google-deepmind/open-spiel> (Zugriff: 19.01.2026).
- [Lanctot et al., 2009] Lanctot, M., Waugh, K., Zinkevich, M., and Bowling, M. (2009). Monte carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 1078–

1086. PDF: https://proceedings.neurips.cc/paper_files/paper/2009/file/00411460f7c92d2124a67ea0f4cb5f85-Paper.pdf (Zugriff: 19.01.2026).
- [Osborne and Rubinstein, 1994] Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press, Cambridge, Massachusetts. PDF: <https://sites.math.rutgers.edu/~zeilberg/EM20/OsborneRubinsteinMasterpiece.pdf> (Zugriff: 19.01.2026).
- [Reis, 2015] Reis, J. (2015). A gpu implementation of counterfactual regret minimization. Master's thesis, University of Porto. Master's thesis, PDF: <https://repositorio-aberto.up.pt/bitstream/10216/83517/2/35409.pdf> (Zugriff: 19.01.2026).
- [Sandholm and Anagnostides, 2025] Sandholm, T. and Anagnostides, I. (2025). Cs 15-888: Computational game solving (fall 2025), course website. <https://www.cs.cmu.edu/~sandholm/cs15-888F25/>. Zugriff: 19.01.2026.
- [Sandholm and Gilpin, 2005] Sandholm, T. and Gilpin, A. (2005). Optimal rhode island hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, page 1551. PDF: <https://www.cs.cmu.edu/~sandholm/RIHoldEm-ISD.aaai05proceedings.pdf> (Zugriff: 19.01.2026).
- [Shi and Littman, 2001] Shi, J. and Littman, M. L. (2001). Rhode island hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 602–607.
- [Southey et al., 2005] Southey, F., Bowling, M., Larson, B., Piccione, C., Burch, N., Billings, D., and Rayner, C. (2005). Bayes' bluff: Opponent modelling in poker. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005)*. PDF: <https://poker.cs.ualberta.ca/publications/UAI05.pdf> (Zugriff: 19.01.2026).
- [Zinkevich et al., 2007] Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. (2007). Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 1729–1736. PDF: <https://proceedings.neurips.cc/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf> (Zugriff: 19.01.2026).