Präferenzbasierte Monte Carlo Baumsuche

Preference based Monte Carlo Tree Search
Master-Thesis von Tobias Joppen
Tag der Einreichung:

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Christian Wirth

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Präferenzbasierte Monte Carlo Baumsuche
Preference based Monte Carlo Tree Search

Vorgelegte Master-Thesis von Tobias Joppen

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Christian Wirth

Tag der Einreichung:

Erklärung zur Master-Thesis

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

Darmstadt, den 9. Oktober 2016

(T. Joppen)

Zusammenfassung

In dieser Arbeit wird Realtime Monte Carlo Baumsuche auf die Verwendung von präferenzbasiertem Feedback angepasst. Dazu wird die Iterationsvorschrift von Monte Carlo Baumsuche (MCTS) soweit abgewandelt, dass während der Selektion nicht eine Folgeaktion gewählt wird, sondern mehrere. In der Folge werden während der Backpropagation die Simulationen der unterschiedlichen Aktionen gegeneinander verglichen, womit durch präferenzbasiertes Feedback gelernt wird. Der resultierende Algorithmus wird Relative MCTS genannt und verwendet Relative UCB als Selektionskriterium.

Weiterhin werden die einzelnen Veränderungen des Algorithmus auf dessen Auswirkungen analysiert, sowie eine Erweiterung von Relative MCTS eingeführt und getestet.

Als Validierungsgrundlage wird das 8Puzzle verwendet. Hier müssen auf einem 3×3 Feld acht Zahlen durch Verschieben in die richtige Reihenfolge gebracht werden. Hierbei erreichten Relative MCTS und dessen Erweiterung eine signifikant höhere Siegrate als Realtime MCTS.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Eine der Hauptanwendungen der Informatik ist das Lösen von Problemen unterschiedlichster Art. Ein Typ von Problemen sind die sequentiellen Entscheidungsprobleme. Bei diesen müssen nacheinander mehrere Entscheidungen getroffen werden. Die Stellen, an denen Entscheidungen getroffen werden oder ein sequentielles Entscheidungsproblem endet, werden Zustände genannt.

Das Ziel eines sequentiellen Entscheidungsproblems ist es, zuverlässig zu erwünschten Zuständen zu gelangen. Eine Lösung ist dementsprechend die Angabe, in welchem Zustand welche Entscheidung zu treffen ist.

Die aktuell gängigsten Methoden sequentielle Entscheidungsprobleme zu lösen arbeiten mit numerischem Feedback. Das bedeutet, dass während der Suche nach einer Lösung des Problems Zustände oder Entscheidungen durch eine heuristische Bewertungsfunktion mit einem numerischen Gütewert versehen werden. Für manche Problemstellungen ist das Festlegen von solch einem numerischem Feedback ein einfacher und natürlicher Vorgang. Beispielsweise lässt sich für kürzeste-Wege-Probleme die Güte von Zuständen, also beispielsweise die wahre Distanz eines Ortes zum Ziel, durch die Luftlinie approximieren.

Es gibt aber auch Bereiche, in denen das Zuordnen von Zuständen zu numerischen Werten nicht leicht und oft widersprüchlich ist. Beispiele aus der Literatur sind das Zuordnen der Bewertung -60 in medizinischen Behandlungen für den Fall, dass der Patient stirbt oder der Gewichtung von unterschiedlichen Features zum Bewerten von (möglichst optimalem) Fahrverhalten von Autos auf der Straße[7, 1].

Es kann somit alles andere als trivial sein, Entscheidungsprobleme mit numerischem Feedback zu lösen. Aus Bereichen, in denen es schwer ist, solche Bewertungsfunktionen zu definieren, haben sich in der Vergangenheit neue Ideen des maschinellen Lernens entwickelt. Das Apprenticeship Learning versucht anstatt von Bewertungsfunktionen durch Verhalten von Experten in dieser Domäne deren Verhalten zu modellieren und zu adaptieren[1].

Das präferenzbasierte Lernen versucht wie das Apprenticeship Learning, von einem Experten gegebene Informationen zu lernen. Anders ist, dass weder Zuständen numerische Werte zugeordnet werden, noch ein Experte eine Lösung zeigt, wovon es zu lernen heißt. Stattdessen bewerten Experten Zustände untereinander durch Präferenzen. Eine Präferenz ist die Angabe, welcher Zustand welch anderem Zustand vorzuziehen ist [6].

Anders als numerisches Feedback ist das eine Art der Bewertung, die in vielen Bereichen intuitiv erlangt werden kann. Für das Beispiel aus medizinischen Tests wäre eine intuitive Präferenz das Präferieren jedes Zustands gegenüber dem Zustand *Patient ist tot*. Dafür lässt sich bei Präferenzen nicht ermittlen, wie groß der Unterschied zwischen zwei Zuständen ist. Es ist nur bekannt, welcher von beiden als besser wahrgenommen wird und nicht wie viel besser er ist.

Besonders interessant ist, dass Lösungen, die durch Expertenwissen trainiert und erstellt wurden, besseres Verhalten als die Experten zeigen können [14].

1.1 Sequentielle Entscheidungsprobleme und Bäume

Sequentielle Probleme können als gerichteter Baum modelliert werden. Im Weiteren sind Bäume als gerichtete Bäume zu verstehen. Ein Baum ist ein gerichteter azyklischer Graph, wobei ein Knoten maximal einen Elternknoten besitzt und keine bis beliebig viele Kinder hat (Vergleiche Abbildung 1). Ist einem Knoten kein Kindknoten zugeordnet, so wird er Terminal genannt, und hat ein Knoten keinen Elternknoten, so heißt er Wurzel. Die Wurzel ist eindeutig. Im Bezug auf

sequentielle Entscheidungsprobleme können Knoten den Zuständen zugeordnet werden und Kanten den Entscheidungen. Ein Knoten hat eine Kante zu einem anderen Knoten, wenn in dem entsprechenden Zustand eine Entscheidung getroffen werden kann, sodass man in den neuen Zustand gelangen kann.

Die Lösung eines sequentiellen Entscheidungsproblems, welches als Baum modelliert ist, ist das Finden eines besten Pfads von der Wurzel zu einem Knoten, also von dem Ausgangszustand zu einem erwünschten Zustand.

Zur Lösung solcher Probleme hat sich A* zu einem der Standardlösungsansätze entwickelt[12]. Bei A* handelt es sich um einen exakten Löser. Das bedeutet, dass, wenn dieser Algorithmus eine Lösung erzeugt, es stets eine Optimallösung ist. Das ist einerseits gut und oft erwünscht, aber für sehr komplexe Probleme dauert das exakte Lösen oft zu lange.

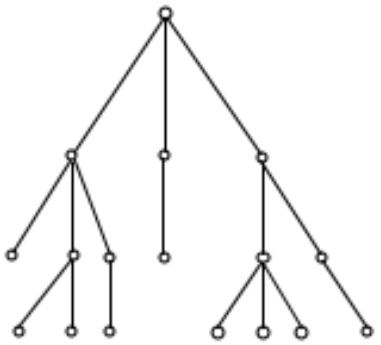


Abbildung 1: Ein Baum

Für diese Fälle gibt es approximierte Löser. Diese Algorithmen lösen gleichartige Probleme wie die exakten Löser. Das Ergebnis muss aber nicht optimal sein. Ein Vorteil dieser Approximationsalgorithmen ist die Geschwindigkeit: Sie können in vielen Problemstellungen mit deutlich weniger Zeitaufwand sehr gute Lösungen oder auch Optimallösungen finden, wo exakte Löser sehr viel Zeit benötigen würden.

Ein Beispiel für Approximationsalgorithmen ist die Monte Carlo Baumsuche (im Folgenden MCTS) [3]. Eine Variante, die auf besonders laufzeitkritische Probleme zugeschnitten wurde, ist Realtime MCTS und wird in dieser Arbeit verwendet und erweitert [3]. Die Funktionsweise von Realtime MCTS wird im Abschnitt 2.1 erläutert. Realtime MCTS verwendet im Gegensatz zum Standard MCTS Bewertungsfunktionen für Zustände, wodurch sich ein Zeitvorteil im Algorithmus ergibt. Allerdings gehen, wie in Abschnitt 1 bereits kurz erwähnt, mit der Verwendung von solch numerischem Feedback auch Nachteile mit einher.

1.2 Problemstellung, Motivation und Forschungsfragen

In diesem Abschnitt werden die Problemstellung und die Forschungsfragen gestellt, die in dieser Arbeit bearbeitet werden.

MCTS ist ein Algorithmus, der in jüngster Vergangenheit in vielen Anwendungsbereichen, wie beispielsweise Echtzeit Computerspielen, kombinatorischen Optimierungsproblemen und Bedingungserfüllungsproblemen, sehr erfolgreich eingesetzt wurde [3]. In zeitkritischen Domänen werden mit Realtime MCTS zusätzlich numerische Bewertungsfunktionen benötigt, die nichtterminale Zustände bewerten. Diese Bewertungsfunktionen sind in der Regel heuristisch, was bedeutet, dass Schätzwerte angegeben werden, die fehlerhaft sein dürfen. Weiterhin wurden im Bereich des maschinellen Lernens interessante Alternativen für numerische Bewertungsfunktionen erforscht. Unter anderem präferenzbasierte Ansätze, in denen Präferenzen angeben, welcher von mehreren Zuständen als bester Zustand angesehen wird. Es konnte gezeigt werden, dass es plausibler sein kann, Paare von Aktionen zu vergleichen, statt einzelne Aktionen heuristisch zu bewerten [4].

Es stellt sich die Frage: Lässt sich durch das Verwenden von Präferenzen statt numerischen Bewertungen in Realtime MCTS ein Gewinn erzielen? Denn das scheint gut möglich zu sein in einem Problem, in dem numerische Bewertungen fehlerhaft sind und Präferenzen diesen Fehler reduzieren könnten.

Um diese Frage beantworten zu können, muss die Korrektheit der verwendeten Bewertungsfunktion und die Korrektheit der Präferenzen vergleichbar sein, was im Allgemeinen schwer ist. Ist aber eine numerische Bewertungsfunktion gegeben, so lassen sich aus dieser leicht Präferenzen abbilden, indem die Bewertung von den entsprechenden Zuständen verglichen wird (Siehe Abschnitt 4.1). Die vorherige Frage lässt sich somit umformulieren:

> **Fragestellung:** *Lässt sich durch das Übersetzen von heuristischen Bewertungen in Präferenzen in Realtime MCTS ein Gewinn erzielen?*

Durch das Übersetzen von heuristischen Bewertungen in Präferenzen gehen Informationen verloren. Ist Aktion a_i gegenüber Aktion a_j zu präferieren, geben Präferenzen im Gegensatz zu Bewertungsfunktionen nicht an, wie groß der Unterschied der beiden Aktionen ist. Große und kleine Unterschiede in den Bewertungen werden durch Präferenzen gleich behandelt.

Da nicht nur die Bewertung, sondern der ganze algorithmische Ablauf von Realtime MCTS angepasst werden muss, um Präferenzen zu verwenden, stellt sich die Frage, welche Änderung welchen Einfluss auf die Güte von Realtime MCTS hat.

> **Fragestellung:** *Wie wirken sich Informationsverlust und algorithmische Anpassung von Realtime MCTS auf dessen Güte aus?*

Um das zu bestimmen, muss der Ablauf von MCTS angepasst werden, um präferenzbasiertes Lernen verwenden zu können.

> **Problemstellung:** *Wie lässt sich Realtime MCTS auf die Verwendung von Präferenzen anpassen?*

1.3 Aufbau der Arbeit

Im folgenden Kapitel werden grundlegende Informationen über MCTS und präferenzbasierte Lernverfahren genannt, sodass im darauf folgenden Kapitel das präferenzbasierte MCTS erklärt werden kann. In Kapitel 4 wird das experimentelle Setup erläutert, in Kapitel 5 werden die Auswertungen der Experimente geschildert, und im letzten Kapitel werden die gewonnenen Resultate nochmals zusammengefasst und weitere Ausarbeitungsmöglichkeiten für zukünftige Projekte aufgezeigt.

2 Monte Carlo Baumsuche und präferenzbasiertes Lernen

In diesem Kapitel werden aktuelle Algorithmen und Lösungsansätze vorgestellt, auf die in den folgenden Kapiteln eingegangen wird. Es folgt eine Einführung in das präferenzbasierte Lernen und Monte Carlo Baumsuche (MCTS).

2.1 Monte Carlo Baumsuche (MCTS)

In diesem Kapitel wird der Ablauf und die Funktionsweise von Monte Carlo Baumsuche (MCTS) beschrieben und erklärt. Dazu werden zunächst Grundlagen erklärt.

2.1.1 Markov Decision Process (MDP)

In der Einleitung wurden sequentielle Entscheidungsprobleme schon kurz erklärt. Um Löser dieser Probleme algorithmisch aufzuarbeiten ist es notwendig die Problemstellung zu formalisieren, was im Folgenden geschieht.

Ein Markov Entscheidungsprozess (MDP) ist ein Framework, um sequentielle Entscheidungsprobleme in einer diskreten stochastischen Umgebungen zu modellieren [13]. Es besitzt folgenden Aufbau:

- S: Eine Menge an Zuständen
- A: Die Menge alle Aktionen
- $A(s) \subseteq A$: Die Mengen von ausführbaren Aktionen in einem Zustand s
- $\delta(s'|a,s)$: Eine Transitionswahrscheinlichkeit, die angibt, mit welcher Wahrscheinlichkeit das System von Zustand s mit Ausführen der Aktion $a \in A(s)$ in den Zustand s' übergeht.
- $R(s) \in \mathbb{R}$: eine Belohnungsfunktion, die jedem Zustandsübergang zum Zustand s eine Belohnung $\in \mathbb{R}$ zuordnet.
- $s_0 \in S$: dem Startzustand des Entscheidungsproblems

Man redet in diesem Zusammenhang davon, dass ein Agent in einem Zustand ist und eine Aktion ausführt, um in einen anderen Zustand zu gelangen. Zu jedem Zeitpunkt befindet sich der Agent in einem Zustand s^t. Er kann aus mehreren Aktionen $A(s^t)$ eine wählen, die ausgeführt wird. Somit gelangt er mit Wahrscheinlichkeit $\delta(s'|a,s^t)$ in den Zustand s'. Führt er eine Aktion aus, so bekommt er mitgeteilt,

1. in welchen neuen Zustand s' der Agent gelangt und
2. welche Belohnung er für diese Transition bekommen hat, also $R(s')$.

Das Ziel ist es, eine Strategie $\pi : S \times A \to [0,1]$ zu finden, die zu jedem Zustands-Aktions-Paar angibt, mit welcher Wahrscheinlichkeit $p \in [0,1]$ im Zustand s die Aktion a auszuführen ist, sodass der Gewinn über die Zeit maximiert wird. Das bedeutet, dass die Belohnungen auf Dauer maximiert.

Gegeben einer Strategie kann einem Zustands-Aktions-Paar eine Güte entsprechend der daraus resultierenden Belohnungen definiert werden, wobei ein Discountfaktor $\gamma \in [0,1]$ früher auftretende Belohnungen gegenüber später auftretenden höher bewertet:

$$Q^{\pi}(s,a) = \sum_{t=0}^{\inf} \gamma^t \sum_{s \in S} P^t(s) R(s),$$

wobei $P^t(s)$ die Wahrscheinlichkeit ist, mit welcher sich der Agent zum Zeitpunkt t im Zustand s befindet. Diese berechnet sich iterativ durch

$$P^t(s_0) = 1$$

und

$$P(s^{t+1}) = \sum_{s \in S} P^t(s) \sum_{a \in A(s)} \delta(s^{t+1}|a, s^t)\pi(s^t, a).$$

Das Ziel, beziehungsweise eine optimale Lösung des zugehörigen Entscheidungsproblems, ist somit eine Strategie π^*, sodass

$$\pi^* = \operatorname*{argmax}_{\pi} \sum_{a \in A(s_0)} Q^{\pi}(s_0, a)\pi(s_0, a).$$

Es gibt Sonderfälle wie deterministische MDPs. Die Anwendungsdomäne in dieser Arbeit ist deterministisch, sodass es für einen Zustand s und eine Aktion a nur einen Nachfolgezustand $\hat{s}$ gibt. Also

$$\forall s \in S, a \in A(s) \exists \hat{s} \in S : (\delta(\hat{s}|a, s) = 1) \wedge (\forall \bar{s} \neq \hat{s} : \delta(\bar{s}|a, s) = 0).$$

MDPs in dieser Arbeit

Im Rahmen dieser Arbeit wird ein Spezialfall von deterministischen MDPs verwendet:

Transitionen in nichtterminale Zustände liefern immer eine Belohnung von 0. Terminale Zustände lassen sich in *positive* und *negative* Terminale aufteilen. Das Aufteten von negativen Terminalen wird auch mit einer Belohnung von 0 bewertet, wohingegen die positive Terminale eine Bewertung von 1 bekommen. Ein Siegzustand ist somit ein terminaler Zustand s^* mit $R(s^*) = 1$.

Die Aufgabe von Algorithmen im Rahmen dieser Arbeit ist es, gegeben solch eines MDPs und eines Startzustands, eine möglichst kurze Aktionsfolge zu finden, die, ausgehend von Startzustand, in einem positiven Terminal endet (Spiel gewonnen). Da dieses Problem zumindest theoretisch durch A^* oder eine erschöpfende Baumsuche perfekt zu lösen ist, wird eine obere Schranke für die Laufzeit des Algorithmus angegeben.

2.1.2 Baumsuche

Baumsuche beschreibt das konzeptionelle Durchsuchen eines Zustandsraumes durch das iterative Aufspannen eines Suchbaums (Vergleiche Abbildung 2). Ein Zustandsraum ist dabei gegeben als gerichteter Graph (V,A), wobei V (die Menge der Knoten) als die Zustände und A (die Menge der Kanten) als die Aktionen zu interpretieren sind. Besteht zwischen zwei Knoten v' und v'' eine Kante a', so kann man in dem Zustand v' durch Ausführen der Aktion a' in den Zustand v'' gelangen.

Zu jedem Zeitpunkt während der Ausführung eines Baumsuchalgorithmus existieren der aktuelle Suchbaum und eine Menge von Knoten, die in der nächsten Iteration in den Baum hinzugefügt werden können. Für jeden Knoten $\overline{v}$ dieser Menge gilt, dass er durch eine Kante $\overline{a} = (\hat{v}, \overline{v})$ von einem Knoten $\hat{v}$ aus dem aktuellen Suchbaum verbunden ist und noch kein Teil des Suchbaums ist. Diese Menge wird Rand (eng. fringe) genannt.

In jeder Iteration wird mindestens ein Element $\overline{v}$ mit zugehörigem Verbindungsbogen $\overline{a}$ aus dem Rand in den aktuellen Suchbaum übernommen. Die Entscheidung, welches Element aus dem Rand hinzugefügt wird, ist der wesentliche Unterschied der diversen Baumsuchalgorithmen.

Die beiden bekanntesten Vertreter dieser Kategorie sind Breitensuche und Tiefensuche, die die Strategien *wenigste Kanten zur Wurzel zuerst wählen* bzw. *meiste Kanten zur Wurzel zuerst wählen* verfolgen.

Baumsuchalgorithmen starten mit einem gegebenen Startzustand v_0, der Wurzel, von dem aus sich im Laufe der Iterationen ein Baum über dem ursprünglichen Graphen bildet (Vergleiche Abbildung 2).

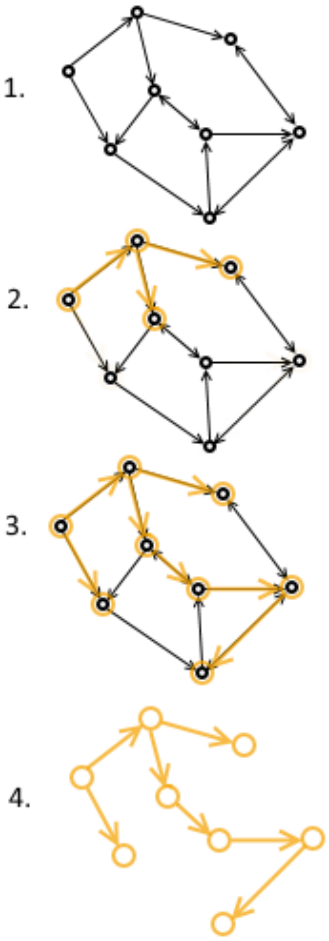


Abbildung 2: Die Aufspannung eines Baums.

2.1.3 K-armiger Bandit (KAB)

Um die Arbeitsweise von Monte Carlo Tree Search zu verstehen, ist es viel wert sich Gedanken über das Problem zu machen, wie man optimal an einem K-armigen Banditen spielt (kurz: KAB) [8]. Unter einer solchen Glücksspielmaschine lässt sich ein zustandsloser Automat vorstellen, der nach Betätigen eines Armes mit einer dem Arm zugehörigen Wahrscheinlichkeit in einem Sieg oder einem Verlust endet. Da es sich um einen zustandslosen Automaten mit K vielen unabhängigen Armen handelt, kann man sich auch vorstellen K viele einarmige Banditen nebeneinander zu betreiben, wobei jeder Automat seine eigene Gewinnchance besitzt. Das Ziel ist, mit maximaler Gewinnchance zu spielen, was nichts anderes heißt, als permanent den besten Arm zu betätigen. Das Problem ist, dass die Gewinnchancen nicht bekannt sind.

Es müssen also die unterschiedlichen Arme ausprobiert werden, um zu ermitteln, welcher der beste Arm ist. Da der Gewinn zufallsabhängig ist, lässt sich die wahre Wahrscheinlichkeit nur mit unendlich vielen Spielen ermitteln. Davor kann man nur mit einer gewissen Sicherheit sagen, dass ein bestimmter Arm der Beste ist. Das K-armige Banditenproblem kann also als Entscheidungsproblem angesehen werden, welcher Arm zu spielen ist, angesichts der schon

bisherigen Resultate. Die Lösung eines K-armigen Banditenproblems ist das Identifizieren eines Armes als des besten Arms bezüglich der Gewinnchance.

Um das zu präzisieren, wird das K-armige Banditenproblem formalisiert betrachtet:

Gegeben sind Zufallsvariablen $X_{i,j}$ für $0 \leq i < K$ und $j > 0$. Der Index i gibt dabei den gewählten Arm des Banditen an. Der Index $X_{i,j}$ ist daher das Ergebnis des j-ten Betätigen des i-ten Armes.

Während des Ermittelns des besten Arms müssen, wie oben beschrieben, alle Arme mehrfach gespielt werden. Das Spielen eines suboptimalen Arms bedeutet Verlust gegenüber dem Spielen des optimalen Arms. Dadurch wird notwendiges Wissen angeeignet, um den besten Arm zu ermitteln. Der spieltheoretische Verlust an Siegwahrscheinlichkeit durch das Spielen der suboptimalen Aktionen gegenüber dem dauerhaften Spielen des optimalen Arms heißt *Regret* (englisch für Bedauern) und ist zu minimieren.

Im Allgemeinen ist Regret auf ein im maschinellen Lernen bekanntes Problem zurückzuführen: *Exploration versus Exploitation*. Im deutschen heißt das soviel wie die Frage, ob man die aktuell beste Lösung nutzt um Gewinn zu machen oder ob man eine andere Möglichkeit ausprobiert mit der Hoffnung eine noch bessere Lösung zu finden.

2.1.4 Von K-armigem Bandit zu MDPs

Es gibt, wie im Folgenden noch beschrieben wird, gute Techniken um das K-armige Banditenproblem zu lösen. Zunächst wird aber beschrieben, wie man gegeben einer Lösung für das K-armige Banditenproblem eine Strategie für endliche MDPs erzeugen kann.

Eine Strategie liefert zu jedem Zustand des MDPs und jeder Aktion, die dort ausgeführt werden kann, mit welcher Wahrscheinlichkeit diese auszuführen ist. Betrachtet man einen Zustand s, für den eine Aktion gespielt werden soll, so kann die optimale Aktion, die mit Wahrscheinlichkeit 1 gespielt werden soll, wie folgt ermittelt werden:

Im Zustand s stehen die Aktionen $A(s)$ zur Verfügung. Wäre jeder Nachfolgezustand s' von s für jede mögliche Aktion a ein terminaler Zustand, so wäre dieses MDP mit Startzustand s ein K-armiger Bandit für $K = |A(s)|$.

Das ist aber nicht unbedingt der Fall, sodass es einen Nachfolgezustand s_1 von s geben kann, der nicht terminal ist. Hier kann aber die stochastische Umgebung ausgenutzt werden, sodass dies doch angenommen werden kann: Es können, ausgehend von s_1, nacheinander zufällig gleichverteilte Aktionen gewählt werden. Somit entsteht eine Kette von Zuständen $s_1 \to s_2 \to s_3 \to ... \to s_t$. Da hier endliche MDPs betrachtet werden, endet diese Kette nach t Zuständen in einem Terminal. Durch die gleichverteilt zufällige Wahl der jeweils nächsten Aktion und der gegebenen Transitionswahrscheinlichkeiten δ des MDP lässt sich ermitteln, wie groß die Wahrscheinlichkeit p ist, dass solch eine Kette in einem positiven Terminal endet (Vergleiche Abschnitt 2.1.1). Entsprechend gibt die Gegenwahrscheinlichkeit $1-p$ an, wie wahrscheinlich ein negatives Terminal auftreten wird. Somit kann das Auftreten des Zustands s_1 mit einer Wahrscheinlichkeit von p als positives, beziehungsweise mit $p-1$ als negatives Terminal interpretiert werden.

Erstellt man zwei neue Zustände s_+ und s_- als positives bzw. negatives Terminal und ersetzt Transitionen auf nichtterminale Zustände mit Transitionen auf s_+ und s_- mit entsprechenden Wahrscheinlichkeiten p und $1-p$, so ist jeder Nachfolgezustand von s entweder s_+ oder s_- und somit terminal, womit dieses umgeänderte MDP mit Startzustand s ein K-armiger Bandit ist.

Die Aktion, welche die Strategie in dem Zustand s wählt, ist diejenige, die durch die Lösung des K-armigen Banditenproblems als Aktion mit der höchsten Gewinnchance identifiziert wurde.

Die Optimalität dieser Strategie ist offensichtlich von der Optimalität der Lösung des K-armigen Banditenproblems abhängig. Wie im vorherigen Kapitel aber beschrieben, lässt sich zu keinem Zeitpunkt der optimale Arm eines K-armigen Banditen identifizieren, da erst im Grenzfall mit unendlich vielen Versuchen die wahren Siegchancen pro Arm feststehen.

Dieses Verfahren schätzt $Q^{\pi}(s,a)$ ab, ohne π zu kennen. Da die Strategie unbekannt ist, müssen bei diesem Verfahren die Aktionen der oben genannten Kette zufällig gleichverteilt gewählt werden, weil nichts über die Güte der einzelnen Aktionen bekannt ist. Im folgenden Abschnitt wird die Idee beschrieben, dass zu diesen Aktionen die beobachteten Gütewerte mitverwaltet werden, sodass auch für Nachfolgezustände s' der Wert $Q^{\pi}(s',a)$ abgeschätzt werden kann.

Das hat unter anderem den Vorteil, dass die Kette an Aktionen nicht mehr zufällig gleichverteilt gewählt werden muss, sondern entsprechend der Gütewerte angepasst werden kann, sodass schlechte Züge seltener ausgeführt werden.

2.1.5 Baumstruktur von KABs

Wie im vorherigen Abschnitt beschrieben, kann jeder Zustand eines MDPs als KAB interpretiert werden. Weiterhin kann der Zustandsraum des MDPs als Baum reinterpretiert werden (Vergleiche Abschnitt 2.1.2). Somit kann jedem Knoten dieses Baumes ein KAB zugeordnet werden.

Dieser schematische Aufbau, also das Aufspannen eines Suchbaums über den Zustandsraum mit Interpretation jedes Knotens als eigenständiges KAB, ist die Grundidee der Monte Carlo Baumsuche.

Statt ein einziges K-armiges Banditenproblem für den Zustand zu lösen, in dem sich der Agent zu einem bestimmten Zeitpunkt befindet, baut Monte Carlo Baumsuche einen Baum von K-armigen Banditen auf. Die Aufgabe bleibt dennoch dieselbe: für den aktuellen Zustand die beste Aktion zu ermitteln.

Die Tatsache, dass ein Baum von KABs aufgespannt wird ergibt vor allem folgenden Vorteil: Es wird nach der Wahl einer der Aktionen des Startzustands keine komplett zufällige Kette an Aktionen von dort aus gestartet, sondern die ersten Aktionen nach dem Startzustand werden von den KABs der entsprechenden Knoten ermittelt.

Dadurch wird nicht nur in dem ersten Knoten eine Abwägung zwischen Exploration und Exploitation gemacht, sondern auch in den darauf folgenden Knoten. Je länger der Algorithmus läuft desto später werden die Aktionen zufällig.

Diese zufällige Aktionswahl ist das, was dieses Verfahren zu einer Monte Carlo Technik macht und ihr somit ihren Namen gibt.

2.1.6 Monte Carlo Methode

Als Monte Carlo Methode bezeichnet man die Idee, durch Einsatz vieler gleichverteilter Zufallsvariablen eine approximierte Lösung für ein Problems zu bekommen. Je mehr Zufallsvariablen verwendet werden, um so genauer wird dabei das Ergebnis. Der Einsatz lohnt sich besonders dann, wenn das Berechnen einer exakten Lösung sehr aufwendig, wenn nicht sogar unmöglich ist. Beispielsweise lässt sich π durch Monte Carlo Methoden beliebig genau berechnen (Vergleiche Abbildung 3).

Entscheidend für den Erfolg ist die vielfache Wiederholung beziehungsweise die hohe Anzahl an Zufallsvariablen, da der Erfolg auf das *Gesetz der großen Zahlen* zurückzuführen ist: Nimmt

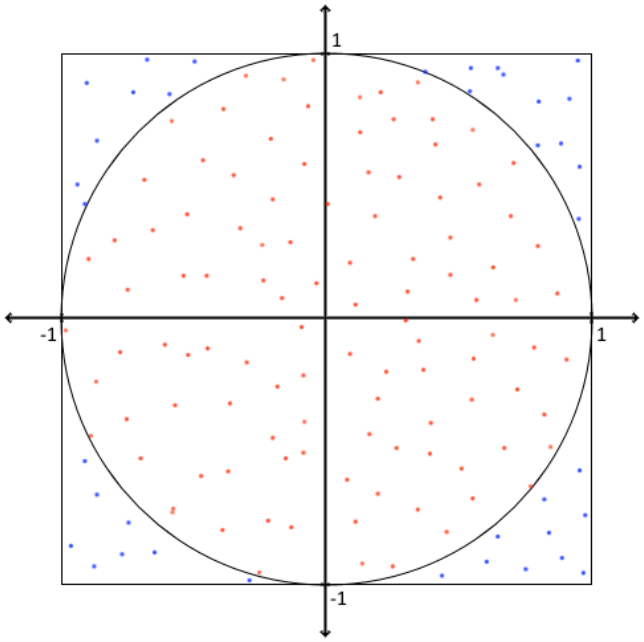


Abbildung 3: Eine Monte Carlo Methode zur Bestimmung von π: Mit zunehmender Anzahl an zufälligen Punkten in $[-1, 1]^2$ liegt der Faktor, wie viele Punkte sich innerhalb des Einheitskreises befinden gegenüber denen außerhalb, beliebig nah an $\pi/4$.

man genug Stichproben, so wird das Ergebnis irgendwann der Wahrheit beliebig nahe sein. Man kann kein *Pech* haben.

2.1.7 Optimieren der nächsten Aktion

MCTS wird vorwiegend in Bereichen eingesetzt, in denen eine zeitliche Obergrenze für die Berechnung der als nächstes auszuführenden Aktion existiert. Die Idee einen kompletten Suchbaum aufzuspannen und alle zugehörigen KABs für alle Zustände des MDPs zu lösen, ist im Allgemeinen selbst approximativ mit einer zeitlichen Beschränkung nicht möglich[3].

Ist ein Agent im Zustand s, so kann MCTS allerdings durch die Verwendung der KABs die nächste auszuführende Aktion approximativ optimieren, ohne einen kompletten Lösungsweg zu erzeugen. MCTS optimiert die nächste auszuführende Aktion im Rahmen einer zeitlichen Obergrenze. Eine Ausführung des Algorithmus liefert daher für einen Zustand eine auszuführende Aktion zurück und keine komplette Lösung des MDPs. MCTS muss daher in der Anwendung in jedem Zeitschritt (also für jede zu spielende Aktion) neu ausgeführt werden.

Das ist anders als viele andere Baumsuchalgorithmen wie A^* oder Tiefensuche, die explizit einen optimalen Lösungsweg benötigen, um die nächste Aktion gut wählen zu können. Ist der optimale Lösungsweg noch nicht bekannt, so können diese Algorithmen im Allgemeinen keine zuverlässige Aussage über die Güte der Aktionen treffen.

Daraus ergeben sich Vorteile und Nachteile von MCTS: MCTS kann in einem gegebenen Zustand eine gute Aktion finden, auch wenn die Zeit nicht genügt um einen kompetten Suchbaum aufzuspannen. Die Aktion, die MCTS nach Ablauf der Zeit zurückgibt, muss allerdings nicht optimal sein, anders als bei beispielsweise A^* oder einer Tiefensuche.

Eine weitere Auflistung der Eigenschaften von MCTS, sowie eine detailliertere Beschreibung des Ablaufs folgen im nächsten Abschnitt.

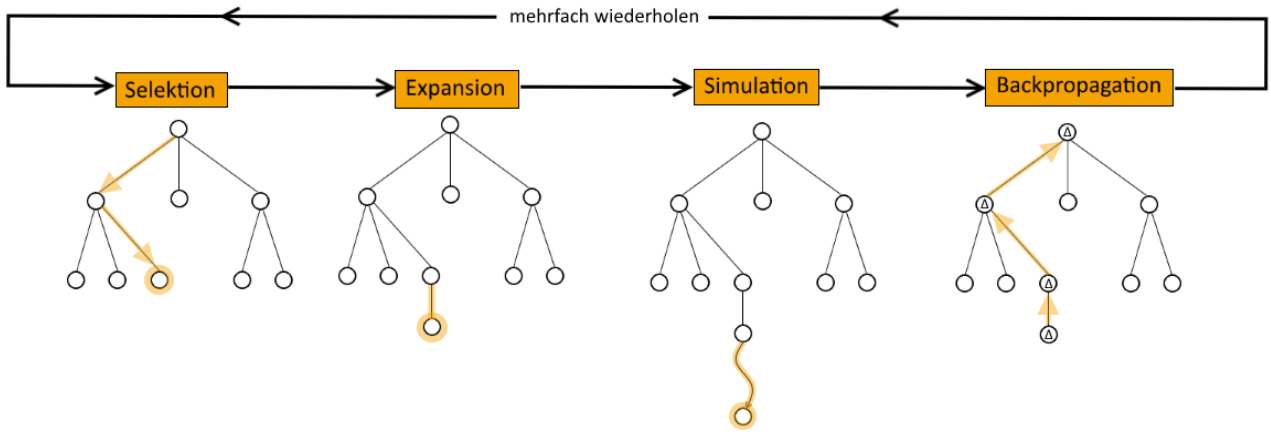


Abbildung 4: Eine Iteration von MCTS verläuft in vier Schritten: Selektion, Expansion, Simulation und Backpropagation. Pro Iteration wird dem Baum ein Knoten hinzugefügt

2.1.8 MCTS im Detail

In diesem Abschnitt wird der Ablauf von MCTS beschrieben. Für offene Fragen und weitergehende Beschreibung wird auf Literatur verwiesen, die im Folgenden zusammengefasst und aufgearbeitet dargestellt wird [3, 9, 11, 14, 8].

Im Laufe des Algorithmus wird ein asynchroner, iterativ wachsender Spielbaum erstellt. Zu Beginn ist nur ein Knoten vorhanden, der dem aktuellen Zustand des Agenten zuzuordnen ist. In jeder Iteration wächst der Baum um einen Knoten, der sich als Kind an einen vorhandenen Knoten anhängt.

Eine Iteration kann in vier Abschnitte unterteilt werden:

- Selektion
- Expansion
- Simulation
- Backpropagation

Diese Iterationen werden so lange wiederholt, bis die zur Verfügung gestellte Rechenzeit aufgebraucht ist. Die vier Abschnitte der Iteration werden nun im Detail behandelt:

Selektion

Zu Beginn jeder Iteration ist eine *Tree Policy* dafür zuständig, einen interessanten Spielzustand innerhalb des aktuellen Suchbaums zu finden, der weiter untersucht werden soll. Interessant heißt hier:

1. Es soll ein möglichst vielversprechender Knoten sein, sodass die zu erwartende Punktzahl hoch ist.
2. Es sollen auch Spielzustände untersucht werden, die nicht optimal erscheinen, da Nachfolgezustände nichtsdestotrotz sehr gut und auch optimal sein können.
3. Der Knoten soll noch nicht komplett expandiert sein. Das soll heißen, dass es noch mindestens eine Aktion gibt, deren Nachfolgezustand noch nicht als Kind dieses Knotens in

dem Spielbaum existiert. In der ersten Iteration wird somit der Wurzelknoten ausgewählt werden.

Hierbei entsprechen die Punkte 1. und 2. dem Exploitation versus Exploration Problem (Vergleiche Abschnitt 2.1.3).

Um diesen Zustand zu finden, wird, startend von der Wurzel, iterativ ein Kindknoten unter Beachtung von Exploitation versus Exploration ausgewählt. Das wird wiederholt, bis ein Knoten erreicht wird, der noch nicht im Suchbaum existiert. Das bedeutet nach Punkt 3, dass sein Elternknoten noch nicht komplett expandiert ist.

Allgemein gesprochen ist eine Tree Policy dafür zuständig, in einem Knoten ein Kind auszuwählen. Die exakte Wahl der Tree Policy ist ein Freiheitsgrad von MCTS. Im Kapitel 2.2.2 wird eine Standardversion (UCT) vorgestellt, die auch in dieser Arbeit verwendet wird.

Diese Tree Policy übernimmt das Auswählen der ersten Aktionen vor der Simulation. Dabei kann das iterative Aussuchen des nächsten Kindknotens durch die Tree Policy als aktuelle Lösung des diesem Knoten zugehörigen K-armigen Banditenproblems interpretiert werden.

Expansion

Ist ein noch nicht komplett expandierter Knoten v gefunden und wurde eine Aktion zum Ausführen gewählt, so kann ein Knoten v' für diesen Spielzustand als Kind von v in den Baum eingefügt werden. So wächst der Baum mit jeder Iteration um genau einen Knoten.

Simulation

Um die Güte des Spielzustands von v zu schätzen, wird von diesem Zustand aus eine Simulation gestartet. Eine Simulation ist das zufällige Ausführen von Aktionen bis entweder eine gewisse Anzahl an Aktionen ausgeführt wurden oder ein terminaler Spielzustand erreicht wurde. Die Obergrenze der Anzahl der Aktionen ist ein Freiheitsgrad des Algorithmus. In der nachfolgenden Auswertung werden unterschiedliche Möglichkeiten miteinander verglichen.

Das Stoppen der Simulation bei nichtterminalen Zuständen ist die Besonderheit von Realtime MCTS gegenüber MCTS. In diesem Fall (bei Realtime MCTS) muss durch eine Bewertungsfunktion der finale Zustand der Simulation bewertet werden, da es sich nicht unbedingt um ein Terminal handelt und die Problemstellung keine eigene Bewertung des Zustands zur Verfügung stellt. Diese Bewertungsfunktion weist jedem möglichen Zustand eine Güte Δ zu. Die exakte Wahl der Bewertungsfunktion ist ein weiterer Freiheitsgrad von Realtime MCTS. Sie muss in der Regel domänenspezifisch angepasst werden.

Es sei angemerkt, dass diese Simulation nicht unbedingt zufällig sein muss. Das bedeutet, dass die Aktionen der Simulation nicht zufällig gleichverteilt gewählt werden. Es gibt Ansätze, die durch das geschickte Wählen von Aktionen dieser Simulation die Performanz von MCTS zu verbessern versuchen[3]. In diesem Zusammenhang geschieht das Wählen der Simulationsaktionen durch eine Rollout Policy. Im Falle des zufälligen Wählens, wie es oben beschrieben ist, und im Weiteren in dieser Arbeit verwendet wird, wird von einer Random Rollout Policy gesprochen.

Backpropagation

Nach dem Ermitteln der Bewertung Δ des finalen Zustands der Simulation $\overline{v}$ wird Δ ausgehend von v' an die Wurzel weitergeleitet. So wird jeder Spielzustand, der ein Vorgänger von $\overline{v}$ ist und im Baum existiert, über einen Spielverlauf benachrichtigt, in dem er vorgekommen ist.

Dadurch werden die entsprechenden KABs der benachrichtigten Knoten aktualisiert, was dem Aktualisieren der Tree Policy entspricht. Somit ist es möglich, dass in der nächsten Iteration im Punkt *Selektion* die Tree Policy andere Knoten selektieren wird als in der vorhergehenden Iteration geschehen.

Eigenschaften

Analysiert man MCTS, lassen sich folgende Eigenschaften ermitteln:

- Anytime
  MCTS ist ein *anytime* Algorithmus, was bedeutet, dass er nach jeder Iteration beendet werden kann und ein sinnvolles Ergebnis liefert. Das ist anders bei üblichen erschöpfenden Suchalgorithmen, da diese die Optimallösung expandiert haben müssen, um Wissen über sie zu haben. Bricht man diese Algorithmen vorzeitig ab, so ist keine Aussage über die Korrektheit dieser Algorithmen möglich. Bei MCTS ist das anders: Durch die Monte Carlo Simulationen gibt es Schätzungen darüber, wie gut ein Knoten ist anhand von Nachfolgezuständen. Es kann nicht immer mit kompletter Sicherheit gesagt werden, dass eine Aktion die beste Option im aktuellen Zustand ist, aber es kann eine Aktion angegeben werden, der aktuell die höchste Wahrscheinlichkeit zugesprochen wird, die beste Aktion zu sein.

- Asymmetrischer Aufbau
  Durch das dynamische Auswählen des zu expandierenden Knotens durch die Tree Policy wird der Baum asymmetrisch aufgebaut. Das bedeutet, dass anders als bei der Breitensuche nicht gleichmäßig erkundet wird, sondern in vielversprechenden Bereichen mehr erforscht wird. Das lässt sich mit einer Bestensuche vergleichen.

- Konvergenz
  Eine wichtige Eigenschaft von MCTS ist die Konvergenzeigenschaft:
  Unter Verwendung bestimmter Tree Policies konvergiert MCTS gegen einen Minimax Baum [3]. Einen Minimax Baum zeichnet aus, dass er in jedem Knoten die optimale Aktion nennen kann. Eine entsprechende Tree Policy und dessen Konvergenz wird im Abschnitt 2.2.2 beschrieben.

- Aheuristisch
  MCTS benötigt in der Grundversion keine Heuristik, da keine Bewertungen nichtterminaler Zustände notwendig sind. Die Bewertung von Zuständen geschieht durch Monte Carlo Simulation, die in terminalen Zuständen endet.

Die Eigenschaften Anytime, Asymetrischer Aufbau und Konvergenz machen MCTS zu einem sehr interessanten Algorithmus zur Anwendung in Echtzeitumgebungen: Konvergenz selbst ist eine Notwendigkeit, um sicherzustellen, dass das Verfahren korrekt funktioniert. Anytime erlaubt es, die Berechnung zu stoppen, wenn die Echtzeitumgebung es erfordert, und so lange weiter zu rechnen, wie es möglich ist. Der asymmetrische Aufbau hilft dabei schneller bessere Lösungen zu finden, was besonders in einer Echtzeitanwendung viel wert ist. Zu bemerken ist allerdings, dass die Monte Carlo Simulation gegenüber Auswertungen simpler Heuristiken einen enormen zeitlichen Mehraufwand bedeutet. Insbesondere in Umgebungen, wo terminale Zustände nicht leicht zu finden sind und solche Simulationen viele Schritte benötigen. Zusätzlich korelliert die Präzision von Monte Carlo Methoden mit der Anzahl der durchgeführten Simulationen, was bedeutet, dass recht viele Iterationen benötigt werden, um ein vielsagendes Ergebnis zu erreichen (Vergleiche Abschnitt 2.2.1 und 2.2.2).

Aus diesen Gründen wird in Echtzeitumgebungen, wie auch in dieser Arbeit, eine Variation von MCTS verwendet: Realtime MCTS. Die Besonderheit an Realtime MCTS ist, dass Monte Carlo Simulationen nicht zwingend bis zu einem terminalen Zustand ausgeführt werden, sondern zwischenzeitlich beendet werden können. In diesen nicht terminalen Zuständen werden Heuristiken eingesetzt, um den entsprechenden Zustand zu bewerten. Damit entfällt die aheuristische Eigenschaft, und das Verfahren ist von der Güte einer Heuristik abhängig. Dadurch ist es allerdings möglich, die Anzahl der Iterationen in gleicher Zeit zu erhöhen und somit die Güte der Monte Carlo Methode in Abhängigkeit der Bewertungsfunktion zu verbessern [3].

2.1.9 Aktionswahl

Durch die Verwendung von MCTS steht ein Algorithmus zur Verfügung, der einen, unter richtigen Umständen, optimal konvergierenden Spielbaum aufspannt. Da die Optimalität erst nach sehr vielen Iterationen gegeben ist, wird der Algorithmus üblicherweise frühzeitig gestoppt. Dies ist dank der *Anytime* Eigenschaft möglich. Das liefert in der Regel gute, wenn auch nicht unbedingt optimale Ergebnisse.

Das Ursprungsproblem ist eine Aktion in einen gegebenen Zustand zu wählen. Diese Aktion muss nach Beendigung der letzten Iteration des Algorithmus ermittelt werden. Der Standardansatz ist, die Aktion zu wählen, die in der Wurzel bisher am häufigsten von der Tree Policy gewählt wurde.

2.2 Tree Policy

Die Aufgabe der Tree Policy ist es, gegeben eines Knotens einen Kindknoten des Knotens zu wählen. Ein wichtiger Bestandteil der Tree Policy ist der Tradeoff zwischen Exploration und Exploitation.

Betrachtet man die mögliche Zuordnung von Zuständen zu KABs, wie es in Kapitel 2.1.4 gezeigt wurde, so lässt sich dieser Ansatz durch Lösen des K-armigen Banditenproblems lösen. Es folgen zwei in der Literatur bekannte, erfolgreiche und aufeinander aufbauende Lösungskonzepte: UCB und UCT.

2.2.1 UCB

Gegeben sind die schon gespielten n Züge eines K-armigen Banditen. Davon sind für $0 \leq i < K$ jeweils n_i Züge auf dem i-ten Arm gespielt worden. Die Resultate des $j$ten Zuges des $i$ten Arms ist $X_{i,j}$.

Sind diese Informationen gegeben, lassen sich Annahmen darüber treffen, welcher Arm der Vielversprechendste ist.

Ein einfacher Ansatz ist, den durchschnittlichen Gewinn jedes Arms zu ermitteln und den Arm mit dem höchsten Wert zu spielen:

$$\overline{X}_i = \sum_{j=0}^{n_i} \frac{X_{i,j}}{n_i} \tag{1}$$

Das Grundproblem dieses Ansatzes ist, dass nicht optimale Arme durch Glück einen höheren durchschnittlichen Gewinn erzielen können als optimale Arme. Ebenso kann der beste Arm Pech haben und einen sehr niedrigen durchschnittlichen Gewinn haben.

Als Beispiel kann ein einfacher KAB betrachtet werden: Aktion a_1 führt zu 50% zum Sieg, Aktion a_2 zu 75%. Beide Aktionen wurden bisher einmal ausgeführt, wobei das Resultat von Aktion

a_1 *Spiel gewonnen*(=1) und von Aktion a_2 *Spiel verloren*(=0) ist. Nun ist der durchschnittliche Gewinn von Aktion a_2 gleich 0. Der durchschnittliche Gewinn von Aktion a_1 ist gleich 1. Somit würde Aktion a_1 gewählt, da dieser Wert höher ist als der Wert von a_2. Egal wie oft in Zukunft die Aktion a_1 zum Ausgang *Spiel verloren* führt, die durchschnittliche Bewertung dieser Aktion wird stets echt größer als 0 sein. Somit fällt die durchschnittliche Bewertung der Aktion a_1 nie auf die Bewertung von Aktion a_2, womit Aktion a_2 nie mehr ausgeführt wird.

Um dieses Problem zu umgehen, ist es notwendig, auch die bisher als schlecht wahrgenommenen Züge zu verwenden, um herauszufinden, ob sie nicht vielleicht besser sind, als sie zu sein scheinen. Im Bezug auf den *Exploration vs. Exploitation Tradeoff* muss hier mehr Exploriert werden.

Ein Lösungsansatz mit Exploration, der hier vorgestellt wird, geht dieses Problem an, indem er zu ermitteln versucht, wie sehr man dem Wert $\overline{X}_i$ vertrauen kann. Dazu wird ein Mittel der Statistik verwendet: Konfidenzintervalle. Das Konfidenzintervall ist der Bereich, der bei unendlicher Wiederholung eines Zufallsexperiments mit einer gewissen Häufigkeit (dem Konfidenzniveau) die wahre Lage des Parameters einschließt [5, 10].

Eins der ersten Verfahren, das die Konfidenzintervalle in diesem Zusammenhang verwendete, ist UCB1 [2, 3, 8]. UCB steht für *Upper Confidence Bound*, also obere Konfidenzschranke. Hier werden obere Konfidenzschranken gebildet und jene Aktion ausgeführt, die die höchste obere Konfidenzschranke besitzt. Das muss eben nicht jene Aktion sein, die den höchsten durchschnittlichen Gewinn besitzt, wie die Formel von UCB1 zeigt:

$$UCB1 = \overline{X}_i + \sqrt{\frac{2\ln n}{n_i}} \tag{2}$$

Hier ist $n = \sum_{l=0}^{K} n_l$ die Gesamtanzahl aller Spiele.

Der vordere Teil ($\overline{X}_i$) ist eben der durchschnittliche Gewinn einer Aktion (Siehe (1)) . Der hintere Teil ($\sqrt{2lnn/n_i}$) wird kleiner, je öfter dieser Arm in Relation zu den übrigen Armen betätigt wurde und ist unabhängig von den Resultaten der Aktionen.

Durch diese Formel wird daher ein Tradeoff zwischen Exploration und Exploitation erreicht: Exploitation wird dadurch gewährleistet, dass der durchschnittliche Gewinn positiv in die Bewertung eingeht. Exploration geschieht dadurch, dass selten durchgeführte Aktionen einen Bonus bekommen: der hintere Teil der Formel wird größer.

Mit Hilfe der Hoeffding Ungleichung kann gezeigt werden, dass

$$Pr(\overline{X}_i \geq \mu_i + \sqrt{\frac{2\ln n}{n_i}}) \leq n_i^{-4} \tag{3}$$

$$Pr(\overline{X}_i \leq \mu_i - \sqrt{\frac{2\ln n}{n_i}}) \leq n_i^{-4} \tag{4}$$

ist, wobei μ_i der wahre Erwartungswert der Aktion a_i ist [2, 3].

Das bedeutet, dass die gemessene durchschnittliche Belohnung $\overline{X}_i$ von dem wahren Erwartungswert μ_i mit einer gewissen Wahrscheinlichkeit n_i^{-4} nicht mehr als $\sqrt{\frac{2\ln n}{n_i}}$ abweicht. Das wirklich Wertvolle daran ist, dass die Wahrscheinlichkeit n_i^{-4} in Abhängigkeit von den gespielten Zügen des jeweiligen Arms abnimmt. Je mehr Züge gespielt werden desto kleiner ist die

Wahrscheinlichkeit, dass die gemessene durchschnittliche Belohnung von dem Erwartungswert zu weit abweicht. Es macht also Sinn, mehrere Züge zu spielen, um ein genaueres Ergebnis zu erzielen.

Das Anwenden der Hoeffding Ungleichung setzt allerdings voraus, dass die gespielten Züge unabhängig und gleichverteilt gespielt werden. Das ist für einfache K-armige Banditen der Fall, oder wenn im Stil von Abschnitt 2.1.4 alle folgenden Züge zufällig gespielt werden. In der Anwendung von MCTS ist das allerdings nicht der Fall. Hier ist die Tree Policy dafür zuständig die ersten Züge vor der Simulation zu leiten, sodass leicht zu erkennen ist, dass die Aktionen ausgehend von der Wurzel nicht gleichverteilt und vor allem nicht unabhängig gewählt sind, da die Aktionswahlen der Tree Policy das Prinzip von Exploration vs. Exploitation beachten sollten. Das steht in einem direkten Widerspruch zur Unabhängigkeit der Aktionswahl.

2.2.2 UCT

Die UCB1 Formel ist für das Lösen eines K-armigen Banditen geeignet. Für das Verwenden der UCB1 Formel als Tree Policy in MCTS müssen die nachfolgend gespielten Züge unabhängig und gleichverteilt sein, damit die Konvergenzungleichungen (3) und (4) mit Hilfe der Hoeffding Ungleichung hergeleitet werden können. Das ist aber, wie im vorherigen Abschnitt erläutert, nicht der Fall.

Es konnte allerdings gezeigt werden, dass es möglich ist, die UCB1 Formel anzupassen, sodass trotz der nicht gleichverteilten Aktionswahl eine gleichstarke Konvergenzgleichung hergeleitet werden kann [8]. Schlüssel des Erfolgs ist das Einschränken der Belohnungen in das Intervall $[0, 1]$ und das Hinzufügen einer Konstante zum Gewichten des Explorationsterms.

Die entstandene Formel (UCT - upper confidence for trees [8]) lautet wie folgt:

$$UCT = \overline{X}_i + 2C\sqrt{\frac{\ln n}{n_i}}, \tag{5}$$

wobei $n = \sum_{l=0}^{K} n_l$ die Gesamtanzahl aller Spiele ist.

Es wurde für $C = \frac{1}{\sqrt{2}}$ gezeigt, dass die *Hoeffding Ungleichung* gilt und somit die angepassten Konvergenzungleichungen

$$Pr(\overline{X}_i \geq \mu_i + 2\frac{1}{\sqrt{2}}\sqrt{\frac{\ln n}{n_i}}) \leq n_i^{-4} \tag{6}$$

$$Pr(\overline{X}_i \leq \mu_i - 2\frac{1}{\sqrt{2}}\sqrt{\frac{\ln n}{n_i}}) \leq n_i^{-4} \tag{7}$$

gelten[8].

2.3 Präferenzbasiertes Lernen

Es gibt viele Algorithmen im Bereich des maschinellen Lernens, die zum Lösen von MDPs verwendet werden können. Sie werden teilweise sehr erfolgreich in vielen Bereichen eingesetzt.

Der Informationsgewinn von vielen Erfolgreichen dieser Algorithmen ist allerdings auf eine bestimmte Art von Feedback beschränkt: Sie benötigen ganzzahliges Feedback, um die Güte eines Zustands zu beurteilen oder selbstständig ihren Lernerfolg zu messen.

Das kann in einigen Umgebungen ein natürlicher Kennwert sein. Ein Roboter, der das Fortbewegen erlernen soll, kann seinen Lernerfolg durch die zurückgelegte Strecke einschätzen. Mehr Strecke bedeutet, dass der Roboter sich besser fortbewegt hat.

Wie in Kapitel 1 schon angesprochen, ist das Erlangen von numerischen Gütewerten für Zustände von Problemen in vielen Anwendungsdomänen schwer, fehlerhaft oder unintuitiv. Allerdings ist es in solchen Situationen oftmals möglich zwei Zustände direkt miteinander zu vergleichen und den besseren Zustand zu identifizieren. Ein Beispiel aus der Literatur ist (Roboter-)Fußball, bei welchem ein Tor besser als eine Ecke ist und eine Ecke besser als ein Einwurf ist. Das Zuordnen von numerischen Gütewerten zu den jeweiligen Ereignissen ist allerdings wieder nicht eindeutig[7]. Dass ein Tor den Wert 10, eine Ecke den Wert 5 und ein Einwurf den Wert 1 hat, wäre eine Möglichkeit, aber natürlich nicht die einzige Möglichkeit.

Das präferenzbasierte Lernen ist die Idee, Präferenzen statt numerischen Werten zu verwenden. So gibt es keine Bewertungsfunktion, die einem Zustand einen numerischen Wert zuweist, sondern einen Experten (oder eine Präferenzfunktion), der oder die zu jeweils zwei Zuständen angibt, welcher der Bessere ist.

Für MDPs sind Präferenzen wünschenswert, die eine Aussage über die Gütefunktion Q treffen. Eine gute Präferenz wäre hier $a >_s a' \Leftrightarrow Q^{\pi^*}(s,a) > Q^{\pi^*}(s,a')$, also dass die Aktion a der Aktion a' im Zustand s vorgezogen wird, wenn die Aktion a einen höheren Gütewert als die Aktion a' in s besitzt und π^* eine optimale Strategie ist.

2.3.1 Das duellierende Banditenproblem

Das K-armige duellierende Banditenproblem ist die Anpassung des K-armigen Banditenproblems für präferenzbasiertes Lernen: In jedem Zustand wird nicht eine, sondern werden zwei Aktionen a_i und a_j ausgeführt. Das Resultat ist die Information, welche der beiden Aktionen zu einem besseren Ergebnis geführt hat (oder die Information über Gleichstand).

In diesem Zusammenhang wird eine Aktion als Condorcet-Sieger bezeichnet, wenn die Aktion gegen jede andere Aktion bei unendlich vielen Vergleichen häufiger gewinnt als verliert.

2.3.2 Relative UCB

Ein Algorithmus, der die Idee der oberen Konfidenzschranken auf präferenzbasiertes Feedback und das K-armige duellierende Banditenproblem überträgt, ist *Relative UCB* (RUCB) [15]. Im folgenden Abschnitt wird auf die Funktionsweise dieses Algorithmus eingegangen. Vergleiche dazu [15].

Die Grundidee von RUCB ist, zu jedem Aktionspaar zu speichern, wie oft welches als Sieger aus dem direkten Vergleich hervorgegangen ist. Darauf aufbauend wird mit Hilfe einer abgewandelten UCB Formel

1. die interessanteste Aktion von allen gewählt und
2. die interessanteste Aktion gewählt, um sie mit der erstgewählten Aktion zu vergleichen.

Wobei *interessant* wieder einen Tradeoff zwischen Exploration und Exploitation beschreibt.

Das Vorgehen des Algorithmus ist wie folgt: In einer Matrix $W = [w_{ij}] \leftarrow 0_{K \times K}$ werden die Anzahl der Siege gespeichert. Der Eintrag w_{ij} entspricht der Anzahl, wie oft Aktion i Aktion j geschlagen hat.

Ausgehend von diesen Werten wird eine optimistische Obergrenze u_{ij} zu jedem Aktionspaar berechnet, die angibt wie wahrscheinlich es ist, dass die Aktion i die Aktion j schlägt. Dabei wird die UCB Formel wie folgt abgewandelt verwendet:

$$u_{ij} = \frac{w_{ij}}{w_{ij} + w_{ji}} + \sqrt{\frac{\alpha ln t}{w_{ij} + w_{ji}}}$$

Vorderer und hinterer Teil der Addition sind direkt vergleichbar mit denen der UCB Formel: $\frac{w_{ij}}{w_{ij}+w_{ji}}$ ist die bisher ermittelte Siegchance von i gegenüber j, der für Exploitation sorgt. $\sqrt{\frac{\alpha ln t}{w_{ij}+w_{ji}}}$ ist der Explorationsterm, der einen höheren Wert erreicht, je seltener diese Paarung bisher verglichen wurde. Durch den Parameter α lässt sich die Wichtigkeit der Exploration regeln. Aus Konvergenzgründen ist $\alpha > 0.5$ zu wählen.

Ausgehend von diesen Berechnungen wird versucht, den besten Condorcet-Sieger-Kandidaten zu ermitteln. Zunächst werden alle Condorcet-Sieger-Kandidaten C ermittelt. Dies sind genau diese Aktionen a_i, sodass $a_i \in C \Leftrightarrow \forall j \in \{0, 1, ..., K-1\} : u_{ij} \geq \frac{1}{2}$, wobei $u_{ii} = \frac{1}{2}$ gesetzt wird.

Nun müssen drei Fälle unterschieden werden:

1. $C = \emptyset$
   Gibt es keinen Condorcet-Sieger-Kandidaten, so wird eine Aktion zufällig gewählt.

2. $|C| = 1$
   Nur eine Aktion steht zur Auswahl. Diese Aktion wird gewählt.

3. $|C| > 1$
   Mehrere Aktionen stehen zur Auswahl. Zwischen diesen Aktionen wird eine zufällig bestimmt. Dabei haben alle Aktionen die gleiche Wahrscheinlichkeit, mit einer Ausnahme: Gab es in einer vorherigen Iteration schon einen eindeutigen Condorcet-Sieger-Kandidaten und war er in den darauffolgenden Iterationen stets weiterhin ein Kandidat (also $\in C$), so hat diese Aktion die Wahrscheinlichkeit von 50% gewählt zu werden. Die übrigen Aktionen teilen sich die restlichen 50% zu gleichen Teilen.

Auf diese Weise wird die erste Aktion gewählt: Eine möglichst gute Aktion, von der angenommen werden kann, dass es sich um den Condorcet-Sieger handelt. Aktion 2 wird nun gewählt mit der Idee dieser These, dass Aktion 1 der Condorcet-Sieger ist, zu widersprechen. Es wird die Aktion gesucht, die die beste Siegchance gegen die erste Aktion hat. Das ist allerdings nicht unbedingt die Aktion mit der höchsten aus den bisherigen Experimenten gemessenen Siegchance, sondern auch hier ist der Tradeoff von Exploration versus Exploitation zu beachten: Es kann eine Aktion, die bisher nur einmal gespielt wurde und dabei verloren hat, der eigentliche Condorcet-Sieger sein. Daher wird zum Ermitteln der zweiten Aktion wieder auf die im UCB Stile berechneten u-Werte zurückgegriffen. Die zweite Aktion a_j wird gewählt, sodass $j = argmax_j(u_{ji})$.

Sind beide Aktionen a_i und a_j ermittelt, so wird der duellierende-Bandit mit ihnen gespielt. Entsprechend des Ergebnisses wird der Wert w_{ij} oder w_{ji} um eins erhöht.

3 Präferenzbasiertes MCTS

Die Idee dieser Arbeit und die in Abschnitt 1.2 genannte Problemstellung ist, die beiden genannten Themen *MCTS* und *präferenzbasiertes Lernen* zu verknüpfen. Im Folgenden wird ein Algorithmus beschrieben, der MCTS mit RUCB verknüpft und ausschließlich auf präferenzbasiertem Feedback arbeitet. Somit ist dieses Kapitel als Behandlung der Problemstellung *Wie lässt sich Realtime MCTS auf die Verwendung von Präferenzen anpassen?* zu sehen.

Im Prinzip gilt es MCTS an eine Umgebung anzupassen, in der es ausschließlich Präferenzen gibt und keine sonstige Bewertung von Zuständen verfügbar sind.

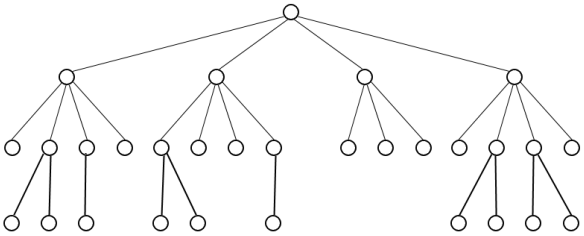
3.1 Vergleich zu MCTS

MCTS wurde in den vorgerigen Kapiteln wie folgt hergeleitet:

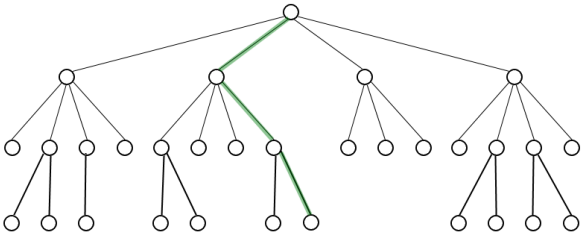
1. Betrachten des K-armigen Banditenproblems
2. Herleiten der UCB Formel zum Lösen dieses Problems
3. Erweitern der UCB Formel auf Bäume (Anwenden des Banditenproblems in jedem Knoten separat)
4. Definieren des Ablaufs:
   a) Angefangen bei der Wurzel: Wähle iterativ eine Aktion mit der UCT Formel, bis ein Folgeknoten nicht mehr im Baum ist.
   b) Füge diesen einen Knoten dem Baum hinzu.
   c) Bewerte den zugehörigen Zustand durch Monte Carlo Simulation.
   d) Backpropagiere diese Bewertung und speichere diese in jedem Knoten von dem neu Erstellten bis hin zur Wurzel.

In einer präferenzbasierten Umgebung ist dieses Schema nicht anwendbar. Wie im letzten Kapitel beschrieben ist nicht das K-armige Banditenproblem, sondern das K-armige duellierende Banditenproblem zu betrachten, wenn durch Präferenzen gelernt werden soll. Das führt zu einer grundlegenden Änderung des Ablaufs, da in einem Zustand nicht eine Aktion, sondern zwei Aktionen parallel ausgeführt werden. Erweitert man dieses Vorgehen auf Bäume, so werden in jedem gewählten Knoten des Baumes zwei Kindknoten gewählt. Somit ergibt sich kein einfacher Pfad von der Wurzel zu einem neu expandierten Knoten, wie es in MCTS der Fall ist. Statt einen Pfad aus dem Baum zu wählen (Punkt 4.a), wird ein Unterbaum selektiert, genauer ein binärer Unterbaum, da in jedem von seinem Elternknoten selektiertem Knoten genau zwei Kindknoten ausgewählt werden. Das geschieht so lange, bis die Blätter des Binärbaums noch nicht expandierte Kinder eines Blattknotens des bisher aufgespannten Suchbaums sind. Somit ergeben sich mehrere Knoten, die noch nicht im Baum existent sind (Vergleiche Abbildung 5c). Verfolgt man weiterhin das oben genannte MCTS-Schema, so werden all diese Blätter des Binärbaums dem ursprünglichen Baum hinzugefügt (Punkt 4.b). Nun müssen noch durch Monte Carlo Simulation und Präferenzen die Bewertungen der Aktionen von Knoten entsprechend aktualisiert werden (Punkt 4.c und 4.d).

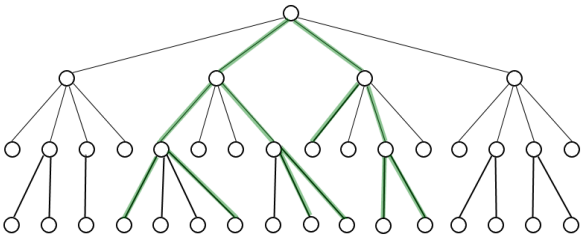
Eine Monte Carlo Simulation wird ausgehend von jedem neu hinzugefügten Blattknoten gestartet und führt zu einem Folgezustand des entsprechenden Zustands des Blattknotens. Dieser



(a) Ein nicht vollständig erkundeter Baum.



(b) Ein Pfad von der Wurzel aus bis zu einem Knoten, der neu in den Baum aufgenommen wird.



(c) In jedem Knoten werden zwei Kinder gewählt bis die Blätter der Auswahl neue Knoten sind, die in den Baum aufgenommen werden. Diese Knoten müssen nicht auf gleicher Höhe des Baums sein.

Abbildung 5: Unterschiedliche Selektionsverfahren: Pfad und Baum.

simulierte Zustand wird verwendet, um den Zustand des Blattknotens in dieser Iteration zu bewerten. Bleibt die Frage, welche dieser Zustände miteinander verglichen werden und wie das Ergebnis verwaltet werden soll.

Betrachtet man sich einen Knoten v des Binärbaums, so gibt es zu jeder der beiden gewählten Aktionen a_i und a_j jeweils eine Menge von daraus folgenden Blattknoten des Binärbaums. Diese Mengen werden $\overline{B}_{vi}$ und $\overline{B}_{vj}$ genannt. Durch die Monte Carlo Simulation jedes Blattknotens kann somit auch die Menge der simulierten Zustände der entspechenden Blattknoten gebildet werden: B_{vi} und B_{vj}. B_{vj} ist entsprechend die Menge der simulierten Zustände, ausgehend von Blattknoten von dem Teils des Binärbaums, der sich ausgehend vom Knoten v mit Aktion a_j aufspannt. Diese Mengen sind nicht leer und sind größer, je größer der entsprechende Teil des Binärbaums ist.

Betrachtet man nun die Zustandspaare $(s_l, s_m) \in B_{vi} \times B_{vj}$ so ergeben sich entsprechend $|B_{vi}| \cdot |B_{vj}|$ viele Vergleiche der Aktionen a_i und a_j. Durch Präferenzen kann nun ermittlet werden, wie oft Aktion a_i der Aktion a_j in dieser Iteration überlegen oder unterlegen war. Diese Präferenzen können genutzt werden, um in folgenden Iterationen die Güte von a_i gegenüber a_j zu bewerten.

3.2 Nutzung von Relative UCB

Relative UCB lässt sich in diesem Zusammenhang wie folgt anwenden:

Pro Knoten des Baums wird eine eigene Instanz von RUCB gespeichert. Für jede Aktion a_i des entsprechenden Zustands des Knoten wird für jede andere dieser Aktionen a_j in w_{ij} gespeichert, wie oft die Aktion a_i der Aktion a_j überlegen war (Verleiche Abschnitt 2.3.2). Somit kann RUCB jederzeit zwei Aktionen auswählen, die miteinander zu vergleichen sind.

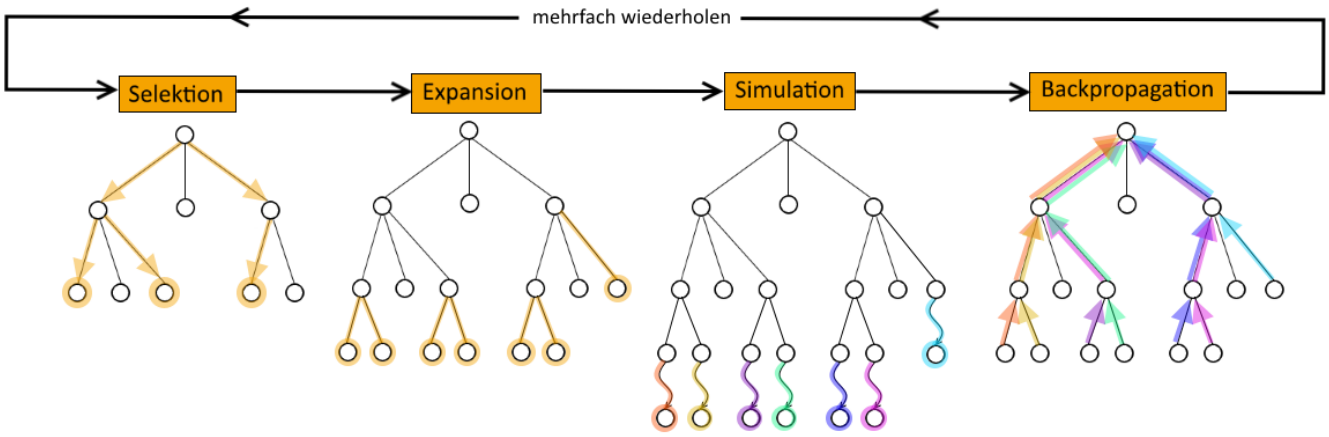


Abbildung 6: Der Ablauf von Relative MCTS. Er unterscheidet sich von dem MCTS Ablauf.

Im Backpropagation-Schritt werden die Mengen der Monte Carlo Simulationen B_{vi} und B_{vj} der beiden Aktionen a_i und a_j zurückgeliefert. Durch eine paarweise Zusammensetzung der Zustände dieser Mengen ergeben sich Zustandspaare (s_l, s_m). Auf Basis von Präferenzen kann bestimmt werden, wie oft Aktion a_i gegenüber Aktion a_j in den Simulationen überlegen war. Der Wert w_{ij} erhöht sich um diese Anzahl. Entsprechend wird w_{ji} um die Anzahl erhöht, wie oft die Aktion a_j der Aktion a_i überlegen war.

Anpassung der Relative UCB Formel für die Baumsuche

Betrachtet man die Formeln von UCB und der Variante für die Baumsuche UCT, so ist der Unterschied eine Konstante zum Gewichten des Explorationstermes:

$$UCB1 = \overline{X}_i + \sqrt{\frac{2 \ln n}{n_i}} \tag{8}$$

$$UCT = \overline{X}_i + 2C\sqrt{\frac{\ln n}{n_i}} \tag{9}$$

Diese Anpassung ist bei der Verwendung von Relative UCB nicht notwendig, da das Hinzufügen eines solchen Parameters wie folgt umgeschrieben werden kann:

$$\begin{aligned} u_{ij} &= \frac{w_{ij}}{w_{ij} + w_{ji}} + 2C\sqrt{\frac{\alpha \ln t}{w_{ij} + w_{ji}}} \\ &= \frac{w_{ij}}{w_{ij} + w_{ji}} + \sqrt{\frac{4C^2 \alpha \ln t}{w_{ij} + w_{ji}}} \\ &= \frac{w_{ij}}{w_{ij} + w_{ji}} + \sqrt{\frac{\hat{\alpha} \ln t}{w_{ij} + w_{ji}}} \end{aligned} \tag{10}$$

Damit ändert sich die Parametrisierung von $0 < C$ und $\frac{1}{2} < \alpha$ in $0 < \hat{\alpha}$.

3.3 Bedingte binäre Splits

Relative MCTS benötigt die Ausführung von zwei Aktionen in einem Knoten, um die Güte der unterschiedlichen Aktionen miteinander zu vergleichen. Somit entsteht ein Baum an Simulationen anstatt eines Pfades wie bei Realtime MCTS. Einen Baum an Simulationen zu berechnen ist mehr Aufwand als einen einzigen Pfad zu berechnen.

Selbst wenn eine Aktion die restlichen Aktionen klar dominiert, wird stets eine der dominierten Aktionen getestet. Das macht einerseits Sinn, da nur so bessere Aktionen gefunden werden können, die fälschlicherweise zu schlecht bewertet wurden. Andererseits kosten die von dieser Aktion ausgehenden Simulationen kostbare Rechenzeit.

Bedingt Relatives MCTS ist ein Ansatz, der durch Angabe eines Konfidenzniveaus ermittelt, ob in einem gegebenen Knoten nur die beste Aktion ausgewählt werden soll oder ob eine weitere Aktion ausgeführt werden soll. Die Hoffnung ist, dass durch weniger binäre Splits des Simulationsbaums mehr Iterationen in der gegebenen Rechenzeit durchgeführt werden können und so ein besseres Ergebnis erzielt werden kann. Dagegen steht die Gefahr, dass eine suboptimale Aktion als Condorcet-Sieger angenommen wird und seine Alternativen nicht weiter getestet werden.

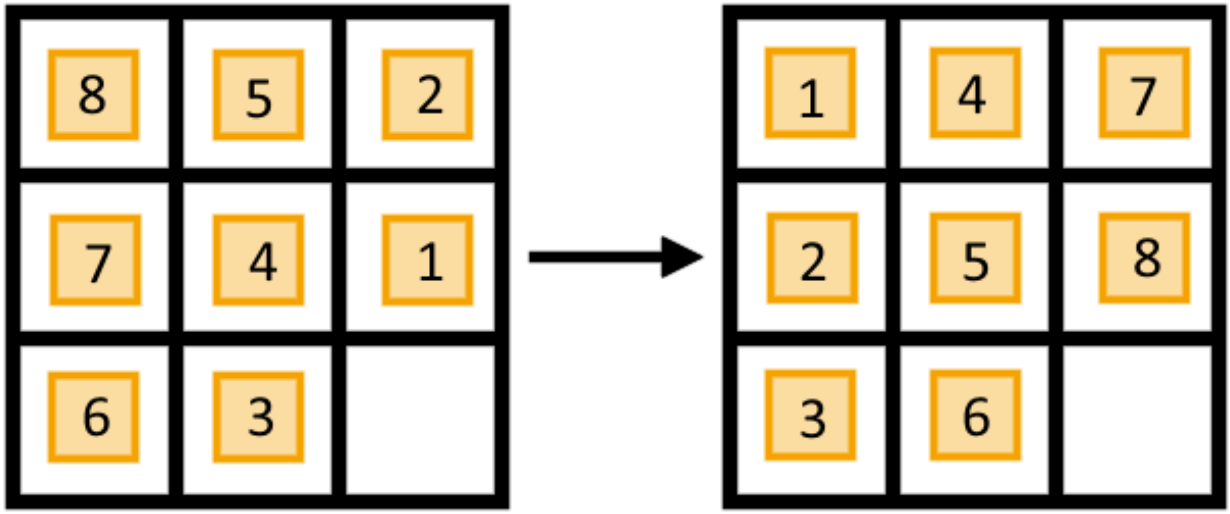


Abbildung 7: Die Aufgabe von 8Puzzle ist es, aus dem Urprungszustand (links) in den Endzustand (rechts) zu gelangen. Ein Zug ist das Wechseln der Position des leeren Kästchens mit einer direkt angrenzenden Zahl (rechts, links, oben oder unten)

4 Experiment Setup

Zum Bewerten des präferenzbasierten MCTS wird er mit Realtime MCTS verglichen. Dies geschieht auf einer festgelegten Domäne, dem 8Puzzle.

4.1 8Puzzle

Das 8Puzzle ist ein Spiel mit quadratischem Spielbrett mit neun Feldern, die mit acht Zahlen und einem freien Platz belegt sind. Die Aufgabe des Spiels ist, jede Zahl auf ein vorher festgelegtes Kästchen zu verschieben. Dazu kann das leere Kästchen auf ein benachbartes Feld (horizontal oder vertikal) verschoben werden, womit es mit der dort liegenden Zahl den Platz tauscht.

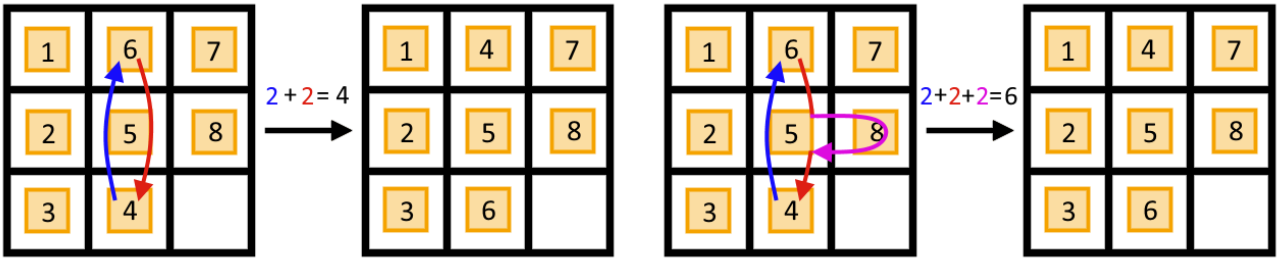
In dieser Arbeit soll der Spielzustand (von oben nach unten und links nach rechts) 8-7-6-5-4-3-2-1-[frei] in 1-2-3-4-5-6-7-8-[frei] überführt werden [Siehe Abbildung XY]. Dazu stehen 99 Züge zur Verfügung. Wird innerhalb dieser Züge das Spiel in den Zielzustand überführt, so gilt das Spiel als gewonnen (Ergebnis: 1) ansonsten ist es verloren (Ergebnis: 0).

Als Heuristik wird die Manhattan-Distanz mit linearen Konflikten verwendet. Diese Heuristik gibt eine echte untere Schranke für die notwendigen Züge bis zum Erreichen des Zielzustands an und ist somit zulässig. Diese Anzahl der Züge wird auch Distanz genannt. Die Distanz zwischen zwei Zuständen ist somit die Anzahl der Züge, die benötigt werden, um vom einen Zustand in den anderen zu gelangen.

Berechnet wird die Manhattan-Distanz mit linearen Konflikten in zwei Schritten:

1. Berechnen der Manhattan-Distanz
2. Hinzufügen von Strafdistanzen für lineare Konflikte

Die Manhattan-Distanz lässt sich wie folgt ermitteln: Für jede Zahl wird die Anzahl der notwendigen Züge ermittelt, die benötigt werden, um diese Zahl auf ihren Zielort zu verschieben.



(a) Manhattan-Distanz **(b)** Manhattan-Distanz mit linearen Konflikten

Abbildung 8: Die Manhattan-Distanz ist die Summe der mindestens notwendigen Züge pro Zahl. Zusätzlich zur Manhattan-Distanz wird bei linearen Konflikten die Notwendigkeit beachtet, dass eine Zahl einer anderen Zahl Platz machen muss, damit beide aneinander vorbei können.

Befindet sich die 7 beispielsweise unten links im Spielfeld und muss nach oben rechts, so muss sie zwei mal nach oben und zwei mal nach rechts verschoben werden, womit sich für die 7 eine Distanz von vier ergibt. Die Manhattan-Distanz ist die Summe dieser Werte über alle Zahlen.

Diese Schätzung ist eine untere Grenze der wirklich benötigten Züge, da mit jedem Zug genau eine Zahl verschoben wird (den Platz mit dem leeren Feld tauscht). Jede Zahl muss mindestens so oft verschoben werden, wie es in der Manhattan-Distanz berechnet wird. Somit überschätzt die Manhattan-Distanz die Anzahl der wirklich benötigten Züge niemals.

Allerdings unterschätzt die Manhattan-Distanz die wirkliche Distanz in vielen Situationen. Das ist nicht erwünscht.

Eine dieser Situationen ist, wenn ein linearer Konflikt vorliegt: Zwei Zahlen i und j stehen in einem linearen Konflikt zueinander, wenn die Zielorte dieser Zahlen in der gleichen Spalte (oder Zeile) sind, sich beide Zahlen in dieser Spalte (oder Zeile) befinden und die relative Anordnung der Zahlen aktuell nicht die gleiche ist, wie die relative Anordnung der Zielorte der Zahlen. Das bedeutet es besteht beispielsweise ein linearer Konflikt, wenn i aktuell rechts neben j steht, i aber links neben j stehen muss. Um diese Situation aufzulösen, müssen auf jeden Fall zwei Züge mehr gespielt werden als die Manhattan-Distanz angibt: Eine der Zahlen muss auf eine andere Spalte (oder Zeile) verschoben werden, damit sie an der anderen Zahl vorbeikommt. Das ergibt mindestens zwei zusätzliche Züge pro linearem Konflikt: das Ziehen auf eine Ausweichspalte (-zeile) und das Zurückziehen in die Zielspalte (-zeile).

Die Anzahl der Paare mit linearem Konflikt wird ermittelt. Der Strafterm der linearen Konflikte ist zweimal die Anzahl der linearen Konflikte. Die Distanz erhöht sich somit pro linearem Konflikt um zwei.

Diese Heuristik (Manhattan-Distanz mit linearen Konflikten) ist im Folgenden die verwendete Heuristik.

Um präferenzbasiertes Lernen zu verwenden, werden Präferenzen benötigt. Aus einer eindimensionalen Heuristik H lässt sich eine Präferenz P erzeugen, indem $P(a, b) = a$, wenn $H(a) < H(b)$, $P(a, b) = b$, wenn $H(b) < H(a)$ und keine Präferenz bei $H(b) = H(a)$ existiert. In dieser Arbeit wurde solch eine Präferenz verwendet. Genauer die generierte Präferenz der verwendeten Heuristik (Manhattan-Distanz mit linearen Konflikten).

4.2 Die Algorithmen

Zwischen dem präferenzbasierten MCTS und Realtime MCTS gibt es diverse Unterschiede.
Als Hauptunterschiede wurden folgende Punkte ausgemacht:

- Unterschiedliche Expandierungsstrategie und daraus resultierender Unterschied im Ablauf einer Iteration.
- Informationsverlust der Heuristik: Die Präferenz bewertet nur besser, gleich oder schlechter. Wie viel besser oder wie viel schlechter ein Zustand als ein anderer Zustand bewertet wird, wird nicht mit angegeben.
- Die Nutzung der RUCB Formel gegenüber UCT.

Um nicht nur zu ermitteln, welcher Algorithmus auf der gewählten Domäne besser funktioniert, sondern weshalb es so ist, werden weitere Algorithmen eingeführt, die diese Unterschiede teilweise nicht haben.
Daraus resultieren folgende Algorithmen:

- Realtime MCTS mit Nutzung der Heuristik bei nicht terminalen Zuständen.
- Parallel MCTS, das der parallelen Expandierung von Relative MCTS gleich ist, allerdings weiterin die UCT Formel und numerische Heuristik verwendet.
- Binary MCTS, welches ähnlich wie Parallel MCTS funktioniert. Hier werden hier die Entscheidungen der UCT Formel nicht durch numerische Werte der Heuristik bestimmt, sondern nur durch binäres Feedback, welche Aktion der aktuellen Iteration welche andere Aktion geschlagen hat.
- Relative MCTS, wie oben beschrieben.
- Bedingtes Relative MCTS: Relative MCTS mit der Einschränkung nur dann einen Binärsplit zu machen, wenn der Condorcet-Sieger nicht mit festgelegtem Konfidenzniveau ermittelt werden konnte. Wenn kein Split geschieht, wird der beste Condorcet-Sieger-Kandidat gewählt.

Diese Algorithmen werden im Folgenden detailliert beschrieben und die Unterschiede weiter hervorgehoben.

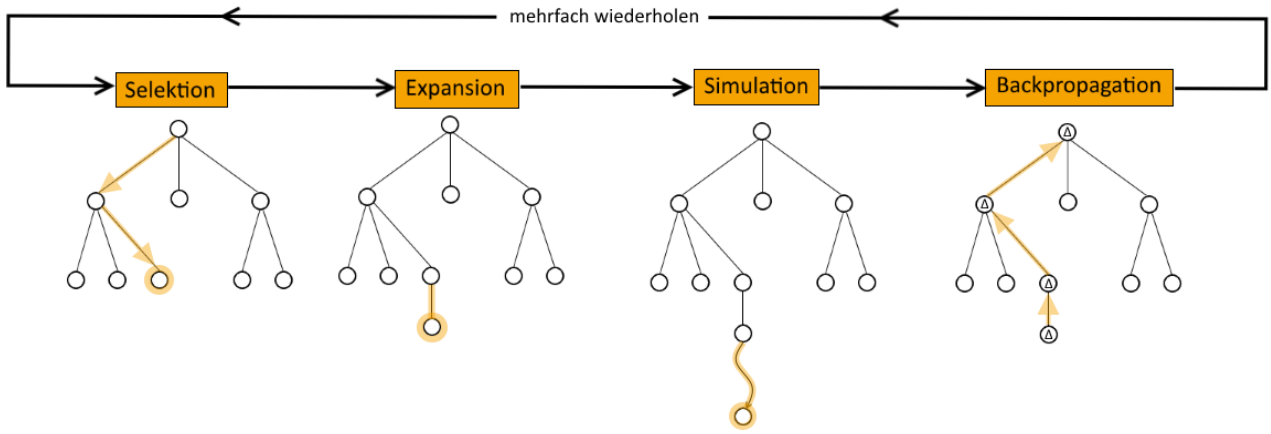


Abbildung 9: Der Ablauf von Realtime MCTS.

4.2.1 Realtime MCTS

Dieser Algorithmus entspricht der Realtime Variante, wie sie in Kapitel 2.1.8 beschrieben wurde. Zum Vergleich hier nochmals ein kurzer Ablauf:

- Selektion
  Ausgehend von der Wurzel wird pro selektiertem Knoten wiederholend *das beste* Kind ausgewählt, bis ein Kind nicht vollständig expandiert ist. Zum Bewerten der Kinder wird die UCT Formel verwendet.

- Expansion
  Dem nicht vollständig expandierten Knoten wird ein Kind hinzugefügt, indem zufällig eine Aktion gewählt wird, die noch keinen Nachfolgeknoten im Baum besitzt.

- Simulation
  Ausgehend von dem neu hinzugefügten Knoten wird eine Simulation gestartet, die zufällig Aktionen ausführt, bis entweder ein terminaler Zustand erreicht wird oder eine Obergrenze für die Anzahl der zufälligen Züge erreicht ist.

- Backpropagation
  Der letzte Zustand der Simulation wird heuristisch bewertet. Diese Bewertung wird verwendet, um die UCT Formel in allen Knoten von dem neu hinzugefügten bis hoch zur Wurzel zu aktualisieren.

Diese Konfiguration besitzt folgende Parameter, die in den angegebenen Wertbereichen getestet wurden:

- Simulationslänge: 5, 10, 15, 20, 50, 70, 100($\equiv$ bis Terminal)

- UCT Parameter C: 10 äquidistante Werte von (jeweils inklusive) 0.5 bis 3:
  $\frac{1}{2} = 0.5$, $\frac{7}{9} = 0.\overline{7}$, $\frac{19}{18} = 1.0\overline{5}$, $\frac{4}{3} = 1.\overline{3}$, $\frac{29}{18} = 1.6\overline{1}$, $\frac{17}{9} = 1.\overline{8}$, $\frac{13}{6} = 2.1\overline{6}$, $\frac{22}{9} = 2.\overline{4}$, $\frac{49}{18} = 2.7\overline{2}$, $\frac{3}{1} = 3.0$

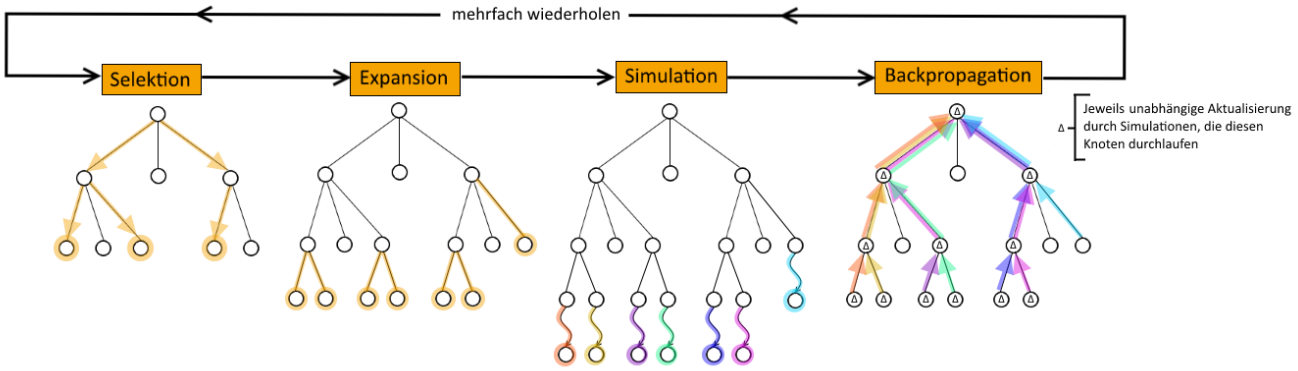


Abbildung 10: Der Ablauf von Parallel MCTS.

4.2.2 Parallel MCTS

Dieser Algorithmus läuft mit den Formeln von Realtime MCTS. Die Abläufe sind allerdings anders, sodass mehrere Knoten pro Iteration in den Baum aufgenommen werden.

- Selektion
  Ausgehend von der Wurzel werden pro selektiertem Knoten wiederholend *die zwei besten* Kinder ausgewählt, bis ein Kind nicht vollständig expandiert ist. Zum Bewerten der Kinder wird die UCT Formel verwendet.

- Expansion
  Den nicht vollständig expandierten Knoten werden jeweils zwei Kinder hinzugefügt, indem zufällig zwei Aktionen gewählt werden, die von diesem Knoten noch keinen Nachfolgeknoten im Baum besitzen.

- Simulation
  Ausgehend von den neu hinzugefügten Knoten wird jeweils eine Simulation gestartet, die zufällig Aktionen ausführt bis entweder ein terminaler Zustand erreicht wird oder die Obergrenze der Anzahl der zufälligen Züge erreicht ist.

- Backpropagation
  Der letzte Zustand der Simulation wird heuristisch bewertet. Diese Bewertung wird verwendet, um die UCT Formel in allen Knoten von dem jeweils zugehörigen neu hinzugefügten Knoten bis hoch zur Wurzel zu aktualisieren. Die unterschiedlichen Simulationen können iterativ nacheinander abgearbeitet werden. Ihre Backpropagation ist unabhängig von den anderen Simulationen in der gleichen Iteration.

Diese Konfiguration besitzt folgende Parameter, die in den angegebenen Wertbereichen getestet wurden:

- Simulationslänge: 5, 10, 15, 20, 50, 70, 100(≡ bis Terminal)

- UCT Parameter C: 10 äquidistante Werte von (jeweils inklusive) 0.5 bis 3:
  $\frac{1}{2} = 0.5$, $\frac{7}{9} = 0.\overline{7}$, $\frac{19}{18} = 1.0\overline{5}$, $\frac{4}{3} = 1.\overline{3}$, $\frac{29}{18} = 1.6\overline{1}$, $\frac{17}{9} = 1.\overline{8}$, $\frac{13}{6} = 2.1\overline{6}$, $\frac{22}{9} = 2.\overline{4}$, $\frac{49}{18} = 2.7\overline{2}$, $\frac{3}{1} = 3.0$

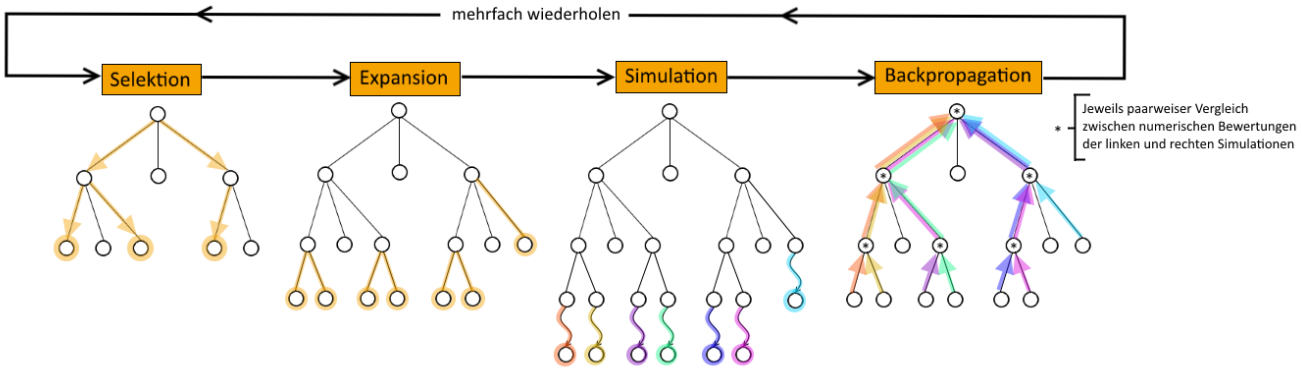


Abbildung 11: Der Ablauf von Binary MCTS.

4.2.3 Binary MCTS

- Selektion
  Ausgehend von der Wurzel werden pro selektiertem Knoten wiederholend *die zwei besten* Kinder ausgewählt, bis ein Kind nicht vollständig expandiert ist. Zum Bewerten der Kinder wird die UCT Formel verwendet.

- Expansion
  Den nicht vollständig expandierten Knoten werden jeweils zwei Kinder hinzugefügt, indem zufällig zwei Aktionen gewählt werden, die von diesem Knoten noch keinen Nachfolgeknoten im Baum besitzen.

- Simulation
  Ausgehend von den neu hinzugefügten Knoten wird jeweils eine Simulation gestartet, die zufällig Aktionen ausführt bis entweder ein terminaler Zustand erreicht wird oder die Obergrenze der Anzahl der zufälligen Züge erreicht ist.

- Backpropagation
  Der letzte Zustand der Simulation wird heuristisch bewertet. Diese Bewertung wird verwendet, um die UCT Formel in allen Knoten von dem jeweils zugehörigen neu hinzugefügten Knoten bis hoch zur Wurzel zu aktualisieren. Dies geschieht allerdings in Abhängigkeit von den anderen Simulationen der gleichen Iteration. Ein schon vor der Iteration im Baum existenter Knoten, der mindestens einmal in der Selektionsphase ausgewählt wurde, bekommt mehrere Simulationen backpropagiert. Für jedes Simulationspärchen von Aktion a_i und a_j mit $i \neq j$, die zu seinen entsprechenden Kindern führen, wie es die UCT Formel in der Selektionsphase ermittelt hat, wird anhand der Bewertung der zugehörigen Simulationen die bessere Aktion ermittelt. Die bessere Aktion wird mit einem Reward von 1, die schlechtere mit einem Reward von 0 und im Falle eines Gleichstands werden beide mit 0.5 bewertet.

Diese Konfiguration besitzt folgende Parameter, die in den angegebenen Wertbereichen getestet wurden:

- Simulationslänge: 5, 10, 15, 20, 50, 70, 100($\equiv$ bis Terminal)

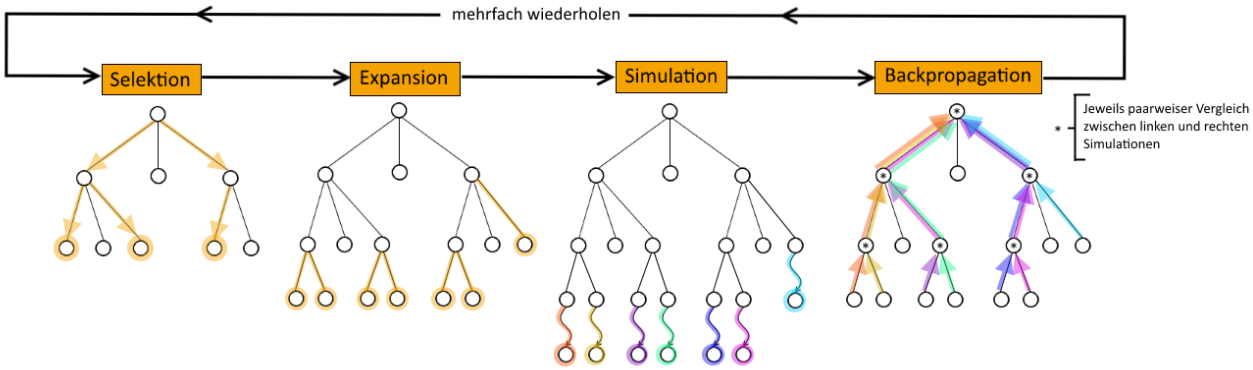


Abbildung 12: Der Ablauf von Relative MCTS.

- UCT Parameter C: 10 äquidistante Werte von (jeweils inklusive) 0.5 bis 3:
  $\frac{1}{2} = 0.5$, $\frac{7}{9} = 0.\overline{7}$, $\frac{19}{18} = 1.0\overline{5}$, $\frac{4}{3} = 1.\overline{3}$, $\frac{29}{18} = 1.6\overline{1}$, $\frac{17}{9} = 1.\overline{8}$, $\frac{13}{6} = 2.1\overline{6}$, $\frac{22}{9} = 2.\overline{4}$, $\frac{49}{18} = 2.7\overline{2}$, $\frac{3}{1} = 3.0$

4.2.4 Relative MCTS

- Selektion
  Ausgehend von der Wurzel werden pro selektiertem Knoten wiederholend *die zwei besten* Kinder ausgewählt, bis ein Kind nicht vollständig expandiert ist. Zum Bewerten der Kinder wird die RUCB-Formel verwendet.

- Expansion
  Den nicht vollständig expandierten Knoten werden jeweils zwei Kinder hinzugefügt, indem zufällig zwei Aktionen gewählt werden, die von diesem Knoten noch keinen Nachfolgeknoten im Baum besitzen.

- Simulation
  Ausgehend von den neu hinzugefügten Knoten wird jeweils eine Simulation gestartet, die zufällig Aktionen ausführt bis entweder ein terminaler Zustand erreicht wird oder die Obergrenze der Anzahl der zufälligen Züge erreicht ist.

- Backpropagation
  Der letzte Zustand der Simulation wird heuristisch bewertet. Diese Bewertung wird verwendet, um die RUCB-Formel in allen Knoten von dem neu hinzugefügten bis hoch zur Wurzel zu aktualisieren. Vergleiche Abschnitt 3.2.

Diese Konfiguration besitzt folgende Parameter, die in den angegebenen Wertbereichen getestet wurden:

- Simulationslänge: 5, 10, 15, 20, 50, 70, 100(≡ bis Terminal)

- Relative UCT α: 15 äquidistante Werte von (jeweils inklusive) 0.1 bis 1.5:
  0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5,

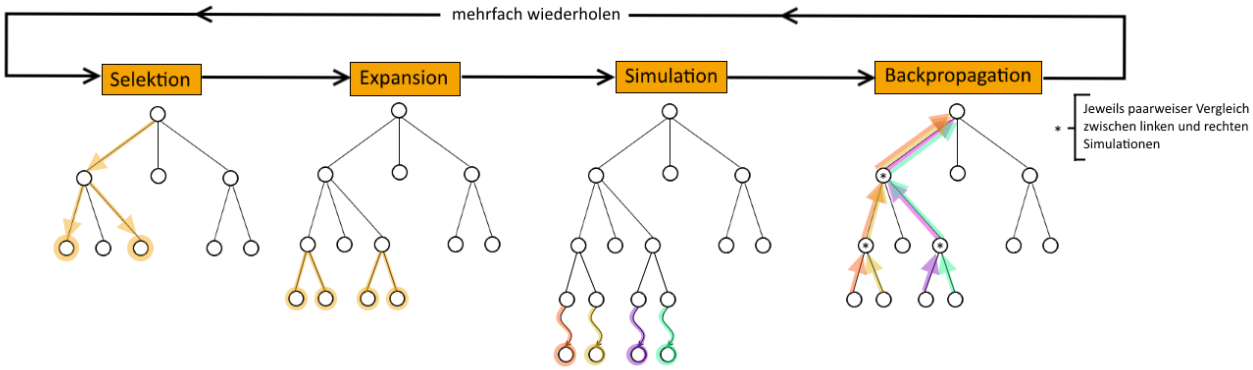


Abbildung 13: Der Ablauf von bedingtem Relative MCTS.

4.2.5 Bedingt Relative MCTS

- Selektion
  Pro selektiertem Knoten, ausgehend von der Wurzel, wird versucht den besten Condorcet-Sieger-Kandidaten zu ermitteln. Abhängig von einem parametrisierten Konfidenzniveau und einer Mindestanzahl von bisherigen Vergleichen wird bestimmt, ob es sich lohnt, in diesem Knoten einen Binärsplit zu machen oder nicht. Entsprechend der Mindestanzahl und dem Ausgang des Konfidenztests wird *das Beste* oder werden *die zwei besten* Kinder ausgewählt, bis ein Kind nicht vollständig expandiert ist. Zum Bewerten der Kinder wird die RUCB Formel verwendet.

- Expansion
  Den nicht vollständig expandierten Knoten werden jeweils zwei Kinder hinzugefügt, indem zufällig zwei Aktionen gewählt werden, die von diesem Knoten noch keinen Nachfolgeknoten im Baum besitzen.

- Simulation
  Ausgehend von dem neu hinzugefügten Knoten wird eine Simulation gestartet, die zufällig Aktionen ausführt, bis entweder ein terminaler Zustand erreicht wird oder eine Obergrenze der Anzahl der zufälligen Züge erreicht ist.

- Backpropagation
  Der letzte Zustand der Simulation wird heuristisch bewertet. Diese Bewertung wird verwendet um die RUCB-Formel in allen Knoten von dem neu hinzugefügten bis hoch zur Wurzel zu aktualisieren. Vergleiche Abschnitt XY, wobei im Falle von keinem Binärsplit keine Aktualisierung stattfindet.

Diese Konfiguration besitzt folgende Parameter, die in den angegebenen Wertenbereichen getestet wurden:

- Simulationslänge: hier wird der ermittelte beste Wert für RUCB verwendet.
- Relative UCT α: hier wird der ermittelte beste Wert für RUCB verwendet.
- Signifikanzniveau, um den Condorcet-Sieger-Kandidaten nicht gegenzutesten (für keinen Split): 0.6, 0.7, 0.8, 0.9, 0.95, 0.98, 0.99.

- Die minimale Anzahl von Vergleichen, die schon gemacht worden sein müssen, bevor durch den Signifikanztest ein Condorcet-Sieger ermittelt werden kann: 0, 5, 10, 15, 20, 50

4.3 Auswertung

Die Auswertung der oben genannten Algorithmen geschieht durch das Anwenden auf das 8Puzzle-Problem. Dabei wird jeder Algorithmus für jede Konfiguration, also jede mögliche Zusammenstellung der Parameter, wiederholt (100x) auf das 8Puzzle-Problem angewandt. Als Instanz wird dabei ein Spiel einer Konfiguration eines Algorithmus beschrieben. Beispielsweise werden für Realtime MCTS $7 \cdot 10 \cdot 100 = 7000$ Instanzen gespielt.

Das Spielen von Instanzen läuft schematisch immer gleich ab:

1. Zustand $s \leftarrow$ Startzustand
2. Solange nicht 99 Züge gespielt wurden:
   a) Wende den entsprechenden Algorithmus mit entsprechender Konfiguration auf s an.
   b) Nach 2 Sekunden: Beende Algorithmus und ermittle beste Aktion a.
   c) Spiele Aktion a. Neuer Zustand wird in s gespeichert.
   d) Wenn s der Endzustand ist: Beende Instanz mit Ergebnis 1.
3. Beende Instanz mit Ergebnis 0

Zu bemerken ist, dass in Punkt 2a) der Algorithmus jedes mal neu gestartet wird. Es werden keine Informationen aus einem vorherigen Lauf übernommen und der Spielbaum wird ausgehend von s immer neu aufgespannt.

Für jeden Algorithmus werden zunächst alle Parameterkombinationen ausgewertet, um die optimale Konfiguration zu ermitteln. Die optimale Konfiguration wird ermittelt, indem nach folgenden Kriterien sortiert wird:

1. Anzahl der gewonnenen Spiele
2. Durchschnittliche Anzahl an Zügen der gewonnenen Spiele
3. Durchschnittliche heuristische Bewertung aller Endzustände

Ist die optimale Konfiguration von jedem Algorithmus gegeben, werden die Güten der unterschiedlichen Algorithmen untereinander verglichen. Durch einen solchen direkten Vergleich der Algorithmen untereinander lassen sich die Fragestellungen der Arbeit beantworten: Das Auswerten der Algorithmen Realtime MCTS und Relative MCTS, insbesondere ein Zusammenhang zwischen der Anzahl der gewonnenen Spiele und der durchschnittlichen heuristischen Bewertung der Nicht-Siegzustände, gibt eine Antwort auf die Fragestellung:

> **Fragestellung:** *Wie wirken sich Informationsverlust und algorithmische Anpassung von Realtime MCTS auf dessen Güte aus?*

Durch das Vergleichen von diesen beiden Algorithmen mit den Algorithmen Binäres MCTS und Paralleles MCTS lassen sich Auswirkungen von binären Splits, binären Splits mit binären Vergleichen und binären Splits mit Anwendung von Relative UCB beurteilen und somit eine Antwort auf die folgende Fragestellung finden:

> **Fragestellung:** *Lässt sich durch das Übersetzen von heuristischen Bewertungen in Präferenzen in Realtime MCTS ein Gewinn erzielen?*

Für die Beantwortung dieser Frage, siehe Abschnitt 5.2.

Framework

Die Auswertung geschieht mit Hilfe des ECLiPSe Frameworks[1], da es sich hier um ein gut bewährtes Framework zum Anwenden von Algorithmen auf diverse Problemstellungen und Spiel handelt.

[1] Siehe http://www.eclipseclp.org/

5 Resultate der Vergleiche und Auswertung

Nachfolgend werden die Resultate der Auswertung aufgezeigt und beschrieben. Zunächst werden die unterschiedlichen Algorithmen einzeln betrachtet und die optimalen Konfigurationen ermittelt. Darauf aufbauend werden die Algorithmen untereinander verglichen, wobei jeweils die optimale Konfiguration verwendet wird. Abschließend werden die offenen Fragestellungen bearbeitet.

5.1 Ermitteln der optimalen Konfiguration pro Algorithmus

In diesem Abschnitt wird für jeden Algorithmus die optimale Konfiguration ermittelt. Dabei werden die unterschiedlichen Konfigurationen, wie sie im Kapitel 4.2 aufgezeigt sind, anhand der in Kapitel 4.3 aufgezeigten Güte sortiert:

1. Anzahl der gewonnenen Spiele
2. Durchschnittliche Anzahl an Zügen der gewonnenen Spiele
3. Durchschnittliche heuristische Bewertung aller Endzustände

Diese Werte werden aufgearbeitet dargestellt und interessante Zusammenhänge aufgezeigt. Im nachfolgenden Abschnitt 5.2 werden die Algortihmen verglichen.

Platz	Konfiguration	C	Länge	# Siege	⌀ Ticks (gewonnen)	⌀ Heuristik
1	27	$1.\overline{3}$	50	21	61.19	0.802
2	15	$0.\overline{7}$	100	20	72	0.785
3	21	$1.0\overline{5}$	70	20	74.4	0.768
4	63	$2.7\overline{2}$	70	19	68.37	0.734
5	14	$0.\overline{7}$	70	19	68.68	0.744

Tabelle 1: Die besten fünf Konfigurationen von Realtime MCTS

5.1.1 Realtime MCTS

Die beste Konfiguration von Realtime MCTS löst das 8Puzzle in 21 von 100 Instanzen. Dabei ist der C-Parameter mit $1,\overline{3}$ und die Rollout-Length mit 50 gesetzt. Interessant ist, dass dieser C-Parameter dem in Kapitel 2.2.2 genannten Parameter von $C = \frac{1}{\sqrt{2}}$ am nächsten liegt, mit welchen die Konvergenz von UCT für Belohnungen in $[0,1]$ gezeigt wurde. In Tabelle 14 a) lässt sich allerdings erkennen, dass dieser Wert nicht der für jede Rollout-Länge der optimale sein muss: Beispielsweise erreicht für eine Rollout-Länge von 5 der C-Wert $1,0\overline{5}$ 10 Siege, $1,\overline{3}$ nur einen einzigen.

Heuristische Bewertung der verlorenen Spiele gegenüber den Siegen

In den Tabellen der Abbildung 14 bildet sich ein interessanter Zusammenhang zwischen den erfolgreichen Konfigurationen der Siegzustände gegenüber denen der Konfigurationen, die eine hohe heuristische Bewertung der Endzustände von verlorenen Spielen darstellen, ab. Betrachtet man die Konfigurationen mit hoher Anzahl von Siegen, so besitzen diese Konfigurationen tendenziell eine schlechtere heuristische Bewertung der Nicht-Sieg-Instanzen als Konfigurationen mit weniger Siegen. Das lässt sich weiter in Grafik 15 erkennen. Es bildet sich kein proportionaler Zusammenhang zwischen diesen beiden Werten.

Das bedeutet insbesonders, dass unabhängig von der Parametrisierung, die Aussage *Je besser die heuristische Bewertung wird, desto eher werden Spiele gewonnen* nicht bestätigt werden kann.

Betrachtet man ausschließlich den Parameter Rollout-Länge und marginalisiert den C-Parameter, so findet sich dieser Zusammenhang in den Abbildungen 14a und 14c: Die meisten Siege sind mit einer Rollout-Länge von 100 zu erreichen, und mit abnehmender Länge sinkt die Anzahl der Siege. Diese Verteilung findet sich nicht in der durchschnittlichen heuristischen Bewertung aller Endzustände. Hier sind die besten Werte bei einer Rollout-Länge zwischen 15 und 50 zu finden. Auch sind die Lösungen mit hoher Rollout Länge nicht die kürzesten: Je länger die Rollouts sind, desto mehr Ticks benötigen die gewonnenen Spiele (siehe Abbildung 14b).

Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	3	5	14	13	11	14	16	10,86	4,5803
0,778	5	9	3	13	16	19	20	12,14	6,1974
1,056	10	13	12	11	17	20	15	14,00	3,2950
1,333	1	7	10	14	21	14	15	11,71	5,9453
1,611	6	6	3	13	15	10	11	9,14	3,9795
1,889	5	8	10	11	15	12	17	11,14	3,7580
2,167	4	6	3	15	14	10	16	9,71	5,0346
2,444	7	9	12	8	16	14	12	11,14	3,0439
2,722	7	7	16	11	16	19	17	13,29	4,5580
3	4	14	8	9	14	15	18	11,71	4,4949
Avg	5,20	8,40	9,10	11,80	15,50	14,70	15,70		
SD	2,3580	2,8355	4,5044	2,0881	2,4187	3,4366	2,5318		

(a) Anzahl der Siege pro Konfiguration von Realtime MCTS

Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	60,33	63,40	59,14	57,77	67,00	70,43	71,75	64,26	5,1566
0,778	60,20	59,00	63,00	57,15	65,38	68,68	72,00	63,63	4,9884
1,056	55,40	59,92	59,67	64,45	60,06	74,40	68,33	63,18	5,9437
1,333	37,00	66,43	58,20	64,43	61,19	74,86	73,27	62,20	11,6960
1,611	62,00	72,00	62,33	57,31	63,80	68,00	72,64	65,44	5,2329
1,889	49,80	54,50	66,00	62,45	72,47	72,33	71,59	64,16	8,4245
2,167	55,00	49,33	39,00	54,73	64,00	71,00	64,13	56,74	9,9083
2,444	52,14	51,00	65,17	60,75	69,25	69,57	73,50	63,05	8,1426
2,722	43,00	59,86	62,63	58,27	63,38	68,37	64,29	59,97	7,5475
3	51,50	62,29	63,25	56,56	59,57	72,33	71,00	62,36	6,9109
Avg	52,64	59,77	59,84	59,39	64,61	71,00	70,25		
SD	7,5119	6,5395	7,3445	3,2436	3,8717	2,3162	3,3119		

(b) Durchschnittliche Anzahl der Züge gewonnener Instanzen pro Konfiguration von Realtime MCTS

Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	0,76	0,72	0,73	0,80	0,77	0,77	0,77	0,76	0,0240
0,778	0,78	0,78	0,75	0,79	0,78	0,74	0,78	0,77	0,0170
1,056	0,78	0,79	0,81	0,80	0,80	0,77	0,74	0,78	0,0243
1,333	0,73	0,76	0,80	0,81	0,80	0,75	0,76	0,77	0,0275
1,611	0,73	0,78	0,79	0,80	0,79	0,76	0,78	0,78	0,0218
1,889	0,76	0,79	0,81	0,78	0,80	0,73	0,77	0,78	0,0242
2,167	0,76	0,80	0,79	0,79	0,79	0,74	0,76	0,77	0,0193
2,444	0,75	0,79	0,81	0,79	0,79	0,75	0,75	0,78	0,0225
2,722	0,77	0,78	0,79	0,78	0,78	0,73	0,76	0,77	0,0175
3	0,75	0,78	0,78	0,77	0,76	0,75	0,80	0,77	0,0166
Avg	0,76	0,78	0,79	0,79	0,79	0,75	0,77		
SD	0,0162	0,0208	0,0245	0,0107	0,0119	0,0130	0,0166		

(c) Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von Realtime MCTS

Abbildung 14: Tabellarische Ergebnisse von Realtime MCTS. Eine größere Darstellung ist im Anhang zu finden.

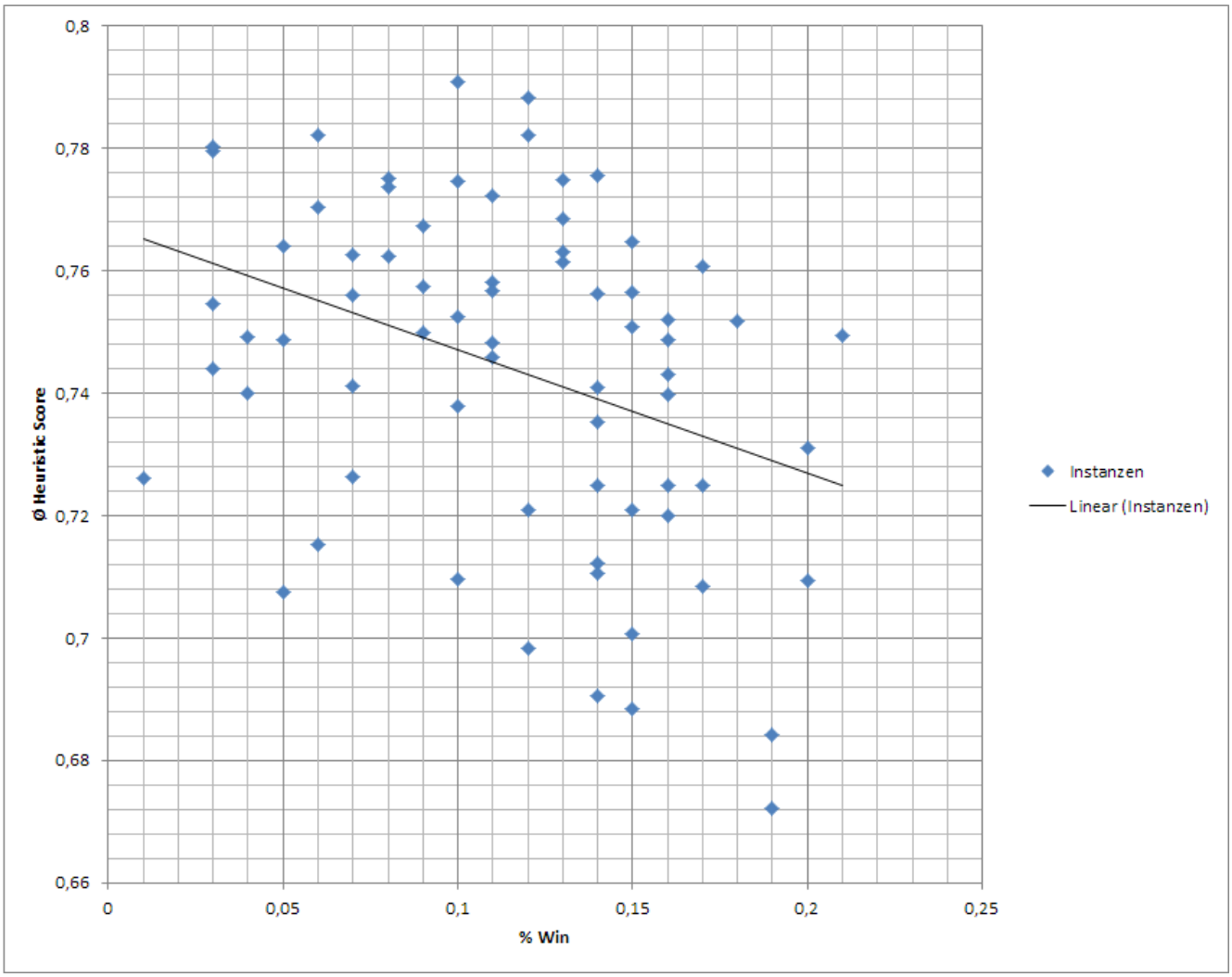
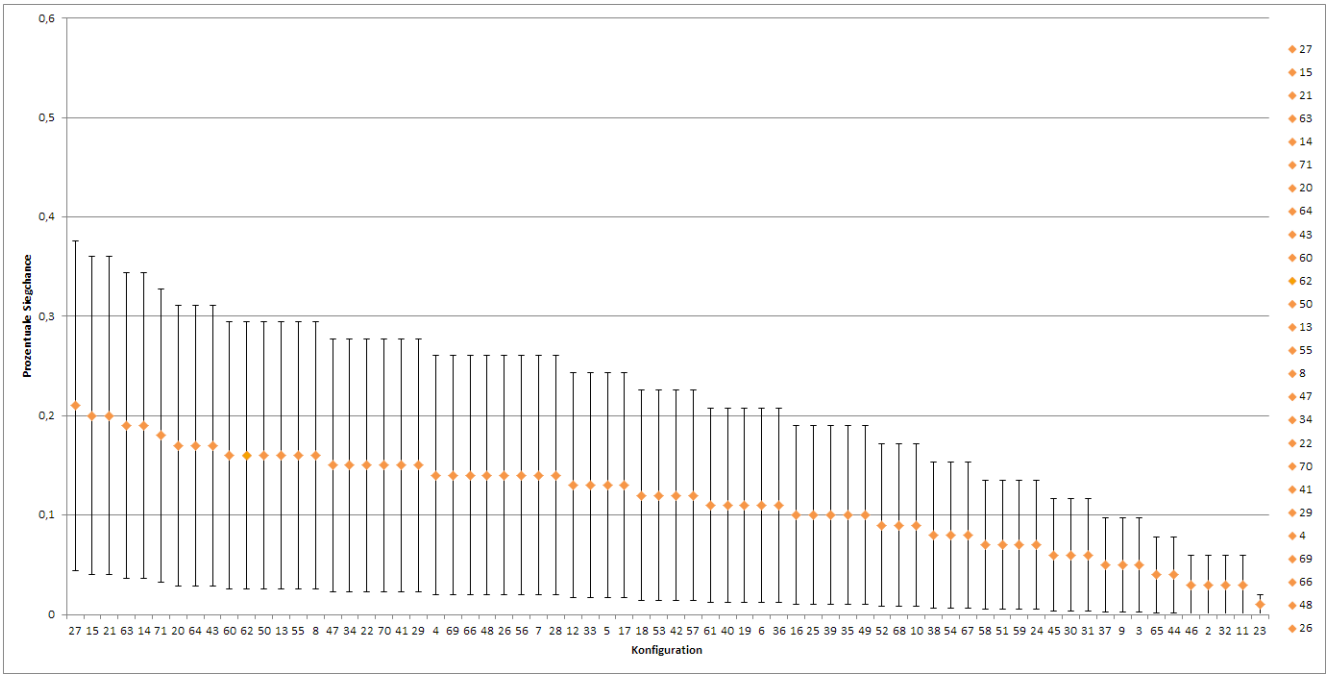
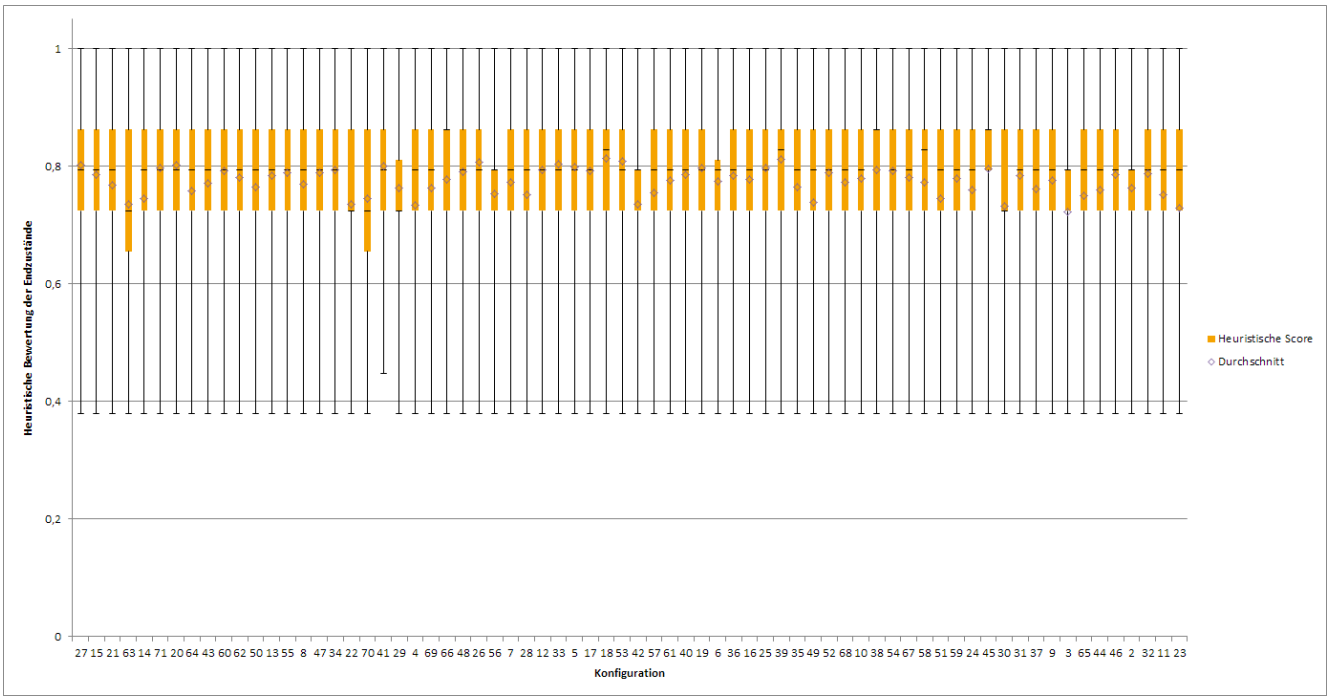


Abbildung 15: Die Anzahl der Siege und die heuristische Bewertung von Realtime MCTS verhalten sich nicht proportional.



(a) Pro Konfiguration von Realtime MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.



(b) Pro Konfiguration von Realtime MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

Abbildung 16: Pro Konfiguration von Realtime MCTS werden die Siege und die heuristische Bewertung der Endzustände dargestellt. Eine größere Darstellung ist im Anhang zu finden.

Platz	Konfiguration	C	Länge	# Siege	⌀ Ticks (gewonnen)	⌀ Heuristik
1	22	$1.0\overline{5}$	100	2	86	0.581
2	59	$2.7\overline{2}$	10	2	92	0.577
3	58	$2.7\overline{2}$	5	1	77	0.521
4	41	$1.\overline{8}$	50	1	79	0.593
5	38	$1.\overline{8}$	10	1	89	0.557

Tabelle 2: Die besten fünf Konfigurationen von Parallel MCTS

Platz	Konfiguration	C	Länge	# Siege	⌀ Ticks (gewonnen)	⌀ Heuristik
1	59	$2.7\overline{2}$	10	1	71	0.513
2	11	$0.\overline{7}$	15	1	73	0.526
3	13	$0.\overline{7}$	50	1	97	0.517
4	52	$2.\overline{4}$	10	0	n/a	0.541
5	2	0.5	5	0	n/a	0.539

Tabelle 3: Die besten fünf Konfigurationen von Binary MCTS

5.1.2 Parallel MCTS und Binary MCTS

Die beste Konfiguration von Parallel MCTS löst das 8Puzzle in 2 von 100 Instanzen. Dabei ist der C-Parameter mit $1,0\overline{5}$ und die Rollout-Length mit 100 gesetzt. Insgesamt wurden nur sieben der 7000 Instanzen, also genau 1‰ gelöst. Das sind sehr wenige gegenüber von 804 gelösten Instanzen von Realtime MCTS.

Da Parallel MCTS und Realtime MCTS sich ausschließlich in den binären Splits unterschieden, muss das der Grund der Verschlechterung sein. Eine Erläuterung, weshalb der Einsatz der binären Splits in Realtime MCTS zu einer Verschlechterung führt wird in Kapitel 5.2 beschrieben.

Binary MCTS löst das 8Puzzle noch seltener. Die beste Konfiguration von Binary MCTS löst das 8Puzzle in einem von 100 Instanzen. Dabei ist der C-Parameter mit $2,7\overline{2}$ und die Rollout-Length mit 10 gesetzt. Insgesamt wurden nur drei der 7000 Instanzen gelöst.

Das schlechte Abschneiden von Binary MCTS lässt sich auf die schlechte Performanz von Parallel MCTS zurückführen. Weitere Informationen dazu sind in Abschnitt 5.2 gegeben.

Wegen der geringen Anzahl von Siegen lassen sich bei diesen beiden Algorithmen leider keine Zusammenhänge zwischen Anzahl der Siege und hohen heuristischen Werten erzeugen.

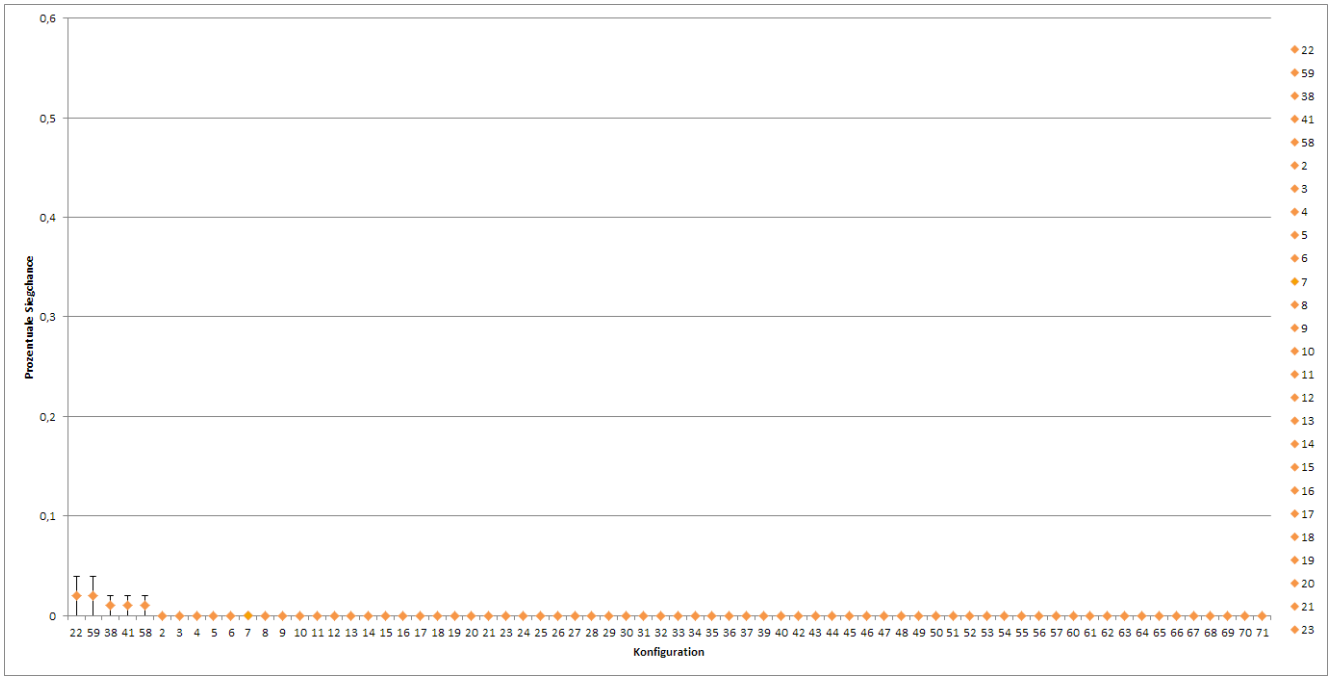
Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	0	0	0	0	0	0	0	0,00	0,0000
0,778	0	0	0	0	0	0	0	0,00	0,0000
1,056	0	0	0	0	0	0	2	0,29	0,6999
1,333	0	0	0	0	0	0	0	0,00	0,0000
1,611	0	0	0	0	0	0	0	0,00	0,0000
1,889	0	1	0	0	1	0	0	0,29	0,4518
2,167	0	0	0	0	0	0	0	0,00	0,0000
2,444	0	0	0	0	0	0	0	0,00	0,0000
2,722	1	2	0	0	0	0	0	0,43	0,7284
3	0	0	0	0	0	0	0	0,00	0,0000
Avg	0,10	0,30	0,00	0,00	0,10	0,00	0,20		
SD	0,3000	0,6403	0,0000	0,0000	0,3000	0,0000	0,6000		

(a) Anzahl der Siege pro Konfiguration von Parallel MCTS

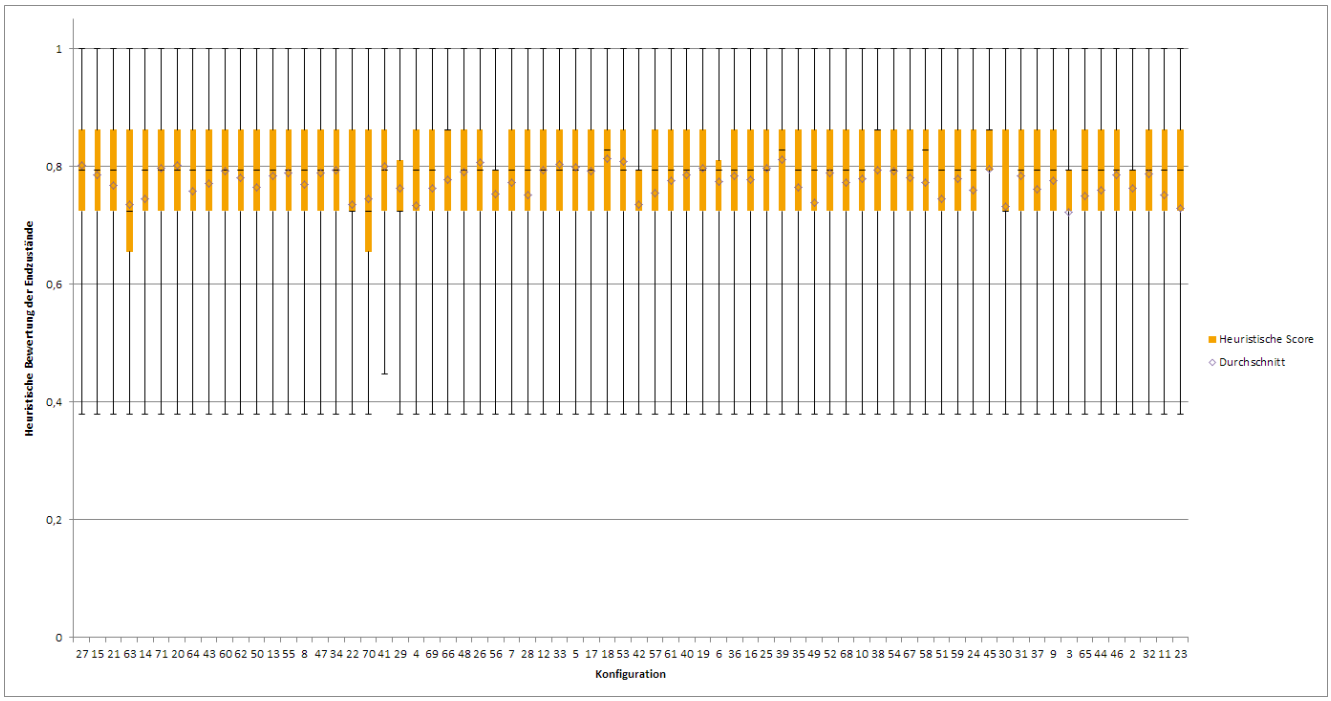
Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	0,42	0,55	0,51	0,42	0,61	0,52	0,51	0,51	0,0609
0,77778	0,53	0,52	0,64	0,57	0,60	0,58	0,56	0,57	0,0381
1,05556	0,55	0,53	0,58	0,63	0,60	0,61	0,58	0,58	0,0318
1,33333	0,57	0,57	0,56	0,58	0,59	0,59	0,57	0,58	0,0106
1,61111	0,55	0,58	0,53	0,54	0,59	0,59	0,57	0,56	0,0232
1,88889	0,56	0,56	0,52	0,54	0,59	0,59	0,58	0,56	0,0253
2,16667	0,56	0,54	0,53	0,54	0,59	0,58	0,57	0,56	0,0199
2,44444	0,56	0,55	0,51	0,54	0,58	0,57	0,55	0,55	0,0199
2,72222	0,52	0,58	0,52	0,55	0,58	0,57	0,55	0,55	0,0247
3	0,49	0,53	0,52	0,55	0,57	0,56	0,54	0,54	0,0244
Avg	0,53	0,55	0,54	0,55	0,59	0,58	0,56		
SD	0,0427	0,0201	0,0388	0,0495	0,0105	0,0211	0,0190		

(b) Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von Parallel MCTS

Abbildung 17: Tabellarische Ergebnisse von Parallel MCTS.



(a) Pro Konfiguration von Parallel MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.



(b) Pro Konfiguration von Parallel MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

Abbildung 18: Pro Konfiguration von Parallel MCTS werden die Siege und die heuristischen Bewertungen der Endzustände dargestellt.

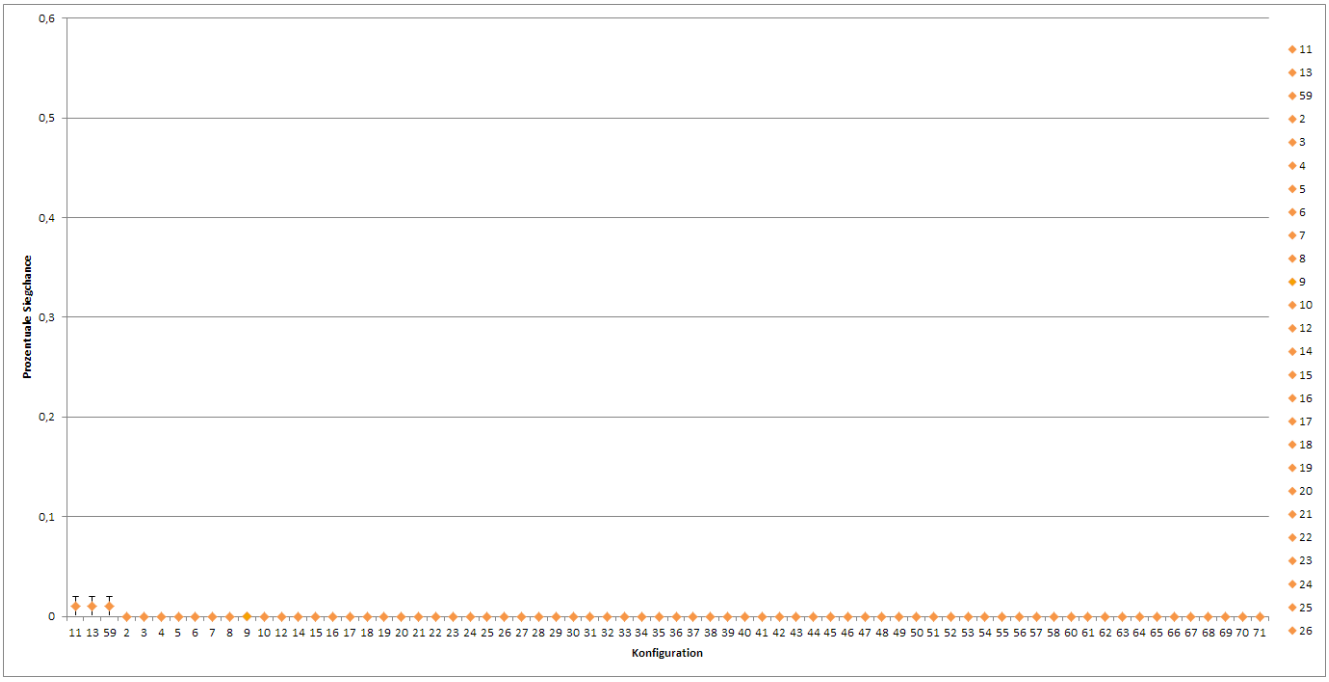
Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	0	0	0	0	0	0	0	0,00	0,0000
0,778	0	0	1	0	1	0	0	0,29	0,4518
1,056	0	0	0	0	0	0	0	0,00	0,0000
1,333	0	0	0	0	0	0	0	0,00	0,0000
1,611	0	0	0	0	0	0	0	0,00	0,0000
1,889	0	0	0	0	0	0	0	0,00	0,0000
2,167	0	0	0	0	0	0	0	0,00	0,0000
2,444	0	0	0	0	0	0	0	0,00	0,0000
2,722	0	1	0	0	0	0	0	0,14	0,3499
3	0	0	0	0	0	0	0	0,00	0,0000
Avg	0,00	0,10	0,10	0,00	0,10	0,00	0,00		
SD	0,0000	0,3000	0,3000	0,0000	0,3000	0,0000	0,0000		

(a) Anzahl der Siege pro Konfiguration von Binary MCTS

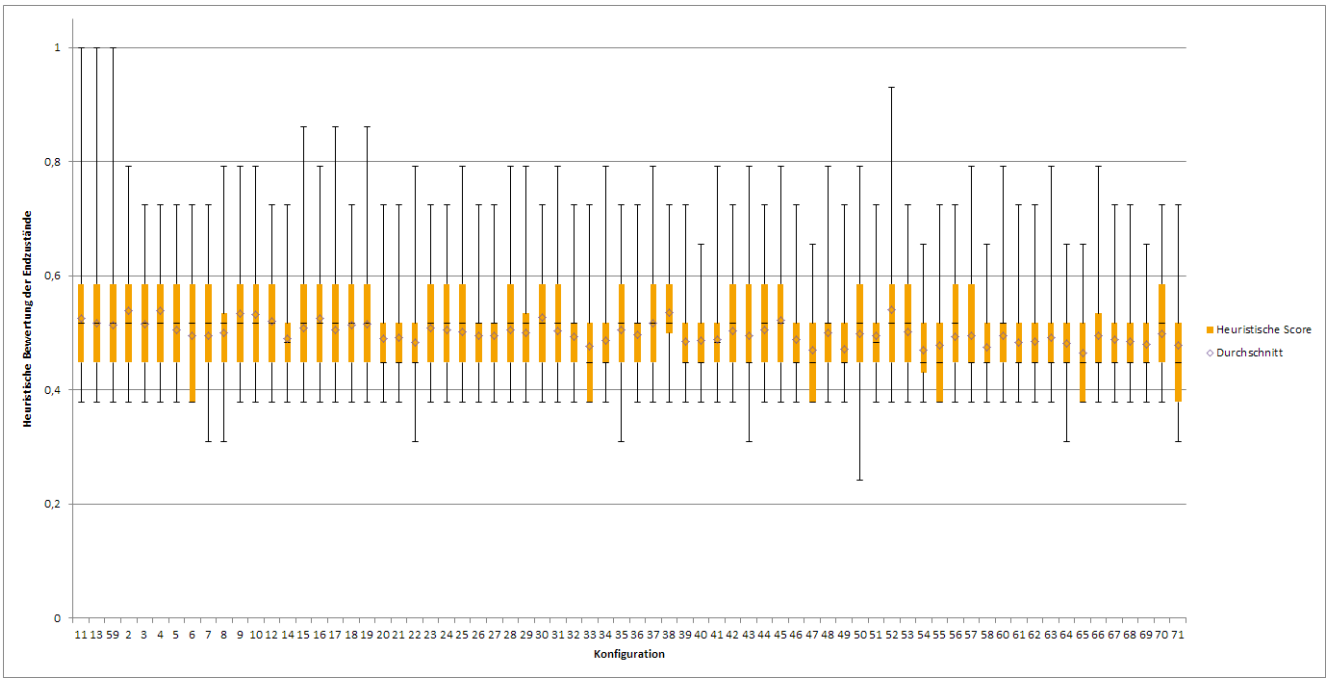
Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	0,54	0,52	0,54	0,51	0,50	0,50	0,50	0,51	0,0175
0,77778	0,53	0,53	0,53	0,52	0,52	0,49	0,51	0,52	0,0139
1,05556	0,53	0,51	0,51	0,52	0,49	0,49	0,48	0,50	0,0144
1,33333	0,51	0,51	0,50	0,49	0,50	0,51	0,50	0,50	0,0049
1,61111	0,53	0,50	0,49	0,48	0,49	0,51	0,50	0,50	0,0149
1,88889	0,52	0,54	0,48	0,49	0,49	0,50	0,49	0,50	0,0176
2,16667	0,51	0,52	0,49	0,47	0,50	0,47	0,50	0,49	0,0172
2,44444	0,50	0,54	0,50	0,47	0,48	0,49	0,49	0,50	0,0207
2,72222	0,47	0,51	0,50	0,48	0,49	0,49	0,48	0,49	0,0117
3	0,46	0,49	0,49	0,49	0,48	0,50	0,48	0,48	0,0107
Avg	0,51	0,52	0,50	0,49	0,49	0,50	0,49		
SD	0,0237	0,0147	0,0169	0,0170	0,0105	0,0093	0,0092		

(b) Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von Binary MCTS

Abbildung 19: Tabellarische Ergebnisse von Binary MCTS.



(a) Pro Konfiguration von Binary MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.



(b) Pro Konfiguration von Binary MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

Abbildung 20: Pro Konfiguration von Binary MCTS werden die Siege und die heuristischen Bewertungen der Endzustände dargestellt.

Platz	Konfiguration	α	Länge	# Siege	⌀ Ticks (gewonnen)	⌀ Heuristik
1	10	0.2	10	31	59.97	0.845
2	9	0.9	5	26	61.54	0.836
3	65	0.1	10	25	63.88	0.852
4	69	1.3	10	25	70.04	0.837
5	59	1.0	5	24	54.75	0.850

Tabelle 4: Die besten fünf Konfigurationen von Relative MCTS

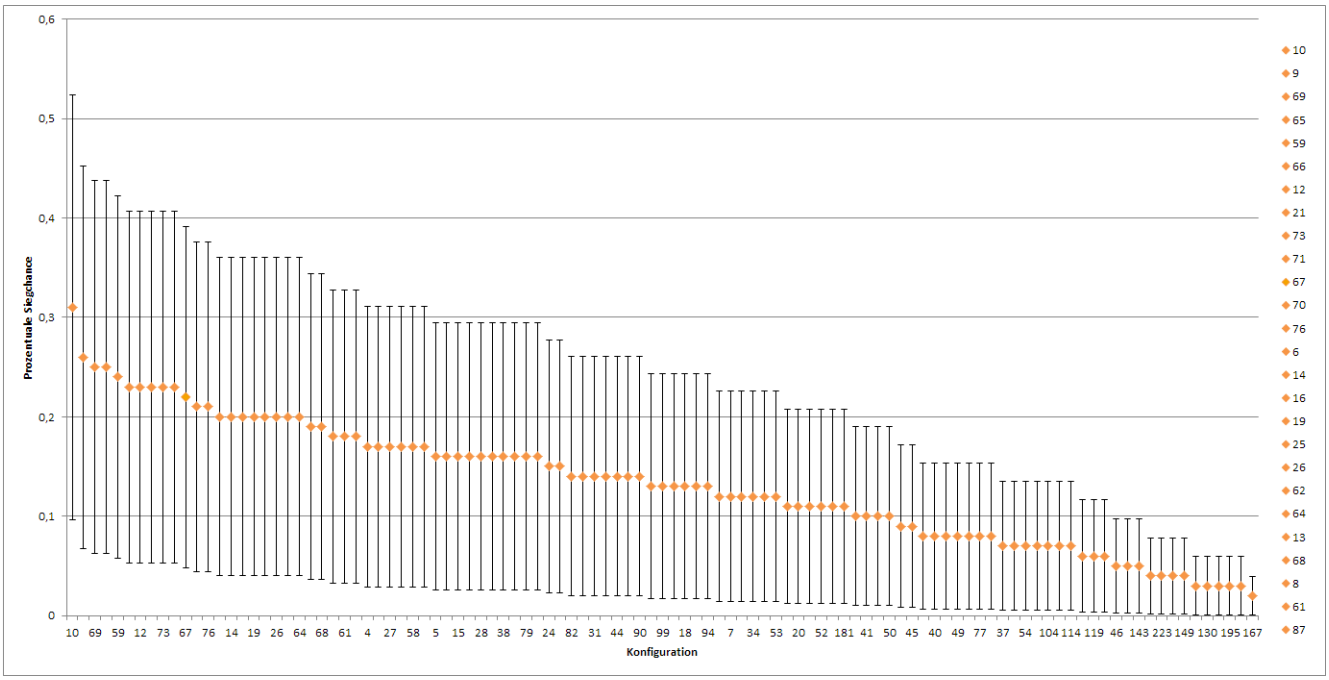
5.1.3 Relative MCTS

Die beste Konfiguration von Relative MCTS löst das 8Puzzle in 31 von 100 Instanzen. Dabei ist der α-Parameter mit 0.2 und die Rollout-Length mit 10 gesetzt.

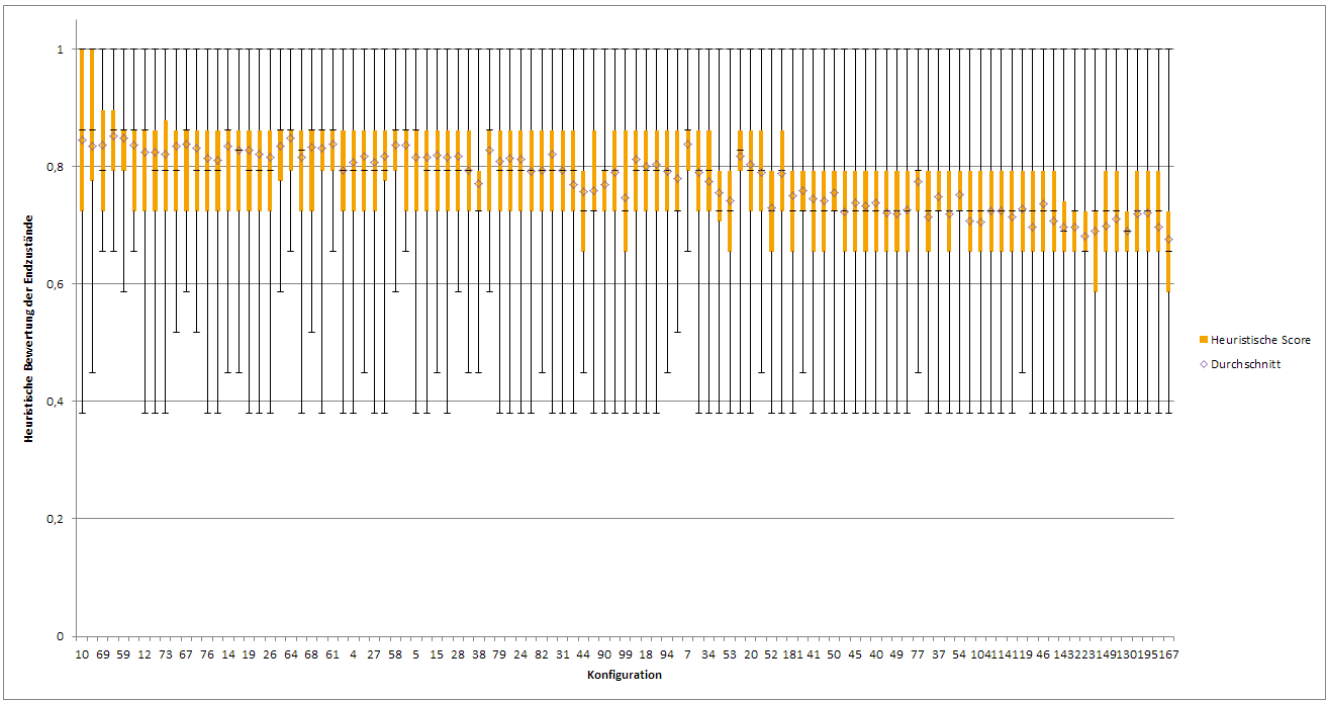
Heuristische Bewertung der verlorenen Spiele gegenüber den Siegen

Betrachtet man in diesen Daten den Zusammenhang zwischen der durchschnittlichen heuristischen Bewertung der Endzustände und der Anzahl der Siege, so existiert ein Unterschied zu Realtime MCTS: In diesen Daten korreliert die Anzahl der Siege positiv mit der heuristischen Bewertung (Siehe Abbildung 23) und nicht negativ wie bei Realtime MCTS.

Eine Interpretation dieser Zusammenhänge folgt in Kapitel 5.2.



(a) Pro Konfiguration von Relative MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.



(b) Pro Konfiguration von Relative MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

Abbildung 21: Pro Konfiguration von Relative MCTS werden die Siege und die heuristischen Bewertungen der Endzustände dargestellt. Eine größere Darstellung ist im Anhang zu finden.

Parameter α \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,1	17	25	16	16	13	10	11	15,43	4,6247
0,2	11	31	13	20	12	10	10	15,29	7,1657
0,3	14	16	20	17	12	9	14	14,57	3,2888
0,4	17	23	11	16	14	14	11	15,14	3,8333
0,5	16	19	23	17	7	9	12	14,71	5,2567
0,6	20	20	16	11	16	5	7	13,57	5,5769
0,7	12	16	17	14	8	7	8	11,71	3,8065
0,8	18	20	15	16	8	8	3	12,57	5,8029
0,9	26	13	20	12	10	8	7	13,71	6,3856
1	24	23	23	15	7	5	3	14,29	8,5643
1,1	16	22	11	14	6	4	3	10,86	6,4681
1,2	18	19	13	12	7	8	3	11,43	5,4210
1,3	20	25	21	18	6	4	7	14,43	7,8714
1,4	17	21	8	14	6	2	5	10,43	6,4776
1,5	20	23	13	13	3	4	4	14,29	8,5643
Avg	17,73	21,07	16,00	15,00	9,00	7,13	7,20		
SD	3,8552	4,2343	4,4572	2,3944	3,5214	3,0302	3,5814		

(a) Anzahl der Siege pro Konfiguration von Relative MCTS

Parameter α \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,1	59,35	63,88	64,00	61,00	79,92	78,60	77,55	69,19	8,3882
0,2	54,82	59,97	58,85	72,60	81,50	85,40	77,60	70,10	11,2719
0,3	59,29	57,88	59,60	71,00	81,83	87,00	82,14	71,25	11,5668
0,4	63,24	64,30	59,91	72,63	73,86	81,57	79,91	70,77	7,8349
0,5	51,38	66,26	68,22	75,59	79,57	82,78	86,33	72,88	11,0780
0,6	62,60	60,20	62,00	69,36	81,13	82,20	75,57	70,44	8,5983
0,7	56,50	64,75	66,18	73,86	84,00	80,14	86,50	73,13	10,2939
0,8	59,89	62,80	55,53	66,13	86,00	79,75	93,00	71,87	13,2769
0,9	61,54	55,46	69,60	73,83	87,20	85,25	92,14	75,00	12,7686
1	54,75	64,30	72,65	75,27	86,71	83,40	89,67	75,25	11,6650
1,1	62,50	70,27	67,00	75,71	82,67	89,50	89,00	76,66	9,9192
1,2	53,44	64,89	69,46	82,00	84,43	81,50	87,00	74,68	11,4695
1,3	56,90	70,04	71,57	76,78	91,33	91,00	86,71	77,76	11,7941
1,4	62,47	64,81	77,25	77,43	94,67	93,00	88,20	79,69	11,9695
1,5	63,00	66,22	66,08	75,92	89,00	94,50	85,00	75,25	11,6650
Avg	58,78	63,74	65,86	73,27	84,25	85,04	85,09		
SD	3,7692	3,9124	5,7227	4,8424	4,9898	4,8039	5,2102		

(b) Durchschnittliche Anzahl der Züge gewonnener Instanzen pro Konfiguration von Relative MCTS

Parameter α \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,1	0,84	0,85	0,81	0,81	0,75	0,76	0,75	0,80	0,0397
0,2	0,82	0,84	0,80	0,82	0,78	0,74	0,76	0,79	0,0343
0,3	0,82	0,82	0,83	0,81	0,76	0,72	0,76	0,79	0,0378
0,4	0,81	0,83	0,80	0,82	0,77	0,76	0,73	0,79	0,0327
0,5	0,82	0,82	0,82	0,82	0,75	0,74	0,74	0,79	0,0374
0,6	0,81	0,84	0,82	0,79	0,77	0,74	0,75	0,79	0,0330
0,7	0,84	0,82	0,82	0,79	0,73	0,72	0,73	0,78	0,0466
0,8	0,83	0,83	0,81	0,79	0,74	0,72	0,71	0,78	0,0480
0,9	0,84	0,81	0,82	0,79	0,75	0,72	0,71	0,78	0,0475
1	0,85	0,84	0,82	0,79	0,71	0,70	0,72	0,77	0,0604
1,1	0,83	0,84	0,79	0,79	0,72	0,70	0,72	0,77	0,0527
1,2	0,84	0,83	0,80	0,78	0,73	0,71	0,70	0,77	0,0540
1,3	0,83	0,84	0,81	0,79	0,73	0,70	0,72	0,78	0,0537
1,4	0,84	0,83	0,77	0,77	0,70	0,68	0,71	0,76	0,0596
1,5	0,85	0,84	0,79	0,79	0,69	0,69	0,68	0,77	0,0604
Avg	0,83	0,83	0,81	0,80	0,74	0,72	0,73		
SD	0,0125	0,0107	0,0147	0,0133	0,0255	0,0238	0,0219		

(c) Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von Relative MCTS

Abbildung 22: Tabellarische Ergebnisse von Relative MCTS. Eine größere Darstellung ist im Anhang zu finden.

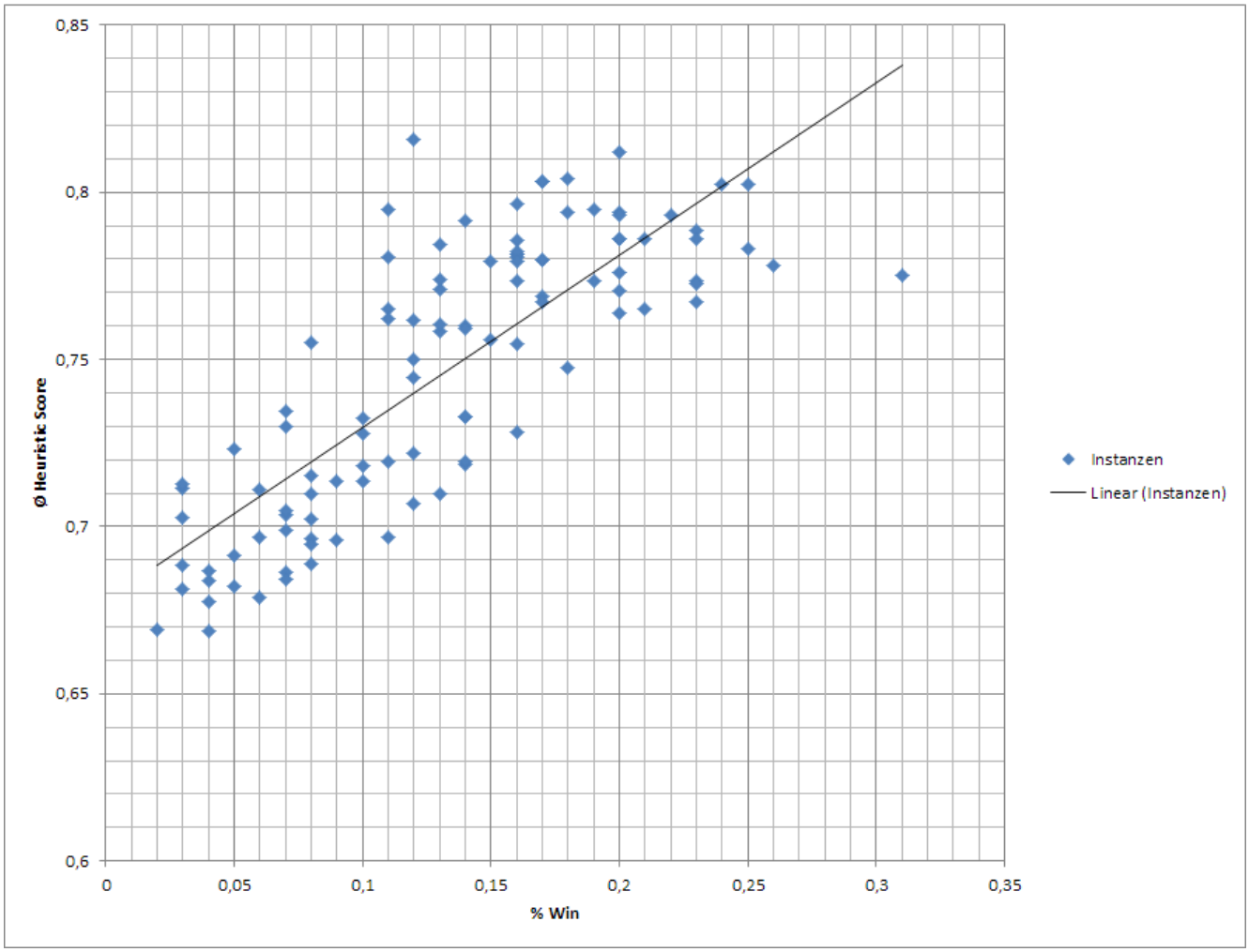


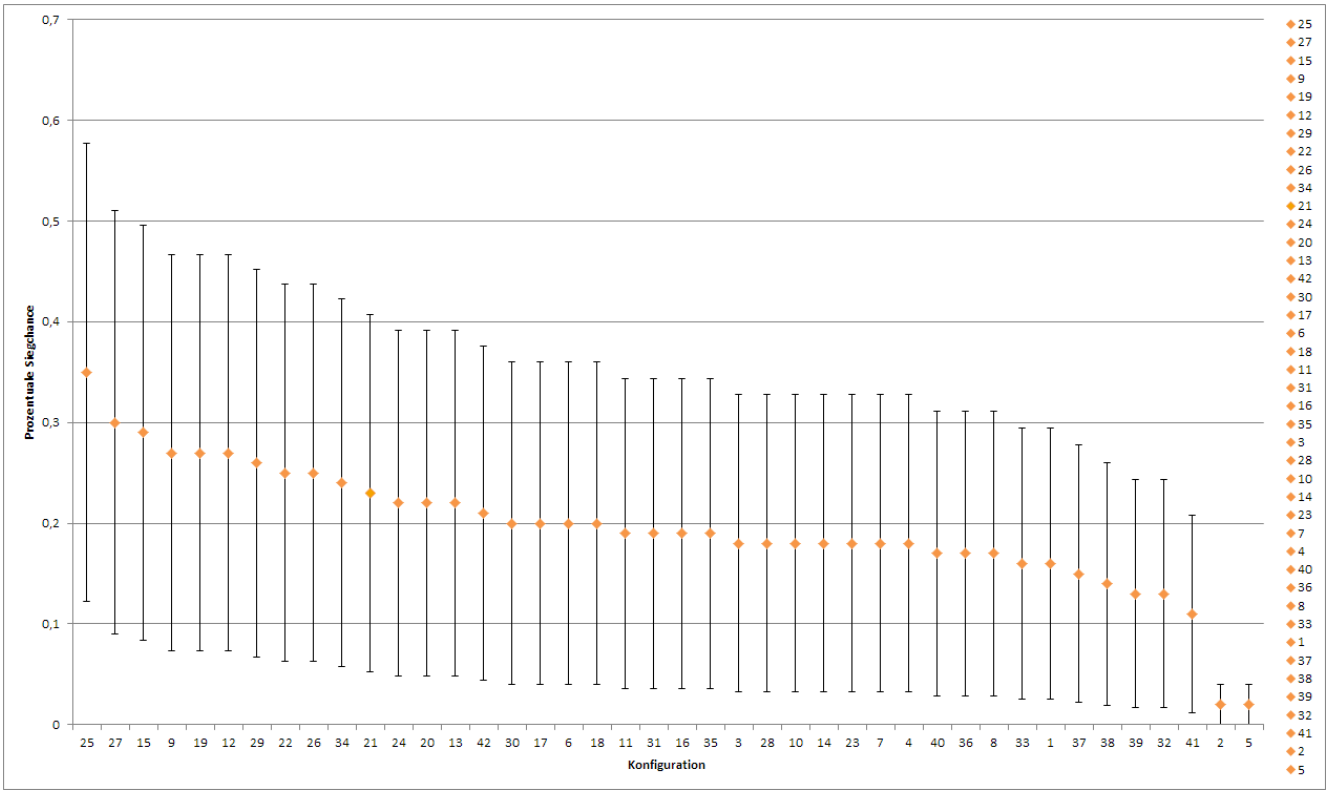
Abbildung 23: Die Anzahl der Siege und die heuristische Bewertung von Realtime MCTS verhalten sich proportional.

Platz	Konfiguration	Signifikanzniveau	# Siege	⌀ Ticks (Sieg)	⌀ Heuristik	% Split
1	25	0.8	35	57.74	0.861	99.93%
2	27	0.99	30	63.20	0.865	100%
3	15	0.9	29	62.45	0.868	100%
4	9	0.99	27	51.94	0.850	100%
5	19	0.95	27	64.41	0.859	99.85%

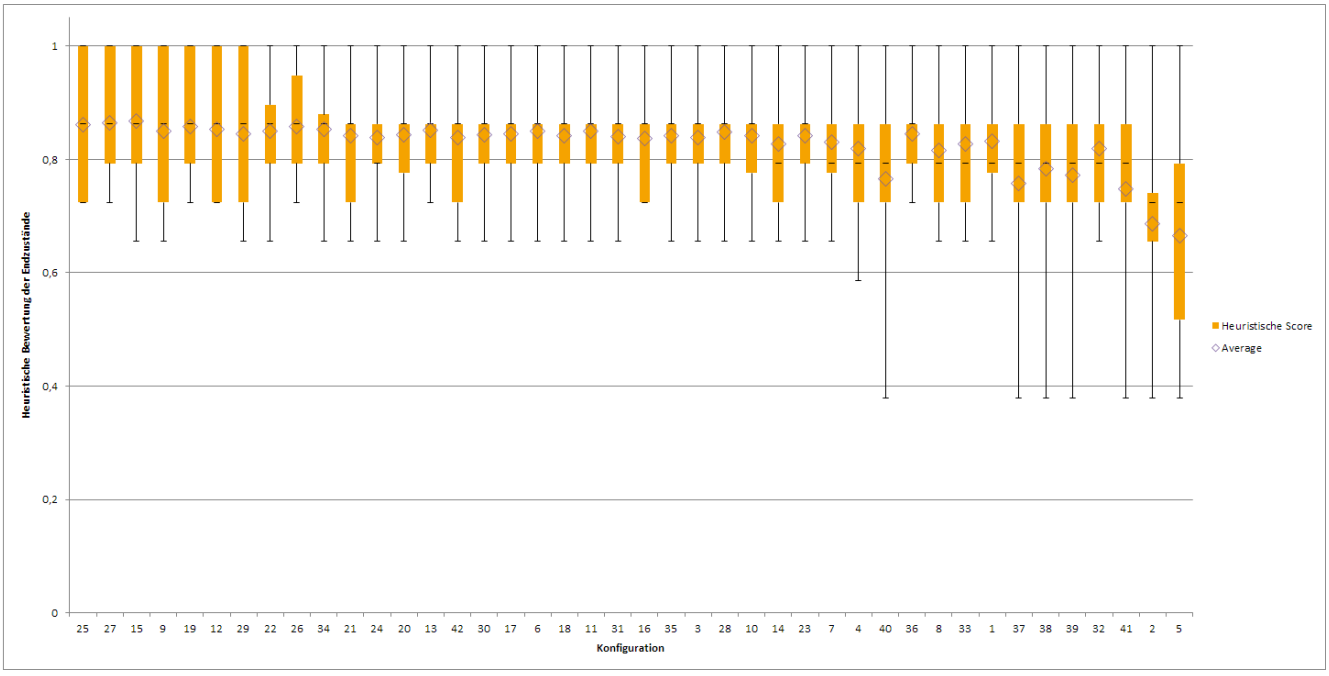
Tabelle 5: Die besten fünf Konfigurationen von bedingtem Relative MCTS

5.1.4 Bedingt Relative MCTS

Die beste Konfiguration von bedingtem Relative MCTS löst das 8Puzzle in 35 von 100 Instanzen. Dabei ist der α-Parameter mit 0.2 und die Rollout-Length mit 10 gesetzt, wie in der besten Konfiguration von Relative MCTS. Als optimales Signifikanznivaeu wurde 0.8 ermittelt.



(a) Pro Konfiguration von bedingtem Relative MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.



(b) Pro Konfiguration von bedingtem Relative MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

Abbildung 24: Pro Konfiguration von bedingtem Relative MCTS werden die Siege und die heuristische Bewertung der Endzustände dargestellt. Eine größere Darstellung ist im Anhang zu finden.

Signifikanzniveau \ MinTests	0	5	10	15	20	50	Avg	SD
0,6	2	11	17	15	13	14	12,00	4,8305
0,7	2	16	13	19	17	24	15,17	6,7680
0,8	17	19	20	22	18	35	21,83	6,0942
0,9	18	25	20	26	19	29	22,83	4,0586
0,95	20	25	27	23	18	22	22,50	2,9861
0,98	16	19	21	18	20	27	20,17	3,4359
0,99	18	18	18	22	27	30	22,17	4,7755
Avg	13,29	19,00	19,43	20,71	18,86	25,86		
SD	7,2252	4,5670	3,9590	3,3685	3,9071	6,2204		

(a) Anzahl der Siege pro Konfiguration von bedingtem Relative MCTS

Signifikanzniveau \ MinTests	0	5	10	15	20	50	Avg	SD
0,6	0,66	0,75	0,77	0,76	0,77	0,78	0,75	0,0392
0,7	0,69	0,83	0,82	0,84	0,84	0,85	0,81	0,0571
0,8	0,82	0,84	0,85	0,84	0,85	0,86	0,84	0,0140
0,9	0,82	0,85	0,84	0,85	0,84	0,87	0,84	0,0147
0,95	0,84	0,86	0,86	0,84	0,84	0,84	0,85	0,0084
0,98	0,83	0,85	0,84	0,84	0,84	0,85	0,84	0,0068
0,99	0,83	0,84	0,83	0,85	0,85	0,86	0,84	0,0128
Avg	0,78	0,83	0,83	0,83	0,83	0,85		
SD	0,0693	0,0350	0,0287	0,0304	0,0255	0,0267		

(b) Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von bedingtem Relative MCTS

Signifikanzniveau \ MinTests	5	10	15	20	50	70	Avg	SD
0,6	88,00	75,73	69,71	67,40	66,08	72,00	73,15	7,3421
0,7	81,00	70,13	70,08	70,26	71,12	60,08	70,44	6,0484
0,8	79,47	62,89	68,60	63,09	59,78	57,74	65,26	7,1875
0,9	77,56	59,96	63,90	61,69	64,68	62,45	65,04	5,7994
0,95	73,70	64,76	64,41	64,04	59,11	65,73	65,29	4,3149
0,98	75,88	60,58	61,48	65,67	65,70	65,30	65,77	4,9621
0,99	76,33	61,00	62,44	69,18	51,96	63,20	64,02	7,4823
Avg	78,85	65,01	65,80	65,91	62,63	63,79		
SD	4,3443	5,4350	3,3143	2,9557	5,7554	4,2406		

(c) Durchschnittliche Anzahl der Züge gewonnener Instanzen pro Konfiguration von bedingtem Relative MCTS

Signifikanzniveau \ MinTests	0	5	10	15	20	50	Avg	SD
0,6	65,28	20,64	15,88	14,40	11,33	7,11	22,44	19,5999
0,7	60,14	8,96	5,89	4,90	2,50	0,89	13,88	20,8468
0,8	27,47	3,11	1,30	0,71	0,28	0,07	5,49	9,8792
0,9	24,48	0,38	0,18	0,03	0,02	0,00	4,18	9,0771
0,95	24,27	0,21	0,15	0,01	0,00	0,00	4,11	9,0175
0,98	24,03	0,09	0,04	0,00	0,00	0,00	4,03	8,9480
0,99	24,21	0,08	0,04	0,00	0,00	0,00	4,05	9,0119
Avg	35,70	4,78	3,35	2,86	2,02	1,15		
SD	17,1773	7,1321	5,4756	4,9916	3,8954	2,4516		

(d) Der Prozentsatz, wie oft kein Binärsplit gemacht wurde, unter der Bedingung, dass ein Condorcet-Sieger-Kandidat ermittelt werden konnte.

Abbildung 25: Tabellarische Ergebnisse von bedingtem Relative MCTS. Eine größere Darstellung ist im Anhang zu finden.

Algorithmus	# Siege	⌀ Ticks (gewonnen)	⌀ Heuristik
Bedingtes Relative MCTS	35	57.74	0.861
Relative MCTS	31	59.97	0.845
Realtime MCTS	21	61.19	0.802
Parallel MCTS	2	92	0.513
Dual MCTS	1	73	0.581

Tabelle 6: Die jeweils besten Konfigurationen im Vergleich

5.2 Vergleich der Algorithmen

In den vorherigen Kapiteln wurden die Algorithmen unabhängig voneinander analysiert und es wurde die jeweils beste Konfiguration ermittelt. In diesem und dem folgenden Abschnitt werden die besten Konfigurationen miteinander verglichen, um die Forschungsfragen zu beantworten.

Auswirkung von binären Splits

Binäre Splits verändern das Explorationsverhalten von MCTS. Es kann nicht ausschließlich in die Tiefe erkundet werden, da sich der Pfad von der Wurzel aufspaltet und somit gezwungenermaßen auch in die Breite erkundet wird je tiefer gesucht wird.

Inwiefern diese Änderung Auswirkungen auf die Performanz des Algorithmus hat, kann durch Vergleich der Algorithmen Realtime MCTS und Binary MCTS ermittelt werden, da Binary MCTS Realtime MCTS mit binären Splits entspricht. In diesem Vergleich geht Realtime MCTS mit einer Siegchance von 21% gegenüber 2% von Binary MCTS klar als besserer Algorithmus hervor.

Das Verwenden von binären Splits gegenüber einem linearen Pfad bei der Selektion von MCTS bringt alleine keinen Mehrwert.

Ein Grund, weshalb die Verwendung von binären Splits bei Realtime MCTS zu einer Verschlechterung führen kann, wird im nachfolgenden Kapitel 5.3 gegeben.

Auswirkung von binären Vergleichen

Binäre Vergleiche sind eine alternative Backpropagationstechnik, in welcher Simulationen untereinander verglichen werden anstatt diese unanhängig zu betrachten. Dabei gehen Informationen verloren, wie die Distanz der Güte zweier Simulationen. Nur die Information, welche der Simulationen besser ist, wird verwendet.

Inwiefern diese Änderung Auswirkungen auf die Performanz des Algorithmus hat, kann durch Vergleich der Algorithmen Binary MCTS und Dual MCTS ermittelt werden, da Dual MCTS Binary MCTS mit binären Vergleichen entspricht. Hier ist die Siegchance von Binary MCTS 2% und die Siegchance von Dual MCTS 1%. Es ist zu bemerken, dass die Anzahl der Siege gesunken ist. Allerdings ist die Performanz von Binary MCTS mit 2% schon sehr schlecht, weshalb kaum Informationen über binäre Vergleiche gegenüber nicht binären Vergleichen gewonnen werden können. Es kann allerdings bemerkt werden, dass die Güte durch die Verwendung von binären Vergleichen nicht signifikant besser wurde. Die Forschungsfrage *Wie wirken sich Informationsverlust und algorithmische Anpassung von Realtime MCTS auf dessen Güte aus?* lässt sich, solange die UCT Formel als Selektionskriterium verwendet wird, wie folgt beantworten: Die Auswirkung der alleinigen algorithmischen Änderungen und des Informationsverlusts durch Präferenzen gegenüber heuristischen Werten ist negativ, wenn die UCT Formel als Selektionskriterium verwendet

wird. Wird nicht die UCT Formel verwendet sondern die RUCB Formel, so folgt die Antwort im folgenden Abschnitt.

Auswirkung von Relative UCB

Die Auswirkungen von Relative UCB können mit zwei Algorithmen vergleichen werden.

1. gegenüber Binary MCTS, um die Güte der Verwendung der Relative UCB-Formel gegenüber der Verwendung der UCT-Formel zu ermitteln.

2. gegenüber Relative MCTS, um die Performanz von binären Splits, binären Vergleichen und der Relative UCB-Formel als Selektionskriterium zusammen gegenüber der Baseline *Realtime MCTS* zu vergleichen.

Im ersten Fall, dem Vergleich von Relative UCB gegenüber UCT, ist eine starke Verbesserung durch die Verwendung der Relative UCB-Formel festzustellen: Relative MCTS hat eine Siegchance von 31% gegenüber einer Siegchance von 1% für Binary MCTS. Das Verwenden der Relative UCB-Formel gegenüber der UCT-Formel bringt somit einen enormen Performanzgewinn unter der Verwendung von binären Splits und binären Vergleichen.

Für Zweites ist Relative MCTS mit Realtime MCTS zu vergleichen. Dieser Vergleich führt auch zu einer Antwort der Fragestellung: *Lässt sich durch das Übersetzen von heuristischen Bewertungen in Präferenzen in Realtime MCTS ein Gewinn erzielen?*

Relative MCTS ist eine Anpassung von Realtime MCTS an die Verwendung von Präferenzen. Da Relative MCTS 31 von 100 Spielen gelöst hat und Realtime MCTS nur 21 von 100, ist festzustellen, dass in der verwendeten Domäne die Nutzung von Präferenzen gegenüber heuristischen Werten einen Mehrwert erzielen können.

Bedingte Splits

Bei Relative MCTS werden in jedem Knoten zwei Aktionen ausgewählt. Wie in Abschnitt 3.3 beschrieben, werden diese binären Splits unter bestimmten Bedingungen bei bedingt Relative MCTS nicht gemacht, sondern es wird ausschließlich die als bestes wahrgenommene Aktion ausgeführt. Das führt zu weniger Exploration aber mehreren Iterationen.

Die Konfigurationen von bedingt Relative MCTS führen zu unterschiedlichem Verhalten, also ob ein binärer Split erfolgt oder nicht (Siehe Abbildung 25d). Darunter gibt es Konfigurationen mit vielen binären Splits, die dem Verhalten und Resultat von Relative MCTS ähneln, Konfigurationen mit deutlich weniger binären Splits, deren Sieganzahlen deutlich niedriger sind und wenigen Konfigurationen dazwischen. Vergleicht man die jeweils beste Konfiguration von Relative MCTS und bedingt Relative MCTS, so schneidet das bedingt Relative MCTS besser ab: Mit 35 Siegen gegenüber 31 Siegen.

Eine Interpretation der Ergebnisse ist im folgenden Kapitel gegeben.

5.3 Interpretation der Ergebnisse

In den vorherigen Abschnitten wurden die Ergebnisse der Auswertung offengelegt und gegeneinander aufgelistet. In diesem Abschnitt werden diese Tatsachen analysiert und interpretiert.

Die in den letzten Abschnitten gelisteten Tatsachen sind insbesonders:

- Bedingtes Relative MCTS hat eine höhere Siegchance als die restlichen Algorithmen in der verwendeten Domäne.

- Relative MCTS schneidet als zweitbestes ab.
- Realtime MCTS hat die dritthöchste Siegchance.
- Sowohl Binary MCTS als auch Parallel MCTS lösen das Problem nur sehr selten.
- Bei Realtime MCTS besteht ein negativer Zusammenhang zwischen den durchschnittlichen heuristischen Bewertungen der Endzustände und der Anzahl der Siege.
- Bei Relative MCTS ist dieser Zusammenhang positiv.

Im Folgenden werden mögliche Gründe für diese Ergebnisse diskutiert.

Das schlechte Abschneiden von Parallel MCTS und Binary MCTS

Zunächst ist interessant, dass die Performanz von Realtime MCTS enorm schlechter wird, wenn binäre Splits verwendet werden (Vergleiche Realtime MCTS mit Parallel MCTS). Ein Grund dafür ist, dass durch diese Veränderung die Konvergenzeigenschaft von Realtime MCTS verloren geht (Vergleiche Abschnitt 2.1.8). Durch das Ausführen der besten und zweitbesten Aktion in jedem Knoten wird die Verteilung der Simulationen stark verändert. Das führt dazu, dass der ermittelte spieltheoretische Wert eines Knotens verfälscht wird. Hat ein Knoten beispielsweise nur zwei Aktionen zur Auswahl, so werden stets beide ausgeführt. Die Güte des Knotens wird durch die Simulationen ermittelt, wobei beide Aktionen gleichermaßen diesen Gütewert beeinflussen. Bei Realtime MCTS nähert sich die Güte eines solchen Knoten an die Güte der besseren Aktion an. Werden aber stets beide Aktionen ausgeführt, so nähert sich die Güte an den Mittelwert der beiden Aktionen an. Das ist nicht erwünscht und führt insbesondere dazu, dass die ermittelten Konvergenzungleichungen (6) und (7) aus Abschnitt 2.2.2 ihre Gültigkeit verlieren.

Eine Verwendung von binären Vergleichen ändert nichts an dieser Tatsache. Somit ist nachzuvollziehen, dass auch Binary MCTS keine signifikante Verbesserung gegenüber Parallel MCTS darstellt.

Relative MCTS verwendet dagegen nicht die UCT Formel, sondern die Relative UCB Formel, welche für das Lernen aus binären Vergleichen geschaffen ist und nicht auf den Konvergenzeigenschaften der UCT Formel aufbaut.

Realtime MCTS gegenüber Relative MCTS

Relative MCTS schlägt Realtime MCTS was die Sieganteile der besten Konfiguration betrifft deutlich (31 zu 21). Es ist allerdings zu bemerken, dass die Testdomäne absichtlich so gewählt wurde, dass Realtime MCTS diese nicht gut löst. Dass dies der Fall ist, lässt sich in dem antiproportionalen Zusammenhang zwischen den heuristischen Bewertungen der Endzustände und der Siege pro Konfiguration erkennen. Realtime MCTS ist mit der verwendete Heuristik offenbar nicht in der Lage, aus heurisitisch gut bewerteten Zuständen einen Siegzustand zu generieren. Eine naheliegende These ist, dass diese Heuristik lokale Optima im Zustandsraum erzeugt die keinen Siegzustand in der direkten Nachbarschaft besitzen und Realtime MCTS nur selten in der Lage ist, aus diesem Optimum zu entfliehen. Somit schafft es Realtime MCTS zwar die Heuristik lokal zu optimieren, nicht aber global (also zu siegen).

Das bedeutet insbesondere, dass während des Laufs des Algorithmus ein hoher heuristischer Wert nicht auf das baldige Siegen des Spiels hinweist. Das ist problematisch, da genau das erwünscht ist.

Relative MCTS scheint dieses Problem nicht zu besitzen: Hat eine Konfiguration eine hohe Sieganzahl, so ist auch die heuristische Bewertung der Endzustände hoch. Je höher die heuristische Bewertung eines Zustandes ist, desto größer die Chance, dass in seiner Nähe ein Siegzustand gefunden werden kann.

Dass Relative MCTS somit eine höhere Siegchance als Realtime MCTS hat ist somit nachvollziehbar.

Relative MCTS gegenüber bedingtem Relative MCTS

Bedingt Relative MCTS arbeitet mit den als optimal identifizierten Parametern von Relative MCTS. Unter anderem wurde der α-Parameter übernommen, der unter anderem Exploration und Exploitation gegeneinander abwiegt.

Die Annahme, dass diese Parameter optimal für bedingt Relative MCTS sind ist nicht klar. Insbesondere wird durch weniger binäre Splits weniger exploriert, was einen direkten Einfluss auf die Abwägung von Exploration und Exploitation hat und somit auf den Parameter α.

Dennoch wurde festgestellt, dass es Konfigurationen von bedingt Relative MCTS gibt, die besser abschneiden als die beste Konfiguration von Relative MCTS. Durch die nicht allzu geringe Varianz der Daten und den mit 100 Spielen nicht zu vielen Tests kann es dabei leicht zu Ausreisern kommen. In den Daten lässt sich aber erkennen, dass mit wenigen Splits das Ergebnis nah an dem Ergebnis von Relative MCTS liegt und mit zu wenigen Splits das Verfahren versagt. Interessante Ergebnisse sind zwischen diesen beiden Extremen zu erwarten. Leider gibt es nur sehr wenige Konfigurationen, die dazwischen liegen. Diejenigen Konfigurationen, die zwischen 0.5% und 3% auf die binären Splits verzichteten, scheinen dabei sehr vielversprechend zu sein, ebenso wie diejenigen, die erst nach 50 Tests das Weglassen von Splits ermöglichen. Unter anderem liegt die optimale Konfiguration von bedingt Relative MCTS in diesen Bereichen.

Die Daten zeigen insgesamt, dass es eventuell einen Mehrwert bringen könnte, wenn binäre Splits unter gewissen Bedingungen weggelassen werden. Das Belegen dieser Aussage, sowie das Identifizieren von konkreten Bedingungen ist mit den gewonnenen Daten aber nicht möglich. Dazu bedarf es weiteren Untersuchungen.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde Realtime MCTS angepasst, sodass der Unterschied zwischen Zuständen durch Präferenzen statt durch numerische Bewertungsfunktionen ermittelt wird. Der dabei entstandene Algorithmus wird Relative MCTS genannt und liefert in der hier untersuchten Testdomäne bessere Ergebnisse als Realtime MCTS.

Die Hauptunterschiede zwischen Relative MCTS und Realtime MCTS sind:

- Die Verwendung einer relativen Bewertung der Zustände untereinander statt einer absoluten Bewertung pro Zustand.
- Es wird in einem Knoten nicht eine Aktion ausgewählt sondern zwei.
- Die Relative UCB-Formel wird als Tree Policy verwendet.
- Es ergeben sich je nach Baumgröße mehrere Simulationen pro Iteration.
- Während der Backpropagation werden die Simulationen der unterschiedlichen Aktionen pro Knoten verglichen und mithilfe der Präferenz bewertet.

Um die Güte des Algorithmus zu testen wurden Relative MCTS und Realtime MCTS zusammen mit weiteren Algorithmen auf einer Testdomäne verglichen. Dazu wurde das 8Puzzle verwendet: Ein Schiebespiel mit einem 3×3 Feld, wo acht Zahlen aus einer Startposition durch Verschieben des freien Felds in eine fest definierte Reihenfolge gebracht werden müssen. Als Heuristik wurde die Manhattan-Distanz mit linearen Konflikten verwendet. Die Präferenzen ergeben sich aus der Heuristik, indem Zustände mit höherem Wert den Zuständen mit geringerem heuristischen Wert vorgezogen werden. Dieses Anwendungsszenario wurde gewählt, da diese Heuristik für dieses Problem zwar eine der besseren Heuristiken sind, sie aber dennoch viele lokale Optima besitzen. Entsprechend hat sich bestätigt, dass Realtime MCTS dieses Problem nicht zuverlässig löst. Insbesondere sind die Parametrisierungen, die einen hohen heuristischen Wert erreichen, jene, welche wenige Spiele lösen. Es scheint, dass die Heuristik Realtime MCTS in lokale Optima führt, aus denen es nicht mehr zuverlässig entkommen kann.

Relative MCTS scheint sich aus diesen lokalen Optima befreien zu können oder sie umgehen zu können, da die Anzahl der Siege mit höheren durchschnittlichen heuristischen Bewertungen steigt, anders als bei Realtime MCTS. Die Tatsache, dass bei Relative MCTS eine höhere Sieganzahl als Realtime MCTS ermittelt wurde, ist somit nachvollziehbar. Die Verwendung von Präferenzen statt der direkten heuristischen Werte brachte in dieser Domäne einen Mehrwert (31 gegenüber 21 von 100 Spielen gewonnen). Das Ergebnis der Arbeit lässt somit vermuten, dass Relative MCTS unter Umständen ein besseres Ergebnis liefert als Realtime MCTS. Die intuitive Vermutung ist, dass die Verwendung von nicht optimalen Heuristiken ein ausschlaggebender Grund dafür ist.

Es ist eine interessante Frage, ob sich dieses Ergebnis in anderen Domänen reproduzieren lässt, oder ob sich explizite Bedingungen für die Dominanz von Relative MCTS gegenüber Realtime MCTS finden lassen.

Weiterhin wurde bedingt Relative MCTS entwickelt und evaluiert. Hier handelt es sich um eine Erweiterung von Relative MCTS, welche nicht immer zwei Aktionen in einem Knoten auswählt, sondern unter bestimmten Bedingungen nur die beste Option wählt. Die Idee dieser Erweiterung ist, dass eine Aktion mit so hoher Wahrscheinlichkeit der Condorcet-Sieger ist,

sodass die alternativen Aktionen nicht mehr exploriert werden. Somit sinkt die Anzahl der Simulationen pro Iteration, und die Anzahl der Iterationen kann sich erhöhen, wodurch andere Knoten mehr exploriert werden können. In der Evaluation hat diese Erweiterung eine höhere Siegrate als Relative MCTS erzielt (35 gegenüber 31 von 100 Spielen gewonnen). Die Arbeit lässt vermuten, dass eine weitergehende Untersuchung vielversprechend ist.

Literatur

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[4] Weiwei Cheng, Johannes Fürnkranz, Eyke Hüllermeier, and Sang-Hyeun Park. Preference-based policy iteration: Leveraging preference learning for reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 312–327. Springer, 2011.

[5] Frederick J Dorey. In brief: statistics in brief: confidence intervals: what is the real result in the target population? *Clinical Orthopaedics and Related Research®*, 468(11):3137–3138, 2010.

[6] Johannes Fürnkranz and Eyke Hüllermeier. *Preference Learning*. Springer, 2011.

[7] Johannes Fürnkranz, Eyke Hüllermeier, Weiwei Cheng, and Sang-Hyeun Park. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine learning*, 89(1-2):123–156, 2012.

[8] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[9] Marc Lanctot, Mark HM Winands, Tom Pepels, and Nathan R Sturtevant. Monte carlo tree search with heuristic evaluations using implicit minimax backups. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[10] Bruno Lecoutre and Jacques Poitevineau. The significance test controversy and the bayesian alternative. *StatProb: The Encyclopedia Sponsored by Statistics and Probability Societies, http://statprob. com/encyclopedia/SignificanceTestControversyAndTheBayesianAlternative. html, accessed: 2016-09-28*, 2010.

[11] Tom Pepels, Mark HM Winands, and Marc Lanctot. Real-time monte carlo tree search in ms pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):245–257, 2014.

[12] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3 edition, 2014.

[13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[14] Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2007.

[15] Masrour Zoghi, Shimon Whiteson, Rémi Munos, Maarten de Rijke, et al. Relative upper confidence bound for the k-armed dueling bandit problem. In *JMLR Workshop and Conference Proceedings*, number 32, pages 10–18. JMLR, 2014.

Anhang

Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	3	5	14	13	11	14	16	10,86	4,5803
0,778	5	9	3	13	16	19	20	12,14	6,1974
1,056	10	13	12	11	17	20	15	14,00	3,2950
1,333	1	7	10	14	21	14	15	11,71	5,9453
1,611	6	6	3	13	15	10	11	9,14	3,9795
1,889	5	8	10	11	15	12	17	11,14	3,7580
2,167	4	6	3	15	14	10	16	9,71	5,0346
2,444	7	9	12	8	16	14	12	11,14	3,0439
2,722	7	7	16	11	16	19	17	13,29	4,5580
3	4	14	8	9	14	15	18	11,71	4,4949
Avg	5,20	8,40	9,10	11,80	15,50	14,70	15,70		
SD	2,3580	2,8355	4,5044	2,0881	2,4187	3,4366	2,5318		

(a) Anzahl der Siege pro Konfiguration von Realtime MCTS

Parameter C \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,5	60,33	63,40	59,14	57,77	67,00	70,43	71,75	64,26	5,1566
0,778	60,20	59,00	63,00	57,15	65,38	68,68	72,00	63,63	4,9884
1,056	55,40	59,92	59,67	64,45	60,06	74,40	68,33	63,18	5,9437
1,333	37,00	66,43	58,20	64,43	61,19	74,86	73,27	62,20	11,6960
1,611	62,00	72,00	62,33	57,31	63,80	68,00	72,64	65,44	5,2329
1,889	49,80	54,50	66,00	62,45	72,47	72,33	71,59	64,16	8,4245
2,167	55,00	49,33	39,00	54,73	64,00	71,00	64,13	56,74	9,9083
2,444	52,14	51,00	65,17	60,75	69,25	69,57	73,50	63,05	8,1426
2,722	43,00	59,86	62,63	58,27	63,38	68,37	64,29	59,97	7,5475
3	51,50	62,29	63,25	56,56	59,57	72,33	71,00	62,36	6,9109
Avg	52,64	59,77	59,84	59,39	64,61	71,00	70,25		
SD	7,5119	6,5395	7,3445	3,2436	3,8717	2,3162	3,3119		

(b) Durchschnittliche Anzahl der Züge gewonnener Instanzen pro Konfiguration von Realtime MCTS

Abbildung 26: Ergebnisse von Realtime MCTS

Parameter C	Rollout Length: 5	10	15	20	50	70	100	Avg	SD
0,5	0,76	0,72	0,73	0,80	0,77	0,77	0,77	0,76	0,0240
0,778	0,78	0,78	0,75	0,79	0,78	0,74	0,78	0,77	0,0170
1,056	0,78	0,79	0,81	0,80	0,80	0,77	0,74	0,78	0,0243
1,333	0,73	0,76	0,80	0,81	0,80	0,75	0,76	0,77	0,0275
1,611	0,73	0,78	0,79	0,80	0,79	0,76	0,78	0,78	0,0218
1,889	0,76	0,79	0,81	0,78	0,80	0,73	0,77	0,78	0,0242
2,167	0,76	0,80	0,79	0,79	0,79	0,74	0,76	0,77	0,0193
2,444	0,75	0,79	0,81	0,79	0,79	0,75	0,75	0,78	0,0225
2,722	0,77	0,78	0,79	0,78	0,78	0,73	0,76	0,77	0,0175
3	0,75	0,78	0,78	0,77	0,76	0,75	0,80	0,77	0,0166
Avg	0,76	0,78	0,79	0,79	0,79	0,75	0,77		
SD	0,0162	0,0208	0,0245	0,0107	0,0119	0,0130	0,0166		

Abbildung 27: Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von Realtime MCTS

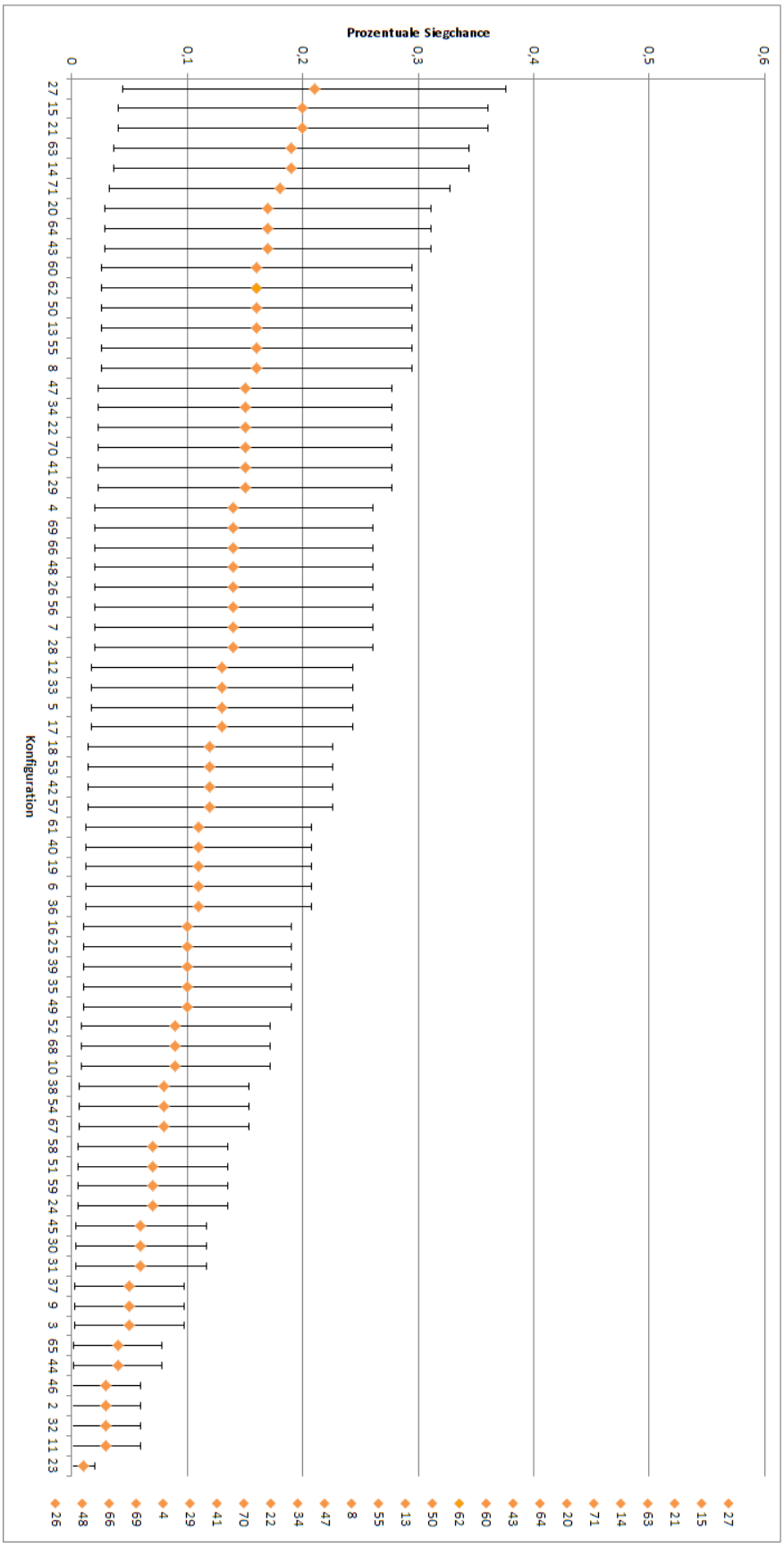


Abbildung 28: Pro Konfiguration von Realtime MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.

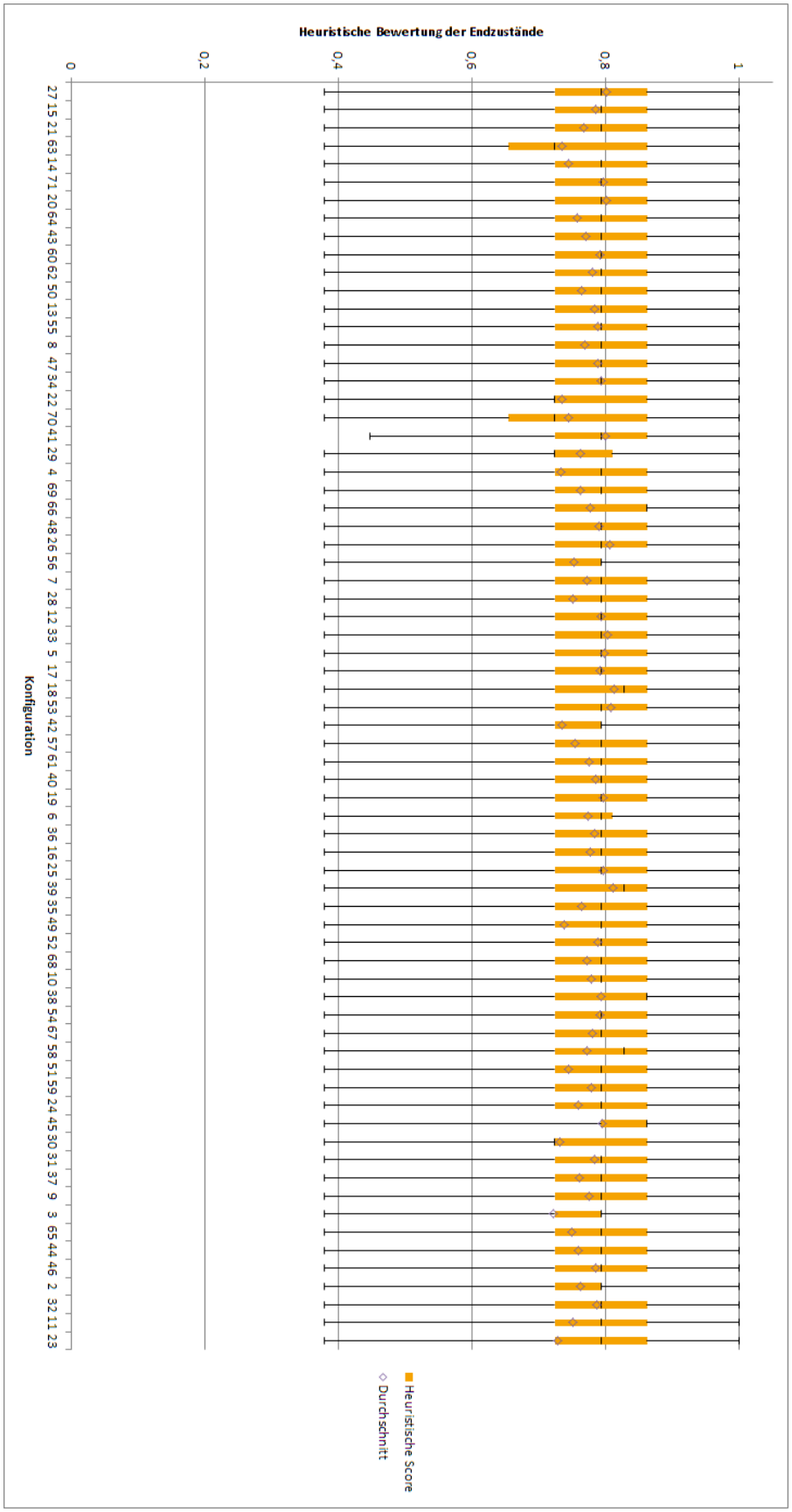


Abbildung 29: Pro Konfiguration von Realtime MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

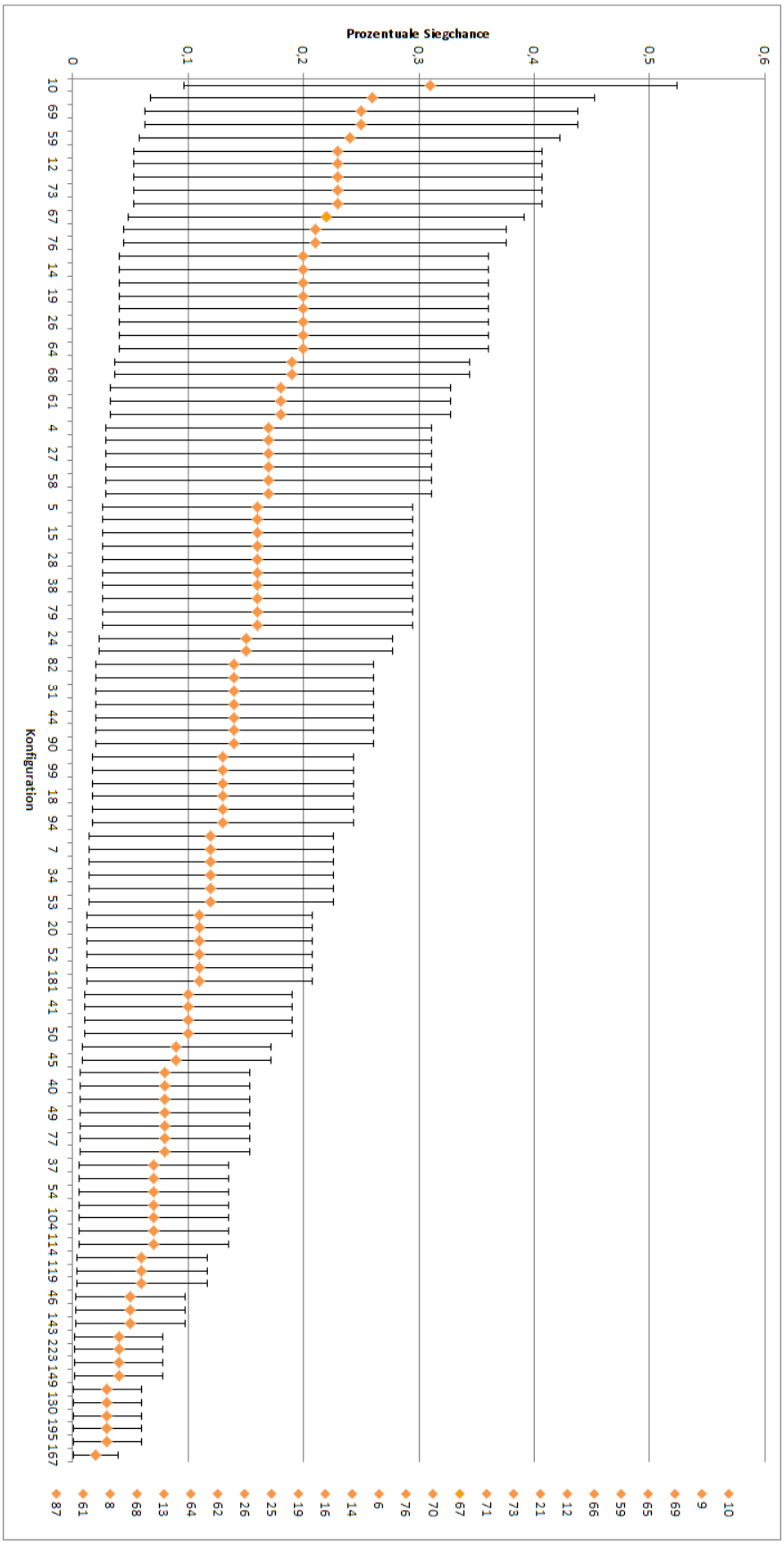


Abbildung 30: Pro Konfiguration von Relative MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.

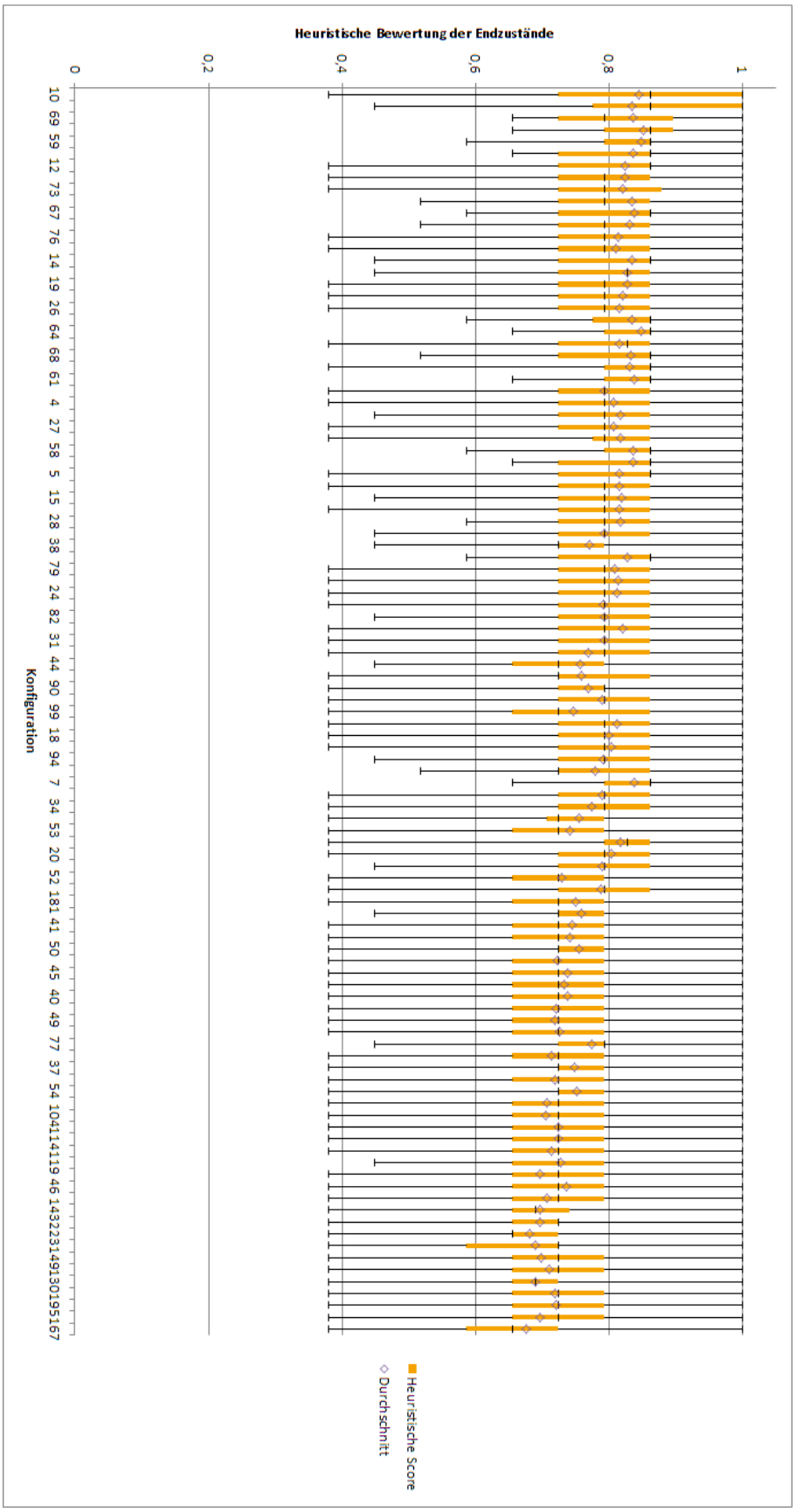


Abbildung 31: Pro Konfiguration von Relative MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

Parameter α \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,1	17	25	16	16	13	10	11	15,43	4,6247
0,2	11	31	13	20	12	10	10	15,29	7,1657
0,3	14	16	20	17	12	9	14	14,57	3,2888
0,4	17	23	11	16	14	14	11	15,14	3,8333
0,5	16	19	23	17	7	9	12	14,71	5,2567
0,6	20	20	16	11	16	5	7	13,57	5,5769
0,7	12	16	17	14	8	7	8	11,71	3,8065
0,8	18	20	15	16	8	8	3	12,57	5,8029
0,9	26	13	20	12	10	8	7	13,71	6,3856
1	24	23	23	15	7	5	3	14,29	8,5643
1,1	16	22	11	14	6	4	3	10,86	6,4681
1,2	18	19	13	12	7	8	3	11,43	5,4210
1,3	20	25	21	18	6	4	7	14,43	7,8714
1,4	17	21	8	14	6	2	5	10,43	6,4776
1,5	20	23	13	13	3	4	4	14,29	8,5643
Avg	17,73	21,07	16,00	15,00	9,00	7,13	7,20		
SD	3,8552	4,2343	4,4572	2,3944	3,5214	3,0302	3,5814		

(a) Anzahl der Siege pro Konfiguration von Relative MCTS

Parameter α \ Rollout Length	5	10	15	20	50	70	100	Avg	SD
0,1	59,35	63,88	64,00	61,00	79,92	78,60	77,55	69,19	8,3882
0,2	54,82	59,97	58,85	72,60	81,50	85,40	77,60	70,10	11,2719
0,3	59,29	57,88	59,60	71,00	81,83	87,00	82,14	71,25	11,5668
0,4	63,24	64,30	59,91	72,63	73,86	81,57	79,91	70,77	7,8349
0,5	51,38	66,26	68,22	75,59	79,57	82,78	86,33	72,88	11,0780
0,6	62,60	60,20	62,00	69,36	81,13	82,20	75,57	70,44	8,5983
0,7	56,50	64,75	66,18	73,86	84,00	80,14	86,50	73,13	10,2939
0,8	59,89	62,80	55,53	66,13	86,00	79,75	93,00	71,87	13,2769
0,9	61,54	55,46	69,60	73,83	87,20	85,25	92,14	75,00	12,7686
1	54,75	64,30	72,65	75,27	86,71	83,40	89,67	75,25	11,6650
1,1	62,50	70,27	67,00	75,71	82,67	89,50	89,00	76,66	9,9192
1,2	53,44	64,89	69,46	82,00	84,43	81,50	87,00	74,68	11,4695
1,3	56,90	70,04	71,57	76,78	91,33	91,00	86,71	77,76	11,7941
1,4	62,47	64,81	77,25	77,43	94,67	93,00	88,20	79,69	11,9695
1,5	63,00	66,22	66,08	75,92	89,00	94,50	85,00	75,25	11,6650
Avg	58,78	63,74	65,86	73,27	84,25	85,04	85,09		
SD	3,7692	3,9124	5,7227	4,8424	4,9898	4,8039	5,2102		

(b) Durchschnittliche Anzahl der Züge gewonnener Instanzen pro Konfiguration von Relative MCTS

Abbildung 32: Ergebnisse von Relative MCTS

Parameter α	Rollout Length 5	10	15	20	50	70	100	Avg	SD
0,1	0,84	0,85	0,81	0,81	0,75	0,76	0,75	0,80	0,0397
0,2	0,82	0,84	0,80	0,82	0,78	0,74	0,76	0,79	0,0343
0,3	0,82	0,82	0,83	0,81	0,76	0,72	0,76	0,79	0,0378
0,4	0,81	0,83	0,80	0,82	0,77	0,76	0,73	0,79	0,0327
0,5	0,82	0,82	0,82	0,82	0,75	0,74	0,74	0,79	0,0374
0,6	0,81	0,84	0,82	0,79	0,77	0,74	0,75	0,79	0,0330
0,7	0,84	0,82	0,82	0,79	0,73	0,72	0,73	0,78	0,0466
0,8	0,83	0,83	0,81	0,79	0,74	0,72	0,71	0,78	0,0480
0,9	0,84	0,81	0,82	0,79	0,75	0,72	0,71	0,78	0,0475
1	0,85	0,84	0,82	0,79	0,71	0,70	0,72	0,77	0,0604
1,1	0,83	0,84	0,79	0,79	0,72	0,70	0,72	0,77	0,0527
1,2	0,84	0,83	0,80	0,78	0,73	0,71	0,70	0,77	0,0540
1,3	0,83	0,84	0,81	0,79	0,73	0,70	0,72	0,78	0,0537
1,4	0,84	0,83	0,77	0,77	0,70	0,68	0,71	0,76	0,0596
1,5	0,85	0,84	0,79	0,79	0,69	0,69	0,68	0,77	0,0604
Avg	0,83	0,83	0,81	0,80	0,74	0,72	0,73		
SD	0,0125	0,0107	0,0147	0,0133	0,0255	0,0238	0,0219		

Abbildung 33: Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von Relative MCTS

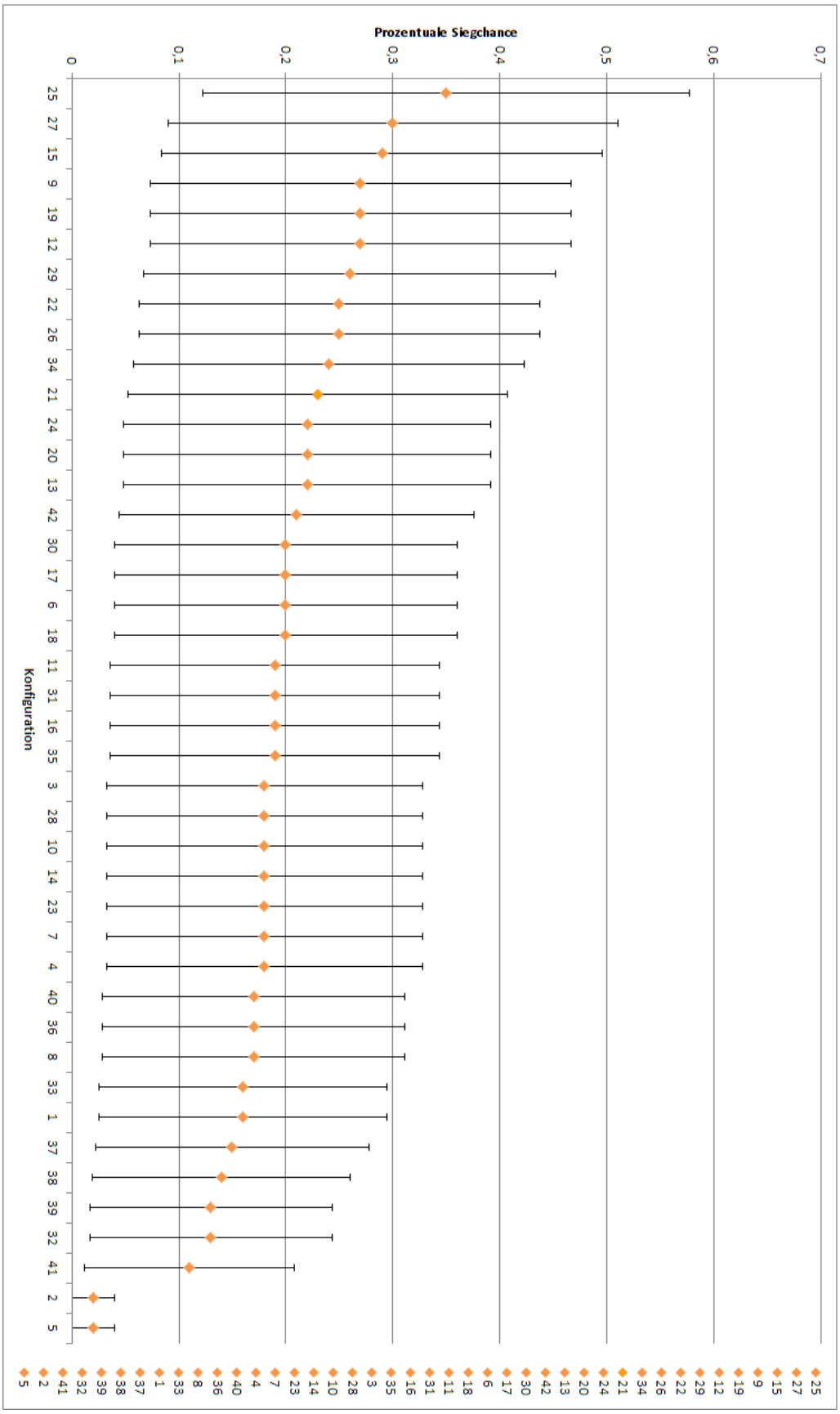


Abbildung 34: Pro Konfiguration von bedingtem Relative MCTS wird die ermittelte prozentuale Siegchance mit Varianz dargestellt.

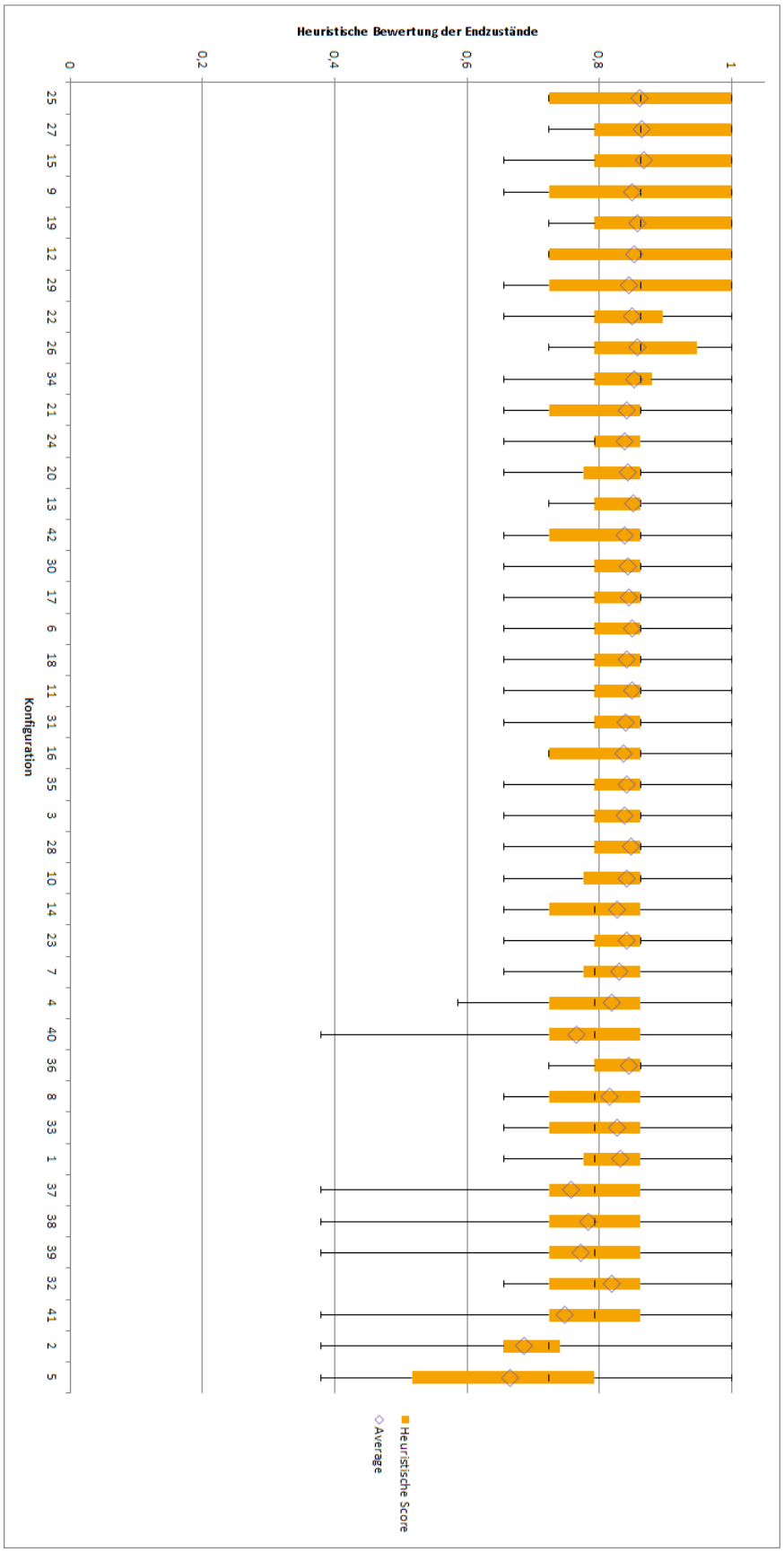


Abbildung 35: Pro Konfiguration von bedingtem Relative MCTS wird die heuristische Bewertung der Endzustände mit oberem und unterem Quantil, Minimum, Maximum und Durchschnitt dargestellt.

Signifikanzniveau	MinTests 0	5	10	15	20	50	Avg	SD
0,6	2	11	17	15	13	14	12,00	4,8305
0,7	2	16	13	19	17	24	15,17	6,7680
0,8	17	19	20	22	18	35	21,83	6,0942
0,9	18	25	20	26	19	29	22,83	4,0586
0,95	20	25	27	23	18	22	22,50	2,9861
0,98	16	19	21	18	20	27	20,17	3,4359
0,99	18	18	18	22	27	30	22,17	4,7755
Avg	13,29	19,00	19,43	20,71	18,86	25,86		
SD	7,2252	4,5670	3,9590	3,3685	3,9071	6,2204		

(a) Anzahl der Siege pro Konfiguration von bedingtem Relative MCTS

Signifikanzniveau	MinTests 0	5	10	15	20	50	Avg	SD
0,6	0,66	0,75	0,77	0,76	0,77	0,78	0,75	0,0392
0,7	0,69	0,83	0,82	0,84	0,84	0,85	0,81	0,0571
0,8	0,82	0,84	0,85	0,84	0,85	0,86	0,84	0,0140
0,9	0,82	0,85	0,84	0,85	0,84	0,87	0,84	0,0147
0,95	0,84	0,86	0,86	0,84	0,84	0,84	0,85	0,0084
0,98	0,83	0,85	0,84	0,84	0,84	0,85	0,84	0,0068
0,99	0,83	0,84	0,83	0,85	0,85	0,86	0,84	0,0128
Avg	0,78	0,83	0,83	0,83	0,83	0,85		
SD	0,0693	0,0350	0,0287	0,0304	0,0255	0,0267		

(b) Durchschnittliche heuristische Bewertung aller Endzustände pro Konfiguration von bedingtem Relative MCTS

Abbildung 36: Ergebnisse von bedingtem Relative MCTS Teil 1

Signifikanzniveau	MinTests 5	10	15	20	50	70	Avg	SD
0,6	88,00	75,73	69,71	67,40	66,08	72,00	73,15	7,3421
0,7	81,00	70,13	70,08	70,26	71,12	60,08	70,44	6,0484
0,8	79,47	62,89	68,60	63,09	59,78	57,74	65,26	7,1875
0,9	77,56	59,96	63,90	61,69	64,68	62,45	65,04	5,7994
0,95	73,70	64,76	64,41	64,04	59,11	65,73	65,29	4,3149
0,98	75,88	60,58	61,48	65,67	65,70	65,30	65,77	4,9621
0,99	76,33	61,00	62,44	69,18	51,96	63,20	64,02	7,4823
Avg	78,85	65,01	65,80	65,91	62,63	63,79		
SD	4,3443	5,4350	3,3143	2,9557	5,7554	4,2406		

(a) Durchschnittliche Anzahl der Züge gewonnener Instanzen pro Konfiguration von bedingtem Relative MCTS

Signifikanzniveau	MinTests 0	5	10	15	20	50	Avg	SD
0,6	65,28	20,64	15,88	14,40	11,33	7,11	22,44	19,5999
0,7	60,14	8,96	5,89	4,90	2,50	0,89	13,88	20,8468
0,8	27,47	3,11	1,30	0,71	0,28	0,07	5,49	9,8792
0,9	24,48	0,38	0,18	0,03	0,02	0,00	4,18	9,0771
0,95	24,27	0,21	0,15	0,01	0,00	0,00	4,11	9,0175
0,98	24,03	0,09	0,04	0,00	0,00	0,00	4,03	8,9480
0,99	24,21	0,08	0,04	0,00	0,00	0,00	4,05	9,0119
Avg	35,70	4,78	3,35	2,86	2,02	1,15		
SD	17,1773	7,1321	5,4756	4,9916	3,8954	2,4516		

(b) Der Prozentsatz wie oft kein Binärsplit gemacht wurde unter der Bedingung, dass ein Condorcet-Sieger-Kandidat ermittelt werden konnte.

Abbildung 37: Ergebnisse von bedingtem Relative MCTS Teil 2