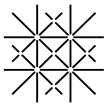


Universität
Basel

Change Detection in THTS

Bachelor Arbeit

Philosophisch-Naturwissenschaftliche Fakultät
Departement Mathematik und Informatik
Artificial Intelligence
https://ai.dmi.unibas.ch/

Beurteiler: Prof. Dr. Malte Helmert
Zweitbeurteiler: Dr. Thomas Keller

Joël Yannik Silvio Grossenbacher
joel.grossenbacher@stud.unibas.ch
2014-054-802

31.7.2019

Anerkennung

Ich möchte mich gerne bei Prof. Dr. Malte Helmert für die Möglichkeit diese Arbeit zu schreiben sowie auch Dr. Thomas Keller für die Begleitung dieses Projekts und seiner wertvollen Einsicht und Rückmeldungen bedanken. Die Kalkulationen wurden auf sciCore (http://scicore.unibas.ch/) das wissenschaftliche Rechnungszentrum an der Universität Basel ausgeführt. Alle Illustrationen sind von mir selbst mit dem Online Tool draw.io hergestellt worden.

Abstrakt

Ein wichtiges Feld in der Wissenschaft der künstliche Intelligenz sind Planungsprobleme. Man hat das Ziel, eine künstliche intelligente Maschine zu bauen, die mit so vielen verschiedenen Probleme umgehen und zuverlässig lösen kann, indem sie ein optimaler Plan herstellt.
Der Trial-based Heuristic Tree Search(THTS) ist ein mächtiges Werkzeug um Multi-Armed-Bandit-ähnliche Probleme, Marcow Decsision Processe mit verändernden Rewards, zu lösen. Beim momentanen THTS können explorierte gefundene gute Rewards auf Grund von der grossen Anzahl der Rewards nicht beachtet werden. Ebenso können beim explorieren schlechte Rewards, gute Knoten im Suchbaum, verschlechtern. Diese Arbeit führt eine Methodik ein, die von der stückweise stationären MABs Problematik stammt, um den THTS weiter zu optimieren.

Inhaltsverzeichnis

1 Einleitung

Im Fachgebiet Automatischen Planens ist man daran interessiert, Maschinen zu bauen, die selbstständig ein Problem lösen können. Oft wird davon ausgegangen, dass eine ausgeführte Aktion ein festes Resultat erbringen. Beispiel wäre hier der selbst saugende Staubsauger. Wenn er saugt ist der Boden staubfrei. Trotzdem können Aktionen nicht immer hundertprozentige Erfolgschance versprechen.
Darum hat man in der Informatik Verfahren entwickelt die Berücksichtigen, dass das Ausführen einer Aktion unter gleichen Bedingungen nicht das gleiche Resultat zurück gibt. Der MAB-Planer ist ein Produkt, das durch diese Anstrengungen entwickelt wurde.
Um die Arbeitsweise des Planers einfach zu erklären, wird das folgende Bild präsentiert: Wir haben ein Raum, in dem sich ein Kind befindet, das mit Spielzeuge spielt. Diese sind pyramidenförmig und anstelle der Spitze befindet sich ein Loch. Gespielt wird indem man eine Kugel in das Loch eines Spielzeug hineinfallen lässt. Der Fall der Kugel wird im Spielzeug durch Querstäbchen per Zufall nach Links oder nach Rechts abgeleitet. An welche Stelle die Kugel auf dem Boden landet determiniert ob der Spieler gewinnt oder verliert.
Die einzelnen Geräte haben unterschiedliche Gewinnchancen, die das Kind nicht kennt. Dafür merkt es sich wie viele Male es bei einem Spielzeug gewonnen hat. Das Kind möchte die beste Gewinnquote von einer Anzahl von Versuchen erreichen. Darum wird es die Geräte alle eine Zeit lang ausprobieren und Anfangen, die Geräte zu spielen, wo die grössten Gewinnchancen von den gesammelten Erfahrungen versprechen. Und seltener werden die wenig erfolgversprechenden Geräte ausprobiert.
Das MAB-Problem ist die einfachste Form des Markow Decision Process. Das MDP erlaubt es Problemstellungen mit Aktionen, die mehrere Ausgänge haben, zu modellieren. Im normal Fall sind im MDP die Chancen, wie wahrscheinlich ein Ausgang erfolgen wird, bekannt. Dies ist nicht der Fall für das MAB-Problem. Das Lösen des Problems ermöglicht uns Ansätze und Werkzeuge zu erarbeitet, die komplexere MDPs mit ähnlichen Bedienungen lösbar machen. Der THTS ist eines davon.
Diese Arbeit präsentiert ein Verfahren, das als eine Optimierung für den THTS dienen könnte.

2
Hintergrund

2.1 Markow Decision Process

Definition 1. *Ein Markow Decision Process, ist ein* 6*er-Tupel* $\mathcal{M} = \langle S, A, \mathcal{T}, R, S_0, H \rangle$, *wo:*

- S *ist eine endliche Menge von Zuständen;*
- A *ist eine endliche Menge von Aktionen;*
- $\mathcal{T} : S \times A \times S \to [0, 1]$ *ist die Transitionsfunktion, wobei* $\mathcal{T}(s, a, s')$ *die Wahrscheinlichkeit angibt, dass die im Zustand* s *ausgeführte Aktion* a *in den Zustand* s' *übergeht;*
- $R : S \times A \to \mathbb{R}$ *ist der Reward, der durch das Ausführen von Aktion* a *im Zustand* s, *gewonnen wird;*
- $s_0 \in S$ *ist der Initialzustand;*
- $H \in \mathbb{N}$ *ist der Horizont.*

Ein MDP erlaubt das Modellieren eines stochastischen, stationären Problems, wo der Akteur nur teilweise Kontrolle hat. Die im Problem definierten Aktionen sind voneinander unabhängig. Der Agent führt nur exakt H Aktionen aus und hört danach auf. Die Aufgabe ist es die Rewards zu maximieren. Die Lösung eines MDPs ist eine Policy.
Eine Policy π ist eine Funktion, die einem gegebenen Zustand s eine Aktion a zuweist ($\pi : S \times H \to A$). Wobei $a \in A(s)$ und $A(s)$ die Menge alle ausführbaren Aktionen in s sind. Eine Aktion a ist dann in s ausführbar, wenn ein $s' \in S$ existiert, so dass $\mathcal{T}(s, a, s') > 0$. Zusätzlich wird hinzugefügt das $\sum_{s' \in S} \mathcal{T}(s, a, s') = 1$ gelten muss. Sei nun $V_\pi(s, h)$ die State-Value Function, die wie folgt definiert ist:

Definition 2.

$$V_\pi(s, h) = \begin{cases} 0 & \textit{falls } h = 0 \\ Q_\pi(s, h, a) & \textit{sonst} \end{cases}$$

wobei $0 \leq h \leq H$, $Q_\pi(s,h,a) := R(s,a) + \sum_{a \in A(s)} (T(s,a,s') \cdot V_\pi(s', h-1)$ die Aktion Value Funktion ist.
Sei nun π^* die optimale Policy des MDPs. Eine Policy π ist dann die optimale Policy π^* wenn:

$$\pi^*(s,h) \in \arg\max_{\pi \in \Pi} V_\pi(s_0, h) \forall s, h$$

Wobei Π die Menge aller Policy[2] ist.

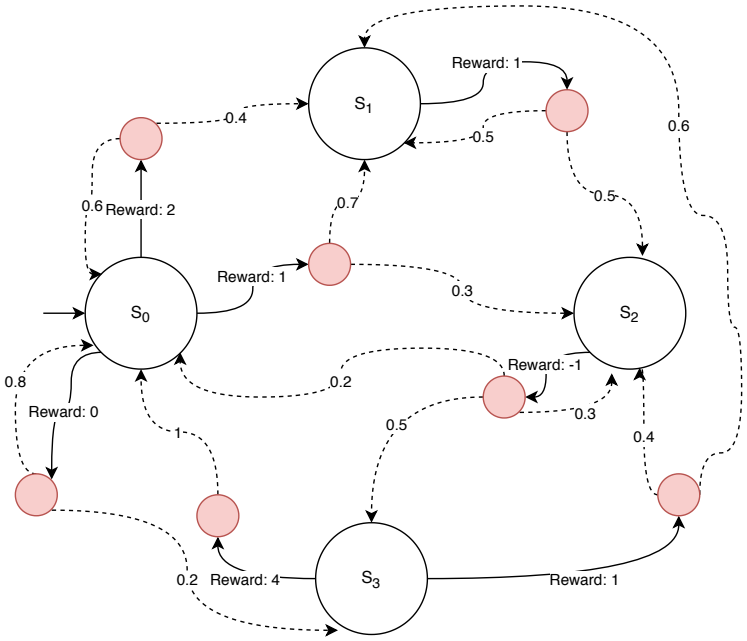


Abbildung 2.1: State Space von einem MDP

Da uns grosse MDPs interessieren, lohnt es sich nicht alle möglichen Kombinationen von Aktionen und deren Rewards auf einmal auszurechnen. Diese grossen MDPs werden mithilfe der Eingabesprache RDDL kompakt beschrieben. Um mit grossen MDPs zu rechnen wird Stück für Stück mit mehreren Durchläufen ein sogenannte Suchbaum aufgebaut und pro Durchlauf wird jedes mal kalkuliert, welche Aktionen angewendet werden.

Definition 3. *Unser Suchbaum ist wie folgt definiert:*

- *Sei* $d = \langle s, h, V^t, \mathbb{L}^t \rangle$ *ein Entscheidungssuchknoten (Decision Node).*
- *Sei* $c = \langle s, h, a, \mathbb{L}^t, Q^t \rangle$ *ein Wahrscheinlichkeitsknoten (Chance Node).*

wobei, $s \in S$, $\mathbb{L}^t$ die Anzahl Besuche des Knotens nach t Durchgänge sind. V^t die State-Schätzervalue des Knotens ist und $Q^t = Q(s,h,a)$, wobei $a \in A(s)$, nach t Durchgänge. Wobei $t \in 1, ..., T$ und $T \in \mathbb{N}$ die maximale Anzahl der Durchgänge ist. In den Decision Nodes werden die Entscheidung des Akteurs gefällt und in den Chance Nodes werden die stochastischen Ausgänge einer Aktionsanwendung bestimmt. Stück für Stück aufbauend heisst, dass bei jedem Durchlauf eine neue Chance Node erkundet und an den Baum angehängt wird. Nach einem Chance Node wird immer eine Decision Node als Blatt an den Suchbaum

gehängt. Das Trial-based Heuristic Tree Search[7] (THTS) so wie das Monte-Calro Tree Search [3] (MCTS) bieten das Framework ein MDP zu ein Suchbaum zu übersetzen und zu lösen. In dieser Arbeit wurde das THTS Framework benutzt. Die Arbeitsweise des THTS kann in drei Phasen unterteilt werden:

1. *Policy Anwendung*: Der Baum wird via Policy durchgelaufen.
2. *Erweiterung des Suchbaums*: Falls die Policy nicht mehr benutzbar ist, da die Mindestvoraussetzung nicht erfühlt ist, wird eine weitere Chance Node an den Baum angehängt.
3. *Backup*: die erzielten Rewards werden in die Knoten, hier Chance Nodes, mithilfe einer Backupfunktion abgespeichert.

Die folgenden Unterkapitel detaillieren, welche Policy- und Backupfunktion im THTS benutzt wurden.

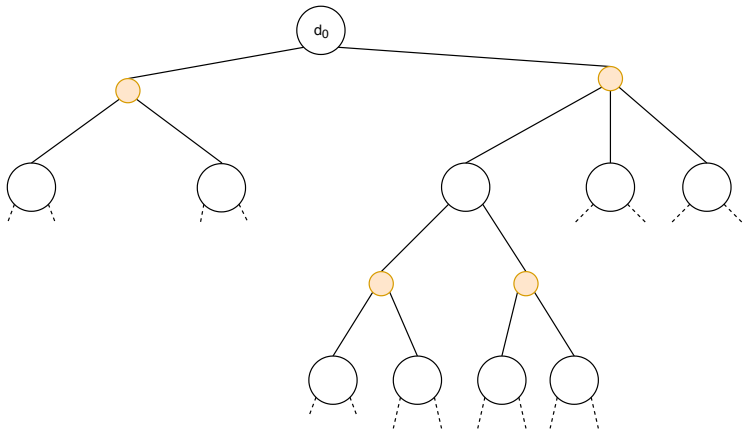


Abbildung 2.2: Beispiel eines Suchbaums mit rot als Chance Nodes und gross als Decision Nodes, wo d_0 die Wurzel des Baums ist und somit $d_0(s) = s_0$

2.1.1 Monte-Carlo-Backup

Das Monte-Carlo-Backup ist die beliebteste Backupfunktion[6] in Sachen Planungsprobleme. In den Chance Nodes werden alle erzielten Rewards kontinuierlich zu einem Durchschnittswert ausgerechnet und abgespeichert. Der Durchschnitt wird mit der folgenden Formel jeweils neu berechnet:

$$Q^{t+1}(c) = Q^t(c) + \frac{R(c) - Q^t(c)}{\mathbb{L}^t},$$

wobei $R(c)$ der Reward ist, der beim durchlaufen erzielt wurde. Diese Formel rechnet den Durchschnitt von einer ständig wachsender Mengen von Werten. Bei genügend Durchläufen stechen die Knoten raus die eine gute Reward Verteilungen haben. Das Problem beim Benutzen vom Monte Carlo ist, dass beim Explorieren ein schlechter Arm gespielt wird und dieses Resultat in die Elternknoten weiter verpflanzt wird. Dadurch kann ein eigentlich guter Arm weniger oft gespielt werden. Das andere Problem ist eher ein stochastisches Problem.

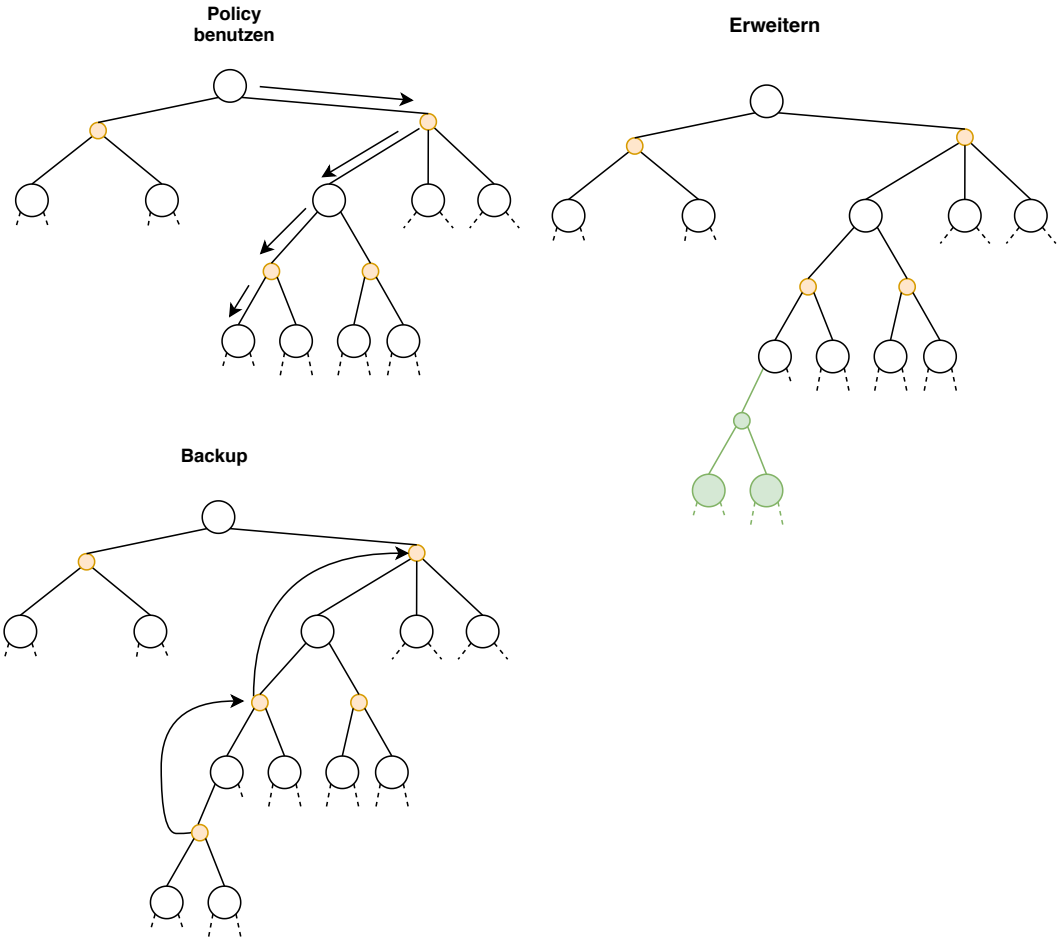


Abbildung 2.3: Die drei Phasen des THTS einfach Dargestellt

Ein eigentlich guter Arm wird nur selten gespielt wegen Pechresultate. Es kann eine grosse Zeitspanne vergehen bis der Durchschnitt der Rewards dieses Arms gut genug ist, dass er vermehrt gespielt wird.

2.1.2 UCB1

Der Upper Confidence Bound[1] (UCB) Algorithmus ist der in dieser Arbeit benutzte Policy-Algorithmus. Sei U der Decision Wert. U wird bei einem gegeben Decision Node d und gegebenes Kindes Chance Node von d folgend errechnet:

Definition 4. $U^t(c) = Q^{t-1}(c) + \sqrt{\frac{\log(\mathbb{L}^{t-1}(d))}{\mathbb{L}^{t-1}(c)}}$

Der UCB1 wird nur angewendet, wenn alle möglichen Chance Nodes am Baum angefügt sind. Die nächste ausgewählte Chance Node $c' \in \mathcal{C}$, wobei $\mathcal{C}$ die Menge aller Kindes Chance Nodes von Decision Node d ist, wird folgend gewählt:

$$c' = \arg\max_{c \in \mathcal{C}}(U^t(c))$$

Würden mehrere Chance Nodes für c' in Frage kommen, wird eines von ihnen zufällige ausgewählt.

Der Effekt ist hier, das Anhand der Qualität der Aktion die besten oft gespielt werden. Der Wurzelteil eines proportional häufig gespielten Knoten bleibt klein. Eine Aktion, die zur schlechteren Qualität gehört wird dann ausgewählt, wenn der Wurzelteil der schlechteren Aktionen zu gross wird und grösser als die Qualität der besseren Aktionen ist.

3

Stückweise stationäre Multi-armed Bandit

3.1 Multi-Armed-Bandit-Problem

Das Multi-Armed-Bandit-Problem[13](MAB) ist das kleinstmögliche MDP. Speziell am MAB ist nicht der MDP selbst vielmehr die zusätzlich existierende Rahmenbedingung, dass der Akteur kein Wissen über R und $\mathcal{T}$ hat. Dadurch ist der MAB in Grunde genommen kein Planungsproblem wie andere MDPs sondern ein Learning Problem. Der Agent steht vor einem Dilemma: Er muss Informationen über R und $\mathcal{T}$ sammeln um mit dieser Information die beste Policy zu erarbeiten. Da er nur durch viel Testen ein gutes Bild von R und $\mathcal{T}$ kriegen würde, ihn aber nur T viele Durchgänge zur Verfügung stehen, muss der Agent zwischen Informationsgewinnung und Ausnutzung vom gesammelter Information abwägen. Die im MAB-Problem beschriebenen Anzahl von Einarmige-Banditen korrespondiert zu der Anzahl, der möglichen Aktionen eines Markov Decision Process.

3.2 Stückweise stationäre Umgebung

Der MAB Ansatz lohnt sich sehr um Planner aufzubauen, die mit einem gewissen Grad an Unsicherheit in Hinsicht der Erfolgschancen rechnen müssen. Eine Einschränkung des MABs ist seine stationäre Annahme. Es gibt Probleme, deren Wahrscheinlichkeitsverteilungen nicht konstant bleiben. In vielen Problemen kann sich die Erfolgschance einer Aktion ändern. Darum wird in der Forschung Anstrengungen gemacht, das MAB zu erweitern um solche Probleme lösen zu können. Zwar wird nie davon ausgegangen, dass sich die Verteilung nach jedem Durchgang verändert. Wäre dies der Fall, so wäre das Problem mit dem Ansatz vom MAB nicht lösbar. Man nimmt an, dass nur nach einer mindestangenommenen Zeit eine Veränderung einer Verteilung eines Arms passiert. Man bezeichnet diese Umgebung als stückweise stationäre Umgebung. Es gibt zwei Ansätze, die aktive adaptive Policy und die passive adaptive Policy um den MAB für diese Umgebung anzupassen.
Passive adaptive Policies merken nie, wenn sich die Verteilung verändert hat, aber passen ihre Entscheidungen anhand der letzten Beobachtungen an. Beispiel für eine passive adative Policies ist der Discounted UCB Algorithmus von Kocsis und Szepesvári[8].
Aktive adaptive Policies haben zusätzlich einen Change Detection Algorithmus implementiert. Dieser beobachtet die vorliegende, sich verändernde Umgebung und startet den Al-

gorithmus neu, wenn er eine Veränderung entdeckt hat. Beispiele hierfür sind der AdaptEvE Algorithmus von Hartland[4] und die erweiterte Form vom Sliding-Window UCB, SW-UCL[15]
Die in dieser Thesis vorgestellten adaptiven Policies werden in dieser Arbeit nicht weiter behandelt. Im Folgenden wird sich die Arbeit auf einen Change Detection Algorithmus fokussieren, der noch nicht vorgestellt wurde. Vorerst muss noch klargestellt werden, was für Probleme sich stellen wenn man das Change Detection Problem in das MAB-Problem integriert will. Das grösste Problem ist, dass in Change Detection Probleme die Annahme gemacht wird, dass die Verteilungen vor und nach dem Change bekannt sind. Dies ist im MAB-Problem nicht gegeben. Weiterhin wird für jeden Arm eine Change Detection gebraucht. Somit laufen zur selben Zeit $|A|$ Change Detection Algorithmen für $|A|$ Arme. Diese sind immer Hungrig nach Samples um überhaupt richtig laufen zu können

3.2.1 Erweiterung des Settings

Definition 5. *Ein stückweise stationäre MAB ist ein* 4*-Tupel* $\mathcal{B} = \langle A, T, \gamma_T, R\rangle$ *wo:*

- A*: eine endliche Menge von Aktionen.*
- T *die Menge der Durchgänge ist*
- γ_T *die Anzahl an abrupten Veränderungen bis zum Zeithorizont* T*, was folgend als breakingpoints referiert wird, ist*
- $\{R_1, R_2, \ldots, R_{\gamma_T}\}$ *eine Menge von Rewards, wobei* $R_i : A \times \{0, 1\} \to [0, 1]$

Es werden folgende 2 Annahmen gemacht:

- ***Annahme 1:*** *kleinstes Interval zwischen zwei breakingpoints ist* $|A|M$*, wo* $M \in \mathbb{N}$ *ein Parameter ist, damit genügend Samples vorhanden sind um den breakingpoint zu erkennen*
- ***Annahme 2:*** *Es gibt ein bekannter Parameter* $\epsilon > 0$*, so dass* $\forall i \in |A|$ *und* $\forall t \leq T-1$*, falls* $Q_i \neq Q_{i+1}$ *dann* $|Q_k(c) - Q_{k+1}(c)| \geq 3\epsilon$*.*

3.3 CUSUM

Ein Change Detection Algorithmus überprüft eine Sequenz von unabhängigen Zufallswerten $y_1, y_2, \ldots, y_k$ und schlägt Alarm wenn eine Veränderung erkannt wurde. In klassischen Change Detection Problemen stammen die Zufallswerte von zwei verschiedenen Wahrscheinlichkeitsdichteverteilungen $p(\cdot|\theta)$, wobei θ der jeweilige Durchschnittswert der Verteilung ist. Diese Verteilungen sind im klassischen Fall bekannt.
Die Grundidee vom Cumulative Sum control chart[12] (CUSUM) Algorithmus ist, dass de Zufallswert y_i mithilfe der Funktion $\mathbf{F} = \log \frac{p(y_k|\theta_1)}{p(y_k|\theta_0)}$ als einen Schritt von einem Random Walk simuliert wird. Gemeint mit einem Random Walk ist, dass wir einen Wert mit dem Wert der Funktion $\mathbf{F}(y_k)$ aufsummieren. Durch die Natur des Logarithmus wächst der Random Walk positiv, wenn eine Change vorhanden ist und somit gehört der neue Wert eher zur

Verteilung $p(\cdot|\theta_1)$, oder er wächst negativ, wenn keine Change vorhanden ist und somit der Wert eher zur Verteilung $p(\cdot|\theta_0)$ gehört. Wenn der Random Walk einen positiven Grenzwert überschreitet, wird der CUSUM Alarm schlagen. Um Verzögerung der Change Detection durch negativ Werte zu verhindern, ist der minimal Wert des Random Walks Null. Durch die Natur des Random Walks ist die Change Detection einseitig Gerichtet, sprich man erkennt nur wenn es von $p(\cdot|\theta_0)$ zu $p(\cdot|\theta_1)$ wechselt. Um ein Wechsel von $p(\cdot|\theta_1)$ zu $p(\cdot|\theta_0)$ zu erkennen muss ein zusätzlicher Random Walk mit $\mathbf{F}' = \log \frac{p(y_k|\theta_0)}{p(y_k|\theta_1)}$ aufgebaut werden.

3.3.0.1 CUSUM im MAB

Der CUSUM, der im stückweise stationären MAB-Setting benutzt wird, ist ein zweiseitiger CUSUM und wurde für dieses MAB-Setting zurecht geschnitten, damit er funktioniert. Da uns die Wahrscheinlichkeitsdichteverteilungen unbekannt sind, benutzen wir eine Approximation, die von den erhaltenen Werten aufgebaut wird. Wie wir die Funktion $\mathbf{F}$ approximieren wird folgend erläutert.
Es werden die ersten M Samples abgespeichert. Mit den M Samples wird der Durchschnitt $\widehat{Q}_0 \triangleq (\sum_{k=1}^{M} y_k)/M$ berechnet. Ausserdem werden zwei Random Walks konstruiert die je entweder den positiven, oder den negativen Durchschnittswerteverlagerung der Samples zu darstellen.
Sei s_k^+, respektiv s_k^-, der momentaner Schritt des positiven, respektiv der negativen, Random Walk. s_k^+,s_k^- sind folgendermassen definiert:

$$(s_k^+, s_k^-) = (y_k - \widehat{Q}_0 - \epsilon, \widehat{Q}_0 - y_k - \epsilon)\mathbb{1}_{k>M}$$

g_k^+, respektiv g_k^-, traktieren die positive Tendenz des positven, respektiv den negativen Random Walk. Sie sind wie folgend Definiert:

$$g_k^+ = \max(0, g_{k-1}^+ + s_k^+), g_k^- = \max(0, g_{k-1}^- + s_k^-)$$

Eine Veränderung wird dann erkannt, wenn einer der beiden den Grenzwert h überschreitet. Für jeden spielbaren Arm wird ein CUSUM aufgebaut, also werden $|A|$ CUSUMs parallel laufen .
Als Beispiel sei $M = 5$. In M sind folgende Werte gespeichert: $y = \{4, 4, 5, 1, 1\}$ somit ist $\widehat{Q}_0 = 3$. Sei $h = 6$, $\epsilon = 0.33$, $g_{k-1}^+ = 0$ und $g_{k-1}^- = 0$.
Nun sei $y_k = 2$ so wird

$$(s_k^+, s_k^-) = (2 - 3 - 0.33, 3 - 2 - 0.33) = (-1.33, 0.67) \text{ und somit } g_k^+ = 0 \text{ und } g_k^- = 0.67.$$

Keine der beiden Werte ist grösser als h, daher wird keine Change erkannt.

Sei $y_{k+1} = 2.7$ so wird

$$s_{k+1}^+ = -0.63 \text{ und } s_{k+1}^- = -0.03 \text{ somit wird } g_k^+ = 0 \text{ und } g_k^- = 0.64.$$

Nun sei $y_{k+2} = 17$ somit ist

$$s_{k+2}^+ = 13.67 \text{ und } s_{k+2}^- = -14.33 \text{ darum werden } g_k^+ = 13.67 \text{ und } g_k^- = 0.$$

Da nun $13.67 > 5$ wurde ein positive Change Detection entdeckt.
Ein Cousin des CUSUMs ist der PHT Algorithmus[5]. PHT kann als eine Variante des CUSUM betrachtet werden mit dem Unterschied der Berechnung von s_k^+,s_k^- nämlich $(s_k^+, s_k^-) = (y_k - \widehat{y}_k - \epsilon, \widehat{y}_k - y_k - \epsilon)$ mit $\widehat{y}_k = \frac{1}{k}\sum_{s=1}^{k} y_s$. Der Unterschied zwischen dem PHT und dem CUSUM ist, dass beim CUSUM die ersten M erzielten Werte für die Berechnung von $\widehat{Q}_0$ genommen werden und im PHT alle erzielten Werten in die Berechnung von $\widehat{y}_k$ einfliessen. Beim Alarm schlagen durch eine Change wird der in Frage gestellte Arm auf den Initialzustand, nämlich die erst Benutzungs des Arms, zurückgesetzt.
Die Fusion vom UCB1 und dem für MAB zugeschnittenen CUSUM nennt man CUSUM-UCB[9]. Der CUSUM-UCB gehört zu den aktiven Change Detectoren.

4

Change Detection in MDPs

4.1 Motivation

Motivation dieser Arbeit, ist eine Optimierung des THTS mit UCB1 Policy und Monte-Carlo Backup mithilfe des Change Detection Algorithmus CUSUM. Es ist primär ein Versuch den Fokus des Algorithmus auf wenig explorierte Arme um zu lenken, wenn bei einer Exploration ein grosser Reward von diesen erzielt wurde. Als Beispiel haben wir ein Ausschnitt von unserem Suchbaum (4.1). Genauer sind nur der Wurzelknoten und zwei Chance Nodes normal abgebildet, die darauf folgende Teilbäume sind nur Abstrakt dargestellt. Links ist der weniger explorierter Teilbaum und rechts ist der zur zeitige beste Teilbaum. Zusehen sind die erzielten Rewards die durch den Monte-Carlo Backup als Durchschnitt in den Chance Nodes abgespeichert wurden. Obwohl es durch Exploration im wenig explorierter Teilbaum ein sehr guter Reward erzielt wurde, reicht es nicht aus dem Durchschnitt des Besten zu übertragen oder anzuschliessen. Der Algorithmus würde sich wenig Zeit geben diesen Teilbaum weiter aus zu Kundschaften. Mithilfe von Change Detection, wäre es möglich solche Ausreisser dem Algorithmus besser zu präsentieren, indem sie $Q^{t+1}(c) = R(c)$,$\mathbb{L}^{t+1}(c) = 1$ setzt. Wobei R der neue Reward, $\mathbb{L}$ die Besuche von der Chance Node c ist.

Sekundär ist es ein Versuch mithilfe vom CUSUM die negativen Aspekte vom Monte-Carlo Backup zu negieren. Ein solch negativer Aspekt ist die Fortpflanzung von schlechten Explorations-Resultate in die Elternknoten. Zwar werden kleine Ausreisser ins Negative von der Monte-Carlo Backup Funktion selbst korrigiert aber stärkere Ausreisser können sehr grossen Schaden verursachen.
Als Beispiel ist hier ein aus Schnitt eines Suchbaums (4.2). Genauer gesagt ist das der beste Ast des Suchbaums wo abgebildet wird. Der Rote Wert ist durch Exploration erreichte Wert, was in diesem Fall jeweils der minimaler Wert ist, sowie sind die Rot umrandeten Knoten die schlecht gespielten Knoten. Dieses Resultat würde nun weiter an die Elternknoten weiter verpflanzt. Die Qualität des Astes verschlechtert sich stark. Eine Change detection könnte solche Ausreisser erkennen und Gegenmassnahmen zu Verfügung stellen.

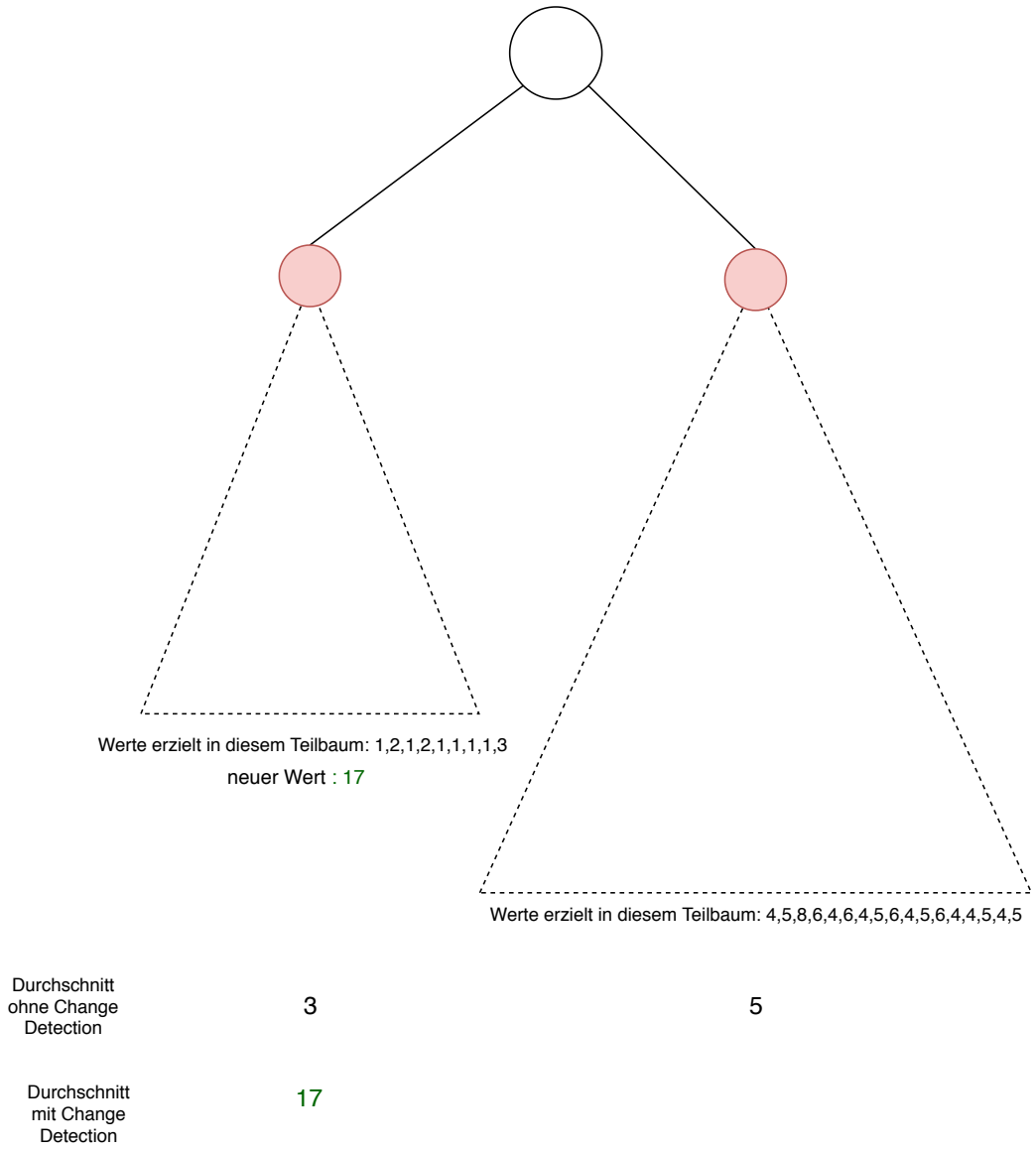


Abbildung 4.1: Fokuswechsel via Change Detection

4.1.1 Übernahme von stückweise stationäre Umgebung

Es handelt sich immer noch um ein MAB mit UCB-Policy. Die zwei Annahmen aus **Definition 5** aus Kapitel 3.2.1 gelten auch hier. Ausserdem wird der Grenzwert h so gewählt mit der Annahme das γ_T und T uns schon bekannt sind[9]:

$$h = \frac{1}{C_1} \log(\frac{T}{\gamma_T})$$

Wobei T Anzahl maximale Durchgänge, γ_T breaking points sind, $C_1 \triangleq \min(C_1^+, C_1^-)$, wobei $C_1^+ \triangleq \log(\frac{4\epsilon}{(1+\epsilon)^2}\binom{M}{\lceil 2\epsilon M \rceil}(2\epsilon)^M + 1)$ und $C_1^- \triangleq \log(\frac{4\epsilon}{(1-\epsilon)^2}\binom{M}{\lfloor 2\epsilon M \rfloor}(2\epsilon)^M + 1)$
Es wird angenommen, dass der Wert bei uns auch gilt. Diese Annahme wurde getroffen, damit ein Parameter weniger vorhanden ist um den wir uns kümmern müssen.

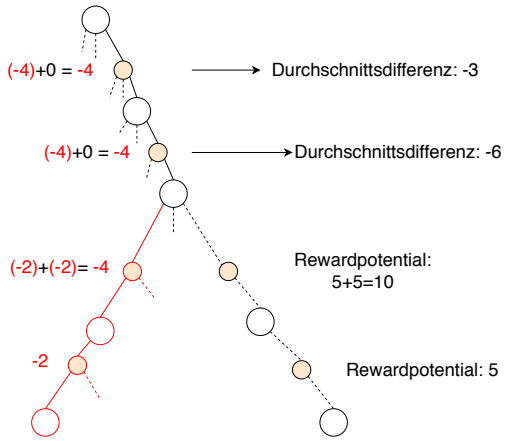


Abbildung 4.2: Beispiel eines Fehltritt

4.1.2 Unterschiede zur stückweise stationären Umgebung

Ein grosser Unterschied liegt darin, das unsere Umgebung keine stückweise stationäre Umgebung ist sondern eine rein stationäre. Dadurch wird γ_T nur als Kalibrieren der Feinfühligkeit der Change Detection benutzt. Ausserdem werden nicht die ersten M Samples gespeichert und mit diesen $\widehat{Q}_0$ ausgerechnet sondern die letzten M Samples am Zeitpunkt t. $\widehat{Q}_0$ wird bei jedem Durchgang aktualisiert. Welche Folgeaktionen bei einer Change Detection ausgeführt werden, hängt vom jeweiligem Random Walk ab. Beim positiven Random Walk wird der Arm wie in der Vorlage zurückgesetzt und der letzte Besuch wird als der erst Besuch gehandhabt. Dies hat die Konsequenz, dass bei der UCB1 Berechnung (siehe **Definition 4**) der Wurzelteil gross ist und $Q(c)$ ebenso einen grossen Wert besitzt. Dadurch werden diese Teilbäume vermehrt untersucht und falls so vermutet, wenn der Wurzelteil durch die erhöhten besuche kleiner wird und $Q(c)$ gleich bleibt, hat der Algorithmus eine weiterer optimaler Pfad erarbeitet.
Beim negativen Random Walk liegen 2 Strategien vor wie man diesen Arm behandelt:

- *the forgiving Strategy*, der letzte Wert wird ignoriert um ein gute Suchknoten gut zu halten, der Besuch des Knotens wird dennoch gezählt.
- Der Suchknoten wird ganz normal behandelt und der negativer Wert werden wie üblich verarbeitet

Die Forgiving Strategie gibt den Knoten einen Puffer, der erst mindestens nach M Durchgänge aktiv ist. Dieser wird verbraucht wenn ein zu schlechter Reward verarbeitet werden sollte. Der Puffer wird erst wieder nach M Durchgänge aktiv. Dies sollte verhindern, dass bei Sackgassensitutaionen, der Algorithmus nicht an den vorerst guten Knoten klammert, welche später nur noch fatale Nachfolger hervorbringt. Ob die Anzahl von Samples für die Forgiving Strategy gleich gross wie die Anzahl, die für die positive Change detection benutzt wird, ausreicht, ist fraglich.

4.1.3 lineare Aufteilung der Maximale Besuche

T wird beim einem nicht festsetzen hoch geschätzt in dem man eine simplere Problemstellung durchlaufen lässt und die Anzahl der Durchgänge zählt. Ausserdem wird T anhand der Tiefe des Suchknotens im Baum angepasst um die Chance für eine Change Detection in tieferen Regionen des Baums zu erhöhen. Grund dafür ist, dass diese weniger oft besucht werden können in Gegensatz zu den Knoten die sich oberen Regionen des Baums aufhalten. Zwei Arten werden in dieser Arbeit vorgestellt wie T aufgeteilt werden kann.

4.1.3.1 Statische T-Verteilung

Die Verteilung von T liegt der Annahme unter das alle durchführbaren Aktionen gleichmässig benutzt werden ohne dem Einfluss der Qualität der Aktion. Zwar wird T mit der Übergangswahrscheinlichkeiten der Chance Nodes modifiziert, aber die oberen Ebenen haben keinen Einfluss auf den T-Wert in den unteren Ebenen. Konkret

$$T_i(c) = \frac{T_{i-1}(d)}{|\mathcal{A}(d)|}\mathcal{W}(c)$$

wobei $c \in \mathcal{A}(d)$ die Aktion in frage, d der Vaterknoten und $i > 0$ die Tiefe der Ebene im Baum und $\mathcal{W}$ die Annäherung an $\mathcal{T}$ ist. Sei noch Anzumerken das T_0 der unskalierte Wert ist.

4.1.3.2 Dynamische T-Vereilung

T wird mithilfe der Boltzmann-Explorationsfunktion reskaliert. Die Funktion kalkuliert wie wahrscheinlich eine Aktion gegeben ihrer Qualität ausgewählt wird. Konkret $\mathbb{P} = \frac{\mathcal{Q}^t(d,c)}{\Sigma_{c' \in \mathcal{A}(d)}\mathcal{Q}^t(d,c')}$. Ausserdem ist $\mathcal{Q}(d,c) = e^{\frac{Q(c)}{\tau}}$, wo τ ein Reskalierungsfaktor ist und bei uns den Wert 0.15 besitzt. [6]. Die kalkulierte Wahrscheinlichkeit wird mit dem Erwartungswert des Vaterknotens multipliziert. Dieser wird mit der Wahrscheinlichkeiten der Chance Nodes weiter modifiziert. Konkret

$$T_i(c) = T_{i-1}(d)\mathbb{P}(c)\mathcal{W}(c)$$

Bei der Verteilung von T kann es zu einem Randfall kommen. Durch die Verteilung von T kann T_i kleiner werden als γ_T. Falls es dann $\mathbb{L}^t(c) + T_i(c) < \gamma_T$ würde das gegen die Definition von γ_T gehen. Wobei $\mathbb{L}^t(c)$ die Anzahl der Besuche des Chance Nodes c beim t-ten Durchgang ist. Falls dies eintreten würde, wird keine Change Detection betrieben. Da γ_T sehr klein ist wird dieser Fall sehr unwahrscheinlich eintreten.

5
Experimente

Die in den vorherigen Kapitel 3 und 4 beschriebenen Algorithmen wurden im Trial-based Heuristic Tree Search (THTS) Framework implementiert.
THTS[7] ist ein Framework, der Algorithmen beschreiben kann mit Hilfe von sechs Zutaten, die beliebig gemischt werden können. Die Komponenten sind: Action Selection (act [option]"), Outcome Selection (out [option]"), Trial length (T [options] [amount]"), Backup Function (backup [option]"), Recommendation Function (rec [option]") und Initializations (-init [option]"). In Klammern ist wie diese Komponente beim Aufrufen definiert werden kann.
Es wurden folgende zwölf Benchmarks zum Experimentieren verwendet, die ebenso in den Internationalen Probalistischen Planungswettbewerbe (IPPC) benutzt wurden

1. *IPPC 2011*:

    a) *recon*, wo der Agent einen Rover steuert die mit Wassersensoren und Lebenssensoren ausgestattet ist. Der Agent muss Objekte nach Wasser und Leben scannen und mögliche Schäden an den Sensoren mit einrechnen und wenn möglich reparieren

    b) *traffic*, wo der Agent via Ampeln die Verkehrssituation steuern muss

    c) *sysadmin*, wo der Agent ein Systemadministrator ist. Der Agent muss ein System mit Computern managen und je nachdem ein ausgegangene Computer neu starten

    d) *elevators*, wo der Agent Aufzüge steuert.

    e) *navigation*, wo ein Steuerbarer Roboter von A nach B fährt. Der Roboter hat die Chance anhand dem Aufenthaltsort verloren zu gehen.

    f) *game-of-life*, wo der Agent das Game-of-Life spielt

    g) *skill-teaching*, wo der Agent ein Schüler via Hinweise und Multiple-choice Fragen lehren muss. Der Agent muss die Hinweise und Frage der Kompetenz des Schülers anpassen.

    h) *crossing-traffic*, wo der Agent der eine befahrene Strasse über kreuzen will, wobei die Fahrzeuge nicht bremsen.

2. *IPPC 2014*:

a) *tamarisk*, wo bei einem Fluss die nativen Uferpflanzen durch eine invasive Pflanze verdrängt werden. Der Agent muss die Uferplätze der invasiven wegräumen und freie Plätze mit nativen bepflanzen.

b) *wildfire*, wo der Agent als Krisenmanager ein wildes Feuer kontrollieren und wichtige Ziele schützen muss.

c) *academic-advising*, ein Student möchte einen Abschluss machen und muss Kurse belegen und diese bestehen. Manche Kurse sind auf andere aufgebaut.

d) *trianle-tireworld*, wo der Agent mit dem Auto von A nach B will, wobei das Auto eine Reifenpanne kriegen kann und der Agent nur via Umwegen ein Ersatzreifen aufsammeln kann[1]

Die Benchmarks von IPPC 2011 und IPPC 2014 besitzen jeweils 10 Instanzen. Jede Instanz ist eine Beschreibung des Problems und variieren in ihrer Komplexität. Diese sind mit der Inputsprache RDDL[14] beschrieben. Die Relational Dynamic Influence Diagram Language (RDDL) ist eine Input Sprache die entwickelt wurde um speziellere Probleme zu beschreiben. Die Input Sprachen PDDL[10] (Planning Domain Definition Language) und die erweiterte Sprache PPDDL[16] (Probabilistic Planning Domain Definition Language), die normalerweise in Plannungsprobleme benutzt werden um die Probleme zu beschreiben, können folgende Problemstellungen als Beispiel nicht beschreiben. Steuermann von mehreren Aufzügen mit unabhängigen zufälligen Beanspruche von ankommenden Passagieren, Überwachung von einer logistische Domänen mit unabhängige bewegende Fahrzeuge und eingehende Störsignale und die Steuerung von einem unbemannten Luftfahrzeug mit Sensoren, die aber keine komplettes Umgebungsbild übermitteln können.
Anstelle, dass PPDDL erweitert wird um stochastische Probleme abzudecken und dennoch an Concurrency zu garantieren, was schwierig wäre weil sie auf Effekt-basierende Definitionen operiert, wurde RDDL, welches auf dBn[11] (dynamic Bayes Network) Formalisierung basiert, entwickelt. Es wird jeweils deklariert was für Objekte, Aktionen und Rewards es gibt und wie Wahrscheinlich ein Ereignis eintreten wird. In den Instanzen ist angegeben wie viele Objekte es gibt.
Die wichtigsten Parameter fürs experimentieren sind ϵ,γ_T und M. Ausserdem wird beim experimentieren getestet ob die beiden linearen T Aufteilungsstrategien einen Einfluss auf die Resultate der Experimente haben, die folgend mit dyn T für dynamische T-Veteilung und mit sta T für statische T-Verteilung abgekürzt werden. Die Resultate werden mit einem ohne Change Detection verglichen. Ausserdem wird verglichen ob Resultate mit der zusätzlichen Forgiving Strategy, in den Tabellen als FS abgekürzt, besser abschneiden als die mit nur positiven Reset-Methoden.
Es wurden zwei Experimentserien durchgeführt mit einer festen Anzahl von Trials. Die erste Reihe hat 100 Trials pro Runde und die zweite 1000 Trials pro Runde, beide Serien mit je 30 Runden pro Durchgang und 100 Durchgänge für jede Instanz. Beide Reihen benutzen ein fest gewähltes γ_T,M und ϵ. Nämlich $\gamma_T = 10$, $M = 4$ und $\epsilon = 0.27$. Der THTS Grundbefehl war bei beiden Experimente: THTS -act [UCB1] -out [MC] -backup [MC] -init [Expand -h

[1] besseres Verb

[Uniform] -iv 0]. Mit MC ist die Monte-Carlo Backup funktion gemeint und beim Erweitern des Baums wird Uniform erweitert.

5.1 Resultate

Die Endresultate(IPPC Scores) der Experimentserie mit 100 Trials sind Folgend:

	wildfire	triangle	elevators	recon	tamarisk	sysadmin	academic	game	traffic	crossing	skill	navigation	Total
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ohne Change Detection	0.77	0.76	0.82	0.17	0.8	0.08	0.55	0.33	0.0	0.3	0.2	0.21	0.42
sta T ohne FS	0.88	0.44	0.27	0.19	0.78	0.07	0.56	0.29	0.0	0.64	0.39	0.47	0.42
sta T mit FS	0.79	0.77	0.8	0.28	0.73	0.2	0.45	0.29	0.0	0.63	0.18	0.31	0.46
dyn T ohne FS	0.8	0.51	0.38	0.41	0.8	0.06	0.54	0.38	0.0	0.43	0.66	0.35	0.45
dyn T mit FS	0.7	0.71	0.69	0.49	0.81	0.1	0.57	0.54	0.0	0.68	0.66	0.3	0.53

Die Endresultate der anderen Serie mit 1000 Trials sieht wie folgt aus:

	wildfire	triangle	elevators	recon	tamarisk	sysadmin	academic	game	traffic	crossing	skill	navigation	Total
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ohne Change Detection	0.6	0.4	0.81	0.2	**0.98**	0.39	0.2	0.36	0.0	0.44	0.28	0.16	0.43
sta T ohne FS	0.91	0.49	0.52	0.24	0.72	0.07	0.43	0.59	0.0	0.44	0.54	0.38	0.48
sta T mit FS	0.48	0.39	0.75	0.24	0.88	0.37	0.21	0.39	0.0	0.65	0.29	0.22	0.44
dyn T ohne FS	0.68	0.39	0.8	0.29	0.72	0.31	0.25	0.76	0.0	0.45	0.19	0.36	0.47
dyn T mit FS	0.56	0.39	0.77	0.32	0.94	0.36	0.23	0.39	0.0	0.55	0.7	0.28	0.49

Beim betrachten der Resultate wird eine wichtige Erkenntnis klar: Ob eine Change Detection eine Verbesserung für ein THTS ist hängt stark an der Problemstellung selbst ab und wie die Change Detection eingestellt ist. Es kann kein Gewinner unter den Strategien ausgemacht werden. Die Ergebnisse der 1000 Trials Serie ist weniger zuverlässig in manchen Instanzen, da zu viele Trials ausgeführt wurden als es nötig wäre. Das grösste Optimierungspotential durch positive Change Detection liegt bei Problemen, die Rewardsarm sind, sprich es werden selten Rewards erzielt. Beispiel dafür sind Navigationsprobleme, wo der Akteur sich zu einem Ziel bewegen muss, da ein Reward nur beim erreichen des Ziels ausgeschüttet wird. Darunter fallen die Benchmarks recon,navigation und crossing. Was in den Resultaten widergespiegelt wird.
Das Potential von der Forgiving Strategie ist dann gross, wenn wir eine Problemstellung haben, die unabhängig von der Aktion des Akteurs, schlechte Rewards ausgeben kann. Ein Beispiel wäre die sysadmin Benchmark. Der Focus liegt hier eher auf dem Resultat der 100 Trials Serie. Hier sind die Resultate der ersten drei Instanzen von sysadmin:

	1	2	3
min	0.0	0.0	0.0
ohne Change Detection	0.84	0.0	0.0
sta T	0.69	0.0	0.0
sta T mit FS	**1.0**	0.0	**1.0**
dyn T	0.59	0.0	0.0
dyn T mit FS	**0.96**	0.0	0.0

Die Restlichen Instanzen dieser Benchmarks haben 0.0 als Resultat und sind daher uninteressant. Im sysadmin Problem, können per Zufall viele Computer ausfallen, auch wenn eine optimale Strategie geführt wird. Die Forgiving Strategie kann als gute Gegenmassnahme für solche Situationen dienen.

Der wichtigste Aspekt für die Qualität der Change Detection ist, wie feinfühlig die Change Detection eingestellt ist. Diese Feinfühligkeit wird durch γ_T,M, ϵ sowie die Art, wie T skaliert wird, beeinflusst. Beim wildfire Benchmark hat die Change Detection mit der statischen T-Verteilung und ohne Forgiving Strategy eine sehr optimale Einstellung in Falle der Feinfühligkeit.

6 Fazit

Obwohl Planer Algorithmen ein breites Spektrum von Problemen effizient lösen können, gibt es immer noch Problemstellungen, die durch Unsicherheit für Erfolg in den Aktionen nicht lösbar sind. Das Markow Decisions Process ist eine Möglichkeit um diese Problemstellung zu modellieren. Das Lösen vom Multi-Armed-Bandit-Problems, die kleinste Form des Markow Decision Process, gibt uns die Werkzeuge, komplexere Formen des Markow Decision Processes mit unbekannter Erfolgschancen oder sich verändernden Rewards zu lösen, wie zum Beispiel der Trial-based Heurstic Tree Search. Es werden Anstrengungen unternommen diese Algorithmen zu verbessern und zu erweitern. Sei es mit Hilfe der Einbeziehung von Change Detection um Verlagerung in den Rewards zu erkennen. In dieser Arbeit wird experimentiert, ob die Change Detection, verwendet werden kann um den Trial-based Heuristic Tree Search zu optimieren. Der primäre Fokus der Change Detection ist hier, den Algorithmus auf gute Rewards in wenig besuchten Knoten aufmerksam zu machen. Eine zusätzliche Art von Strategie wurde vorgeschlagen, wie der Knoten im Suchbaum behandelt wird, wenn eine Change Detection ins Negative erkannt wurde. Diese ist die Forgving Strategy, die eine Puffer bereit stellt, der den letzten Wert ignorieren lässt. Ausserdem wurden zwei Massnahmen eingeführt um die Chancen einer Change Detection für spätere Aktionen zu erhöhen. In den Experimenten hat sich folgende Ergebnisse herauskristallisiert:
Welche Art von Change Detection optimal ist hängt stark von der Problemstellung selbst ab. Die rein positive Change Detection sind am besten bei rewardsarme Problemen, wie zum Beispiel Navigationsproblemen. Die Forgiving Strategy ist dann von Vorteil wenn ein Problemstellung vorliegt, wo schlechte Rewards erzielt werden können, die unabhängig von der Aktion des Akteurs stammen. Wie gut eine Change Detection den Trail-based Heuristic Tree Search optimieren kann, hängt von der Feinfühligkeit der Change Detection ab. Diese Feinfühligkeit wird durch von uns eingeführten Parameter M,ϵ,γ_T wie auch die Art der Chancenerhöhung beeinflusst.

6.0.1 Ausblick für zukünftige Arbeiten

Eine Offene Frage ist wie stark die jeweiligen Parametern M,ϵ,γ_T die Feinfühligkeit der Change Detection beeinflussen. So wie auch bei gegebener Problemstellung die optimale

Konfiguration gefunden werden kann und welche T-Verteilung sich am besten eignet. Ausserdem muss getestet werden wie die Change Detection bei einem nicht festgesetzten T sich schlägt und es mit einem geschätzten T arbeiten muss, sowie auch wenn in allen Trials Aktionen bis zum Horizont ausgeführt werden.

Literaturverzeichnis

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Journal of Machine Learning Research*, 47:235–256, 3 2002.

[2] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, USA, 1st edition, 1957.

[3] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Institute of Electrical and Electronics Engineers (IEEE) Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 03 2012.

[4] Cédric Hartland, Nicolas Baskiotis, Sylvain Gelly, Michèle Sebag, and Olivier Teytaud. Change point detection and meta-bandits for online learning in dynamic environments. *Conférence Francophone sur l'Apprentissage Automatique (CAp)*, pages 237–250, 2007.

[5] David V. Hinkley. Inference about the change-point from cumulative sum tests. *Biometrika*, 58(3):509–523, 12 1971.

[6] Thomas Keller. *Anytime Optimal MDP Planning with Trial-based Heuristic Tree Search*. PhD dissertation, Albert Ludwigs Universität, Freiburg, 7 2015.

[7] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 06 2013.

[8] Kocsis Levente and Szepesvári Csaba. Discounted UCB. *2nd PASCAL Challenges Workshop*, pages 784–791, 4 2006.

[9] Fang Liu, Joohyun Lee, and Ness Shroff. A change-detection based framework for piecewise-stationary multi-armed bandit problem. *The Ohio State University*, 11 2017.

[10] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – the planning domain definition language – version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University, 1998.

[11] Kevin P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD dissertation, University of California, Berkeley, 2002.

[12] Ewan S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 06 1954.

[13] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 09 1952.

[14] Scott Sanner. Relational dynamic influence diagram language (RDDL): Language description. *National ICT Australia (NICTA) and the Australian National University*, 01 2011.

[15] Vaibhav Srivastava, Paul Reverdy, and Naomi E. Leonard. Surveillance in an abruptly changing world via multiarmed bandits. *53rd Institute of Electrical and Electronics Engineers (IEEE) Conference on Decision and Control*, pages 692–697, 12 2014.

[16] Hakan Younes and Michael Littman. PPDDL: The probabilistic planning domain definition language. *Carnegie Mellon University, Pittsburgh*, 10 2004.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Joël Yannik Silvio Grossenbacher

Matriculation number — Matrikelnummer
2014-054-802

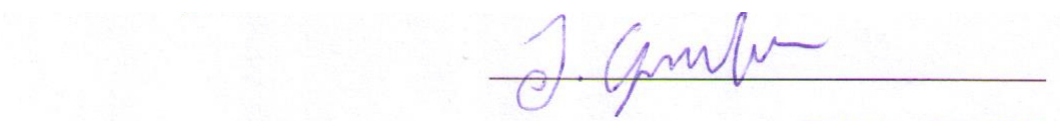
Title of work — Titel der Arbeit
Change Detection in THTS

Type of work — Typ der Arbeit
Bachelor Arbeit

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 31.7.2019



Signature — Unterschrift