Vergleich von Z-Learning und Q-Learning auf diskreten Markov Decision Problems

Bachelorarbeit

Technische Universität Berlin
Fakultät IV Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Methoden der Künstlichen Intelligenz (KI)

Vincent Froese

Matrikelnr. 314519

6. November 2010

1. Gutachter: **Prof. Dr. Manfred Opper**
2. Gutachter: **Prof. Dr. rer. nat. Klaus Obermayer**
Betreuer: **Dr. Andreas Ruttor**

Eidesstattliche Erklärung

Die selbständige und eigenhändige Ausfertigung versichert an Eides statt

Berlin, den 6. November 2010

.....

Unterschrift

Abriss

Diese Arbeit beschäftigt sich mit dem Thema Reinforcement Learning, einem Teilgebiet der Künstlichen Intelligenz. Es wird ein neuer Off-Policy Algorithmus namens Z-Learning vorgestellt, der von Emanuel Todorov für eine spezielle Klasse stetiger Markov Decision Problems entwickelt wurde. Dieser Algorithmus wird mit dem bisherigen Standard, dem Q-Learning, auf klassischen Markov Decision Problems verglichen. Zunächst wird eine kurze Zusammenfassung der Theorie des Reinforcement Learnings, sowie des Artikels von Todorov gegeben. Danach werden beide Algorithmen in verschiedenen Experimenten auf ihre Leistungsfähigkeit geprüft. Es zeigt sich, dass Z-Learning auch zum Lösen diskreter Markov Decision Problems eingesetzt werden kann und dabei sogar effizienter als das Q-Learning ist.

Abstract

This bachelor thesis deals with the topic of reinforcement learning. A new off-policy algorithm (Z-Learning) for continuous Markov Decision Problems as proposed by Emanuel Todorov is described, analyzed and compared to the state-of-the-art Q-Learning algorithm on discrete Markov Decision Problems. This work starts with an overview of the fundamentals of reinforcement learning followed by a brief summary of the original paper by Todorov. Both algorithms were subsequently tested through various experiments. The results show that Z-Learning is able to solve discrete tasks and in doing so outperforms Q-Learning.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Beim *Reinforcement Learning* (dt. Verstärkendes Lernen, kurz RL) handelt es sich um ein Teilgebiet der Künstlichen Intelligenz. Das typische Szenario beinhaltet einen *Agenten* (z. B. einen Roboter oder ein Computerprogramm), der in einer ihm unbekannten *Umgebung* agiert und eine bestimmte Problemstellung (z. B. Navigieren oder Schach Spielen) lösen soll. Der Agent besitzt also kein *Modell* der Umgebung, d. h. er kennt das meist probabilistische Verhalten von Aktionen und deren Auswirkungen nicht und muss somit durch zunächst zufälliges Ausprobieren ein solches aufbauen. Damit der Agent dies überhaupt tun kann, benötigt er ein *Feedback* (eine *Belohnung* oder *Verstärkung*, daher der Name), das ihm mitteilt, ob die eben ausgeführte Aktion gut oder schlecht war. Anhand dieser Information *lernt* der Agent eine *Strategie* (engl. Policy), um seine Aufgabe zu lösen.

Die Idee des Reinforcement Learnings besteht also darin, den Agenten durch Interaktion mit dem System an Erfahrung gewinnen zu lassen, und diese in eine optimale Strategie umzusetzen. Reinforcement Learning eignet sich dadurch zum Lösen einer großen Klasse von Problemstellungen und ist vor allem bei sehr komplexen Problemen häufig die einzige Möglichkeit, zu guten Ergebnissen zu gelangen. So wurde bspw. ein Programm erfolgreich darauf trainiert, Backgammon zu spielen [Tes94, Tes95] und ein autonomer Roboter lernte erfolgreich, beim Staubsaugen zu navigieren und Kollisionen mit Personen im Raum zu verhindern [Ger10].

1.2 Problemabgrenzung

Ein typisches Modell beim Reinforcement Learning ist das Markov-Entscheidungsproblem (engl. *Markov Decision Process*, kurz MDP), dessen Übergangswahrscheinlichkeiten und Belohnungen (siehe Kapitel 2) dem Agenten unbekannt sind. Die klassische formale Definition eines MDP geht hierbei von diskreten Zustands- und Aktionsmengen aus, weshalb auch von einem *diskreten* MDP gesprochen wird. Die diskrete Formulierung eines MDP führt zu einer nichtlinearen Problemstellung, deren Lösung nur aufwendig approximiert werden kann. Das hierfür aktuell wohl meistverwendete Verfahren stammt von Watkins aus dem Jahre 1989 und trägt den Namen *Q-Learning* [Wat89].

Emanuel Todorov[1] führte 2006 in seinem Artikel „*Linearly-solvable Markov decision problems*“ [Tod06] eine neue Klasse stetiger MDP ein, welche analytische Konzepte erlauben und das Optimierungsproblem linearisieren. Todorov schlägt auch gleich einen entsprechenden Algorithmus zum Approximieren einer Lösung vor, den er *Z-Learning* tauft. Das Z-Learning soll dank der Konstruktion der stetigen MDP effizienter sein als das bisher verwendete Q-Learning. Allerdings basiert die Formulierung stetiger MDP auf einigen Annahmen, die in klassischen MDP so nicht auftreten. Zwar zeigt Todorov in seinem Artikel, wie ein diskretes MDP in ein stetiges eingebettet werden kann, doch benötigt man dazu vollständige Kenntnis des Modells, was in den häufigsten Anwendungsfällen nicht gegeben ist. Die Ergebnisse, die Todorov vorbringt, belegen zwar die Überlegenheit von Z-Learning gegenüber Q-Learning, allerdings stammen diese von Experimenten auf einem stetigen MDP und einem speziell dazu entworfenen diskreten MDP, welches wiederum Wissen über das Modell voraussetzte [Tod06, Tod09].

Es bleibt also offen, ob sich der Z-Learning Algorithmus auch für klassische Problemformulierungen in Form von diskreten MDP eignet und wie er unter diesen Bedingungen im Vergleich zu Q-Learning abschneidet. Diese

[1] Applied Mathematics and Computer Science & Engineering
University of Washington

Frage soll Gegenstand der vorliegenden Arbeit sein. Um dies zu untersuchen, werden Experimente mit beiden Algorithmen durchgeführt. Als Problemstellung werden sogenannte *Gridworlds* (siehe Kapitel 3) verwendet werden. Aufgabe des Agenten ist es, einen optimalen Weg ins Ziel zu finden. Die Algorithmen werden hinsichtlich ihrer Effizienz untersucht und bewertet.

1.3 Aufbau der Arbeit

Im Anschluss an die Einleitung werden im zweiten Kapitel die theoretischen Grundlagen des Reinforcement Learnings behandelt. Diese beinhalten u. a. die beiden Klassen Markovscher Entscheidungsprobleme, sowie die beiden Algorithmen Q- und Z-Learning. Im darauffolgenden dritten Kapitel werden die Algorithmen dann in verschiedenen Experimenten auf ihre Leistungsfähigkeit hin untersucht. Hierzu wird zunächst der experimentelle Rahmen gegeben, woraufhin dann die Resultate vorgestellt und analysiert werden. Im abschließenden, vierten Kapitel dieser Arbeit findet sich eine Zusammenfassung und Auswertung der Ergebnisse, sowie die Beantwortung der Fragestellung.

1.4 Schreibweisen

Wie in der Fachliteratur zum Thema Reinforcement Learning üblich, werden auch hier englische Begriffe, wie Reinforcement Learning, Policy, Reward, Return, State-Value etc. verwendet. In Anlehnung an [Tod06] wird vom Markov Decision Problem gesprochen, statt vom Markov Decision Process, was inhaltlich jedoch keinen Unterschied macht. Für häufig auftretende Begriffe werden Abkürzungen verwendet, welche alle bei der ersten Verwendung erklärt werden.

2 Reinforcement Learning

In diesem Kapitel werden die theoretischen Grundlagen zum Reinforcement Learning eingeführt, welche für die Lösung der in Kapitel 3 vorgestellten Probleme notwendig sind. Es werden formale Definitionen der Problemstellungen in Form von klassischen (diskreten) und stetigen Markov Decision Problems eingeführt. Zudem werden entsprechende Lösungsmethoden, darunter das Q- und Z-Learning, vorgestellt.

2.1 Klassische Markov Decision Problems

Das zu Grunde liegende Modell eines RL Problems wird üblicherweise als ein Markov Decision Problem formuliert. Der Begriff des MDP wurde bereits von Bellman in [Bel57, BD62] verwendet. In [RN04] wird ein MDP zusammenfassend als ein sequentielles Entscheidungsproblem mit Markov-Eigenschaft und additiven Gewinnen bei vollständig beobachtbarer Umgebung umschrieben. Es soll nun genauer darauf eingegangen werden, worum es sich dabei handelt. Begriffe, Definitionen und Notation sind im Folgenden weitestgehend aus [SB98] entnommen.

2.1.1 Die Markov-Eigenschaft

Zuerst soll der fundamentale Begriff erklärt werden, der dem Modell seinen Namen gibt: Die Markov-Eigenschaft.

Beim RL befindet sich der Agent zu jedem Zeitpunkt in einem gewissen Zustand. Dieser Zustand beinhaltet alle Informationen der Umgebung, die dem Agenten momentan zugänglich sind. Zu jedem dieser Zeitpunkte kann

der Agent aus einer Menge von Aktionen wählen, die im aktuellen Zustand möglich sind. Je nach durchgeführter Aktion landet der Agent mit einer bestimmten Wahrscheinlikeit in einem der Nachfolgezustände und erhält ein bestimmtes Feedback (Reward). Abbildung 2.1 stellt diesen Ablauf schematisch dar. Falls diese Wahrscheinlichkeiten des Übergangsmodells nur vom aktuellen Zustand und der gewählten Aktion abhängen, erfüllt das Modell die Markov-Eigenschaft. Anders ausgedrückt, ist es egal, welche Zustände der Agent bisher besucht hat und welche Aktionen er dort ausgeführt hat, entscheidend für das weitere Geschehen ist der aktuelle Zustand. Formal lässt sich dies folgendermaßen ausdrücken:

Definition 1 *Sei s_t der aktuelle Zustand zum Zeitpunkt t, a_t die gewählte Aktion und r_t der erhaltene Reward, dann erfüllt das Modell die Markov-Eigenschaft, wenn gilt*

$$P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, \ldots, r_1, s_0, a_0) = P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t).$$

Der gegenwärtige Zustand enthält also alle wesentlichen Informationen über den bisherigen Verlauf und bestimmt somit den zukünftigen Verlauf bei gegebener Aktion. Wie die Zustandsrepräsentation gewählt wird, um alle relevanten Informationen zu kodieren, wird hierbei nicht spezifiziert.

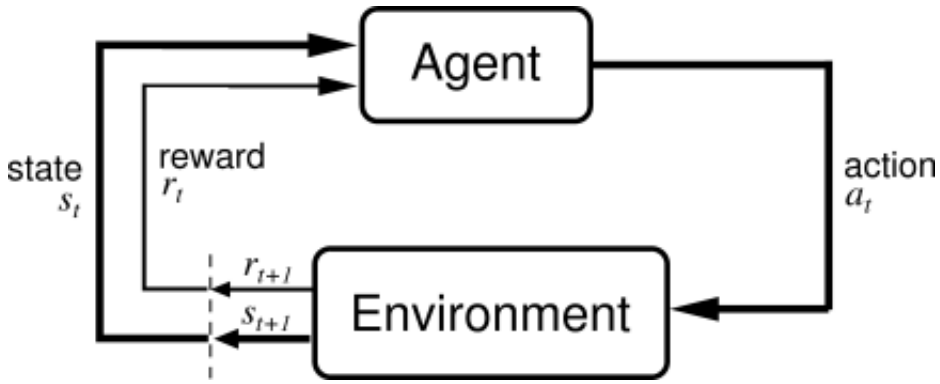

Abbildung 2.1: RL Ablauf aus [SB98, S. 52]

2.1.2 Nutzen und Horizont

An dieser Stelle soll nun erklärt werden, was eigentlich das Ziel des Agenten ist. Die Lösung des Problems besteht für den Agenten darin, den sogenannten *erwarteten Nutzen* (engl. expected return) zu maximieren. Während des Lernens erhält der Agent bestimmte Rewards $r_1, r_2, r_3, \cdots \in \mathbb{R}$. Beim Return handelt es sich um eine Funktion R_t dieser Rewards. Im einfachsten Fall

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T = \sum_{k=t+1}^{T} r_k \tag{2.1}$$

werden einfach alle erhaltenen Rewards vom Zeitpunkt t bis zu einem letzten Schritt T aufsummiert. Diese Definition macht natürlich nur Sinn, wenn sichergestellt ist, dass der Agent nach endlich vielen Schritten im Ziel (*terminaler* Zustand) ist. Solche Aufgabenstellungen werden *episodisch* genannt. Allerdings gibt es auch Problemstellungen beim RL, bei denen der Agent unendlich viele Schritte ausführt und nie terminiert. In diesem Fall hat man das Problem, dass der zu maximierende Return unendlich groß werden kann. Um dieses Problem zu lösen, führt man einen *Verminderungsfaktor* (engl. discounting rate) $0 \leq \gamma \leq 1$ ein. Der verminderte Return sieht dann wie folgt aus:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.2}$$

Interessant ist hierbei die inhaltliche Interpretation des Faktors γ. Dieser gibt nämlich an, wie stark zukünftige Rewards abgewertet werden. Für Werte nahe bei 0 ist der Agent sehr „kurzsichtig“, und versucht den unmittelbar nächsten Gewinn zu maximieren. Je größer γ wird, desto „weitsichtiger“ handelt der Agent und plant zukünftige Rewards mit ein.

Man zeigt leicht, dass für $\gamma < 1$ die unendliche Summe R_t einen endlichen Wert annimmt, falls die Folge der Rewards $\{r_k\}$ eine obere Schranke r_{max} besitzt:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \leq \sum_{k=0}^{\infty} \gamma^k r_{max} = \frac{r_{max}}{1-\gamma}. \tag{2.3}$$

Problemstellungen ohne Terminalzustände werden *kontinuierlich* genannt. Für $\gamma = 1$ enspricht R_t dem Return einer episodischen Aufgabe.

Zum Abschluss soll noch kurz auf das Konzept des Horizonts eingegangen werden. In der RL Literatur findet man üblicherweise drei Klassifizierungen von Problemen. Bei der ersten ist der Agent nach einer festen Anzahl N an Schritten auf jeden Fall fertig. Solche Aufgaben besitzen einen *endlichen* Horizont. Wenn hingegen die Dauer nicht von vornherein feststeht, der Agent aber sicher irgendwann einen terminalen Zustand erreicht, spricht man von *indefinitem* Horizont. Dies entspricht den episodischen Aufgaben. Wie schon erwähnt, terminieren kontinuierliche Probleme nicht und besitzen daher einen *unendlichen* Horizont. Im Folgenden werden wir episodische Probleme betrachten, wobei wir dennoch den verminderten Return verwenden. Dazu definieren wir den Terminalzustand als absorbierend mit Reward 0, wodurch die unendliche Summe kein Problem macht. Ist der Agent also einmal in einem Zielzustand angelangt, bleibt er dort und erhält 0 Rewards.

Als letzte Bemerkung sei noch erwähnt, dass die Rewards r_t durchaus negativ interpretiert werden könnnen, also als Verlust oder Kosten. Das Ziel des Agenten ändert sich dann insofern, als dass er den „erwarteten Nutzen“ zu minimieren versucht.

2.1.3 Markov Decision Problem

Nun kann ein MDP formal wie folgt definiert werden:

Definition 2 *Ein (endliches) Markov Decision Problem ist ein Tupel* $(\mathcal{S}, \{\mathcal{A}_s\}_{s\in\mathcal{S}}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'})$*, bestehend aus*

- $\mathcal{S}$ *(endliche) Menge von Zuständen*
- $\{\mathcal{A}_s\}_{s\in\mathcal{S}}$ *Familie von (endlichen) Mengen von möglichen Aktionen im jeweiligen Zustand* s
- $\mathcal{P}^a_{ss'} = P(s_{t+1} = s' \mid s_t = s, a_t = a)$ *markovsches Übergangsmodell vom Zustand* s *nach* s' *mit Aktion* a
- $\mathcal{R}^a_{ss'} = \mathbb{E}(r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s')$ *erwarteter Reward nach Übergang von* s *zu* s' *mittels* a*.*

Zustands- und Aktionsmengen können auch unendlich groß sein, allerdings soll hier der Einfachheit halber von endlichen Mengen ausgegangen werden. Ein diskretes MDP besitzt diskrete Zustands- und Aktionsmengen (z. B. endliche).

Dieses Tupel beinhaltet die wichtigsten Informationen zur Dynamik eines MDP. Nur die explizite Verteilung der Rewards ist als Erwartungswert verdichtet. Vollständige Beobachtbarkeit bedeutet in diesem Fall, dass der Agent zu jedem Zeitpunkt weiß, in welchem Zustand s er sich gerade befindet, die Zustände selbst unterliegen also keinem probabilistischen Modell.

2.1.4 Policy

Die Lösung eines MDP besteht darin, eine Strategie zu finden, die den erwarteten Return maximiert. Der Agent muss also bei jedem Schritt entscheiden, welche Aktion er in welchem Zustand ausführt. Formal lässt sich eine solche Policy π_t wie folgt definieren:

Definition 3 *Sei* $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$. *Eine Policy ist eine Abbildung*

$$\pi_t : \mathcal{S} \times \mathcal{A} \to [0, 1]$$
$$\pi_t(s, a) = P(a_t = a \mid s_t = s)$$

mit der Bedingung $\sum_{a \in \mathcal{A}} \pi_t(s, a) = 1$ *und* $a \notin \mathcal{A}_s \Rightarrow \pi_t(s, a) = 0$.

Es handelt sich also um eine probabilistische Policy. Sie gibt an, mit welcher Wahrscheinlichkeit Aktion a im Zustand s vom Agenten gewählt wird. Dabei dürfen nur Aktionen, die im aktuellen Zustand auch möglich sind, mit positiver Wahrscheinlichkeit gewählt werden. Der Index t deutet dabei an, dass der Agent seine Strategie in jedem Schritt an die Erfahrungen anpasst. Die RL Verfahren spezifizieren hierbei, wie der Agent seine Policy verändert (siehe Abschnitt 2.2).

Bemerkung 1 *Ein diskretes, endliches MDP mit einer festen Policy* π *entspricht einer homogenen, diskreten, endlichen Markov-Kette.*

2.1.5 Bewertungsfunktionen

Als nächstes kommen wir zu einem Begriff, der eng mit dem der Policy verknüpft ist. Die meisten RL Algorithmen basieren auf der Approximation einer Bewertungsfunktion (engl. value function), welche angibt, wie gut es für den Agenten ist, in einem bestimmten Zustand zu sein (engl. *state-value function*) bzw. eine bestimmte Aktion in einem bestimmten Zustand auszuführen (engl. *action-value function*). Als Maß für die Güte wird der zu erwartende Return verwendet. Hier entsteht der Zusammenhang zur Policy. Der Return hängt natürlich davon ab, welcher Policy π der Agent folgt. Formal lässt sich der erwartete Return von Zustand s unter Policy π wie folgt definieren:

$$V^\pi(s) = \mathbb{E}_\pi(R_t \mid s_t = s) = \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right), \qquad (2.4)$$

also als zu erwartende verminderte Gewinne, wenn sich der Agent in Zustand s befindet und ab hier Policy π verfolgt. Dementsprechend lässt sich der Wert der Aktion a im Zustand s mit darauffolgendem Handeln nach π durch den erwarteten Return definieren:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi(R_t \mid s_t = s, a_t = a) \\ &= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a\right). \end{aligned} \tag{2.5}$$

Eine wichtige Eigenschaft solcher Bewertungsfunktionen ist der folgende rekursive Zusammenhang zwischen dem Wert eines Zustands s und dem seiner möglichen Nachfolgezustände s':

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi(R_t \mid s_t = s) \\ &= \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s\right) \\ &= \mathbb{E}_\pi\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s\right) \\ &= \sum_{a \in \mathcal{A}_s} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_{t+1} = s'\right)\right] \\ &= \sum_{a \in \mathcal{A}_s} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]. \end{aligned} \tag{2.6}$$

Die resultierende Gleichung (2.6) heißt *Bellman-Gleichung* für V^π und gilt für alle π und s. Sie besagt, dass der Wert eines Zutands gleich dem Mittel der verminderten Werte der Nachfolgezustände, gewichtet nach der Wahrscheinlichkeit ihres Auftretens, plus dem unmittelbar zu erwartenden Reward ist.

2.1.6 Bellman-Optimalitätsgleichung

Ziel des Agenten ist es, eine optimale Policy zu finden, die den erwarteten Return maximiert. Jede Policy π induziert eine bestimmte Bewertungsfunktion V^π bzw. Q^π. Dadurch lässt sich eine Ordnung $\geq$ auf Policies definieren:

$$\pi \geq \pi' \Leftrightarrow \forall s \in \mathcal{S} : V^\pi(s) \geq V^{\pi'}(s). \tag{2.7}$$

Policy π ist also „besser als“ oder „gleich gut wie“ Policy π', falls ihr erwarteter Return größer oder gleich dem von π' ist. Eine optimale Policy π^* ist also „mindestens so gut wie“ alle anderen Policies, erfüllt also

$$\forall \pi : \pi^* \geq \pi. \tag{2.8}$$

Es kann verschiedene optimale Policies geben, die allerdings alle die gleiche optimale Bewertungsfunktion

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in \mathcal{S} \tag{2.9}$$

beziehungsweise

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}_s \tag{2.10}$$

besitzen. Da V^* eine Bewertungsfunktion ist, erfüllt sie die Bellman-Gleichung (2.6). Da es sich um die optimale Bewertungsfunktion handelt, lässt sie sich ohne Bezug auf eine Policy formulieren.

Die daraus resultierende *Bellman-Optimalitätsgleichung* für V^* besagt dann, dass der Value eines Zustands s unter einer optimalen Policy π^* gleich dem erwarteten Return der besten Aktion in diesem Zustand sein muss:

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}_s} Q^{\pi^*}(s,a) \\
&= \max_{a \in \mathcal{A}_s} \mathbb{E}_{\pi^*}(R_t \mid s_t = s, a_t = a) \\
&= \max_{a \in \mathcal{A}_s} \mathbb{E}_{\pi^*}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right) \\
&= \max_{a \in \mathcal{A}_s} \mathbb{E}_{\pi^*}\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s, a_t = a \right) \\
&= \max_{a \in \mathcal{A}_s} \mathbb{E}(r_{t+1} + \gamma V^*(s_{t+1}) \;\mid\; s_t = s, a_t = a) \\
&= \max_{a \in \mathcal{A}_s} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^*(s')]. \qquad (2.11)
\end{aligned}
$$

Entsprechend ergibt sich die Bellman-Optimalitätsgleichung für Q^*:

$$
\begin{aligned}
Q^*(s,a) &= \mathbb{E}_{\pi^*}(R_t \mid s_t = s, a_t = a) \\
&= \mathbb{E}_{\pi^*}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right) \\
&= \mathbb{E}_{\pi^*}\left(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \;\middle|\; s_t = s, a_t = a \right) \\
&= \mathbb{E}(r_{t+1} + \gamma \max_{a' \in \mathcal{A}_{s_{t+1}}} Q^*(s_{t+1}, a') \;\mid\; s_t = s, a_t = a) \\
&= \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma \max_{a' \in \mathcal{A}_{s'}} Q^*(s', a')].
\end{aligned}
$$

Diese Gleichung (2.11) wurde von Richard Bellman in [Bel57] entwickelt. Für endliche MDP besitzt die Bellman-Optimalitätsgleichung eine eindeutige Lösung V^*. Streng genommen, handelt es sich bei der Bellman-Optimalitätsgleichung um ein ganzes Gleichungssystem. Für ein MDP mit $|\mathcal{S}| = N$ Zuständen erhält man N Gleichungen in N Unbekannten. Allerdings sind die Gleichungen nichtlinear auf Grund des max Operators. Falls nun das Modell ($\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$) bekannt ist, kann man einfach ein beliebi-

ges Verfahren zum Lösen nichtlinearer Gleichungen anwenden, um V^* zu erhalten.

Im Falle vollständigen Wissens über das Modell lassen sich beispielsweise Algorithmen benutzen, die unter den Sammelbegriff *Dynamische Programmierung* (DP) [Bel57, BD62] fallen. Diese Algorithmen sind mathematisch fundiert und ausgearbeitet, treffen allerdings die unrealistische Annahme des vollständig bekannten Modells und benötigen meist sehr hohen Rechenaufwand. Dennoch sind sie aus theoretischer Sicht interessant, weshalb im folgenden Unterabschnitt ein solcher Algorithmus, mit dem sich V^* berechnen lässt, genauer betrachtet werden soll. Hat man einmal V^* ausgerechnet, erhält man auch automatisch eine optimale Policy, indem man einfach in jedem Zustand s nur Aktionen mit positiver Wahrscheinlichkeit auswählt, die das Maximum in der Bellman-Optimalitätsgleichung annehmen. Der Agent folgt dabei einer *greedy* (dt. gierig) Policy, die nur den besten nächsten Schritt betrachtet. Die Konstruktion der Bellman-Optimalitätsgleichung gewährleistet hierbei, dass diese Policy auch tatsächlich optimal ist.

Elementar für das Lösen von RL Aufgaben ist also das Berechnen einer optimalen Value Function V^* bzw. Q^*. Beim RL ist das exakte Modell allerdings in der Regel nicht bekannt. Daher müssen RL Algorithmen die Value Function aus Erfahrungen approximieren. Darauf wird im Abschnitt 2.2 eingegangen.

2.1.7 Value Iteration

Wie angekündigt, soll an dieser Stelle ein Algorithmus zur Berechnung der optimalen State-Value Function V^* besprochen werden, der dem Prinzip der Dynamischen Programmierung folgt. DP beruht auf dem *Optimalitätsprinzip* von Bellman [Bel57, BD62], welches dann erfüllt ist, wenn das ursprüngliche Problem in gleichartige Teilprobleme zerlegbar ist und eine optimale Lösung des Problems durch Zusammensetzen optimaler Lösungen der Teilprobleme generiert werden kann. Man löst also zuerst kleinere Teilprobleme und speichert deren Ergebnisse zur späteren Wiederverwendung.

Die Bellman-Optimalitätsgleichung wurde eben nach diesem Prinzip hergeleitet und birgt implizit eine mögliche Update-Regel für einen iterativen Algorithmus namens *Value Iteration*. Bei der Value Iteration geht man von beliebigen initialen Werten $V_0(s) \in \mathbb{R}$ für alle Zustände s aus und verbessert diese mit jedem Schritt nach Regel (2.12), um V^* zu approximieren.

$$\begin{aligned} V_{k+1}(s) &= \max_{a \in \mathcal{A}_s} \mathbb{E}(r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a) \\ &= \max_{a \in \mathcal{A}_s} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]. \end{aligned} \tag{2.12}$$

Man kann zeigen, dass für $k \to \infty$ die Folge $\{V_k\}$ gegen die eindeutige Lösung V^* konvergiert (siehe z. B. [RN04]). In der Praxis lässt man die Iteration abbrechen, sobald die größte Änderung eines Values in einem Schritt kleiner ist als eine kleine positive Zahl θ. Algorithmus 1 zeigt den Pseudocode nach [SB98].

Algorithmus 1 Value Iteration

Initialize V arbitrarily, e.g., $V(s) = 0 \quad \forall s \in \mathcal{S}$
repeat
  $\Delta \leftarrow 0$
  for all $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in \mathcal{A}_s} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$ (a small positive number)
return deterministic policy, π, such that
$\pi(s) = \arg\max_{a \in \mathcal{A}_s} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$

2.2 Q-Learning

Im vorigen Abschnitt wurde gezeigt, wie ein endliches MDP gelöst werden kann, falls das stochastische Verhalten der Rewards und der Zustandsübergänge bekannt ist. Beim RL kennt der Agent jedoch das Modell seiner Umwelt nicht. Daher muss er die optimale Value Function approximieren. Dies gelingt ihm nur durch Erfahrung, also durch direkte Interaktion mit der Umgebung oder Simulation. Dem Agenten stehen also nur Folgen nach dem Schema: Zustand, Aktion, Reward, Zustand, ... zur Verfügung. Innerhalb des RL gibt es zwei große Klassen von Verfahren zur Lösung eines Problems durch Erfahrung. Sogenannte *Monte Carlo Methoden* approximieren Value Functions, indem sie ganze Episoden betrachten und anhand des erhaltenen Returns die Schätzungen der Values aktualisieren. Auf diese Algorithmen soll in dieser Arbeit jedoch nicht weiter eingegangen werden. Vielmehr wird hier ein Vertreter der *Temporal-Difference Learning* (TD) Methoden vorgestellt. TD Learning gilt als zentrale und grundlegend neue Idee des RL. Es bildet eine Art Zusammenschluss von DP und Monte Carlo Methoden, indem es aus einzelnen Schritten online lernt, also aus direkter Interaktion mit der Umwelt. Zudem schätzen solche Verfahren die Values basierend auf bisherigen Schätzwerten. Als bekanntester Vertreter wird hier der *Q-Learning* Algorithmus vorgestellt. Die simpelste Form nennt sich *one-step Q-Learning* und führt nach jedem Schritt einer Episode ein Update der Action-Value Schätzungen Q_t durch. An großer Bedeutung gewann das Q-Learning, weil zum ersten mal die Konvergenz zur optimalen Policy bewiesen werden konnte (siehe Abschnitt 2.2.3).

Bevor allerdings der Algorithmus selbst gezeigt wird, müssen noch einige Grundlagen zum TD Learning behandelt werden.

2.2.1 Temporal Difference Learning

Zeile (2.13) zeigt die Update-Regel der State-Value Function beim TD Learning:

$$V(s_t) \leftarrow V(s_t) + \alpha_t[r_{t+1} + \gamma V(s_{t+1}) - V(s)]. \tag{2.13}$$

Der Agent braucht also nur den nächsten Reward r_{t+1} abwarten, um seine Approximation der State-Value Function zu verbessern, und nicht, wie bei Monte Carlo Methoden, eine ganze Episode. Um nun die State-Value Function einer Policy π zu berechnen, aktualisiert der Agent diese einfach fortlaufend, während er π folgend mit der Umgebung interagiert. Dieses Verfahren nennt sich TD(0) und ist ein Spezialfall vom sogenannten TD(λ), auf das hier nicht weiter eingegangen wird. Bei α handelt es sich um eine Lernrate, die meist mit der Zeit abnimmt (z. B. $\alpha_t = \frac{1}{t}$).

Da das Modell dem Agenten nicht bekannt ist, ist es unpraktisch die State-Value Function V zu lernen, da daraus noch keine Policy generiert werden kann. Um die beste Aktion zu wählen, müsste der Agent ja das Modell kennen. Deswegen wird beim TD Learning die Action-Value Function Q angenähert. Mit dieser ist eine deterministische Policy durch

$$\pi(s) = \arg\max_{a \in \mathcal{A}_s} Q(s, a) \tag{2.14}$$

sofort gegeben.

2.2.2 Erkundung und Ausbeutung

Der Agent hat zu Beginn des Lernens das Problem, dass ihm keine Werte der Action-Value Function bekannt sind. Daher hat er das Bestreben, möglichst den gesamten Zustandsraum zu erkunden, um die Values zu lernen. Andererseits muss er möglichst die besten Aktionen wählen, um eine gute Approximation der Action-Value Function, die den tatsächlichen Nutzen einer Aktion darstellt, zu erhalten. Um die optimale Policy π^* zu lernen,

muss der Agent also zwischen *Erkundung* (engl. Exploration) neuer Aktionen und *Ausnutzung* (engl. Exploitation) bisherigen Wissens abwägen.

Folgt der Agent beim Lernen der Greedy Policy, wählt also immer die bisher bestgeschätzte Aktion, wird er vielleicht nie die optimale Policy finden, da er nicht alle Aktionen ausprobiert. Dies lässt sich am einfachsten durch eine ϵ-*Greedy* Policy ändern, die mit einer Wahrscheinlichkeit von ϵ eine andere als die bisher am besten erscheinende Aktion wählt. Dieser naive Ansatz hat noch den Nachteil, dass unter allen anderen Aktionen gleichverteilt gewählt wird, ohne Rücksicht auf ihren geschätzten Wert. Um einen gestaffelten Wahrscheinlichkeitsverlauf unter allen Aktionen zu gewährleisten, kann man eine *softmax* Auswahl vornehmen. Dabei wird jeder Aktion eine Wahrscheinlichkeit gemäß ihrem Schätzwert zugeordnet. Die bestgeschätzte Aktion hat dabei immernoch die größte Wahrscheinlichkeit. Häufig wird eine Gibbs-Boltzmann-Verteilung benutzt:

$$\pi_t(s,a) = \frac{e^{\frac{Q_t(s,a)}{\tau}}}{\sum\limits_{a' \in \mathcal{A}_s} e^{\frac{Q_t(s,a')}{\tau}}}. \tag{2.15}$$

Der Parameter $\tau > 0$ (Temperatur) bestimmt dabei die Staffelung. Für große Werte ist die Auswahl annähernd gleichverteilt. Für $\tau \to 0$ wird die Policy greedy.

Das Problem der Exploration und Exploitation ist auch als *n-Armed Bandit Problem* bekannt. Es zeigt sich, dass schon eine ϵ-Greedy Policy tatsächlich bessere Ergebnisse erziehlt als eine Greedy-Policy (siehe z. B. [SB98]).

2.2.3 One-Step Q-Learning

An dieser Stelle soll nun der One-Step Q-Learning Algorithmus vorgestellt werden. Die entscheidende Update-Regel der Action-Value Function sieht wie folgt aus:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \big[r_{t+1} + \gamma \max_{a \in \mathcal{A}_{s_{t+1}}} Q(s_{t+1}, a) - Q(s_t, a_t) \big]. \quad (2.16)$$

Dieser Algorithmus gilt als Durchbruch im RL, da es mit Q-Learning zum ersten mal gelang, einen *Off-Policy* TD Algorithmus zu entwickeln. Off-Policy beudetet, dass zur Episodengenerierung eine andere Policy benutzt wird als zur Auswertung. So ist es möglich, dass der Agent beim Lernen z. B. einer ϵ-Greedy Policy folgt, dabei jedoch direkt eine Greedy Policy ohne Exploration lernt, die gegen die optimale Policy konvergiert. Dies bewirkt der max Operator in (2.16). Eine erste Beweisskizze für die Konvergenz befindet sich schon in [Wat89], der ausführliche Beweis wurde dann in [WD92] ausgearbeitet. Die wichtigste Bedingung ist dabei, dass alle Action-Values unendlich oft aktualisiert werden. Dies ist eine notwendige Bedingung, um überhaupt garantieren zu können, dass eine optimale Policy gefunden wird. Unter dieser Annahme konnte gezeigt werden, dass mit Wahrscheinlichkeit 1 die Folge $Q_t(s, a)$ für $t \to \infty$ für alle $s \in \mathcal{S}, a \in \mathcal{A}_s$ gegen $Q^*(s, a)$ konvergiert, falls die Rewards beschränkt sind $|r_t| \leq r_{max}$ und die Lernrate $0 \leq \alpha_t < 1$ folgende Bedingungen erfüllt:

$$\sum_{i=1}^{\infty} \alpha_t = \infty, \qquad \sum_{i=1}^{\infty} \alpha_t^2 < \infty.$$

Algorithmus 2 zeigt den One-Step Q-Learning Algorithmus als Pseudocode nach [SB98].

Die Konvergenzbedingungen für den Q-Learning Algorithmus können sogar noch abgeschwächt werden, wie u. a. mit Hilfe der Theorie der stochastischen Approximation bewiesen werden konnte [Tsi94, JJS94].

Algorithmus 2 One-Step Q-Learning

1: Initialize $Q(s, a)$ arbitrarily
2: **for all** episodes **do**
3: Initialize s
4: **repeat**
5: Choose a from s using policy derived from Q (e.g., ϵ-greedy)
6: Take action a, observe r, s'
7: $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a' \in \mathcal{A}_{s'}} Q(s', a') - Q(s,a)\big]$
8: $s \leftarrow s'$
9: **until** s is terminal
10: **end for**

2.3 Stetige Markov Decision Problems

Nachdem nun der klassische Formalismus des RL als diskretes MDP behandelt worden ist, soll die stetige Version eines MDP vorgestellt werden, wie sie von Todorov in [Tod06] eingeführt wurde. Todorov konnte zeigen, dass diese Klasse stetiger MDP das RL Problem vereinfacht, indem sie analytische Berechnung einer otimalen Policy erlaubt und durch einfache Transformation die Bellman-Optimalitätsgleichung in ein lineares Eigenwertproblem überführt. Die Herleitung beinhaltet gleichzeitig einen Ansatz für einen Off-Policy Algorithmus zum Approximieren der transformierten, optimalen State-Value Function.

Das sogenannte Z-Learning soll im nächsten Abschnitt vorgestellt werden. Zuerst wird an dieser Stelle die Theorie zu stetigen MDP vorgestellt. Die Notation folgt hierbei [Tod06].

2.3.1 Stetiges Markov Decision Problem

Definition 4 *Ein stetiges Markov Decision Problem ist ein Tupel* $(\mathcal{S}, \mathcal{U}(i), P(u), \ell(i,u))$*, bestehend aus*

- $\mathcal{S}$ *endliche Zustandsmenge*
- $\mathcal{U}(i)$ *Familie zulässiger (überabzählbarer) Aktionsmengen im jeweiligen Zustand* $i \in \mathcal{S}$
- $P(u)$ *stochastische Matrix, wobei* $p_{ij}(u)$ *die Wahrscheinlichkeit angibt vom Zustand* i *unter Aktion* u *nach Zustand* j *zu gelangen*
- $\ell(i,u) \geq 0$ *Kosten, sich im Zustand* i *zu befinden und Aktion* u *zu wählen.*

Weiterhin wird von Aufgabenstellungen mit indefinitem Horizont ausgegangen. Es existiert also eine nicht-leere Teilmenge absorbierender Zustände $\mathcal{A} \subseteq \mathcal{S}$ mit $p_{ij}(u) = \delta_{ij}$ und $\ell(i,u) = 0$ für alle $u \in \mathcal{U}(i), j \in \mathcal{S}$ und $i \in \mathcal{A}$, die mit positiver Wahrscheinlichkeit erreicht werden kann. Dann ergibt sich die Bellman-Optimalitätsgleichung für die unverminderte, optimale State-Value Function v wie folgt:

$$v(i) = \min_{u \in \mathcal{U}(i)} \left\{ \ell(i,u) + \sum_j p_{ij}(u) v(j) \right\}. \tag{2.17}$$

Der min Operator resultiert daher, dass die Rewards $\ell(i,u)$ in diesem Kontext als Kosten interpretiert werden. Der Agent will die erwarteten Kosten (Return) also minimieren.

2.3.2 Modell nach Todorov

Entscheidend für die neue Klasse stetiger MDP ist die Wahl der Aktionsmenge. Todorov definiert seinen Aktionsraum als den reellen Vektorraum $\mathbb{R}^{|\mathcal{S}|}$. Eine Aktion $\mathbf{u}$ ist also ein reeller Vektor, dessen Dimension gleich

der Anzahl an Zuständen des MDP ist. Nun muss noch die Wirkung einer Aktion definiert werden. Todorov geht davon aus, dass dem MDP eine sogennante *unkontrollierte* Markov-Kette zu Grunde liegt. Das unkontrollierte Verhalten der Umgebung kann also durch eine stochastische Matrix $\overline{P}$ beschrieben werden, deren Einträge $\overline{p}_{ij}$ die Übergangswahrscheinlichkeit von Zustand i in Zustand j angeben. Die *kontrollierte* Übergangsmatrix $P(\mathbf{u})$ wird nun wie folgt definiert:

$$p_{ij}(\mathbf{u}) = \overline{p}_{ij} \exp(u_j). \tag{2.18}$$

Daraus folgt offensichtlich $\overline{P} = P(\mathbf{0})$. Natürlich muss jede Aktion $\mathbf{u} \in \mathcal{U}(i)$ die Bedingung erfüllen, dass die i-te Zeile $\mathbf{p}_i(\mathbf{u})$ von $P(\mathbf{u})$ wieder eine diskrete Wahrscheinlichkeitsverteilung ist. Somit lässt sich die Menge der zulässigen Aktionen im Zustand i definieren als

$$\mathcal{U}(i) = \left\{ \mathbf{u} \in \mathbb{R}^{|\mathcal{S}|} \;\middle|\; \sum_j \overline{p}_{ij} \exp(u_j) = 1 \right\}. \tag{2.19}$$

Eine Aktion wirkt also direkt auf das Übergangsmodell ein und verändert dieses. Daher liegt die Idee nahe, die Aktionskosten durch die Unterschiedlichkeit der kontrollierten und der unkontrollierten Übergangswahrscheinlichkeiten zu definieren, also als Grad der Veränderung des Übergangsmodells durch die Aktion. Als Maß für die Differenz zweier Wahrscheinlichkeitsverteilungen wird typischerweise die Kullback-Leibler-Divergenz (KL-Divergenz) verwendet (siehe [KL51]). Damit ergeben sich folgende Aktionskosten:

$$r(i, \mathbf{u}) = KL(\mathbf{p}_i(\mathbf{u}) \parallel \mathbf{p}_i(\mathbf{0})) = \sum_{j:\overline{p}_{ij} \neq 0} p_{ij}(\mathbf{u}) \log \frac{p_{ij}(\mathbf{u})}{p_{ij}(\mathbf{0})}. \tag{2.20}$$

Aus den Eigenschaften der KL-Divergenz folgt, dass $r(i, \mathbf{u}) \geq 0$ und dass $r(i, \mathbf{u}) = 0 \Leftrightarrow \mathbf{u} = \mathbf{0}$. Einsetzen von (2.18) ergibt

$$r(i, \mathbf{u}) = \sum_j p_{ij}(\mathbf{u}) u_j. \tag{2.21}$$

Zu den Aktionskosten kommen noch beliebige Zustandskosten $q(i) \geq 0$ hinzu. Somit ergeben sich die Gesamtkosten $\ell(i, \mathbf{u}) = q(i) + r(i, \mathbf{u})$. Damit lässt sich die Bellman-Optimalitätsgleichung (2.17) umschreiben zu

$$v(i) = \min_{u \in \mathcal{U}(i)} \left\{ q(i) + \sum_j \overline{p}_{ij} \exp(u_j)(u_j + v(j)) \right\} \tag{2.22}$$

2.3.3 Lösung der Bellman-Optimalitätsgleichung

Diese spezielle Konstruktion ermöglicht nun die Anwendung analytischer Methoden zum Lösen des Extremalproblems (2.22) unter der Nebenbedingung (2.19). Eine Möglichkeit, solche Optimierungsprobleme unter Nebenbedingungen zu lösen, bilden Lagrange-Multiplikatoren (siehe z. B. [Bis06]). Dazu bildet man die Lagrange-Funktion

$$\mathcal{L}(\mathbf{u}, \lambda_i) = \sum_j \overline{p}_{ij} \exp(u_j)(u_j + v(j)) + \lambda_i \left(\sum_j \overline{p}_{ij} \exp(u_j) - 1 \right). \tag{2.23}$$

Notwendige Bedingung für ein Extremum von $\mathcal{L}$ bezüglich u_j ist

$$\frac{\partial \mathcal{L}}{\partial u_j} = \overline{p}_{ij} \exp(u_j)(u_j + v(j) + \lambda_i + 1) \stackrel{!}{=} 0. \tag{2.24}$$

Für $\overline{p}_{ij} = 0$ ist $p_{ij}(\mathbf{u}) = 0$ für alle $\mathbf{u}$ und trägt daher nichts zur Summe in (2.19) bei. Für $\overline{p}_{ij} \neq 0$ ergibt sich die eindeutige Lösung

$$u_j^*(i) = -v(j) - \lambda_i - 1. \tag{2.25}$$

Für die zweite Ableitung von $\mathcal{L}$ an der Stelle $u_j^*(i)$ gilt:

$$\frac{\partial^2 \mathcal{L}}{\partial u_j \partial u_j}(u_j^*(i)) = \overline{p}_{ij} \exp(u_j^*(i)) > 0, \tag{2.26}$$

womit auch die hinreichende Bedingung erfüllt ist und es sich daher bei (2.25) tatsächlich um ein Minimum handelt. Die Lagrange-Multiplikatoren lassen sich nun durch Einsetzen von (2.25) in die Nebenbedingung (2.19) berechnen:

$$\begin{aligned} 1 &= \sum_j \overline{p}_{ij} \exp(-v(j) - \lambda_i - 1) \\ \Rightarrow \lambda_i &= \log\left(\sum_j \overline{p}_{ij} \exp(-v(j))\right) - 1. \end{aligned} \tag{2.27}$$

Es ergibt sich also folgende optimale Aktion:

$$u_j^*(i) = -v(j) - \log\left(\sum_k \overline{p}_{ik} \exp(-v(k))\right) \tag{2.28}$$

Da nun die optimalen Aktionen in Abhängigkeit der optimalen State-Value Function bekannt sind, kann diese durch Einsetzen von (2.28) in (2.22) bestimmt werden. Dabei fällt der min Operator weg, da ja die optimale Aktion $\mathbf{u}^*(i)$ eingesetzt wurde:

$$\begin{aligned} v(i) &= q(i) + \sum_j p_{ij}(\mathbf{u}^*(i))(u_j^*(i) + v(j)) \\ &= q(i) + \sum_j p_{ij}(\mathbf{u}^*(i))(-\lambda_i - 1) \\ &= q(i) - \lambda_i - 1 \\ &= q(i) - \log\left(\sum_j \overline{p}_{ij} \exp(-v(j))\right) \end{aligned} \tag{2.29}$$

Negieren und Exponenzieren beider Seiten von (2.29) ergibt

$$\exp(-v(j)) = \exp(-q(i)) \sum_j \overline{p}_{ij} \exp(-v(j)). \tag{2.30}$$

Todorov führt nun die exponentielle Transformation

$$z(i) := \exp(-v(i)) \tag{2.31}$$

ein, welche die Bellman-Optimalitätsgleichung (2.29) linearisiert:

$$z(i) = \exp(-q(i)) \sum_j \overline{p}_{ij} z(j). \tag{2.32}$$

Zum Abschluss fässt man diese Gleichung noch in Matrixschreibweise zusammen. Dazu definiert man den Vektor $\mathbf{z}$ mit Einträgen $z(i)$ und die Diagonalmatrix $G = \text{diag}(\exp(-q(i)))$ und erhält das Eigenwertproblem

$$\mathbf{z} = G\overline{P}\mathbf{z}. \tag{2.33}$$

Gesucht ist also ein Eigenvektor $\mathbf{z}$ von $G\overline{P}$ zum Eigenwert 1 für den außerdem gilt:

$$\forall i \in \mathcal{S} : z(i) > 0 \tag{2.34}$$

$$i \in \mathcal{A} \Rightarrow z(i) = 1 \tag{2.35}$$

Dass ein solcher Vektor existiert und eindeutig ist, folgt daraus, dass die Bellman-Optimalitätsgleichung eine eindeutige Lösung besitzt. Um diese eindeutige Lösung zu finden, schlägt Todorov eine iterative Methode vor, bei der es sich im Grunde um die sogenannte Potenzmethode (ohne Normierung) handelt. Zeile (2.33) stellt einen offensichtlichen Ansatz bereit:

$$\mathbf{z}_{k+1} = G\overline{P}\mathbf{z}_k, \quad \mathbf{z}_0 = \mathbf{1}. \tag{2.36}$$

Diese Iteration konvegiert gegen den Eigenvektor zum größten Eigenwert, welcher in diesem Fall 1 ist, da $\overline{P}$ als stochastische Matrix Spektralradius $\rho(\overline{P}) = 1$ hat und die Matrix G mit Diagonaleinträgen $\exp(-q(i)) \leq 1$ das Spektrum nur runter skaliert. Da die Bellman-Optimalitätsgleichung jedoch eine eindeutige Lösung hat, gilt $\rho(G\overline{P}) = 1$ und $\mathbf{z}$ ist Eigenvektor zum Eigenwert 1. Auch die beiden Bedingungen (2.34) und (2.35) bleiben dabei erfüllt, da negative Werte nirgends auftauchen und für $i \in \mathcal{A}$ die i-te Zeile von $G\overline{P}$ nur Einträge δ_{ij} enthält und damit $z_k(i) = 1$ für alle k gilt.

Hat man die $z(i)$ bestimmt, lassen sich daraus die Werte der State-Value Function durch Retransformation $v(i) = -\log(z(i))$ bestimmen. Gleichung (2.28) gibt an, wie sich dann eine Policy bzw. die optimale Aktion $\mathbf{u}^*(i)$ berechnen lässt. Diesem Verfahren sei hier der Name *Z-Iteration* gegeben. Der Pseudocode wird in Algorithmus 3 gezeigt.

Algorithmus 3 Z-Iteration

1: Initialize $\mathbf{z} = \mathbf{1}$
2: **repeat**
3: $\quad \mathbf{z} \leftarrow G\overline{P}\mathbf{z}$
4: **until** convergence
5: **for all** $i \in \mathcal{S}$ **do**
6: $\quad v(i) \leftarrow -\log(z_i)$
7: **end for**
8: **return** policy $u_j^*(i) = -v(j) - \log\left(\sum_k \overline{p}_{ik} \exp(-v(k))\right)$

2.4 Z-Learning

Im vorherigen Abschnitt wurde eine Klasse spezieller MDP eingeführt, wie sie von Todorov beschrieben wurde. Sie ermöglicht es, die minimierte Bellman-Optimalitätsgleichung als Eigenwertproblem zu formulieren. Im Falle vollständigen Wissens über das Modell (G und $\overline{P}$) lässt sich die State-Value Function somit prinzipiell mit jeder Methode zum Lösen linearer Gleichungssysteme bestimmen.

In der Regel ist jedoch das Modell der Umgebung nicht vollständig bekannt und der Agent ist gezwungen, aus direkter Interaktion oder Simulation zu lernen.

2.4.1 Random Z-Learning

In seiner Konstruktion stetiger MDP hat Todorov implizit einen Ansatz für einen Off-Policy Algorithmus zur stochastischen Approximation der optimalen State-Value Function, ähnlich dem Q-Learning, mitgeliefert.

Dazu betrachte man Gleichung (2.32) und stelle fest, dass

$$z(i) = \exp(-q(i)) \sum_j \overline{p}_{ij} z(j) = \exp(-q(i)) \mathbb{E}_{j \sim \overline{P}(\cdot|i)}(z(j)). \tag{2.37}$$

Die daraus resultierende Update-Regel für eine stochastische Approximation von z lautet:

$$\hat{z}(i_k) \leftarrow \hat{z}(i_k) + \alpha_k [\exp(-q(i_k)) \hat{z}(i_{k+1}) - \hat{z}(i_k)]. \tag{2.38}$$

Bei α_k handelt es sich wieder um eine entsprechend abnehmende Lernrate. Man lässt also Samples $(i_k, q(i_k), i_{k+1})$ der unkontrollierten Markov-Kette $\overline{P}$ generieren und benutzt diese für eine iterative Approximation nach (2.38). Algorithmus 4 zeigt das Z-Learning als Pseudocode in Anlehnung an Q-Learning.

Algorithmus 4 Z-Learning

1: Initialize $z(i) = 1 \quad \forall i \in \mathcal{S}$
2: **for all** episodes **do**
3: Initialize i
4: **repeat**
5: Behave according to $\overline{P}$, observe q, j
6: $z(i) \leftarrow z(i) + \alpha[\exp(-q)z(j) - z(i)]$
7: $i \leftarrow j$
8: **until** i is terminal
9: **end for**

2.4.2 Greedy Z-Learning mittels Importance Sampling

Wie beim Q-Learning, kann auch beim Z-Learning die Approximation verbessert werden, indem man die Policy zur Episodengenerierung an die aktuellen Schätzwerte der State-Value Function anpasst. Anstatt wahllos der unkontrollierten Markov-Kette $\overline{P}$ zu folgen, kann der Agent eine andere Policy verwenden, welche das bisherige Wissen über die State-Value Function ausnutzt (Exploitation) [Tod09]. Diese Greedy Policy verwendet dann die aktuell optimal erscheinenden Aktionen $\hat{\mathbf{u}}^*(i)$. Die Samples werden also nach $P(\hat{\mathbf{u}}^*(i))$ erzeugt. Die einzelnen kontrollierten Übergangswahrscheinlichkeiten sind dann

$$\begin{aligned} p_{ij}(\hat{\mathbf{u}}^*(i)) &= \overline{p}_{ij} \exp\left(-\hat{v}(j) - \log\left(\sum_k \overline{p}_{ik} \exp(-\hat{v}(k))\right)\right) \\ &= \overline{p}_{ij} \frac{\exp(-\hat{v}(j))}{\sum_k \overline{p}_{ik} \exp(-\hat{v}(k))} \end{aligned} \tag{2.39}$$

Diese Form des Greedy Z-Learnings erfordert Importance Sampling. Beim Importance Sampling handelt es sich um eine allgemeine Technik zur Schätzung von Parametern einer Wahrscheinlichkeitsverteilung (siehe z. B. [Bis06]). Will man bspw. den Erwartungswert einer P-verteilten Zufallsvariable durch

Samples schätzen, kann man versuchen die Samples nach einer geschickt gewählten Verteilung Q zu ziehen und so zu gewichten, dass wieder der ursprünglich gesuchte Erwartungswert resultiert. Die Grundidee dabei ist Varianzreduktion. Hat die Verteilung Q eine geringere Varianz als die ursprüngliche Verteilung P, so wird das Schätzverfahren durch Importance Sampling effizienter.

Beim Z-Learning will man den erwarteten Return des nächsten Zustands j in Bezug auf $\overline{P}$ schätzen. Kleine Werte für $\hat{z}(j)$ tragen dabei kaum zu der Summe in (2.32) bei. Daher schlägt Todorov vor, die Samples nach $P(\hat{\mathbf{u}}^*(i))$ zu generieren, da dabei Zustände mit größer geschätzten Z-Values häufiger gezogen werden, was die Approximation hoffentlich beschleunigt. Tatsächlich gilt für den aktuell bestgeschätzten Zustand $m := \arg\max_{i \in \mathcal{S}} \{\hat{z}(i)\}$:

$$p_{im}(\hat{\mathbf{u}}^*(i)) = \overline{p}_{im} \underbrace{\frac{\hat{z}(m)}{\sum_k \overline{p}_{ik} \hat{z}(k)}}_{\geq 1} \geq \overline{p}_{im}. \tag{2.40}$$

Der Zustand mit dem größten Schätzwert wird also mit größerer Wahrscheinlichkeit gezogen als bei unkontrollierter Dynamik $\overline{P}$. Die Samples müssen dann noch mit

$$\frac{\overline{p}_{ij}}{p_{ij}(\hat{\mathbf{u}}^*(i))} \tag{2.41}$$

gewichtet werden, damit gilt

$$\mathbb{E}_{j \sim P(\hat{\mathbf{u}}^*(i))[\cdot|i]} \left(\frac{\overline{p}_{ij}}{p_{ij}(\hat{\mathbf{u}}^*(i))} z(j) \right) = \mathbb{E}_{j \sim \overline{P}(\cdot|i)}(z(j)). \tag{2.42}$$

Somit ergibt sich folgende Update-Regel fürs Greedy Z-Learning:

$$\hat{z}(i_k) \leftarrow \hat{z}(i_k) + \alpha_k \left[\frac{\overline{p}_{i_k i_{k+1}}}{p_{i_k i_{k+1}}(\hat{\mathbf{u}}^*(i_k))} \exp(-q(i_k)) \hat{z}(i_{k+1}) - \hat{z}(i_k) \right]. \tag{2.43}$$

Der kritische Punkt beim Importance Sampling ist die Wahl einer geeigneten Verteilung, nach der Samples generiert werden. Eine gut gewählte

Verteilung zieht die für den zu schätzenden Parameter „wichtigen“ Werte häufiger und verringert damit die Varianz, was den Schätzer effizienter macht. Andernfalls kann eine schlecht gewählte Verteilung die Laufzeit auch verlängern. Beim Greedy Z-Learning lassen sich nur schwer allgemeine Aussagen zur Eignung von $P(\hat{\mathbf{u}}^*(i))$ machen, da die Varianz sowohl von der unkontrollierten $\overline{P}$, als auch von der Qualität der bisherigen Schätzwerte $\hat{z}(i)$ abhängt. Im Prinzip entspricht das Greedy Z-Learning genau der stetigen Umsetzung der ϵ-Greedy Policy beim Q-Learning, weshalb auch hier gute Ergebnisse zu erwarten sein sollten. In [Tod09] wird empirisch gezeigt, dass Greedy Z-Learning in einem Beispiel tatsächlich schneller lernt als einfaches Z-Learning. Allerdings spricht Todorov dort auch einen wichtigen Punkt an: Für die Gewichte (2.41) werden die unkontrollierten Übergangswahrscheinlichkeiten $\overline{p}_{ij}$ benötigt. Sind diese nicht bekannt (wovon ursprünglich ausgegangen wurde), kann man nur versuchen, auch diese zu schätzen. Mit diesem und weiteren Problemen bei der praktischen Umsetzung des Z-Learning Algorithmus' beschäftigt sich das nächste Kapitel.

Im Vergleich mit Q-Learning auf extra dafür konstruierten diskreten MDP zeigt sich, dass Z-Learning die optimalen Values tatsächlich schneller lernt, was darauf zurückgeführt werden kann, dass Z-Learning keinen Minimierungsoperator gebraucht und sich nur im Zustandsraum, statt im Zustands-Aktions-Raum, bewegt.

3 Z versus Q

Im letzten Kapitel wurden die theoretischen Grundlagen zum klassischen Reinforcement Learning vorgestellt. Es wurde gezeigt, wie die Problemstellung als Markov Decision Problem mit diskreten Aktionen formuliert wird und wie mit Kenntnis der Dynamik der Umgebung, die State-Value Function mittels Value Iteration berechnet werden kann, woraus sich eine deterministische optimale Policy bestimmen lässt, die vom Agenten verfolgt wird, um sein Problem zu lösen.

Außerdem wurde mit Q-Learning ein Off-Policy Algorithmus vorgestellt, der es ermöglicht, die Action-Value Function durch Erfahrung zu approximieren und dabei die optimale Policy zu lernen, ohne dass ein Modell der Umgebung erforderlich ist. Dieses Verfahren ist mittlerweile mathematisch fundiert und liefert oft gute Ergebnisse.

Darüber hinaus wurde eine spezielle Klasse von Markov Decision Problems mit überabzählbaren Aktionsmengen vorgestellt, die von Todorov stammt. Das von ihm konstruierte Modell ermöglicht analytisches Lösen der Bellman-Gleichung, welche sich als lineares Eigenwertproblem herausstellt. Zudem schlägt Todorov einen Off-Policy Algorithmus namens Z-Learning zum Lernen der State-Value Function vor, welcher auch bei unvollständiger Kenntnis der Umgebung funktionieren soll und sogar effizienter als Q-Learning sein soll.

Empirische Belege dafür wurden nur auf den speziellen stetigen MDP erbracht, jedoch nicht auf diskreten MDP.

In diesem Kapitel soll nun untersucht werden, wie das Z-Learning auf klassischen, diskreten MDP abschneidet und in welchen Fällen es tatsächlich

besser funktioniert als Q-Learning. Dazu müssen zunächst einige Probleme bei der Umsetzung von Z-Learning auf diskreten MDP behandelt werden. Danach wird das experimentelle Setting vorgestellt. Abschließend werden die Ergebnisse gezeigt und erklärt.

3.1 Z-Learning auf diskreten MDP

Will man das Z-Learning benutzen, um ein diskretes MDP ohne vollständige Kenntnis des Modells zu lösen, stößt man auf folgende Probleme:

(i) Z-Learning geht von einer unkontrollierten Dynamik $\overline{P}$ aus, die in der klassischen Formulierung eines MDP nicht vorkommt.

(ii) Z-Learning verwendet reelle Vektoren, welche die unkontrollierte $\overline{P}$ umskalieren, als Aktionen, und keine diskreten, symbolischen Aktionen.

(iii) Will man Z-Learning mittels Importance Sampling verbessern, so benötigt man Kenntnis der Unkontrollierten $\overline{P}$.

An dieser Stelle sei noch ein wichtiger Punkt angesprochen, der von Todorov gar nicht erwähnt wird. Das Z-Learning (im diskreten wie im stetigen Fall) liefert nur die optimale State-Value Function. Der Agent benötigt jedoch die optimalen Aktionen, um sein Problem zu lösen. Diese lassen sich zwar nach (2.28) aus den $v(i)$ bestimmen, jedoch benötigt man dafür wiederum die $\overline{p}_{ij}$, welche nach Voraussetzung eigentlich nicht vollständig bekannt sein sollten. Wenn man also nicht noch gleichzeitig die unkontrollierte Dynamik $\overline{P}$ mit schätzt, so muss man die stärkere Annahme treffen, dass $\overline{P}$ bekannt ist. In diesem Fall sollte man vom Z-Learning auch erwarten, dass es schneller lernt als das Q-Learning, da man ja mehr Information hineingesteckt hat und sich somit den Aktionsraum beim Lernen spart.

Zur Bewältigung der oben genannten Probleme bieten sich folgende Lösungsansätze an:

(i) Da es im diskreten Fall keine unkontrollierte Dynamik an sich gibt, liegt es mangels besseren Wissens nahe, einfach eine Policy π zur Episodengenerierung zu wählen, welche gleichverteilt aus den möglichen Aktionen wählt:

$$\forall s \in \mathcal{S} \quad \pi(s,a) = \begin{cases} \frac{1}{|\mathcal{A}_s|}, & a \in \mathcal{A}_s \\ 0, & a \notin \mathcal{A}_s \end{cases} \tag{3.1}$$

(ii) Z-Learning liefert nur die optimale State-Value Function. Das Ziel aber ist, daraus eine deterministische optimale Policy zu bestimmen, die für jeden Zustand s die beste symbolische Aktion $a \in \mathcal{A}_s$ auswählt. Um dies zu erreichen, wird eine einfache Heuristik verwendet. Wir nehmen an, dass für jeden Zustand bekannt sei, welche Aktion mit größter Wahrscheinlichkeit einen bestimmten Nachfolgezustand liefert. Als Policy wird dann einfach die Aktion gewählt, die mit größter Wahrscheinlichkeit im besten (in einem Schritt erreichbaren, d.h. $P(s' \mid s) = \sum_{a \in \mathcal{A}_s} \mathcal{P}^a_{ss'} > 0$) Nachfolgezustand landet.

$$\pi(s) = \arg\max_{a \in \mathcal{A}_s}(\mathcal{P}^a_{sm}), \quad m = \arg\min_{\{k \in \mathcal{S} | P(k|s) > 0\}} (V(k)) \tag{3.2}$$

Dies mag zwar nicht immer die beste Wahl sein, aber dieses Vorgehen ermöglicht es einem, ohne vollständige Kenntnis der Übergangswahrscheinlichkeiten der einzelnen Aktionen zu arbeiten. Die Annahme, zu wissen, welche Aktion am wahrscheinlichsten in einen bestimmten Zustand führt, ist immer noch schwächer, als das gesamte stochastische Verhalten als bekannt vorauszusetzen.

(iii) Greedy Z-Learning mittels Importance Sampling stellt im diskreten Fall die größte Herausforderung dar. Zum einen benötigt es die unkontrollierte $\overline{P}$, um die Gewichte (2.41) zu bestimmen. Doch darüber

hinaus muss noch eine Greedy Policy $\pi_{\hat{\mathbf{u}}^*(s)}$ berechnet werden, welche die kontrollierte Dynamik $P(\hat{\mathbf{u}}^*(s))$ simuliert, d. h. es muss gelten:

$$\sum_{a \in \mathcal{A}_s} \pi_{\hat{\mathbf{u}}^*(s)}(s,a) \mathcal{P}^a_{ss'} = p_{ss'}(\hat{\mathbf{u}}^*(s)) \quad \forall s \in \mathcal{S}. \tag{3.3}$$

Daraus folgt also, dass nicht nur $\overline{P}$ bekannt sein muss, sondern auch $\mathcal{P}^a_{ss'}$ (mit dem sich dann automatisch auch $\overline{P}$ bestimmen lässt). Man ist also gezwungen, die Dynamik $\mathcal{P}^a_{ss'}$ des Modells als bekannt vorauszusetzen oder zu schätzen, um das folgende lineare Gleichungssystem aus $N = |\mathcal{S}|$ Gleichungen in $M = |\mathcal{A}_s|$ Unbekannten lösen zu können:

$$\underbrace{\begin{pmatrix} \mathcal{P}^{a_1}_{ss_1} & \cdots & \mathcal{P}^{a_M}_{ss_1} \\ \vdots & \ddots & \vdots \\ \mathcal{P}^{a_1}_{ss_N} & \cdots & \mathcal{P}^{a_M}_{ss_N} \end{pmatrix}}_{=: \boldsymbol{\mathcal{P}}(s) \in \mathbb{R}^{N \times M}} \cdot \underbrace{\begin{pmatrix} \pi_{\hat{\mathbf{u}}^*(s)}(s,a_1) \\ \vdots \\ \pi_{\hat{\mathbf{u}}^*(s)}(s,a_M) \end{pmatrix}}_{=: \boldsymbol{\pi}(\hat{\mathbf{u}}^*(s)) \in \mathbb{R}^M} = \underbrace{\begin{pmatrix} p_{ss_1}(\hat{\mathbf{u}}^*(s)) \\ \vdots \\ p_{ss_N}(\hat{\mathbf{u}}^*(s)) \end{pmatrix}}_{=: \mathbf{p}(\hat{\mathbf{u}}^*(s)) \in \mathbb{R}^N}, \tag{3.4}$$

wobei $\boldsymbol{\pi}(\hat{\mathbf{u}}^*(s))$ folgende zusätzliche Bedingungen erfüllen muss:

$$\pi_{\hat{\mathbf{u}}^*}(s,a) \geq 0 \quad \forall s \in \mathcal{S}, a \in \mathcal{A}_s \tag{3.5}$$

$$\sum_{a \in \mathcal{A}_s} \pi_{\hat{\mathbf{u}}^*(s)}(s,a) = 1 \quad \forall s \in \mathcal{S}. \tag{3.6}$$

Im Allgemeinen muss nicht immer eine zulässige Lösung für $\boldsymbol{\pi}(\hat{\mathbf{u}}^*(s))$ existieren. In diesem Fall jedoch kann man z. B. eine deterministische Policy wählen, welche einfach die Aktion auswählt, deren Übergangsverteilung $\boldsymbol{\mathcal{P}}^a_s$ (a-te Spalte aus $\boldsymbol{\mathcal{P}}(s)$) den geringsten quadratischen Abstand zur gesuchten $\mathbf{p}(\hat{\mathbf{u}}^*(s))$ hat:

$$\pi_{\hat{\mathbf{u}}^*}(s) = \arg\min_{a \in \mathcal{A}_s} (\|\boldsymbol{\mathcal{P}}^a_s - \mathbf{p}(\hat{\mathbf{u}}^*(s))\|^2). \tag{3.7}$$

Nach diesen Vorüberlegungen soll nun die Performance des Z-Learning Algorithmus' anhand von Experimenten untersucht werden und mit der des Q-Learnings verglichen werden.

3.2 Setting

Wie schon erwähnt, werden die Experimente auf sogenannten *Gridworlds* (dt. Gitterwelten) durchgeführt. Dabei handelt es sich um ein schachbrettartiges Raster aus Feldern (siehe Abb. 3.1), wovon einige als Zielfelder (Kästchen) definiert sind. Aufgabe des Agenten ist es, von einem speziellen Startfeld (Kreuz) aus mit minimalem Aufwand eines der Zielfelder zu erreichen. Dabei treten bei jedem Feld bestimmte (Schritt-) Kosten r_t auf, die über die Zeit aufsummiert werden. Diese (unverminderten) Gesamtkosten $R = \sum_{t=1}^{\infty} r_t$ gilt es zu minimieren. Es handelt sich also um eine Problemstellung mit indefinitem Horizont und absorbierenden Zielzuständen. Einige Felder (schwarz) können nicht betreten werden und stellen somit Hindernisse (z. B. Mauern) dar. Der Agent (Kreis) hat die Möglichkeit, sich mit jedem Schritt in eine der 4 Richtungen (vorwärts, rückwärts, links, rechts) zum nächsten Nachbarfeld zu bewegen. Es handelt sich also um eine klassische Wegfindungsaufgabe durch eine labyrinthartige, 2-dimensionale Welt.

Eine solche $n \times m$-Gridworld lässt sich mathematisch als ein diskretes MDP formulieren. Die Menge der Zustände ist dabei die Menge der Felder. Die Felder können beispielsweise durch Koordinatentupel dargestellt werden.

$$\mathcal{S} = \{1, \cdots, n\} \times \{1, \cdots, m\}$$

In jedem Zustand hat der Agent die Möglichkeit, einen Schritt in eine der 4 Richtungen zu tun. Das ergibt die folgenden Aktionen:

$$\forall s \in \mathcal{S} \quad \mathcal{A}_s = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$$

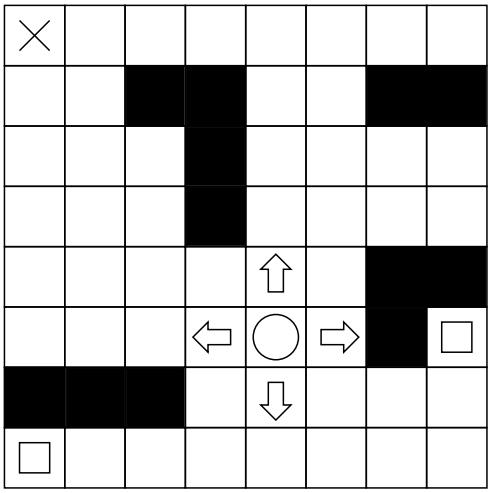

Abbildung 3.1: Schematisches Beispiel einer Gridworld: Startfelder sind mit Kreuzen, Zielfelder mit Kästchen markiert. Schwarze Felder sind Hindernisse, die der Agent (Kreis) umgehen muss. Als Aktion kann der Agent einen Schritt in eine der 4 Richtungen tun.

Das Übergangsmodell $\mathcal{P}^a_{ss'}$ wird schematisch in Abb. 3.2 dargestellt. Der Agent befindet sich dabei auf dem Feld in der Mitte (Pfeil). Die Aktion $\uparrow$ wurde gewählt. Die Zahlen in den Feldern geben die Wahrscheinlichkeiten an, mit denen der Agent unter Aktion $\uparrow$ auf ihnen landet. Die Aktionen sind nicht deterministisch. In 80% der Fälle landet der Agent auf dem gewünschten Feld. Doch manchmal kommt es vor, dass der Agent einen Fehltritt macht (z. B. durch Störung der Navigation, Steuerung oder Sensorik eines Roboters) und auf einem anderen Feld landet. Für die anderen 3 Aktionen sei hier das selbe Schema angenommen, jeweils um 90° gedreht. Falls der Agent mit einer Aktion aus der Gridworld rauslaufen oder in ein Hindernis laufen würde, bleibt er stattdessen auf dem aktuellen Feld stehen.

Zielzustände sind als absorbierend definiert, was praktisch bedeutet, dass der Agent nach Erreichen des Ziels auf den Startzustand zurück gesetzt wird und eine neue Episode generiert wird.

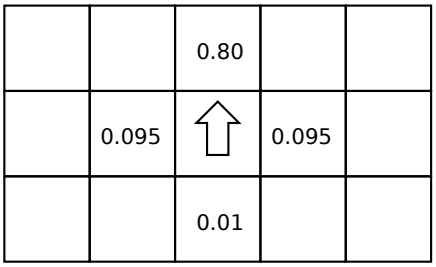

Abbildung 3.2: Aktionsschema $\mathcal{P}^a_{ss'}$: In 80% der Fälle landet der Agent auf dem gewünschten Feld. Fehltritte auf andere Felder passieren mit geringer Wahrscheinlichkeit.

Als erwartete Rewards (Kosten) $\mathcal{R}^a_{ss'}$ kommen im Prinzip beliebige positive Werte in Frage. Im Folgenden wird hier von festen Einheitskosten für jeden Nicht-Zielzustand ausgegangen, unabhängig von der gewählten Aktion und dem Vorgängerzustand (was bedeutet, den Agenten den kürzesten Weg ins Ziel lernen zu lassen). Zielzustände $\mathcal{C} \subseteq \mathcal{S}$ erzeugen keine Kosten.

$$\mathcal{R}^a_{ss'} = \begin{cases} 1, & s' \notin \mathcal{C} \\ 0, & s' \in \mathcal{C} \end{cases}$$

Damit sind alle notwendigen Informationen zur Aufgabenstellung gegeben.

Die gerade beschriebenen Gridworlds sind ein beliebtes Beispiel für klassische diskrete MDP, welche gerne als Grundlage für empirische Auswertung von RL Algorithmen verwendet werden (u. a. auch von Todorov). Sie sind anschaulich und einfach zu verstehen. Sie lassen sich schnell implementieren und sind dabei einfach zu skalieren. Zudem bieten sie Möglichkeiten für etliche Variationen und Erweiterungen, um die Problemstellung noch interessanter zu machen.

3.3 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Testläufe der verschiedenen Algorithmen auf unterschiedlichen Gridworlds vorgestellt und erläutert.

Für die Experimente wurden die Zustandskosten $\mathcal{R}^a_{ss'}$ aus numerischen Gründen auf $0,1$ runterskaliert. Die verwendete Lernrate hat die Form

$$\alpha_t = \frac{c}{c+t}. \tag{3.8}$$

Der Parameter $c \in \mathbb{R}$ wird vor jeder Durchführung in mehreren Probeläufen optimiert.

3.3.1 Z-Learning

Abb. 3.3 zeigt eine Gridworld, auf der das Z-Learning getestet wurde. Die

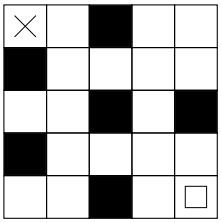

Abbildung 3.3: Gridworld small1

Aufgabe wurde zunächst als ein stetiges MDP im Sinne des Z-Learnings betrachtet, um Funktionstüchtigkeit und Verhalten des Algorithmus' zu demonstrieren. Die Lernkurve in Abb. 3.4 stellt den normierten, mittleren Approximationsfehler über der Anzahl Iterationen dar:

$$e = \frac{\max\limits_{s\in\mathcal{S}} \frac{1}{n} \sum\limits_{k=1}^{n} |v(s) - \hat{v}(s)_k|}{\max\limits_{s\in\mathcal{S}} v(s)}. \tag{3.9}$$

Für das Greedy Z-Learning wurden die benötigten Wahrscheinlichkeiten des Modells für das Importance Sampling als bekannt vorausgesetzt. Beide Algorithmen konvergieren schnell gegen die optimale State-Value Function. Das Importance Sampling beschleunigt die Konvergenz tatsächlich – wenn auch nur geringfügig – in einer solchen kleinen Gridworld. Auf einer größeren Gridworld (siehe Abb. 3.5) hingegen lernt das Greedy Z-Learning die

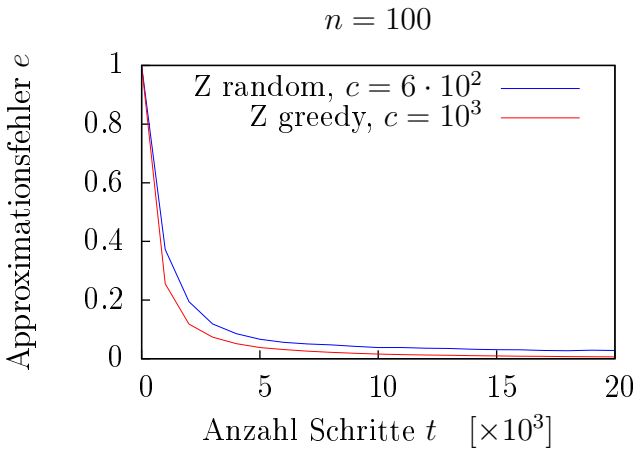

Abbildung 3.4: Verlauf des e-Fehlers über der Iterationszahl t auf small1: Greedy Z-Learning lernt geringfügig schneller als Random Z-Learning.

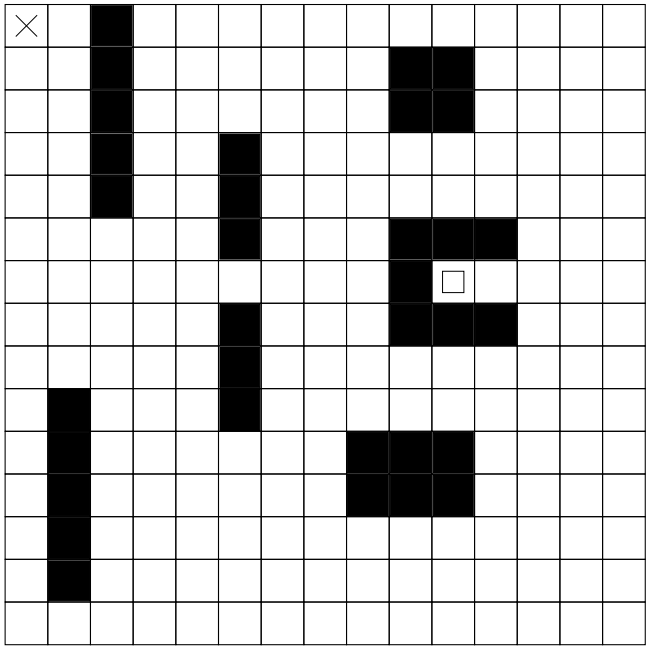

Abbildung 3.5: Gridworld big1

optimale State-Value Function anfangs deutlich schneller, wie die Lernkurve in Abb. 3.6 zeigt. Allerdings verlangsamt sich die Approximation mit zunehmender Iterationszahl, sodass das einfache Z-Learning sogar noch überholt. Dieser Effekt ist dadurch zu erklären, dass Greedy Z-Learning schlechte Zustände mit der Zeit immer seltener erkundet, der Approximationsfehler jedoch als Maximum der Abweichungen aller Zustände definiert ist. Die günstigen Zustände lernt das Greedy Z-Learning aber schneller.

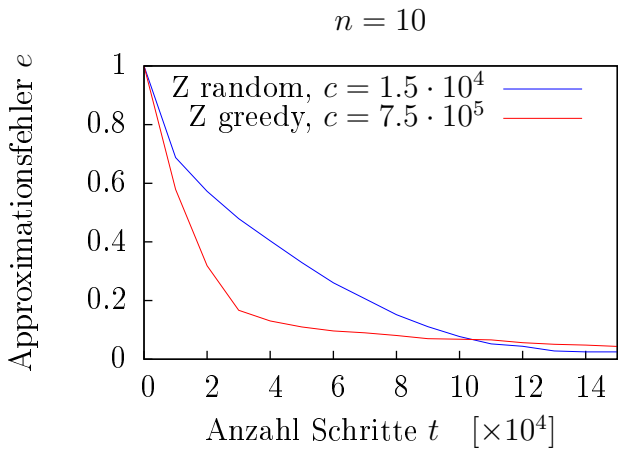

Abbildung 3.6: Verlauf des e-Fehlers über der Iterationszahl t auf big1: Greedy Z-Learning lernt anfangs deutlich schneller als Random Z-Learning.

Z-Learning liefert bekanntermaßen grundsätzlich eine probabilistische optimale Policy. Abb. 3.7 visualisiert die optimal kontrollierte Dynamik $P(\mathbf{u}^*)$ an ausgewählten Stellen (Kreise). Die Zahlen in den Feldern geben die gerundeten Wahrscheinlichkeiten $p_{ij}(\mathbf{u}^*(i))$ in Prozent an. Dabei ist schön zu sehen, wie die optimale Aktion $\mathbf{u}^*(i)$ in bestimmten Zuständen (z. B. direkt vor der Zielecke) die unkontrollierte Dynamik $\overline{P}$ stark in Richtung Zielzustand umskaliert (und dafür hohe Aktionskosten in Kauf nimmt), wohingegen auf anderen Feldern (z. B. auf dem Startfeld) kaum Einfluss auf $\overline{P}$ genommen wird (da sich die Aktionskosten nicht lohnen).

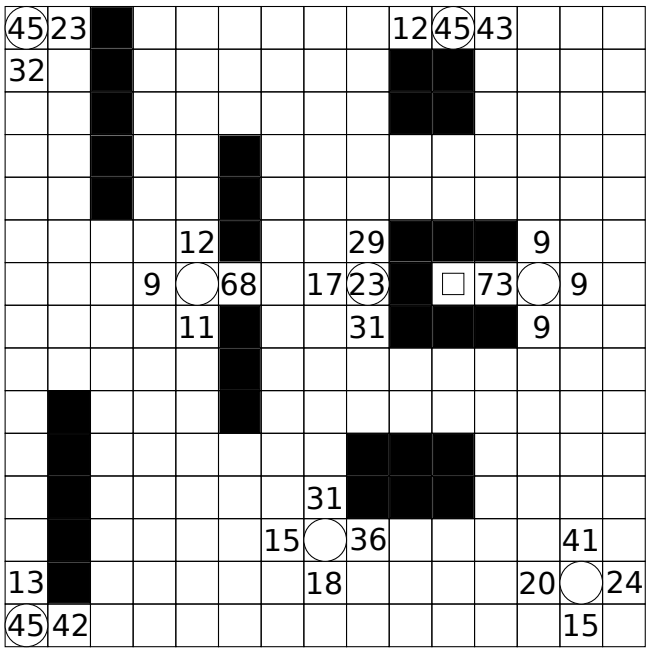

Abbildung 3.7: Optimal kontrollierte Dynamik $P(\mathbf{u}^*)$ an ausgewählten Stellen (Kreise) auf big1. Die Zahlen geben die Übergangswahrscheinlichkeiten $p_{ij}(\mathbf{u}^*(i))$ in gerundeten Prozent an. Auf manchen Feldern (z. B. direkt vor dem Zielfeld) ist gut erkennbar, wie der Agent die Unkontrollierte $\overline{P}$ (Random Walk) stark in Richtung Ziel umskaliert. Auf anderen Feldern hingegen (z. B. dem Startfeld) lohnen sich die Aktionskosten für starkes Kontrollieren nicht.

Die Aktionsbereitschaft des Agenten hängt u. a. von den Zustandskosten $q(i)$ ab. Der Zusammenhang wird in Abb. 3.8 für ein Feld (linke, untere Ecke $(15, 1)$) dargestellt. Es wurden die optimal kontrollierten Wahrscheinlichkeiten $p_{ij}(\mathbf{u}^*(i))$ für einen Übergang von Feld $(15, 1)$ in eines der Nachfolgefelder $(14, 1)$ (oben), $(15, 2)$ (rechts) und $(15, 1)$ für gegebene Kosten von Feld $(15, 1)$ aufgetragen. Mit zunehmenden Zustandskosten wird es für den Agenten immer teurer, in der Ecke zu bleiben. Daher lohnt es sich immer mehr, höhere Aktionskosten in Kauf zu nehmen, um mit immer größerer Wahrscheinlichkeit auf das rechte (beste) Nachbarfeld $(15, 2)$ zu gelangen.

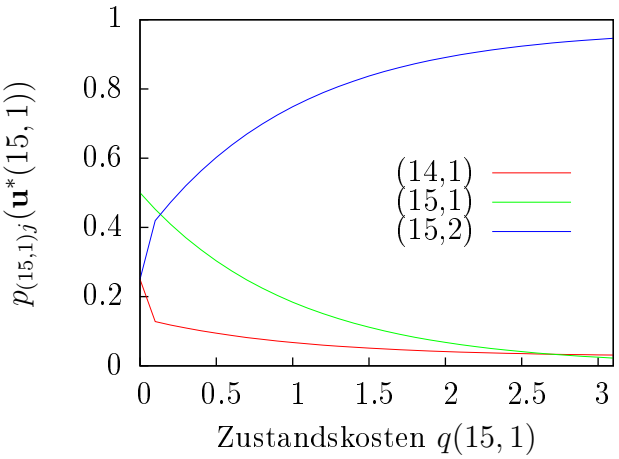

Abbildung 3.8: Mit zunehmenden Zustandskosten q für das Feld $(15, 1)$ nimmt der Agent immer höhere Aktionskosten in Kauf, um sich mit zunehmender Sicherheit auf das Nachbarfeld $(15, 2)$ zu bewegen.

3.3.2 Vergleich mit Q-Learning bei bekannter Dynamik

Die folgenden Experimente dienen dem direkten Vergleich von Z-Learning mit Q-Learning. Dabei werden die Gridworlds als diskrete MDP aufgefasst. Abb. 3.9 zeigt die kleine Gridworld des vorherigen Abschnitts samt optimaler Policy. Da von einem klassischen MDP ausgegangen wird, handelt

es sich um eine deterministische Policy, die jedem Zustand eine optimale Aktion zuordnet. Ausnahmen bilden Felder, bei denen mehrere Aktionen optimal sind. Bei solchen muss sich der Agent einfach für eine der optimalen Aktionen entscheiden.

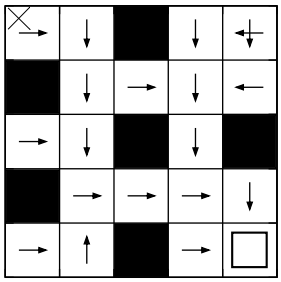

Abbildung 3.9: Optimale Policy small1

Zunächst wird angenommen, dass das Übergangsmodell $\mathcal{P}^a_{ss'}$ bekannt ist. Eine deterministische Policy für das Z-Learning wird dann heuristisch, wie in Abschnitt 3.1(ii) beschrieben, generiert. Das Importance Sampling wird durch Lösen des Gleichungssystems aus Abschnitt 3.1(iii) umgesetzt.

Im diskreten Fall macht der Approximationsfehler e aus (3.9) keinen Sinn, da das Z-Learning gegen andere optimale State-Values $V^*(s)$ konvergiert als das Q-Learning. Grund dafür sind die implizit mitberechneten Aktionskosten $r(i, \mathbf{u})$ beim Z-Learning. Wenn das Z-Learning jedoch im diskreten Fall einsetzbar sein soll, dann sollte es trotz anderer optimaler State-Value Function zur optimalen, deterministischen Policy π^* konvergieren. Da jede Policy eine bestimmte Bewertungsfunktion induziert, liegt es nahe, den folgenden Approximationsfehler zu betrachten:

$$e_\pi = \frac{\max\limits_{s\in\mathcal{S}} \frac{1}{n} \sum\limits_{k=1}^{n} |V^*(s) - V^\pi(s)_k|}{\max\limits_{s\in\mathcal{S}} V^*(s)}. \tag{3.10}$$

Es wird also die Abweichung der State-Value Function der aktuellen Policy π von der optimalen State-Value Function gemessen. $V^*(s)$, sowie π^*

werden mittels Value Iteration bestimmt. Die $V^\pi(s)$ werden iterativ durch Einsetzen in die Bellman-Gleichung (2.6) errechnet (dieses Verfahren ist auch als *Policy Evaluation* bekannt, siehe [SB98]). Leider eignet sich dieser Fehler nicht gut für eine graphische Darstellung, da zu Beginn des Lernens, auf Grund der geringen Anzahl erkundeter Zustände, die Policies stark vom Zufall abhängen, wodurch der Fehler stark schwankt und beliebig groß werden kann. Die folgende Abbildung 3.10 zeigt einen groben Beispielverlauf (logarithmisch). Interessant sind eigentlich nur die Zeitpunkte, zu denen der Approximationsfehler annähernd 0 wird, also wann die aktuelle Policy beinahe optimal ist. In dem dargestellten Beispiel hatten die beiden Z-Algorithmen schneller eine nahezu optimale Policy gelernt.

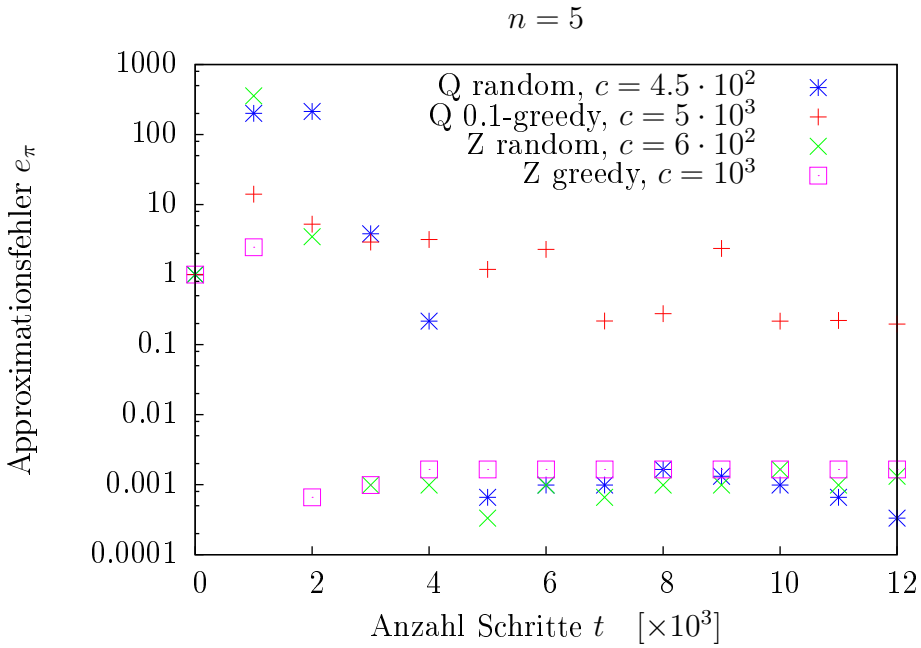

Abbildung 3.10: Logarithmische Darstellung des e_π-Fehlers über der Iterationszahl t auf small1: Der Fehler der Z-Algorithmen geht schneller gegen 0.

Um die verschiedenen Algorithmen zu vergleichen, kann man auch direkt die gelernten Policies betrachten. Dazu definiert man sich die Differenz

zweier Policies als Anzahl an „Fehlstellungen“, also die Anzahl der Zustände, bei denen keine der optimalen Aktionen gewählt wird:

$$d = |\{s \in \mathcal{S} \mid \ \pi(s) \in \pi^*(s)\}|. \tag{3.11}$$

Natürlich ist dies eine sehr einfache Definition, da sie nicht die Richtung der Fehlstellung berücksichtigt, aber man erkennt an ihr dennoch das Wesentliche, nämlich, wann die optimale Policy gefunden wurde. Abb. 3.11 zeigt den Verlauf der Policy-Differenz (gemittelt über n Läufe) auf der Gridworld small1. Die Z-Algorithmen lernen anfangs etwas schneller, insgesamt handelt es sich jedoch nur um geringfügige Unterschiede.

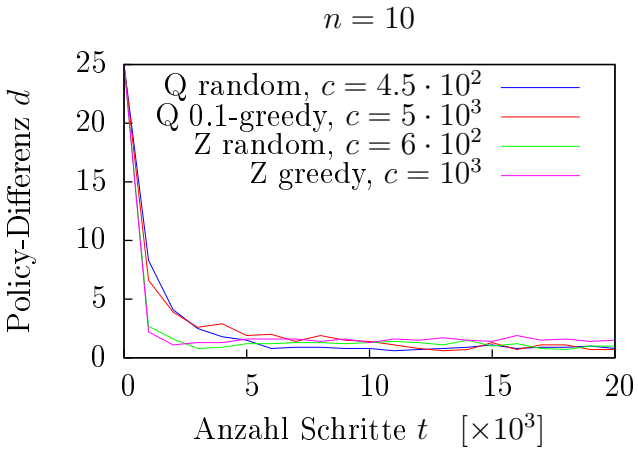

Abbildung 3.11: Verlauf der Policy-Differenz (Anzahl nicht-optimaler Aktionen) über der Schrittzahl t auf small1: Die Z-Algorithmen finden etwas schneller eine nahezu optimale Policy.

Abb. 3.12 zeigt den Verlauf auf der großen Gridworld big1 aus dem vorherigen Abschnitt. Hier wird der Geschwindigkeitsunterschied deutlicher. Die Z-Algorithmen konvergieren schneller als die Q-Algorithmen und die Greedy-Variante jeweils schneller als die Random-Variante. Auffällig ist hierbei jedoch das Greedy Q-Learning, das anfangs sogar schneller als das Random Z-Learning lernt, dann jedoch sehr schnell stagniert und sich nur

noch sehr langsam der optimalen Policy nähert. Das Verhalten kann wieder auf die geringere Erkundung zurückgeführt werden, wodurch in schlechten Zuständen nicht optimal agiert wird. Da der Agent jedoch sowieso lernt solche Zustände zu vermeiden, sollte der erwartete Nutzen dadurch kaum beeinflusst werden. Dieser Effekt tritt auf, wenn es viele eher schlechte Zu-

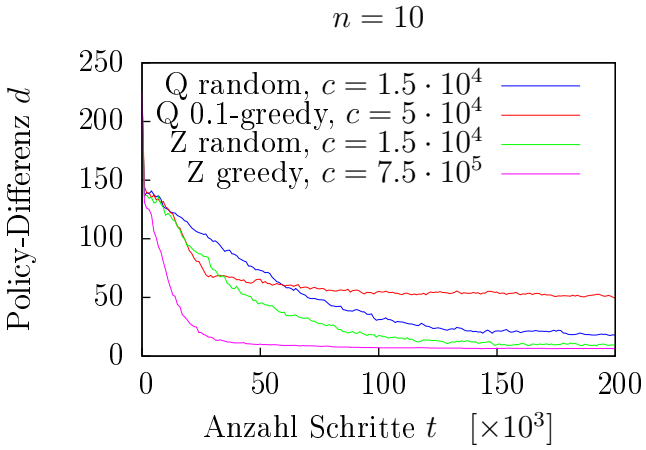

Abbildung 3.12: Policy-Verlauf auf small1: Die Z-Algorithmen lernen deutlich schneller eine nahezu optimale Policy. Greedy Q-Learning lernt anfangs zwar noch verhältnismäßig schnell, stagniert dann jedoch mangels Erkundung.

stände gibt (wie in big1) und verhältnismäßig wenige sehr gute Zustände. Es gibt also fast nur den eindeutigen besten Weg und jede Erkundung anderer Zustände erzeugt hohe Kosten. Ein weiteres Beispiel für eine solche Gridworld (samt optimaler Policy) wird in Abb. 3.13 gezeigt.

Der Verlauf der Policies in Abb. 3.14 lässt erkennen, wie die Greedy Algorithmen zuerst sehr schnell die optimalen Aktionen in den guten Zuständen finden und dann nur langsam die schlechten Zustände erkunden. Bemerkenswert ist hierbei, dass das Greedy Z-Learning weniger anfällig dafür ist, was auf die implizite Erkundung zurückgeführt werden kann, da eine zu gierig kontrollierte Dynamik $P(\hat{\mathbf{u}}^*)$ zu hohe Aktionskosten verursacht.

Eine weitere Möglichkeit, die Algorithmen zu vergleichen, besteht darin, den erwarteten Nutzen der gelernten Policies empirisch zu schätzen,

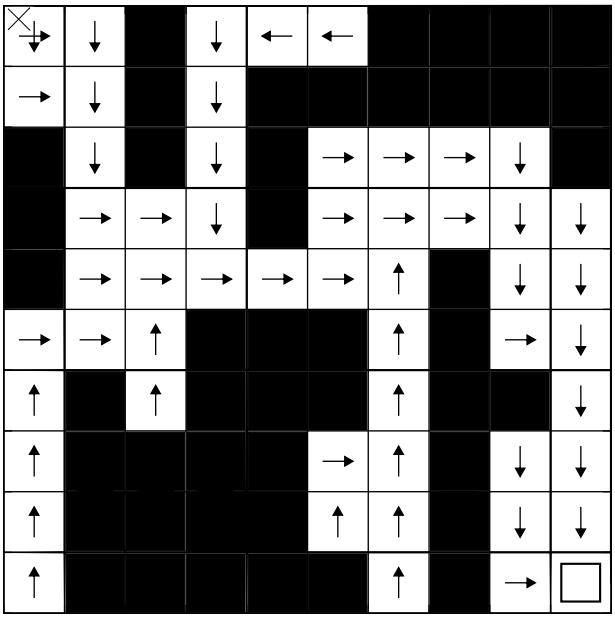

Abbildung 3.13: Gridworld medium1

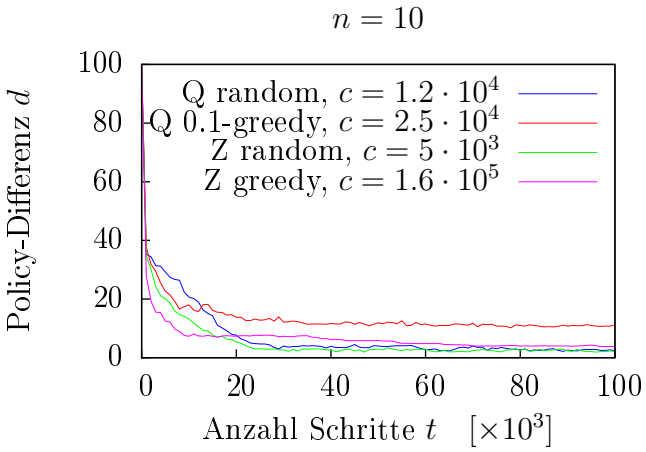

Abbildung 3.14: Policy-Verlauf auf medium1: Auch die Lernrate von Greedy Z-Learning verlangsamt sich mit der Zeit, jedoch ist Greedy Z-Learning auf Grund der implizizen Erkundung weniger anfällig dafür.

statt den exakten Erwartungswert über die Bellman-Gleichung auszurechnen. Dazu lässt man den Agenten wiederholt vom Startzustand aus, der aktuellen Policy folgend, bis ins Ziel laufen und summiert die entstandenen Kosten auf. Das arithmetische Mittel aller entstandenen Gesamtkosten stellt dann eine empirische Schätzung von $V^{\pi}(s)$ für den Startzustand s dar. Allerdings hat man auch beim Schätzen der State-Value Function das Problem, dass die Policies anfangs beliebig schlecht sein können und somit beliebig hohe Kosten (lange Laufzeiten) erzeugen. Abb. 3.15 (oben) zeigt einen groben Verlauf (logarithmisch) der geschätzten Kosten des Startfelds $(1, 1)$ gemittelt über n Durchläufe (die Kosten wurden jeweils über 100 Episoden gemittelt) auf der Gridworld medium1. Die untere Grafik enthält zusätzlich die empirische Standardabweichung der Schätzungen. Um überhaupt messbare Kosten zu erhalten, wurde erst nach 20000 Transitionen angefangen, die Kosten abzutragen. Vorher waren die Policies der Q-Algorithmen zu schlecht. Die geschätzten Kosten der Q-Algorithmen weichen anfangs noch deutlich vom optimalen Wert ($\approx 2,55$) ab, wobei das Greedy Q-Learning tatsächlich besser liegt als das Random Q-Learning. Die Z-Algorithmen jedoch liefern zum Zeitpunkt $t = 20000$ schon geschätzte Kosten, die annähernd den optimalen zu erwartenden Kosten entsprechen. Zudem liegt die Standardabweichung der Z-Schätzwerte nahe bei 0, wohingegen die Q-Algorithmen anfangs noch stärker vom Optimum abweichen.

Abb. 3.16 zeigt eine weitere kleine Gridworld small2 (mit optimaler Policy). In dieser Aufgabe erzeugt das mit dem Punkt gekennzeichnete Feld das Zehnfache der Einheitskosten (also 1). Dennoch führt der schnellste Weg ins Ziel über dieses Feld, da sonst ein enormer Umweg genommen wird. Auch mit dieser Aufgabenstellung kommen alle untersuchten Algorithmen zurecht. Es ergibt sich das übliche Bild (siehe Abb. 3.17) des Policy-Verlaufs. Auch hier konvergieren die Z-Algorithmen schneller.

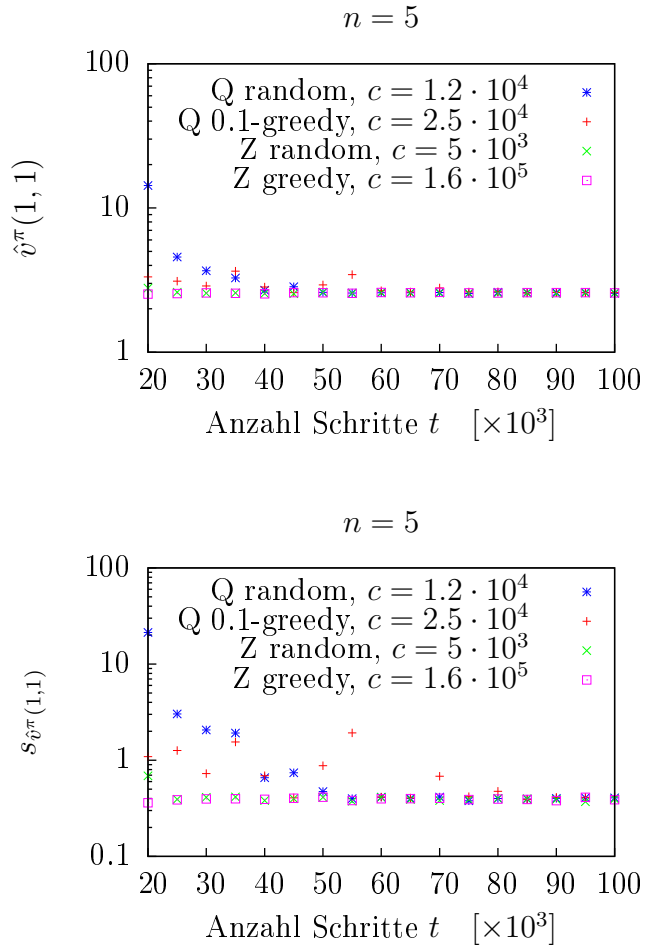

Abbildung 3.15: Oben: empirisch geschätzte Kosten (arithmetisches Mittel über 100 Stichproben) von Feld $(1, 1)$ bei aktueller Policy. Die Z-Algorithmen liefern schneller Schätzwerte nahe am Optimum $V^*(1, 1) \approx 2, 55$.
Unten: empirische Standardabweichung der Schätzungen. Die Z-Algorithmen liefern Schätzungen, die kaum noch vom Optimum abweichen.

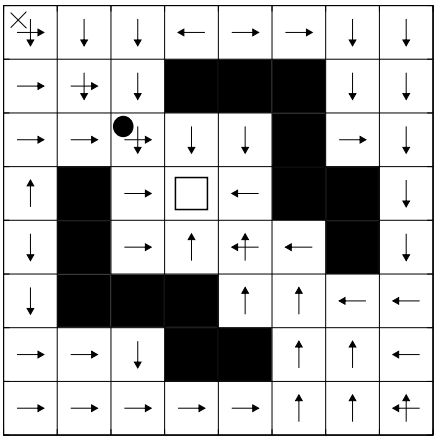

Abbildung 3.16: Gridworld small2: Feld $(3, 3)$ (Punkt) verursacht zehnfache Kosten. Dennoch führt der günstigste Weg ins Ziel darüber, da sonst ein enormer Umweg genommen werden muss.

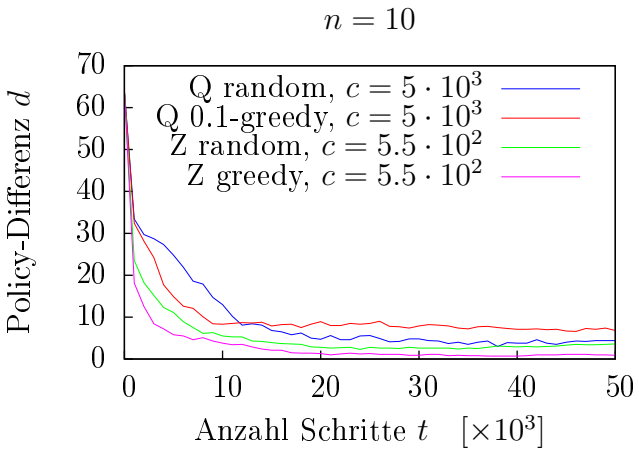

Abbildung 3.17: Policy-Verlauf small2: Alle Algorithmen finden eine annähernd optimale Policy. Z-Algorithmen sind dabei wieder schneller.

3.3.3 Vergleich bei unbekannter Dynamik

Zum Abschluss soll das Z-Learning im klassischen Reinforcement Learning Fall bei völlig unbekanntem Modell auf die Probe gestellt werden. Dafür müssen die Übergangswahrscheinlichkeiten $\mathcal{P}^a_{ss'}$ (eine $|\mathcal{S}| \times |\mathcal{S}|$-Matrix pro Aktion a) während der Interaktion geschätzt werden. Als Schätzwerte werden die relativen Häufigkeiten der jeweiligen Transitionen bei durchgeführter Aktion verwendet:

$$\hat{\mathcal{P}^a_{ss'}} = \frac{h^a_{ss'}}{N^a_s}. \tag{3.12}$$

$h^a_{ss'}$ ist die absolute Häufigkeit der Transition von Zustand s nach s' unter Aktion a, N^a_s ist die Gesamtzahl Transitionen von s unter a. Beide Varianten des Z-Learnings sind auf das Schätzen der Dynamik angewiesen, da aus der State-Value Function eine deterministische Policy generiert werden muss (siehe Abschnitt 3.1(ii)). Zudem benötigt das Greedy Z-Learning die gesamte Dynamik für das Importance Sampling (siehe Abschnitt 3.1(iii)). Mit dem adaptiven Ansatz stößt man beim Greedy Z-Learning aber auf ein grundlegendes Problem. Da man anfangs nichts über die Dynamik weiß, kann der Agent nicht sofort beginnen, nach der kontrollierten Dynamik $P(\hat{\mathbf{u}}^*)$ zu samplen, da er diese noch gar nicht sinnvoll bestimmen kann (geschweige denn, das Gleichungssystem (3.4) lösen).

Ein Lösungsansatz besteht darin, das kontrollierte Samplen um ein bestimmte Zeitspanne T zu verzögern. In dieser Zeit folgt der Agent einfach der Random Policy, um das Modell „kennenzulernen“ und das probabilistische Verhalten der Aktionen zu schätzen. Danach kann er dann beginnen, die aktuell optimal kontrollierte Dynamik zu bestimmen und nach ihr zu samplen. Dieser Ansatz hat allerdings einen Nachteil, denn einen guten Wert für T zu finden, stellt sich dabei als gar nicht so einfach heraus. Zu kleine Werte führen dazu, dass der Agent zu früh anfängt, die Dynamik nach seinem „Weltbild“ zu kontrollieren, obwohl seine Schätzungen noch nicht ausreichend genau sind. Bei zu großen Werten für T hingegen verliert der Agent den Vorteil der beschleunigten Konvergenz gegen die optimale

State-Value Function, da er sein bisheriges Wissen zu spät ausnutzt und zu lange erkundet. Abb. 3.18 veranschaulicht das Problem. Dargestellt ist der Policy Verlauf von Greedy Z-Learning (schätzend) für 3 verschiedene Werte von T auf big1. Zum Vergleich ist zusätzlich die Kurve des Random Z-Learnings (schätzend) eingetragen. Man erkennt deutlich, dass der kleinste T-Wert zu gering gewählt ist, sodass der Agent zu früh anfängt, die Dynamik zu kontrollieren und dabei nicht die optimale Policy findet. Der größte T-Wert führt dazu, dass der Agent zu spät mit der Exploitation beginnt. Der mittlere Wert hingegen scheint für diese Problemstellung gut geeignet zu sein. Man ist also gezwungen, einen weiteren Parameter vor den eigentlichen Experimenten empirisch in Probedurchläufen zu optimieren.

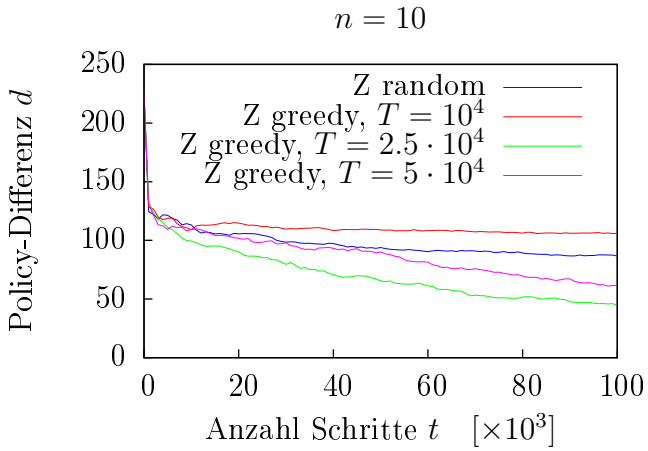

Abbildung 3.18: Policy-Verlauf schätzender Greedy Z-Algorithmen für verschiedene Werte von T auf big1: Ein zu kleiner Wert für T lässt den Agenten zu früh gierig werden. Bei zu großem T beginnt er hingegen zu spät damit, die Dynamik zu kontrollieren.

Zum Leistungsvergleich wurden die schätzenden Z-Algorithmen (mit $\hat{}$ gekennzeichnet), sowie die nicht schätzenden Z- und Q-Algorithmen auf der Girdworld medium2 (siehe Abb. 3.19) laufen gelassen. Abb 3.20 zeigt die einzelnen Policy-Differenzen über der Zeit t. Tatsächlich schneiden die beiden $\hat{Z}$-Algorithmen genauso gut ab wie Random Z mit bekannter Dyna-

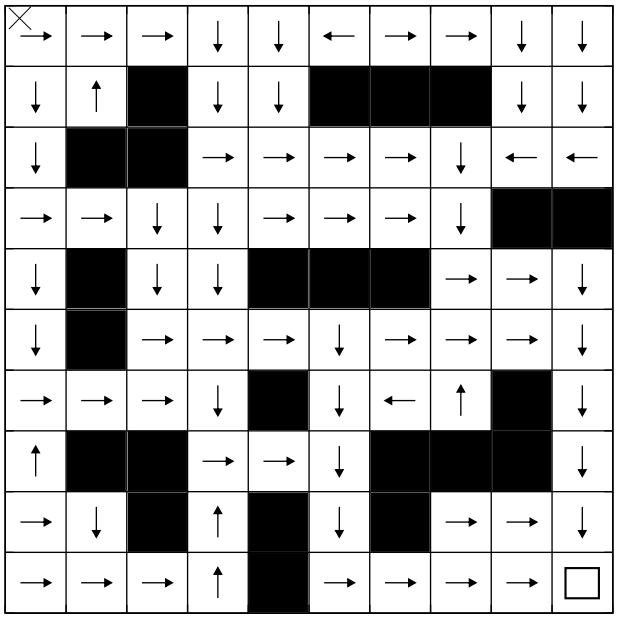

Abbildung 3.19: Gridworld medium2

mik und sind damit besser als die Q-Algorithmen. Wie zu erwarten, lernt Greedy Z am schnellsten. Dass Random $\hat{Z}$ vergleichbar gut abschneidet wie Random Z, lässt sich darauf zurückführen, dass die Episodengenerierung bei beiden gleich abläuft und die Schätzwerte nur zur heuristischen Bestimmung der resultierenden Policy verwendet werden. Hierfür muss aber nur bekannt sein, welche Aktion mit *größter* Wahrscheinlichkeit in einen bestimmten Zustand führt. Der exakte Wert wird nicht benötigt, weshalb die Schätzwerte ausreichen.

Dass Greedy $\hat{Z}$ nicht schneller lernt als Random $\hat{Z}$ liegt an der verhältnismäßig kleinen Gridworld. Die Zeit T, die nötig ist, um akkurate Schätzungen des Modells zu erhalten, ist so groß, dass Random $\hat{Z}$ bis dahin auch schon beinahe an der optimalen Policy angekommen ist. Dem Greedy $\hat{Z}$ bleibt also keine Zeit mehr, seinen Vorteil „auszuspielen“.

Anders sieht es auf der Gridworld big1 aus (siehe Abb. 3.21). Hier braucht einfaches Random $\hat{Z}$ deutlich länger zum Konvergieren, sodass

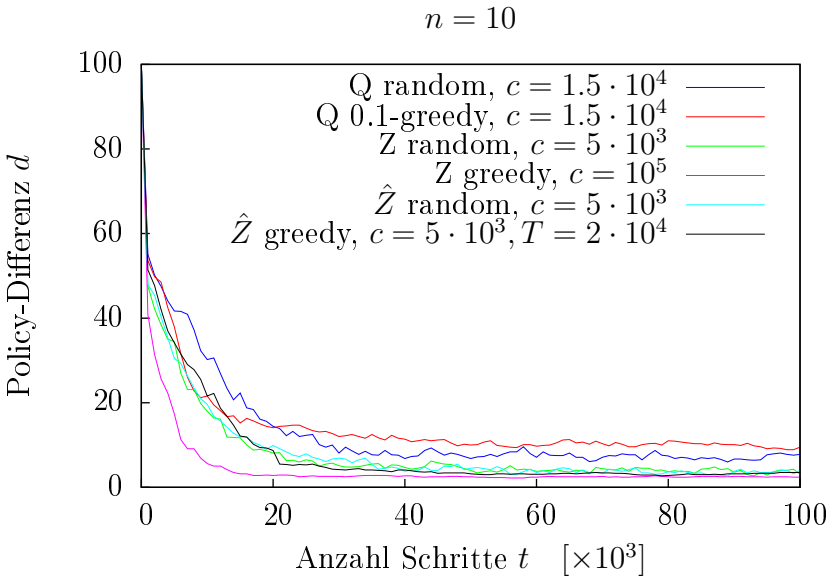

Abbildung 3.20: Policy-Verlauf auf medium2: Schätzende Z-Algorithmen wurden mit ˆ markiert. Die $\hat{Z}$-Algorithmen schneiden vergleichbar gut wie Random Z-Learning ab. Greedy $\hat{Z}$-Learning verliert auf einer eher kleinen Gridworld seinen Konvergenzvorsprung gegenüber Random $\hat{Z}$-Learning, da die nötige Schätzphase T zu lange ist.

Greedy $\hat{Z}$ genug Zeit hat, erst die Dynamik ausreichend zu schätzen und sie dann zu kontrollieren.

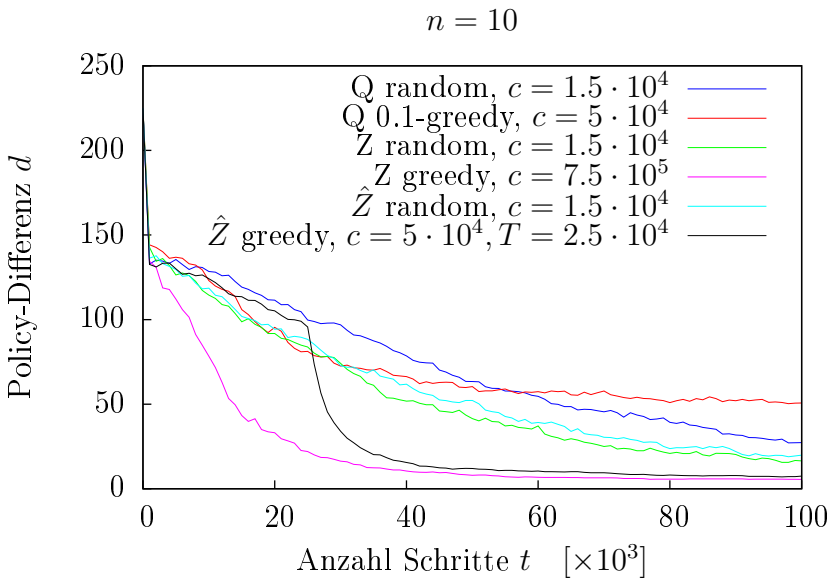

Abbildung 3.21: Policy-Verlauf auf big1: Greedy $\hat{Z}$-Learning hat genug Zeit, seine Konvergenz nach der Schätzphase T deutlich zu beschleunigen.

Abb. 3.22 zeigt eine Variation (small3) der herkömmlichen Gridworlds. Die Gridworld wurde um einen Bestandteil erweitert, der Wind simuliert. In den beiden Zeilen, die auf der linken Seite mit Pfeilen markiert sind, weht ein starker Wind, der bewirkt, dass der Agent bei Betreten eines der Felder der beiden Zeilen um ein (kleiner Pfeil) bzw. zwei (großer Pfeil) Feld(er) nach rechts geschoben wird. Am rechten Rand wird er einfach gegen die Wand „gedrückt“ und bleibt stehen.

Die Lernkurve (Abb. 3.23) der verschiedenen Algorithmen auf dieser Gridworld enthält einige Auffälligkeiten. Zunächst fällt auf, dass bis auf Greedy Z-learning, keiner der Algorithmen in der dargestellten Zeitspanne unter eine Differenz von 20 gelangt. Dies ist durch mangelnde Erkundung zu erklären. Auf Grund des Windes ist es vor allem für die Random Algo-

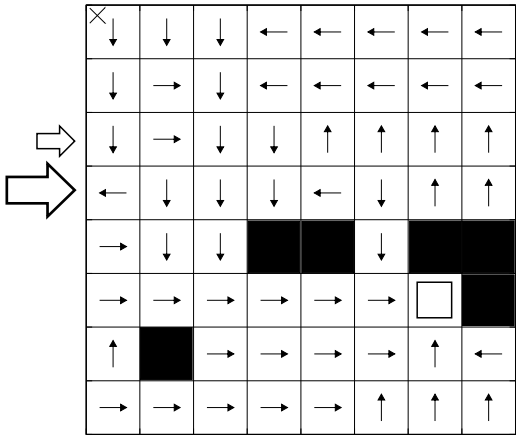

Abbildung 3.22: Gridworld small3: In der 3. und 4. Zeile weht ein starker Wind, der den Agenten jeweils um ein (kleiner Pfeil) bzw. zwei (großer Pfeil) Feld(er) nach rechts schiebt.

rithmen sehr unwahrscheinlich, jemals Zustände in der unteren Hälfte zu passieren, weshalb dort nur sehr langsam eine gute Policy gelernt wird. Die Greedy Algorithmen (vor allem Z) schaffen dies noch eher, da sie kontrollierter in Richtung Ziel steuern. Random $\hat{Z}$-Learning schneidet in diesem Beispiel schlechter ab als Z-Learning, da hinzu kommt, dass ausreichende Schätzwerte für die Heuristik zur Bestimmung einer deterministischen Policy fehlen. Die schlimmsten Auswirkungen finden sich jedoch beim Greedy $\hat{Z}$-Learning. Sobald es anfängt, die Dynamik zu kontrollieren, springt der Fehler dramatisch nach oben. Das Problem der mangelnden Erkundung führt dazu, dass auf selten besuchten Zuständen noch gar nicht alle Aktionen ausprobiert wurden, wodurch es zu einem Laufzeitfehler (Division durch 0) beim Berechnen der relativen Häufigkeiten kommt. Die Zeitspanne T zu verlängern, hilft auch nicht wirklich, da es je nach Gridworld sehr lange dauern kann, bis genug Erfahrung gesammelt wurde, sodass der Vorteil gegenüber Random $\hat{Z}$-Learning verschwindet.

Es lässt sich jedoch ein Kompromiss zur Behebung des Problems finden. Nach Ablauf der Zeitspanne T beginnt der Agent nach $P(\hat{\mathbf{u}}^*)$ zu samplen, falls im aktuellen Zustand jede Aktion schon mindestens ein Mal ausgeführt

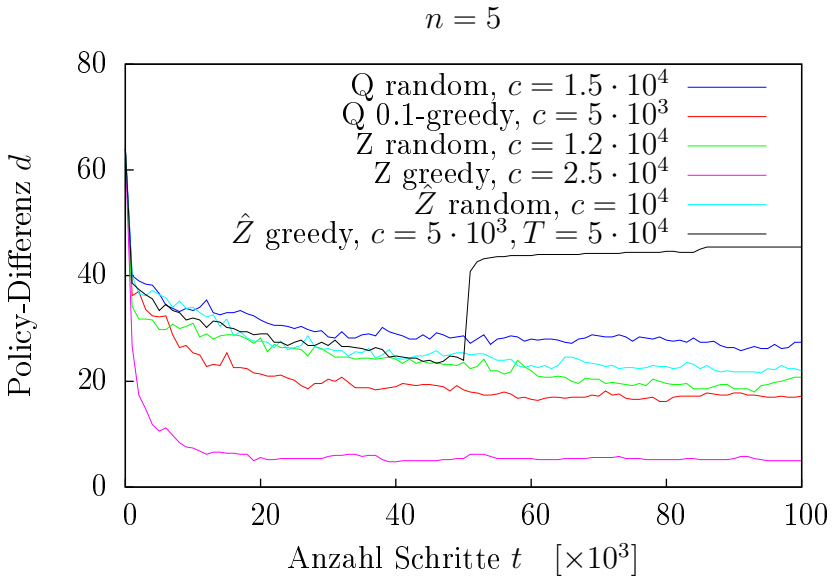

Abbildung 3.23: Policy-Verlauf small3: Auf Grund mangelnder Erfahrung macht der Agent beim Greedy $\hat{Z}$-Learning grobe Fehler, sobald er mit der Kontrolle der Dynamik beginnt.

wurde. Ansonsten folgt er einfach weiter der Unkontrollierten $\overline{P}$. Somit sind die relativen Häufigkeiten immer wohldefiniert. Allerdings muss man noch auf die Gewichte (2.41) beim Importance Sampling achten, da diese immernoch $\frac{0}{0}$ ergeben könnten, falls die Transition in einen Zustand führt, der zuvor noch nie erreicht wurde. In diesem Fall setzt man die Gewichte einfach auf 1. Dieses Vorgehen zeigt gute Ergebnisse (siehe Abb. 3.24). Wie erhofft, lernt das Greedy $\hat{Z}$-Learning die optimale Policy dann schneller als die Q- und Random Z-Algorithmen.

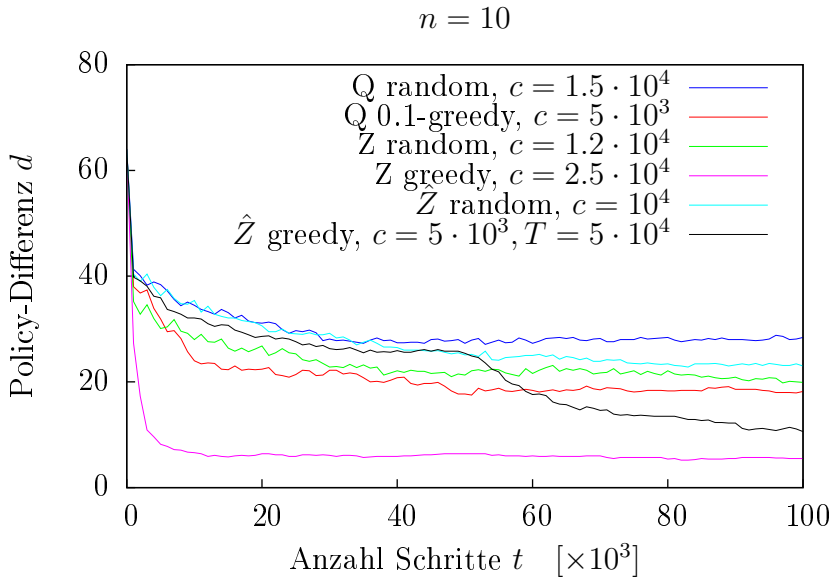

Abbildung 3.24: Policy-Verlauf small3: Der Agent kontrolliert die Dynamik beim Greedy $\hat{Z}$-Learning ab Zeitpunkt T nur in Zuständen, in denen er schon genug Erfahrung gesammelt hat. So vermeidet er grobe Fehler und lernt schneller als die Random Algorithmen.

Diese Methode ermöglicht es sogar, auf die Verzögerungszeit T ganz zu verzichten. Der Agent kann also von Anfang an versuchen, die Dynamik in einem Zustand zu kontrollieren, falls er bereits Schätzwerte zu allen Aktionen in diesem Zustand gesammelt hat. Abb. 3.25 zeigt, dass das Greedy $\hat{Z}$-Learning auf diese Weise sehr schnell lernt. Der entscheidende Vorteil

hierbei ist, dass der Agent selbständig den Grad an Kontrolle der Umgebung steigert, sodass man sich eine manuelle Konfiguration über den Parameter T sparen kann.

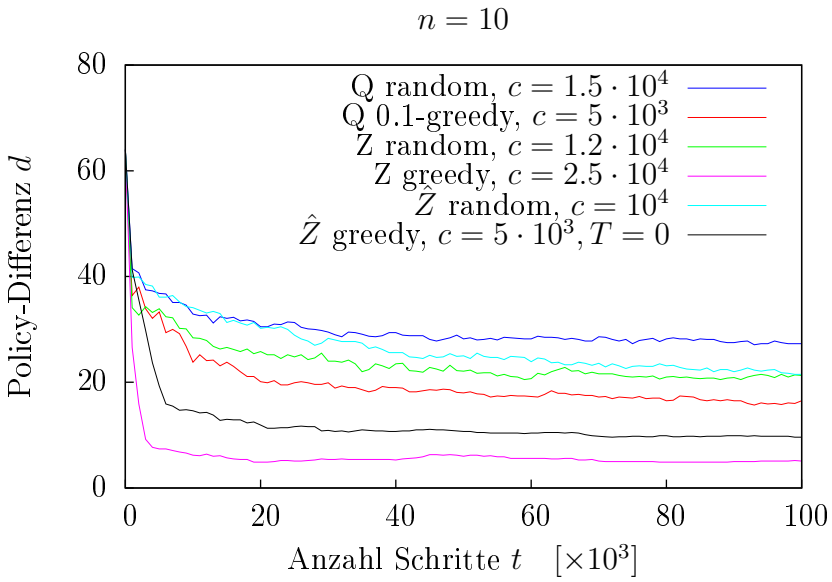

Abbildung 3.25: Policy-Verlauf small3: Greedy $\hat{Z}$-Learning verzichtet auf eine Schätzphase ($T = 0$) und darf von Beginn an kontrolliert agieren, falls der Zustand ausreichend besucht wurde. So steigt der Grad an Kontrolle durch den Agenten zunehmend, wodurch die Effizienz des Greedy $\hat{Z}$-Learnings noch erhöht wird.

4 Zusammenfassung

4.1 Auswertung

Nachdem nun das Z-Learning in den verschiedenen Experimenten untersucht und mit Q-Learning verglichen wurde, lässt sich eine Antwort auf die in dieser Arbeit untersuchte Forschungsfrage geben. Die Ergebnisse der Experimente belegen nicht nur, dass sich das Z-Learning tatsächlich auf diskreten MDP anwenden lässt, sondern sie zeigen auch seine Überlegenheit beim Lösen klassischer RL Probleme gegenüber dem Q-Learning Algorithmus. In allen dargestellten Fällen konvergiert das (Greedy) Z-Learning schneller gegen eine optimale Policy als das (Greedy) Q-Learning. Es profitiert also davon, dass es sich nur im Zustandsraum und nicht im Zustands-Aktionsraum bewegt. Dieser Vorteil wird umso größer, je größer die Gridworld (der Zustandsraum) ist.

Wie erwartet, lernen die Z-Algorithmen unter der optimistischen Annahme, dass das Übergangsmodell $\mathcal{P}^a_{ss'}$ bekannt ist, sehr viel schneller, da sie die zusätzlichen Informationen zum Modell verwerten. Vor allem das Greedy Z-Learning mittels Importance Sampling bewirkt eine deutliche Beschleunigung der Konvergenz. Dabei hat es gleichzeitig die nützliche Eigenschaft der impliziten Erkundung des Zustandsraums, welche die Optimierung eines ϵ Parameters beispielsweise überflüssig macht.

Aber auch bei vollkommen unbekanntem Modell lässt sich das Z-Learning einsetzen. Indem es die relativen Häufigkeiten der Transitionen während des Lernens speichert und aktualisiert, entwickelt es eine immer genauere Schätzung der Dynamik, auf Basis derer es eine deterministische Poli-

cy generieren und sogar Importance Sampling betreiben kann. Auch diese schätzenden Z-Algorithmen lernen schneller als die entsprechenden Q-Algorithmen.

Allerdings ist noch ein weiterer Aspekt zu beachten. Die Z-Algorithmen besitzen zwar eine bessere Konvergenzrate, jedoch steigert sich vor allem bei den schätzenden Z-Algorithmen der Speicher- und Rechenaufwand und damit die effektive Laufzeit. Für die relativen Häufigkeiten wird für jede Aktion eine Matrix der Größe $|\mathcal{S}| \times |\mathcal{S}|$ im Speicher angelegt. Zudem muss beim Importance Sampling in jedem Zeitschritt ein lineares Gleichungssystem der Größenordung $|\mathcal{S}| \times |\mathcal{A}|$ gelöst werden. Bei Problemen mit sehr großen Zustands- und Aktionsmengen steigt die benötigte Rechenzeit somit sehr schnell an. Hier bedarf es weiterer Experimente, um die Anwendbarkeit auf wirklich großen Problemen zu untersuchen.

4.2 Ausblick

Die vielversprechenden Ergebnisse geben Anlass zu weiterer Betrachtung und Analyse des Z-Learnings. An dieser Stelle folgen einige Vorschläge zu weiteren möglichen Experimenten und Verbesserungen.

In dieser Arbeit wurde das Verhalten des Z-Learnings nur auf Gridworlds untersucht. Um die allgemeine Brauchbarkeit zu untersuchen, sind weitere Experimente mit Z-Learning in anderen Problemdomänen verschiedener Größenordnungen nötig.

Der kritische Punkt beim Z-Learning in unbekannter Umgebung ist das Schätzen der Dynamik. Hier könnte man sich effizientere Methoden zum Schätzen überlegen, die speziell auf die Problemstellung zugeschnitten sind. Beispielsweise könnte man versuchen, statt dem gesamten (globalen) Matrizensystem $\mathcal{P}^a_{ss'}$, nur die lokale Dynamik (siehe Übergangsschema in Abb. 3.2) einer Aktion zu lernen (falls man Kenntnisse über ein Schema hat). Unter der Annahme, dass diese in fast allen Zuständen gültig ist, lassen sich damit prinzipiell alle benötigten Wahrscheinlichkeiten berechnen. Für Zu-

stände neben Hindernissen, am Rand der Gridworld oder im Wind ändert sich das lokale Verhalten zwar, doch könnte man sich hierfür womöglich dennoch Lösungen überlegen. Denkbar wäre es auch, den Agenten bereits mit einem (vielleicht unvollständigen) Vorwissen (Expertenwissen) über die Dynamik auszustatten. Dieses könnte er verwenden (und mit der Zeit seinen Erfahrungen anpassen), um noch besser kontrolliert agieren zu können.

Die verwendete Heuristik zur Policygenerierung muss nicht in jeder Problemstellung sinnvolle Ergebnisse liefern. Statt eine Heuristik zu verwenden, kann man auch versuchen, die Schätzwerte zu benutzen, um die optimale Aktion durch Lösen der Gleichung (10) in Algorithmus 1 zu bestimmen.

4.3 Schluss

In der vorliegenden Arbeit wurde ein neuer Algorithmus zum Lösen von Reinforcement Learning Problemen vorgestellt und untersucht. Das Z-Learning, erstmalig in [Tod06] beschrieben, wurde ursprünglich für eine spezielle Klasse stetiger Markov Decision Problems konzipiert. In dieser Arbeit wurde gezeigt, dass sich das Z-Learning auch zum Lösen klassischer, diskreter Markov Decision Problems eignet und dabei sogar effizienter ist als der bisherige Standardalgorithmus Q-Learning. Um die Funktionsweise des Q- und Z-Learnings zu erklären, wurde zunächst ein kurzer Überblick über den aktuellen Stand der klassischen Reinforcement Learning Theorie, sowie über die neu eingeführte Klasse stetiger Probleme gegeben. Daraufhin wurden die Algorithmen in mehreren Experimenten auf diskreten Gridworlds getestet und ausgewertet. Die erzielten Ergebnisse belegen die Überlegenheit von Z-learning in den untersuchten Aufgabenstellungen. Weiterführende Studien zur Nutzbarkeit des Z-Learnings in anderen Problemdomänen wären durchaus erfolgversprechend und wünschenswert, würden jedoch leider über den Umfang dieser Arbeit hinaus gehen.

Literaturverzeichnis

[BD62] Bellman, Richard E. ; Dreyfus, Stuart E.: *Applied Dynamic Programming.* Princeton University Press, 1962

[Bel57] Bellman, Richard E.: *Dynamic Programming.* Dover paperback (2003). Princeton University Press, 1957

[Bis06] Bishop, Christopher M.: *Pattern Recognition and Machine Learning.* Neue Auflage (2007). Springer, 2006

[Ger10] Gerlach, Ruben: *Roboternavigation unter Berücksichtigung menschlicher Bewegungsabläufe*, Technische Universität Berlin, Diplomarbeit, Februar 2010

[JJS94] Jaakkola, Tommi ; Jordan, Michael I. ; Singh, Satinder P.: On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. In: *Neural Computation* 6 (1994), November, Nr. 6, S. 1185–1201

[KL51] Kullback, Solomon ; Leibler, Richard A.: On Information and Sufficiency. In: *Annals of Mathematical Statistics* 22 (1951), Nr. 1, S. 79–86

[RN04] Russel, Stuart ; Norvig, Peter: *Künstliche Intelligenz - Ein moderner Ansatz.* 2. Auflage. Pearson Studium, 2004

[SB98] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning - An Introduction.* 1. Auflage. MIT Press, 1998

[Tes94] TESAURO, Gerald: TD-Gammon, a self-teaching backgammon program, achieves master-level play. In: *Neural Computation* 6 (1994), März, Nr. 2, S. 215–219

[Tes95] TESAURO, Gerald: Temporal Difference Learning and TD-Gammon. In: *Communications of the ACM* 38 (1995), März, Nr. 3, S. 58–68

[Tod06] TODOROV, Emanuel: Linearly-solvable Markov decision problems. In: *Advances in Neural Information Processing Systems* 19 (2006), S. 1369–1376. – Schölkopf et.al. (Herausgeber), MIT Press

[Tod09] TODOROV, Emanuel: Efficient computation of optimal actions. In: *Proceedings of the National Acadamy of Sciences of the United States of America* 106 (2009), Juli, Nr. 28, S. 11478–11483

[Tsi94] TSITSIKLIS, John N.: Asynchronous Stochastic Approximation and Q-Learning. In: *Machine Learning* 16 (1994), September, Nr. 3, S. 185–202

[Wat89] WATKINS, Christopher J. C. H.: *Learning from Delayed Rewards*, Cambridge University, Diss., Mai 1989

[WD92] WATKINS, Christopher J. C. H. ; DAYAN, Peter: Q-Learning. In: *Machine Learning* 8 (1992), Mai, Nr. 3, S. 279–292

Abbildungsverzeichnis

Algorithmenverzeichnis