Reinforcement Learning für kontinuierliche Zustands- und Aktionsräume unter Berücksichtigung der wissenschaftlichen Informationsverarbeitung

Inaugural-Dissertation
zur Erlangung des Doktorgrades der
Fakultät für Mathematik und Physik
der Albert-Ludwigs-Universität Freiburg i. Brsg.
vorgelegt von

Michael C. Röttger
aus Ahaus

März 2009

Dekan:	Prof. Dr. Kay Königsmann
Leiter der Arbeit:	Prof. Dr. J. Honerkamp
Referent:	Prof. Dr. J. Honerkamp
Korreferent:	apl. Prof. Dr. Stefan Waldmann

Tag der Verkündigung des Prüfungsergebnisses: 15. Mai 2009

Veröffentlichungen im Rahmen dieser Arbeit

In Fachzeitschriften und Jahresberichten

- Röttger, Michael C. ; Liehr, Andreas W.: Control task for Reinforcement Learning with known optimal solution for discrete and continuous actions. In: *Journal of Intelligent Learning Systems and Applications* (2009), zur Veröffentlichung angenommen

- Riede, Moritz K. ; Sylvester-Hvid, Kristian O. ; Kühne, Martin ; Röttger, Michael C. ; Zimmermann, Klaus ; Liehr, Andreas. W.: On the Communication of Scientific Results: The Full-Metadata Format / Freiburg Materials Research Center. 2009 (20090302a). – Scientific Information. DOI: 10.1594/fmf.SI20090302a

- Röttger, M. C. ; Liehr, A. W.: Steuerungsprobleme mit bekannten Lösungen als Baustein zur Entwicklung von Reinforcement-Learning-Algorithmen. In: *Jahresbericht FMF* Bd. 2008. Freiburg : Freiburger Materialforschungszentrum, 2009

- Thomann, Y. ; Thomann, R. ; Röttger, M. C. ; Liehr, A. W. ; Mülhaupt, R.: AFM Nano-Indentation und quantitative Bestimmung der E-Modul-Werte von Polymeren. In: *Jahresbericht FMF* Bd. 2007. Freiburg : Freiburger Materialforschungszentrum, 2008, S. 28–30. – URL `http://www.freidok.uni-freiburg.de/volltexte/5321/`

- Liehr, A. W. ; Zimmermann, K. ; Röttger, M. C. ; Belenkaia, L. ; Kühne, Martin ; Sylvester-Hvid, Kristian O. ; Wawrzinek, Ch. ; Braun, S. ; Riede, M. K.: Nachhaltige Informationsverarbeitung für die Materialforschung. In: *Jahresbericht FMF* Bd. 2007. Freiburg : Freiburger Materialforschungszentrum, 2008, S. 26–28. – URL `http://www.freidok.uni-freiburg.de/volltexte/5321/`

- Backofen, R. ; Borrmann, H.-G. ; Deck, W. ; Dedner, A. ; Raedt, L. D. ; Desch, K. ; Diesmann, M. ; Geier, M. ; Greiner, A. ; Hess, W. R. ; Honerkamp, J. ; Jankowski, St. ; Krossing, I. ; Liehr, A. W. ; Karwath, A. ; Klöfkorn, R. ; Pesché, R. ; Potjans, T. ; Röttger, M. C. ; Schmidt-Thieme, L. ; Schneider, G. ; Voss, B. ; Wiebelt, B. ; Wienemann, P. ; Winterer, V.-H.: A Bottom-up approach to Grid-Computing at a University: the Black-Forrest-Grid Initiative. In: *Praxis der Informationsverarbeitung und Kommunikation* 29 (2006), Nr. 2, S. 81–89

Vorträge

- Röttger, M. C. ; Liehr, A. W.: How to decide – Machine Learning with Python. EuroSciPy 2008 (Veranst.), *Leipzig 27.7.2008* – Vortrag

- Röttger, M. C. ; Liehr, A. W. ; Honerkamp, J.: Mesoskopisches Modell zur Steuerung einer Polymerisationsreaktion. 14. internes FMF-Kolloquium (Veranst.), *Europapark Rust 19.-20.10.2006* – Vortrag

- RÖTTGER, M. C. ; LIEHR, A. W. ; HONERKAMP, J.: Anwendung der Optimalen Steuerung zur Nutzung paralleler Resourcen. 13. internes FMF-Kolloquium (Veranst.), *Titisee-Neustadt 6.-7.10.2005* – Vortrag

- THOMANN, Yi ; RÖTTGER, Michael ; LIEHR, Andreas W. ; THOMANN, Ralf ; MÜLHAUPT, Rolf: Quantitative elastic modulus and scratch resistance with AFM nano indentation. 16. internes FMF-Kolloquium (Veranst.), *Europapark Rust, 15.-16.10.2007* – Vortrag

- HONERKAMP, J. ; LIEHR, A. W. ; RÖTTGER, M. C. ; SHAH, B. K.: Datenanalyse und Steuerung von Polymerisationsreaktionen. Workshop des BMBF-Projekts HOPS im Deutschen Kunststoff-Institut (Veranst.), *Darmstadt, 24.11.2005* – Vortrag

- LIEHR, Andreas W. ; RÖTTGER, Michael C. ; ZIMMERMANN, Klaus ; QUACK, Lorenz ; HONERKAMP, Josef: Bildverarbeitung, Datenanalyse und Steuerung. Ergebnisworkshop des BMBF-Projekts HOPS im Freiburger Materialforschungszentrum (Veranst.), *Freiburg, 19.5.2006* – Vortrag

Inhaltsverzeichnis

1 Einleitung

1.1 Konzept des Reinforcement Learning

Viele Prozesse in Wissenschaft und Technik lassen sich als dynamische Systeme darstellen, die mit Hilfe von Differentialgleichungen oder in Form von Übergängen zwischen diskreten Zuständen modelliert werden. Neben der reinen Beschreibung einer Beobachtung entsteht dabei häufig der Wunsch, aktiv in das beobachtete System einzugreifen, um einen gewünschten Zielzustand zu erreichen, ein Zielfunktional zu maximieren bzw. Kosten zu minimieren. Gesucht ist eine aktive, möglichst *optimale Steuerung* des Systems.

Eine solche Beeinflussung eines dynamischen Systems geht über das Prinzip der *Regelungstechnik* hinaus. Bei der Regelung wird stets eine *Regelgröße* mit einer *Vergleichsgröße* verglichen und entsprechend angepasst. Diese fortlaufende Rückkopplung fehlt bei der Steuerung. Im englischen Sprachgebrauch werden beide Arten des Eingriffs unter dem Begriff *control* zusammengefasst – der Unterschied wird explizit durch die Bezeichnungen *open loop control* (Steuerung) und *closed loop control* (Regelung) gemacht.

Im technischen Einsatz einer Steuerung wird die Antwort auf die Frage „Was ist wann zu tun?“ meist im Vorfeld der Anwendung durch Analyse einer bekannten Systemdynamik theoretisch bestimmt und fest implementiert. Dazu existiert eine gut ausgearbeitete Theorie der *optimalen Steuerung* (Bryson und Ho 1975). Wie Bryson Jr. (1996) in einer Übersicht zeigt, ist sie seit langem im produktiven Einsatz und wurde z.B. zur Steuerung des Mondlandemoduls der Apollo 11 eingesetzt (Widnall 1971).

Eine genaue Kenntnis der Systemdynamik ist jedoch nicht immer a priori vorhanden, z.B. wenn ein Roboter sich in einer ungewohnten Umgebung befindet und der Mensch diese nicht aktiv bewerten kann oder will. In solchen Umgebungen kann es sein, dass es zunächst völlig unklar ist, wie diese auf einen aktiven Eingriff reagieren. An an einem solchen Punkt kann das aktive System, im Beispiel der Roboter, sich nur auf bisherige Erfahrungen verlassen. Für den Fall, dass diese Kenntnisse nicht ausreichen, sollte das System die Möglichkeit haben, durch Ausprobieren weitere Informationen zu sammeln, um dadurch sinnvolle Entscheidungen treffen zu können. Das *Reinforcement Learning* (RL) ist eine Disziplin des maschinellen Lernens und zielt darauf ab, dass autonome Systeme durch Interaktion mit einer solchen, zunächst unbekannten Umgebung, lernen sich optimal zu verhalten.

Der Ursprung des Konzepts des Reinforcement Learning ist auf Beobachtungen aus der Verhaltenspsychologie zurückzuführen: Verhaltensexperimente zeigen, dass Lernprozesse signifikant durch Änderung der Erwartungen über zukünftige Belohnungen oder Bestrafungen bestimmt werden (z.B. Cziko 1995). Physiologische Experimente konnten bestimmte Neuronengruppen in Primaten identifizieren, deren Feuerungsrate mit Fehlern in der Vorhersage solcher Belohnungsereignisse zusammenhängt. Diese Erkenntnisse können durch quantitative Methoden erklärt werden, die dem Reinforcement Learning nahe stehen

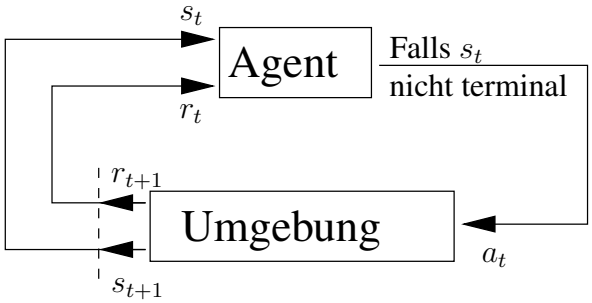


Abbildung 1.1: Steuerungsproblem im Sinne des Reinforcement Learning nach Sutton und Barto (1998). Das betrachtete System wird konzeptionell aufgeteilt in einen Agenten und seine Umgebung. Der Agent beeinflusst seine Umgebung lediglich durch die Wahl von Aktionen, die eine Zustandsänderung hervorrufen. So entscheidet er sich zum Zeitpunkt t für die Aktion a_t. Seine Umgebung antwortet mit einer Belohnung, dem numerischen Wert r_{t+1}, und der Information über ihren neuen Zustand s_{t+1}. Dies geschieht so lange, bis ein *terminaler* Zustand erreicht ist. Damit ist eine *Episode* beendet und der Prozess beginnt von neuem.

(Schultz et al. 1997). Im Sinne des Reinforcement Learning wird ein Steuerungsproblem aus folgender Perspektive betrachtet: Ein *Agent* versucht das Steuerungsziel zu erreichen, in dem er mit seiner *Umgebung* interagiert. Zum Zeitpunkt t erhält der Agent Informationen über den aktuellen Zustand s_t seiner Umgebung und entscheidet sich für eine Aktion a_t. Die Aktion wird durchgeführt, worauf sich der Zustand der Umgebung ändert. Der Agent bekommt den neuen Zustand s_{t+1} mitgeteilt und erhält gleichzeitig eine Belohnung r_{t+1} in Form eines numerischen Wertes. Die Abbildung 1.1 stellt den Informationsfluss zwischen dem Agenten und seiner Umgebung schematisch dar. Der Lernprozess ist in eine oder mehrere *Episoden* unterteilt. Jede Episode beginnt mit einem initialen Zustand s_0 der Umgebung. Sie ist entweder endlos (*unendlicher Zeithorizont*) oder endlich und wird nach T Zeitschritten durch das Erreichen eines *terminalen Zustands* s_T abgeschlossen (*endlicher Zeithorizont*). Einen Lernprozess mit mehr als einer Episode nennt man *episodisch*.

Für einen episodischen Lernprozess ergibt sich die folgende Sequenz von Zuständen, Belohnungen und Aktionen:

$$s_0 \overset{a_0}{\rightarrow} (r_1, s_1) \overset{a_1}{\rightarrow} (r_2, s_2) \overset{a_2}{\rightarrow} \ldots \overset{a_{T-1}}{\rightarrow} (r_T, s_T). \tag{1.1}$$

Der Ablauf einer solchen Sequenz wird ganz wesentlich durch die *Strategie* bestimmt, die der Agent bei seinen Entscheidungen für die jeweiligen Aktionen verfolgt. Diese Strategie wird auch als die *Politik* des Agenten bezeichnet und liefert eine Antwort auf die Frage:

Was ist jeweils im Zustand s zu tun?

Die Antworten darauf sind zunächst völlig unbestimmt und hängen von der Definition des Ziels ab, das der Agent zu erreichen sucht. Während des Lernprozesses ist der Agent in

der Lage, sein Verhalten in Form der verfolgten Politik zu verändern und an die Gegebenheiten anzupassen. Das Ziel des Reinforcement Learning ist es, in einem Trial-and-Error-Verfahren, ohne Vorgaben eines Experten und nur durch die Auswertung der Informationen aus der Sequenz (1.1) eine Politik zu finden, die auf lange Sicht dem Agenten maximale (Gesamt-)Belohnungen einbringt. Diese Politik ist eine *optimale Steuerung*. Diese Aufgabe wird als das *Kontroll-Problem* oder *Steuerungsproblem* bezeichnet.

Bei der Lösung eines Steuerungsproblems wird meist eine bereits vorhandene Politik während des Lernprozesses immer wieder leicht modifiziert, um ein gutes Gesamtergebnis zu erzielen. Ein wichtiger Bestandteil dieses Vorgangs ist die *Bewertung einer Politik*, die durch die Antwort auf die Frage

> *Was wird es bringen, die momentane Politik weiter zu verfolgen?*

bzw. alternativ

> *Welche Belohnung ist in der Summe zu erwarten, wenn die momentane Politik weiter verfolgt wird?*

gegeben ist. Diese Fragestellung wird als *Prediction-Problem* oder *Vorhersage-Problem* bezeichnet. Sie wird meist durch die Schätzung der so genannten *State-Value-Funktion* beantwortet, welche durch das Symbol V repräsentiert wird. Zur Lösung des Steuerungsproblems wird häufig eine zweite *Value-Funktion* verwendet, die *Action-Value-Funktion*, meist mit dem Symbol Q abgekürzt. Sie liefert eine Antwort auf die Frage

> *Welche Belohnung ist in der Summe zu erwarten, wenn zunächst eine bestimmte Aktion ausgeführt wird und danach die momentane Politik weiter verfolgt wird?*

Die *Value-Funktionen* V und Q werden im Kapitel 2 dieser Arbeit näher erläutert, sie seien jedoch an dieser Stelle erwähnt, um die nachfolgende Diskussion zu Steuerungsproblemen in der Literatur und die anschließenden Schlussfolgerungen transparent zu machen. Wichtig ist dabei, dass die Value-Funktionen V und Q zu einer gegebenen Politik eindeutig sind und dass diese häufig ein Mittel darstellen, um die gesuchte optimale Steuerung schließlich zu bestimmen.

Die Aufgabenstellung, allein mit Hilfe der Erfahrungen des Agenten eine optimale Steuerung zu suchen, beinhaltet große Herausforderungen an mögliche Lösungen, z.B.:

- Häufig hängt eine spätere Belohnung von einer richtigen Aktion zum jetzigen Zeitpunkt in der aktuellen Situation ab (engl. *delayed reward*) – wie kann eine solche Aktion verstärkt werden, d.h. wie kann man die Wahrscheinlichkeit erhöhen, dass in Zukunft tatsächlich diese Aktion ausgeführt wird, selbst wenn die sofortige Belohnung mitunter mager ausfällt?

- Wie kann man erreichen, dass eingefahrene Wege vermieden werden und der Agent offen bleibt für Experimente, die eventuell auf lange Sicht für bessere Erfahrungen sorgen?

Die Formulierung eines Steuerungsproblems aus der Perspektive des Reinforcement Learning ist eine sehr allgemeine Problemstellung. Viele Entscheidungsprozesse aus unterschiedlichsten Bereichen können auf diese Weise formuliert werden. Im folgenden Abschnitt werden einige Anwendungen genannt.

1.2 Anwendungen des Reinforcement Learning

In diesem Abschnitt werden einige reale Anwendungen des Reinforcement Learning beschrieben. Das Attribut *real* soll dabei so verstanden werden, dass

- die Aktionen des Agenten messbare physische, über die Änderung von rechnerinternen Speichern hinausgehende Auswirkungen haben, oder
- das Ergebnis eines Lernprozesses in produktive Systeme implementiert wird.

Das populärste Beispiel für eine erfolgreiche Anwendung des Reinforcement Learning ist *TD-Gammon* (Tesauro 1995, Kaelbling et al. 1996). Dieses System lernte nur durch virtuelle Partien gegen sich selbst so gut Backgammon zu spielen, dass es in späteren Versionen schließlich in Lage war, mit den weltbesten Spielern zu konkurrieren. Nach Aussage einer der damaligen Meister hat das System sogar stellenweise die menschlichen Spieler auf grundlegend neue Strategien hingewiesen.

Ein Motor in der Entwicklung des Reinforcement Learning sind zur Zeit sicherlich die Anwendungen in der Robotik, welche immer wieder zu neuen Untersuchungen und Ideen führen. Roboter können dort eingesetzt werden, wo es für den Menschen schwierig ist. Gerade deswegen ist das Konzept interessant, ein autonomes System in eine teilweise unbekannte Umgebung einbringen zu können, in der sich dieses dann zurecht findet. Eine hier in der Literatur mehrfach diskutierte Anwendung des Reinforcement Learning in der Robotik ist es, einen Roboter dazu zu bringen, einem Korridor zu folgen (Rohanimanesh und Mahadevan 2001, Theocharous et al. 2001, Smart und Kaelbling 2000). Besonders stark tritt das Reinforcement Learning innerhalb der Robotik seit einigen Jahren beim *Roboter-Fußball* in Erscheinung. Federführend ist hier das internationale *RoboCup*-Projekt, das sich zum Ziel gesetzt hat, bis zum Jahr 2050 eine Mannschaft von autonomen, humanoiden Robotern bereit zu stellen, die in der Lage ist, gegen die amtierenden menschlichen Fußball-Weltmeister zu gewinnen (The RoboCup Federation 2009). Viele Veröffentlichungen zu Anwendungen des Reinforcement Learning stammen von Gruppen, die sich kontinuierlich auf die ständig steigenden Anforderungen der jährlichen Weltmeisterschaft im Roboter-Fußball vorbereiten (siehe z.B. Kwok und Fox 2004, Duan et al. 2007, Aler et al. 2009).

Es gibt Ansätze, das Reinforcement Learning für medizinische Anwendungen einzusetzen, wie z.B. zur Segmentierung von Ultraschall-Aufnahmen der Prostata (Sahba et al. 2008) oder in einem Gerät zur Unterstützung der Atmung bei Schlafapnoe (Kreutz und Honerkamp 2003, Kreutz 2003).

Eine Aufgabenstellung, die immer wieder Anlass bietet, Reinforcement Learning-Methoden anzuwenden, ist die effiziente Nutzung von Ressourcen, teilweise mit Unterstützung der Industrie (Vengerov 2005, Tesauro et al. 2006, Vengerov 2007) oder auch der NASA, wie z.B. bei der Optimierung des Ladeprozess eines Space-Shuttle (Zhang und Dietterich 1995). In den Bereich der Ressourcen-Verteilung fällt zudem die Verwaltung von Rechner-Ressourcen (Wang und Usher 2004, Galstyan et al. 2005, Röttger et al. 2005).

Weitere Anwendungen sind: Die Verbesserung der Regelung einer Heizschleife (Anderson et al. 1997, Young et al. 2003), die Kontrolle eines nichtlinearen Neutralisationsprozesses (Syafiie et al. 2007), die Ermittlung der Übertragungskosten bei der Breitbandübertragung von digitalen Daten (Ren et al. 2007), ein automatisches Handelssystem für Aktien

(Bertoluzzo und Corazza 2007) oder die Steigerung der Effektivität von intelligenten Tutor-Systemen (Martin und Arroyo 2004). Weitere Studien untersuchen die Anwendbarkeit des Reinforcement Learning zur Verbesserung von Stoßdämpfern (Howell et al. 1997) oder zum Design digitaler Filter (Howell und Gordon 2001). Vielversprechend ist ein Projekt zur Optimierung eines Verbrennungsprozesses bei der Stromerzeugung in Kohlekraftwerken (Fakulät für Maschinenbau, Universität Ilmenau 2009). Weitere Anwendungen finden sich in den Übersichtsartikeln von Keerthi und Ravindran (1995) sowie Kaelbling et al. (1996).

Insgesamt kann man feststellen, dass das Reinforcement Learning innerhalb vieler Untersuchungen in Hinblick auf praktische Anwendungen erforscht wird. Es wird jedoch kaum für produktive Anwendungen eingesetzt, die über das Versuchsstadium hinaus gehen. Das liegt möglicherweise an der Komplexität des Optimierungsproblems, bzw. der Allgemeinheit des Ansatzes und dem Anspruch des selbständigen Lernens ohne eine lehrende Instanz. Die Komplexität der Optimierungsaufgabe hängt oft damit zusammen, dass in den meisten Fällen der realen Anwendung des Reinforcement Learning dicht liegende, reelle Zahlen zur Beschreibung von Zuständen erforderlich sind. Häufig ist in der Praxis auch die stufenlose Wahl einer Aktion gegenüber der Beschränkung auf endlich viele mögliche Aktionen die natürlichere Wahl, wie z.B. bei der Steuerung von Servomotoren. In diese Fällen können die Erfahrungen des Agenten nicht mehr vollständig in Tabellen mit endlich vielen Einträgen gespeichert werden. Der Agent ist gezwungen, seine Erfahrungen zu generalisieren und unter Umständen Informationen wieder zu verwerfen, da er noch nicht weiß, was zukünftig relevant sein wird und sein Speicherplatz begrenzt ist. Dies hat zur Folge, dass unter Umständen ähnliche Erfahrungen mehrfach gemacht werden müssen, bevor das System diese als ähnlich erkennt und eine gemeinsame Generalisierung möglich ist. Diese Vorgehensweise kann es für den Agenten erforderlich machen, zunächst sehr zeitintensive Lernprozesse zu durchleben, bevor sich eine produktive Strategie ergibt. Kaelbling et al. (1996) vertreten die Meinung, dass diese Probleme es in vielen Fällen erzwingen, möglichst viel Vorwissen in das System einzubringen. Dazu gibt es verschiedene Ansätze, die sich von dem Einbringen von extern erworbenen Vorwissen (Hailu und Sommer 1999, Garcia 2003, Framling 2005, Taylor et al. 2008) bis hin zur Übertragung von Wissen aus anderen ähnlichen Lernprozessen erstrecken (Perkins und Precup 1999, Madden und Howley 2004, Sherstov und Stone 2005).

1.3 Klassische Steuerungsprobleme

Bevor ein Reinforcement Learning-Algorithmus in der realen Welt zum Einsatz kommt, sollte er möglichst ausgiebig in einer virtuellen Umgebung getestet werden, da hier in der Regel das notwendige Trial-and-Error-Verfahren deutlich weniger zeit- und kostenintensiv ist.

Hier stellt sich die Frage, wie ein solcher Algorithmus getestet werden könnte – nützlich sind dazu Steuerungsprobleme, für die bereits eine Lösung bekannt ist, um einen Vergleich anstellen zu können. Eine solche bekannte Lösung kann entweder direkt die theoretische optimale Lösung sein, oder aber eine bis dato sehr gute Lösung, die mit anderen Methoden gefunden wurde. Es folgt eine Übersicht mit einer Auswahl von typischen Steuerungspro-

blemen in der Literatur, die in Zusammenhang mit dem Reinforcement Learning behandelt werden. Es wird diskutiert, welche zu testenden Aspekte sie jeweils abdecken und wie der Vergleich mit anderen Lösungen erfolgt.

Zur Strukturierung der Übersicht werden einige weitere Begriffe eingeführt: Im Folgenden sei $\mathcal{S}$ die Menge der möglichen Zustände der Umgebung, wobei diese Menge endlich oder unendlich viele Elemente enthalten kann. Im ersteren Fall wird im Folgenden kurz von *diskreten Zuständen* bzw. *diskreten Zustandsräumen* und im zweiten Fall von *kontinuierlichen Zuständen* bzw. *kontinuierlichen Zustandsräumen* gesprochen. Entsprechendes gilt für die Aktionen: Die Menge $\mathcal{A}(s)$ sei die Menge der Aktionen, die dem Agenten im Zustand s zur Verfügung steht. Sie hängt im Allgemeinen vom aktuellen Zustand ab, da möglicherweise einige Aktionen in bestimmten Zuständen nicht zur Verfügung stehen. Im weiteren Verlauf soll der Zustand durch eine Tupel von n reellen Zahlen repräsentierbar sein, $\mathcal{S} \subset \mathbb{R}^n$, sowie eine Aktion durch ein Tupel von m reellen Zahlen: $\mathcal{A} \subset \mathbb{R}^m$. Wie auch die Zustände werden die Aktionen in zwei Varianten betrachtet: Als endliche Menge $\mathcal{A} = \{\bar{a}_1, \ldots, \bar{a}_m\}$ von *m diskreten Aktionen* oder als Menge mit unendlich vielen Elementen, deren Elemente als *kontinuierliche Aktionen* bezeichnet werden. Ein Beispiel für eine kontinuierliche Aktion ist beispielsweise die Wahl eines Parameters, der innerhalb eines Intervalls stufenlos eingestellt werden kann. In der Literatur wird ähnlich wie bei den Zuständen häufig von *diskreten* bzw. *kontinuierlichen Aktionsräumen* gesprochen. Im Allgemeinen bzw. für einige Beispiele aus realen Anwendungen sind auch Aktionsräume denkbar, die aus diskreten und kontinuierlichen Aktionsmengen zusammengesetzt sind.

1.3.1 Diskrete Zustandsräume

Wie in Abschnitt 1.1 beschrieben, ist das *Vorhersage-Problem*, also die Bewertung einer gegebenen Politik π, Bestandteil vieler Algorithmen zur Suche nach einer optimalen Politik. Eine solche Bewertung geschieht in Form einer Funktion $V^\pi(s)$, die jedem Zustand einen numerischen Wert zuordnet. Dieser Wert ist die zu erwartende gesamte Belohnung für den Agenten, sollte er ausgehend vom Zustand s der Politik π folgen.

Es stellt sich die Frage, welches steuerbare System aus der Literatur sich zum Test von Algorithmen zur Lösung des Vorhersage-Problems mit diskreten Aktionen eignet? Hier ist das *Hop-World-Szenario* ein interessanter Kandidat: Es handelt sich um eine einfache Markov-Kette mit diskreten Zuständen, zwischen denen bestimmte Übergangswahrscheinlichkeiten festgelegt sind. Die Menge der möglichen Aktionen besteht aus Wechseln zu einem benachbarten Zustand, die Politik π legt fest, mit welcher Wahrscheinlichkeit ein jeweils möglicher Wechsel geschehen kann. Das System kann so konfiguriert werden, dass V^π exakt bekannt ist.

Boyan (1999) verwendet 13 Zustände und schätzt V^π mittels *Temporal-Difference Learning* (TD(λ), siehe Abschnitt 2.6) und einem Verfahren namens *Least-Squares Temporal-Difference Learning* oder kurz *LSTD(λ)*. Die wahre Funktion V^π ist in diesem Fall linear gewählt. Der Vergleich der gefundenen Value-Funktionen mit V^π geschieht mittels der Wurzel aus dem mittleren quadratischen Fehler über alle Zustände (RMS-Fehler). Es werden die Ergebnisse von TD(λ) für verschiedene Parametersätze mit denen von LSTD(λ) verglichen. Wegen des diskreten Zustandsraums ohne Einsatz einer Funktionsapproximation für V^π ist

es leicht möglich über alle Zustände zu mitteln. Das gleiche System zusammen mit dem RMS-Fehler verwendet Xu et al. (2002) um einen Algorithmus namens *Recursive Least-Squares Temporal-Difference Learning* (RLS-TD(λ)) zu testen und mit TD(λ) und LSTD(λ) zu vergleichen. Das gleiche Problem greift der Autor später erneut auf, diesmal mit einer nichtlinearen State-Value-Funktion V^π und dem neuen Algorithmus *Kernel Least-Squares Temporal Difference Learning* (KLS-TD(λ)) bzw. einer ressourcenschonenden Variante (Xu et al. 2005, Xu 2006). Für das Vorhersage-Problem existiert also ein Testproblem mit diskreten Zuständen, für das die Lösung analytisch bekannt ist. Sie wird vereinzelt genutzt, um entsprechende Algorithmen mit Hilfe des RMS-Fehlers zu vergleichen.

Ein ähnliches System wie das Hop-World-Szenario wird als das so genannte *Chain-Walk-Problem* beschrieben. Es handelt sich hier ebenfalls im eine Markov-Kette mit linear verbundenen Zuständen, dabei wird jedoch eine optimale Steuerung gesucht. Die Aufgabe des Agenten besteht darin, möglichst nicht die beiden Randzustände zu betreten, sondern in den mittleren Zuständen zu verbleiben. Die optimale Value-Funktion ist bekannt und wird beispielsweise bei der Untersuchung der Policy-Iteration (siehe Abschnitt 2.4) mit dynamischen Bayes'schen Netzwerken zum Vergleich verschiedener Approximationstechniken verwendet (Koller und Parr 2000). Außerdem wird sie genutzt, um ein Verfahren namens *Least-Squares Policy Iteration* (LSPI) zu testen (Lagoudakis und Parr 2003), bzw. um diesen Algorithmus mit der *Kernel-Based Least-Squares Policy Iteration* (KLSPI) zu vergleichen (Xu et al. 2007). Dazu werden grafische Darstellungen der Approximationen von Value-Funktionen mit bekannten optimalen Value-Funktionen verglichen, d.h. eine Bewertung in Form eines numerischen Vergleichsmaßes findet nicht statt.

Für diskrete Zustands und Aktionsräume sind also prinzipiell Steuerungsprobleme vorhanden, die sich für einen Vergleich von verschiedenen Algorithmen mit Hilfe einer bekannten Lösung eignen. Zudem steht hier das Mittel der *Dynamischen Programmierung* zur Verfügung, mit der sich effizient numerisch optimale Lösungen berechnen lassen (Sutton und Barto 1998).

1.3.2 Kontinuierliche Zustandsräume

Im Folgenden werden typische Steuerungsprobleme aus der Literatur für kontinuierliche Zustandsräume aufgelistet und diskutiert, inwieweit sie für einen Vergleich von Algorithmen geeignet sind oder ob eine optimale Lösung bekannt ist.

Das ist zum Einen das *Cart-Pole-Balancing-Problem*, einer der häufig verwendeten Steuerungsaufgaben in der Literatur zum Reinforcement Learning. Es handelt sich dabei um einen drehbar gelagerten Stab auf einem beweglichen Wagen, und es gilt, den Stab aufrecht zu balancieren. Es gibt vier kontinuierliche Zustandsvariablen: Den Ort x des Wagens, dessen Geschwindigkeit $\dot{x}$, den Auslenkungswinkel θ des Stabes im Vergleich zur Vertikalen und dessen zeitliche Änderung $\dot{\theta}$. Dem Agenten stehen zwei diskrete Aktionen zur Verfügung: Die Einwirkung einer Kraft mit festgelegtem Betrag auf die eine oder die andere Seite des Wagens, d.h. der Agent entscheidet sich für das Vorzeichen dieser Kraft. Eine optimale Value-Funktion V^* ist für dieses Problem nicht beschrieben, jedoch ermöglicht die Aufgabe einen Vergleich verschiedener Algorithmen dahingehend, dass man sich anschaut, nach wie vielen Episoden sprich Anläufen der Agent in der Lage ist, den Stab über beliebig lange

Zeit senkrecht zu halten. Xu et al. (2002) nutzen das System, um verschiedene Algorithmen auf diese Weise zu vergleichen.

Mit einer etwas anderen Definition des Problems arbeiteten van Hasselt und Wiering (2007). Sie verwenden kontinuierliche Aktionen und tragen gemittelte Belohnungen gegen absolvierte Episoden auf und vergleichen so Algorithmen, die sie der neuen Klasse der *Continuous Actor Critic Learning Automata* (CALCA) zuordnen. Li und Dagli (2003) verwendet eine abweichende Problemdefinition – so werden u.a. die Kräfte am Wagen mit einem zusätzlichem Rauschterm versehen. Das Ergebnis für das vorgeschlagene Verfahren namens *Hybrid Least-Squares Method* wird anhand der steigenden Zeitdauer, die der Agent den Stab halten kann, im Laufe der Episoden visualisiert.

Für das Cart-Pole-Balancing-Problem existiert keine analytische Darstellung der optimalen Valuefunktionen, daher kann deren Approximation nicht zur Analyse des Lernprozesses herangezogen werden. Prinzipiell wäre es jedoch denkbar ein sinnvolles Kriterium zu definieren, der einen Vergleich von Algorithmen möglich macht und gleichzeitig die Systemparameter festlegt. Tatsächlich ist aber eine einheitliche Verwendung von Parametern oder Vergleichskriterien für dieses Steuerungsproblem nicht in der Literatur feststellbar.

Das *Mountain-Car-Problem* ist ein vielfach verwendetes, anschauliches Problem mit zwei kontinuierlichen Zustandsvariablen und einer diskreten Aktionsvariable für das sich die Näherungen für die optimale State-Value-Funktion V^*, die durch das Reinforcement Learning gewonnen werden, und die Entwicklung des Lernprozesses gut visualisieren lassen (siehe auch Abschnitt 2.7.3). Allerdings existiert auch für dieses Problem keine analytisch bekannte Lösung. Als Referenz-System dient meistens jenes von Sutton (1996), welches in Abschnitt 2.7.3 dieser Arbeit beschrieben ist. Es gibt eine Vielzahl von Veröffentlichungen, die sich auf das Mountain-Car-Problem beziehen, oft verwenden sie jedoch leicht verschiedene Varianten oder eine andere Nomenklatur (Moore 1990, Moore und Atkeson 1995, Boyan und Moore 1995, Kretchmar und Anderson 1997, Smart und Kaelbling 2000, Ozawa und Shiraga 2003, Whiteson und Stone 2006, Epshteyn und DeJong 2006, Taylor et al. 2008), was einen Vergleich der Ergebnisse schwierig macht.

Zusammengefasst lässt sich sagen: In der Literatur existieren viele teilweise recht unterschiedliche Ansätze zur Lösung von Reinforcement Learning-Problemen mit kontinuierlichen Zuständen. Häufig werden sehr individuelle Steuerungsaufgaben herangezogen, um die Leistungsfähigkeit der Verfahren zu demonstrieren. Außerdem zeigt es sich, dass sich die bisherigen Vergleiche mit ähnlichen Steuerungsaufgaben nur bedingt zur Bewertung eines Algorithmus eignen: Entweder fehlt ein Vergleichsmaß, es werden variierende Parametersätze verwendet oder aber es ist keine obere Schranke in Form einer optimalen Lösung bekannt.

Ein idealer Maßstab für den Vergleich verschiedener Algorithmen ist eine bekannte optimale Lösung eines Systems, sofern diese zur Verfügung steht. Eine solche Lösung ermöglicht es außerdem, Zwischenergebnisse während des Lernprozesses einem Vergleich zugänglich zu machen, was eine detailliertere Fehlersuche und -behebung ermöglicht. Ein Steuerproblem mit bekannter Lösung kann darüber hinaus dazu verwendet werden, um ein gegebenes Verfahren, das sich aus mehreren Subsystemen zusammensetzt, systematisch auf dessen Eignung zur Lösung von Reinforcement Learning-Problemen zu testen. In dieser Arbeit wird zu diesem Zweck ein Steuerungsproblem definiert, der *Richtungssucher*, des-

sen optimale Lösung für diskrete als auch kontinuierliche Aktionen in analytischer Form bekannt ist (Röttger und Liehr 2009). Darüber hinaus erlaubt die Verwendung von kontinuierlichen Aktionen bei diesem Problem im Allgemeinen eine höhere Gesamtbelohnung als die Verwendung von diskreten Aktionen. Dies ist nicht die Regel: Viele Systeme für kontinuierliche Aktionen haben die gleiche optimale Lösung wie im Falle diskreter Aktionen, weil letztendlich praktisch nur Extremwerte für die Aktionsvariablen gewählt werden (so genannte *Bang-Bang-Steuerungen*, siehe z.B. Kim 2002).

1.3.3 Kontinuierliche Aktionsräume

In den Ursprüngen des Reinforcement Learning wurden zunächst lediglich Steuerungsprobleme mit endlichen Aktionsmengen (*diskrete Aktionen*) betrachtet (Sutton 1988, Watkins 1989, Mataric 1991). Im Bereich der diskreten Aktionen ist das Reinforcement Learning theoretisch gut untersucht. In Verbindung mit diskreten Zustandsräumen sind effiziente Algorithmen zur Lösung bekannt und es existieren Konvergenzbeweise (Sutton und Barto 1998). Für reale Anwendungen ist jedoch eine festgelegte Diskretisierung des Aktionsraum nicht immer sinnvoll, beispielsweise wenn es in manchen Bereichen des (Zustands-) Aktionsraums mehr Variationen gibt als in anderen und daher dort eine feinere Auflösung nötig ist. Es mag Fälle geben, in denen eine geringe Abweichung von einer optimalen Aktion bereits zu deutlich kleineren Gesamtbelohnungen führt und daher eine Steuerung mit hoher Genauigkeit nötig ist.

Die Arbeit mit kontinuierlichen Aktionen stellt gewisse Herausforderungen an einen Reinforcement Learning-Algorithmus (Lazaric et al. 2008): So ist im Gegensatz zu diskreten Aktionen a priori gar nicht mehr klar, wie eine Aktion zu selektieren ist, wenn man den Aufwand, im gesamten Aktionsraum zu suchen, nicht tragen möchte.

Bislang gibt es, verglichen mit den Arbeiten zu diskreten Aktionen, relativ wenig Literatur zu kontinuierlichen Aktionsräumen. Die zu diesem Zweck entwickelten Algorithmen verfolgen eine Vielzahl unterschiedlicher Strategien und Ideen. Es folgt eine kurze Übersicht durch eine Klassifizierung in Methoden, die einer *Actor-Critic-Architektur* folgen und jene die es nicht (explizit) tun.

Actor-Critic-Methoden sind RL-Algorithmen, die die Strategie des Agenten in einer separaten Speicherstruktur unabhängig von der Value-Funktion verwalten. Der *Actor* ist der Teil, der die Strategie implementiert und der *Critic*, häufig eine State-Value-Funktion, bewertet die Aktionen des Actors. Diese Bewertung geschieht mit Hilfe des so genannten *TD-Fehlers* (TD: *temporal difference*, siehe auch Abschnitt 2.5), aus dem beide, der Actor und der Critic, lernen. Im Falle kontinuierlicher Zustände und Aktionen können für beide Elemente verschiedene Arten von Funktionsapproximatoren eingesetzt werden. Die Actor-Critic-Methoden haben den Vorteil, dass sie auf natürliche Weise kontinuierliche Abbildungen von Zuständen auf Aktionen ermöglichen, um so auf leichte Änderungen in Zuständen mit leichten Änderungen in Aktionen reagieren zu können. Konkret heißt das (siehe Sutton und Barto 1998, Gaskett et al. 1999), sie ermöglichen

1. eine schnelle Aktionsselektion ohne Anwendung von aufwendigen Optimierungsroutinen, was sie für kontinuierliche Aktionen attraktiv macht und

2. die Suche nach stochastischen Politiken.

Die Umsetzung des Actor-Critic-Konzepts variiert stark, u.a. bezüglich der Verarbeitung und Speicherung bereits gemachter Erfahrungen. Lazaric et al. (2008) verwalten im Actor für jeden Zustand Listen von diskreten Aktionen, aus denen mittels eines Monte-Carlo-Verfahrens kontinuierliche Aktionen generiert werden. Andere Autoren approximieren Wahrscheinlichkeitsdichten, aus denen Aktionen gezogen werden (Arie et al. 2006, Satoh 2006, van Hasselt und Wiering 2007). Dabei kommen unter anderem neuronale Netze und Entwicklungen nach orthonormalen Wellenfunktionen zum Einsatz. Weiter gibt es Ansätze, die Monte-Carlo-Verfahren mit der Aktionsselektion mittels Wahrscheinlichkeitsverteilungen kombinieren (Wawrzyński und Pacut 2004). Das erste Reinforcement Learning-System, das einen Critic zusammen mit kontinuierlichen Aktionen verwendet, wird von Millán und Torras (1992) diskutiert.

Klassische Ansätze, die sich nicht auf die separate Repräsentation eines Actor und eines Critic stützen sind das *Q-Learning* (Watkins 1989) und das in dieser Arbeit genutzte *Sarsa-Learning* (siehe Abschnitt 2.7), bei welchen die Politik indirekt durch die Änderung einer gespeicherten Action-Value-Funktion verbessert wird. Speziell das Q-Learning wiederum hat einen Vorteil gegenüber den Actor-Critic-Methoden: Der Agent kann einer anderen Politik folgen, als derjenigen, der er zu verbessern sucht. Dieser Vorteil wird als *exploration insensitivity* bezeichnet (Gaskett et al. 1999). Während diese Ansätze für diskrete Aktionen gut diskutiert sind, gibt es auch hier für die Verwendung von kontinuierlichen Aktionen in der Literatur viele Variationen, die die Speicherung der Erfahrungen und die Approximation betreffen.

Das populäre *Tile-Coding* (Sutton 1996, Sutton und Barto 1998) geht auf den *Cerebellar Model Articulation Controller* (kurz *CMAC*, Albus 1975) zurück, einem neuronalen Netzwerk bzw. assoziativen Speicher basierend auf einem Modell des Kleinhirns von Säugetieren. Gewöhnlich wird es im Reinforcement Learning verwendet, um kontinuierliche Zustände zu generalisieren. Sherstov und Stone (2004) setzen es in Verbindung mit Q-Learning zur Kodierung einer kontinuierlichen Aktionsvariable ein. Santamaría et al. (1997) diskutieren ausführlich die Anwendung eines Sarsa-Algorithmus in Kombination mit Tile-Coding zum Vergleich mit zwei speicherbasierten Funktionsapproximatoren, die die gesammelten Erfahrungswerte in Form einer dynamisch veränderlichen Karte verwalten. Auch hier kommen neuronale Netze zur Anwendung wie auch bei ten Hagen und Kröse (2000) (*Neurales Q-Learning*) oder bei Gaskett et al. (1999). Letztere nutzen einen so genannten *Wirefitter* als Funktionsapproximator (angelehnt an Baird und Klopf 1993). Smart und Kaelbling (2000) präsentieren wiederum einen *Hedger* genannten Algorithmus, der die Erfahrungen des Agenten sammelt und für eine lokal gewichtende Regression (LWR) einsetzt. Fukao et al. (1998) wendet ein Regularisierungsverfahren an, um Probleme mit kontinuierlichen Zuständen und Aktionen zu behandeln. Die Funktion Q wird approximiert durch eine Linearkombination von Gaußfunktionen im gemeinsamen Zustands- und Aktionsraum, die Erfahrungen des Agenten bilden dabei die Basispunkte der Glockenkurven.

Neben der klaren Unterscheidung zwischen Architekturen, die eine separate Datenstruktur für die Politik verwalten (Actor-Critic-Architektur) und die, die es nicht tun, gibt es auch Mischformen. Bei diesen ist zwar prinzipiell die Funktionalität trennbar in einen Actor und

einen Critic, allerdings sind beide in einer gemeinsamen Datenstruktur gespeichert. Dazu gehört beispielsweise die von Bonarini et al. (2006) verwendete Fuzzy-Menge, die im Rahmen eines *Fuzzy Inference Systems* (FIS) genutzt wird (siehe auch Jouffe 1998, Deng und Er 2003), um die Approximation der Action-Value-Funktion zusammen mit einer Datenstruktur abzubilden, aus der sich eine (quasi) kontinuierliche Politik ableiten lässt. Flentge (2006) greift dazu auf ein *Growing Neural Gas* (GNG) zurück. Jedem Neuron werden zusätzliche Größen zugeordnet, die ebenfalls wie die Repräsentation einer State-Value-Funktion aus dem TD-Fehler lernen und zur Aktionsselektion verwendet werden. Millán et al. (2002) benutzen auf ähnliche Weise eine selbstorganisierende Karte, die diskrete Aktionen zusammen mit Q-Werten speichert. Eine so genannte *incremental topology preserving map* wird dabei dynamisch bei Bedarf um neue Erfahrungen ergänzt. Ein weiterer aktueller Ansatz ist die Approximation der Action-Value-Funktion durch einen binären Entscheidungsbaum (Jodogne und Piater 2006). Kimura (2007) nutzt schließlich Ideen aus dem klassischen Tile-Coding zusammen mit Monte-Carlo-Methoden zur Aktionsselektion, also zur Bestimmung einer Politik aus den bisherigen Erfahrungen des Agenten.

Es bleibt festzuhalten, dass die Vielfalt in den Ansätzen zur Verarbeitung von kontinuierlichen Aktionen groß ist und in diesem Teilbereich des Reinforcement Learning bislang noch keine gemeinsame Richtung auszumachen ist, obschon er für die Anwendung des Reinforcement Learning große Bedeutung hat. Bislang hat sich weder ein Lernalgorithmus, noch eine grundsätzliche Architektur bzw. Verarbeitung der Erfahrungen des Agenten als Mittel der Wahl durchgesetzt. Aus diesem Grund wird in dieser Arbeit in neuer, leicht zu implementierender Ansatz vorgestellt, um das Richtungssucherproblem für kontinuierliche Aktionen zu lösen (Röttger und Liehr 2009). Gleichzeitig eignet sich das Richtungssucherproblem dazu, die grundlegende Funktionalität von Algorithmen für kontinuierliche Aktionen zu testen.

1.4 Über den Umgang mit wissenschaftlichen Daten

Praktisch jeder Naturwissenschaftler hat mit einer Fülle von wissenschaftlichen Daten zu tun. Dies gilt auch für Forscher im Bereich des maschinellen Lernens, dem das Reinforcement Learning zuzurechnen ist. Diese Daten erstrecken sich von Messwerten, über Zwischenergebnisse durch Analysen oder grafische Darstellungen bis hin zu Simulationsergebnissen, die durch die Auswertung eines Modells entstehen. Im Mittelpunkt steht dabei die wissenschaftliche Thematik, wie beispielsweise die Suche nach Modellen zur Beschreibung von Beobachtungen, der Versuch Hypothesen durch Experimente zu bestätigen oder die Optimierung von Konstruktionen oder Algorithmen für den praktischen Einsatz. Die Erhebung von wissenschaftlichen Daten ist dabei ein wichtiger Mittler zwischen Messung und Simulation bzw. zwischen Theorie und Praxis. Die Präparation von Experimenten und Simulationen kostet viel Zeit und Geld und sollte auf eine nachhaltige Art und Weise geschehen, die die Wiederverwendbarkeit zur Vermeidung von Wiederholungen ermöglicht und Syntheseeffekte durch den Austausch zwischen verschiedenen Wissenschaftlern, wie z.B. Theoretikern und Experimentatoren, fördert. Auch die Durchführung von numerischen Analysen, Fehlerabschätzungen oder die Bewertung von wissenschaftlichen Daten aufgrund von do-

kumentierten Kriterien kann viel Arbeitszeit und Rechenleistung in Anspruch nehmen und sollte nach Möglichkeit zur wiederholten Verwendung zur Verfügung stehen. Innerhalb jeder wissenschaftlichen Arbeitsgruppe ist der Zugriff auf die Ergebnisse der aktuellen und vorangegangenen Arbeit zu dem gemeinsamen Interessengebiet von zentralem Interesse. Dies steht im Einklang mit den Grundsätzen der Deutschen Forschungsgemeinschaft (DFG) zur *guten wissenschaftlichen Praxis* (DFG 1998). Diese enthält u.a. die Empfehlung, wissenschaftliche Primärdaten zehn Jahre aufzubewahren, welche von der Universität Freiburg übernommen wurde (Senat der Albert-Ludwigs-Universität Freiburg 2007).

Ungeachtet der Wichtigkeit der Arbeitsweise mit wissenschaftlichen Daten, gibt es bislang relativ wenig Veröffentlichungen, die den Anspruch haben, sich allgemein an Naturwissenschaftler zu richten (Oden et al. 2003, Kühne und Liehr 2009, Riede et al. 2009). Auf spezielle Fachgebiete bezogen gibt es jedoch durchaus Literatur, bei denen die Daten im Mittelpunkt stehen, so z.B. in Zusammenhang mit einer Initiative der NASA zur Erforschung des Sonnensystems (Daley et al. 2005). Hervorstechend ist hier sicher das LHC-Experiment am CERN, das derart große und zugleich komplexe Datenmengen produziert, dass ohne eine systematische Aufteilung der Daten und ihrer Analyse auf viele Kooperationspartner das Projekt zum Scheitern verurteilt wäre (siehe z.B. De La Hoz et al. 2008). Wie schon bei den Anfängen des Internets ist das CERN hier möglicherweise ein Vorreiter, was das Bewusstsein im Umgang mit wissenschaftlichen Daten angeht, nicht zuletzt auch wegen des großen Aufwands und der hohen Kosten, die mit dem LHC-Experiment in Verbindung stehen. Interessant ist in diesem Zusammenhang das Entstehen neuer Zeitschriften zu diesem Thema, wie dem frei zugänglichen *Data Science Journal* (International Council for Science 2009), welches von der Unesco unterstützt wird. Darüber hinaus interessieren sich auch die größeren Softwarekonzerne für das Thema (z.B. Gray et al. 2005). Es ist davon auszugehen, dass in Zukunft das Thema stärker in das Bewusstsein von Wissenschaftlern rückt. Dies wird begünstigt durch Veröffentlichungen, in denen der Umgang mit den jeweiligen wissenschaftlichen Daten zwar fachspezifisch erläutert, aber gleichzeitig in den Zusammenhang mit allgemein gültigen Konzepten gestellt wird. Dazu leistet diese Arbeit einen Beitrag.

1.5 Aufbau dieser Arbeit

Das zweite Kapitel gibt einen Überblick der Grundlagen der in dieser Arbeit verwendeten Begriffe und Methoden des Reinforcement Learning. Die dazu benötigte Notation für Funktionen, Parameter und andere Größen orientiert sich im Wesentlichen an derjenigen von Sutton und Barto (1998), da sie in einem Großteil der Literatur zum Thema Reinforcement Learning zur Anwendung kommt.

Im dritten Kapitel werden typische Probleme diskutiert, denen Wissenschaftler im Umgang mit wissenschaftlichen Daten begegnen, zusammen mit Konzepten, wie sich diese vermeiden lassen. Die Umsetzung dieser Konzepte wird anhand der Software für das Reinforcement Learning, die im Rahmen dieser Arbeit entwickelt wurde, dargestellt. Die grundsätzlichen Ideen lassen sich unabhängig vom Reinforcement Learning auch auf andere Bereiche übertragen.

Im vierten Kapitel wird ein neues Steuerungsproblem mit kontinuierlichen Zustandsvariablen diskutiert, der *Richtungssucher*. Für dieses Problem sind die optimalen Lösungen in Form von Value-Funktionen sowohl für diskrete als auch für kontinuierliche Aktionen vollständig bekannt und können analytisch angegeben werden. Es kann als Testproblem für neue Reinforcement Learning-Algorithmen dienen oder als anschauliches didaktisches Beispiel. Zudem wird ein Vergleichsmaß definiert und auf numerische Lösungen des Richtungssucher-Problems angewendet. Das Maß eignet sich auch zur Untersuchung anderer Steuerungsprobleme mit bekannter optimaler Lösung.

Die Arbeit schließt in Kapitel 5 mit einer Zusammenfassung und einem Ausblick mit weiteren Ideen und Ansätzen für eine systematische Arbeitsweise bei der Evaluation von Reinforcement Learning-Algorithmen und die Verwertung der dabei entstehenden Daten.

2 Grundlagen

In diesem Kapitel werden die Grundlagen beschrieben, die zur Lösung des Richtungssuchers benötigt werden. Ausgehend vom Markov'schen Entscheidungsprozess werden Grundbegriffe wie *Politik* oder *Value-Funktion* erläutert. Über die Ideen der Dynamischen Programmierung wird das Konzept des *Temporal Difference Learning* zunächst auf das Vorhersage-Problem, dann auf das Steuerungsproblem angewandt. Der sich daraus ergebende *Sarsa(λ)-Algorithmus* wird für das bekannte Mountain-Car-Problem benutzt. Schließlich wird eine Erweiterung vorgenommen, die die Suche nach Politiken mit kontinuierlichen Aktionen ermöglicht.

2.1 Der Markov'sche Entscheidungsprozess

Von zentraler Bedeutung für einen Großteil der Methoden und Ideen des Reinforcement Learning für endliche Zustands- und Aktionsräume ist die Markov-Eigenschaft des betrachteten Problems, d.h. das Ergebnis der Ausführung der Aktion a_t soll lediglich vom aktuellen Zustand s_t abhängen und nicht von vorherigen Zuständen, Aktionen und Belohnungen. Formal kann diese Eigenschaft als die Gleichheit

$$\Pr\left\{ s_{t+1} = s', r_{t+1} = r \,\middle|\, s_t, a_t \right\} \tag{2.1}$$

$$= \Pr\left\{ s_{t+1} = s', r_{t+1} = r \,\middle|\, s_t, s_{t-1}, \ldots, s_0, a_t, a_{t-1}, \ldots, a_0, r_t, r_{t-1}, \ldots, r_1 \right\} \tag{2.2}$$

zweier bedingter Wahrscheinlichkeitsverteilungen[1] formuliert werden, welche für alle möglichen $s' \in \mathcal{S}$, für alle $r \in \mathbb{R}$ und alle möglichen Folgen von $s_t, \ldots, s_0$, $a_t, \ldots, a_0$ und $r_t, \ldots, r_1$ gelten soll. Diese Eigenschaft ist grundlegend wichtig für das Reinforcement Learning, da der Agent lernen soll sich nur aufgrund des aktuellen Zustands für eine günstige Aktion zu entscheiden. Sie ermöglicht eine theoretische Betrachtung mit Hilfe von Kenntnissen über stochastische Prozesse und der dynamischen Programmierung (Sutton 1988, Watkins 1989) und es gibt Beispiele für die Anwendung von Reinforcement Learning-Algorithmen, die eine schlechte Performance aufweisen, wenn diese Eigenschaft verletzt ist (wie z.B. in Whitehead und Lin 1995). In der Praxis stellt sich heraus, dass diese Eigenschaft nicht immer im strengen Sinne benötigt wird, um mit Hilfe des Reinforcement Learning zu guten Strategien zu kommen. Häufig kann auch die Markov-Eigenschaft hergestellt werden, indem man die Definition des Systemzustands entsprechend anpasst. Besteht z.B. nicht nur eine Abhängigkeit von s_t, sondern zusätzlich auch von s_{t-1}, so kann eine alternative Beschreibung des Systemzustands in der Form $\tilde{s}_t = (s_{t-1}, s_t)$ das Problem lösen.

[1] Das Kürzel „Pr" steht für *probability*.

Sei $\mathcal{S}$ die Menge der möglichen Zustände des Systems und $\mathcal{A}$ die Menge aller möglichen Aktionen. Die Menge der möglichen Aktionen kann dabei unter Umständen für bestimmte Zustände eingeschränkt sein: $\mathcal{A}(s) \subset \mathcal{A}$. Der Einfachheit halber soll unabhängig davon, ob diese Mengen endlich oder unendliche viele Elemente haben, vom *Zustands-* bzw. vom *Aktionsraum* gesprochen werden.

Bei gegebener Markov-Eigenschaft entspricht das Reinforcement Learning-Problem einem *Markov(ian) Decision Process* (MDP), wie ihn Bellman (1957) beschrieben hat. Für endliche Zustands- und Aktionsräume spricht man auch von *endlichen Markov'schen Entscheidungsprozessen*. Ein solches *finites* MDP kann im Allgemeinen stochastisch formuliert sein, d.h. der Übergang von s nach s' nach Ausführung der Aktion a wird durch eine bedingte Wahrscheinlichkeit

$$\mathcal{P}^a_{ss'} = \Pr\left\{ s_{t+1} = s' \middle| s_t = s, a_t = a \right\} \tag{2.3}$$

und die Belohnung durch einen Erwartungswert

$$\mathcal{R}^a_{ss'} = E\left\{ r_{t+1} \middle| s_t = s, a_t = a, s_{t+1} = s' \right\} \tag{2.4}$$

beschrieben. Zusammen mit den Mengen der möglichen Zustände und Aktionen ist so der MDP im Wesentlichen definiert, lediglich die genaue Verteilung der Belohnungen wird für die weiteren Betrachtungen nicht verwendet. Trotz der stochastischen Natur des MDP wird die Umgebung des Agenten als *stationär* bezeichnet, falls die Verteilungen zur Beschreibung der Übergänge und der Vergabe der Belohnungen sich nicht im Laufe der Zeit ändert.

Für *deterministische* Systeme gilt, dass bei Ausführung der Aktion a im Zustand s sowohl der Folgezustand s', also auch die Belohnung $r(s, a, s')$ eindeutig festgelegt ist:

$$\mathcal{P}^a_{ss'} = \begin{cases} 1 & \text{falls } s \xrightarrow{a} s', \\ 0 & \text{sonst,} \end{cases} \tag{2.5}$$

$$\mathcal{R}^a_{ss'} = r(s, a, s'). \tag{2.6}$$

Um im Folgenden nicht zwischen episodischen und endlos andauernden Lernprozessen unterscheiden zu müssen, wird auf die Notation von Sutton und Barto (1998) zurückgegriffen: Episodische Lernprozesse mit einem terminalen Zustand s_T können als endlos andauernde Prozesse betrachtet werden, indem man annimmt, dass sie, einmal an einem terminalen Zustand angelangt, dort verbleiben, ohne dass es zusätzliche Belohnungen gibt:

$$s_{T+k} \equiv s_T \ \wedge \ r_{T+k} \equiv 0 \,\text{für}\ k = \{1, 2, 3, \ldots\}. \tag{2.7}$$

Die (diskontierte) gesamte Belohnung, die der Agent im Zeitintervall $[t, T]$ sammelt, ist

$$R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{k+t+1}, \tag{2.8}$$

wobei γ der so genannte *Diskontierungsfaktor* (engl. *discount factor*) ist. Für $\gamma = 0$ ist der Agent *kurzsichtig* in dem Sinne, dass er lediglich die nächste Belohnung r_{t+1} berücksichtigt.

Für $0 < \gamma < 1$ gilt, dass zu dem gegebenen Grad frühere Belohnungen höher bewertet als spätere. Für andauernde Lernprozesse ohne terminalen Zustand ist dies auch notwendig, um Divergenz zu vermeiden. Für $\gamma = 1$ werden Belohnungen gleich gewichtet, unabhängig vom Zeitpunkt ihrer Vergabe.

Eine *Politik* oder *Strategie* gibt eine Antwort auf die Frage, was der Agent in einem gegebenen Zustand s tun wird. Ist die Politik *deterministisch*, so ist sie eine Abbildung von der Menge der Zustände auf die Menge der möglichen Aktionen und man kann die Antwort direkt ablesen:

$$\pi : \mathcal{S} \to \mathcal{A}.$$

Im Allgemeinen ist eine Politik *stochastisch* - sie gibt die Wahrscheinlichkeit (bzw. bei kontinuierlichen Räumen die Wahrscheinlichkeitsverteilung)

$$\pi : \mathcal{S} \times \mathcal{A}(s) \to \mathbb{R}$$

an, mit der der Agent im Zustand s die Aktion a durchführen wird. Eine deterministische Politik ist ein Spezialfall einer stochastischen Politik, bei dem es für jeden Zustand s nur eine Aktion $a(s)$ gibt, die mit endlicher Wahrscheinlichkeit ausgeführt wird.

Das *Folgen einer Politik* besteht darin, sukzessive den Folgezustand s' als Ergebnis einer durchgeführten Aktion a, wieder als Argument für die Politik zu verwenden, um die nächste Aktion a' zu bestimmen. Sollte dabei innerhalb einer Episode die Umgebung in einen terminalen Zustand überführt werden, so wird die Umgebung neu initialisiert und der Agent startet von neuem.

Das Ziel der Lösung der Steuerungsaufgabe besteht letztendlich darin, eine Politik zu finden, die das Steuerungsziel erfüllt: Der Agent soll seine Aktionen derart wählen, dass auf lange Sicht die Summe seiner Belohnungen maximal wird. Um dies zu konkretisieren und umzusetzen ist es notwendig, eine solche Politik zu bewerten bzw. sie dem Vergleich mit anderen Strategien zugänglich zu machen. Dies kann mit Hilfe der *Value-Funktionen* geschehen, welche im folgenden Abschnitt beschrieben werden.

2.2 Bewertung einer Politik durch Value-Funktionen

Mit der Notation (2.7) für episodische Lernprozesse in Form von endlos andauernden Lernprozessen wird die *State-Value*-Funktion bezüglich einer gegebenen Politik π für alle Zustände $s \in \mathcal{S}$ definiert als:

$$V^{\pi}(s) := E_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right\}. \tag{2.9}$$

Sie enthält die Information, welche diskontierte Gesamtbelohnung zu erwarten ist, wenn man im Zustand s startet und dann der Politik π folgt. Der Erwartungswert wird dabei über alle möglichen Trajektorien gebildet, die vom Zustand s ausgehen und mit der Politik π vereinbar sind.

In Analogie zur Zuordnung eines Wertes zu jedem Zustand bei gegebener Politik π kann man auch jedem Zustands-Aktionspaar (s, a) einen Wert zuordnen. Diese Abbildung wird

Action-Value-Funktion genannt und gibt Auskunft über die zu erwartende Gesamtbelohnung, wenn im Zustand s zunächst die Aktion a ausgeführt wird und der Agent *anschließend* der Politik π folgt:

$$Q^{\pi}(s,a) := E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right\}. \tag{2.10}$$

Um den Erwartungswert der Gesamtbelohnung des Agenten zu maximieren, ist es nützlich, eine *Ordnungsrelation* für Politiken einzuführen: Eine Politik π' ist „gleich gut“ oder „besser als“ eine Politik π, falls

$$\pi' \geq \pi \;:\Leftrightarrow\; V^{\pi'}(s) \geq V^{\pi}(s)\ \forall s \in \mathcal{S} \tag{2.11}$$

gilt. Eine *optimale Politik* π^* erfüllt

$$\pi^* \geq \pi \quad \forall \pi, \tag{2.12}$$

d.h. sie ist genau so gut oder besser als alle anderen Strategien. Ihr sind eindeutig die *optimalen Value-Funktionen* V^* und Q^* zugeordnet:

$$Q^*(s,a) := Q^{\pi^*}(s,a), \tag{2.13}$$

$$V^*(s) := V^{\pi^*}(s) = \max_a Q^*(s,a). \tag{2.14}$$

Einem gegebenen Paar von Funktionen V^* und Q^* können durchaus mehrere verschiedene optimale Politiken π^* zugeordnet sein – letztere sind nicht eindeutig. Ein Beispiel wäre ein Steuerungsproblem, bei dem es zwei aufeinander folgende Aktionen gibt, von deren Reihenfolge die Gesamtbelohnung nicht abhängt. Dieser Fall tritt auch bei dem in Kapitel 4 diskutierten Richtungssucher-Problem auf.

Die Gleichung (2.13) gibt einen Hinweis, wie bei gegebener Action-Value-Funktion $Q^*(s,a)$ die Politik π bestimmt werden kann. Ist der Zustand s gegeben, lässt sich durch Auswertung von

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{2.15}$$

eine Aktion selektieren. Die Aktion $a^*(s)$ ist *greedy* („gierig“) bezüglich der optimalen Action-Value-Funktion Q^*. Im Falle von mehreren möglichen optimalen Aktionen ist diese Wahl nicht eindeutig. Interpretiert man den Ausdruck „$\arg\max_{a'} Q^*(s,a')$“ als eine Menge von gleichwertigen optimalen Aktionen mit einer Anzahl von $|\arg\max_{a'} Q^*(s,a')|$ Elementen, so kann für diskrete Aktionen eine stochastische optimale Politik über die Wahrscheinlichkeiten

$$\pi^*(s,a) = \begin{cases} 1/|\arg\max_{a'} Q^*(s,a')| & \text{für } a \in \arg\max_{a'} Q^*(s,a'), \\ 0 & \text{sonst,} \end{cases} \tag{2.16}$$

definiert werden.

2.3 Klassifizierung von Lösungsverfahren im Reinforcement Learning

Grundsätzlich gibt es im Rahmen des Reinforcement Learning zwei verschiedene Fragestellungen, die mit verschiedensten Methoden behandelt werden können:

Vorhersage-Problem: Gegeben sei eine Strategie π. Wie sieht die State-Value-Funktion $V^\pi(s)$ aus, d.h. welche diskontierte Gesamtbelohnung kann der Agent ausgehend von einem Zustand s erwarten, wenn er der Strategie π folgt?

Kontroll-Problem: Wie kann der Agent eine optimale Strategie π^* bzw. eine möglichst gute Näherung einer solchen finden, die es ihm von allen oder bestimmten Startzuständen aus ermöglicht, seine zu erwartende diskontierte Gesamtbelohnung zu maximieren?

Die erste Fragestellung stellt eine Aufgabe dar, bei deren Bearbeitung der Agent zwar seine Umgebung ändert, aber dabei wie ein Automat agiert: Er folgt seiner Politik, was immer ihm auch begegnet. Dabei lernt er etwas über die eine gegebene Politik π im Kontext seiner Umgebung. Die zweite Fragestellung ist dagegen an einer *aktiven* Steuerung interessiert, die die Umgebung *gezielt* verändert. Es stellt sich heraus, dass die Methoden zur Suche von $V^\pi(s)$ häufig Ideen liefern, die auch bei der Entwicklung von Algorithmen zur Suche einer optimalen Strategie π^* nützlich sind. Bei diesen *Steuerungsalgorithmen* geht oft eine Verbesserung der Schätzung für $Q^\pi(s,a)$ für die momentan verfolgte Politik π mit einer Verbesserung dieser Politik Hand in Hand. Sutton und Barto (1998) bezeichnen diesen Prozess als *generalized policy iteration* (GPI, siehe auch Abschnitt 2.4).

Die Steuerungsalgorithmen im Reinforcement Learning lassen sich wiederum in zwei Klassen aufteilen. Die Algorithmen der ersten Klasse, die *Off-Policy-Algorithmen*, suchen die optimale Politik π^*, folgen dabei aber einer anderen Politik π, die den Zustands-Aktionsraum hinreichend gut durchläuft. Dagegen folgen die *On-Policy-Algorithmen* genau der Politik π, die durch den Lernprozess derart verbessert werden soll, dass sie der optimalen Politik π^* möglichst nahe kommt. Nissen (2007) betrachtet zum Vergleich von Off- und On-Policy-Algorithmen als Beispiel die Algorithmen *Sarsa(λ)* und *Q(λ)*. Der Autor kommt dabei zu dem Schluss, dass Sarsa(λ) zur Lösung großer komplexer Probleme geeigneter ist, da die Kosten der Exploration beim Lernprozess berücksichtigt werden. Beim Q(λ)-Learning geschieht dies wegen der Off-Policy-Eigenschaft nicht und so kann es sein, dass der Agent sich relativ lange in der Nähe schlechter Lösungen aufhält. In dieser Arbeit wird der Sarsa(λ)-Algorithmus verwendet, also ein On-Policy-Verfahren. Zu dessen Erläuterung sind noch einige Vorbemerkungen nötig.

2.4 Bellman-Gleichungen und Dynamische Programmierung

Es ist möglich, den Wert $V^\pi(s)$ eines Zustands bei Anwendung einer bestimmten Strategie π mit Hilfe der Werte der möglichen Folgezustände s' auszudrücken:

$$V^\pi(s) = E_\pi\left\{R_t \,\middle|\, s_t = s\right\} \tag{2.17}$$

$$= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \,\middle|\, s_t = s\right\} \tag{2.18}$$

$$= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{k+t+2} \,\middle|\, s_t = s\right\} \tag{2.19}$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+(t+1)+1} \,\middle|\, s_{t+1} = s'\right\}\right) \tag{2.20}$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right) \tag{2.21}$$

$$= E_\pi\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \,\middle|\, s_t = s\right\} \tag{2.22}$$

Die Ausdrücke (2.21) bzw. (2.22) für $V^\pi(s)$ werden als *Bellman-Gleichung für V^π* bezeichnet. Sie liefert die Grundidee für die *Policy-Iteration*, einem iterativen Verfahren, das es erlaubt, bei Kenntnis des Modells $V^\pi(s)$ zu schätzen.

Die optimale State-Value-Funktion $V^*(s) = V^{\pi^*}(s)$ bezüglich der optimalen Politik π^* kann rekursiv festgelegt werden durch die Annahme, dass im Zustand s genau *diejenige* Aktion a gewählt wird, die im weiteren Verlauf zu einem maximalen Nutzen führt, wenn man danach die optimale Politik verfolgt:

$$V^*(s) = \max_a Q^*(s,a). \tag{2.23}$$

Durch Verwendung der Definition (2.10) für Q sowie mit Hilfe einer äquivalenten Argumentation wie bei der Bellman-Gleichung für V^π für eine allgemeine Politik π folgt die *Bellman-Optimalitätsgleichung für V^**:

$$V^*(s) = \max_a E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s, a_t = a\right\}. \tag{2.24}$$

Analog lässt sich die *Bellman-Optimalitätsgleichung für Q^**

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a\right\}. \tag{2.25}$$

herleiten (Sutton und Barto 1998). Diese Gleichung ist Grundlage vieler Reinforcement Learning-Algorithmen, die sich wiederum mit den Konzepten der *Dynamischen Programmierung* verstehen lassen.

Die Methoden der *Dynamischen Programmierung* (DP) sind nur anwendbar, wenn man ein perfektes Modell der Umgebung in Form der Kenntnis von $\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$ vorliegen hat (siehe (2.3) und (2.4)). Sie ergeben sich auf natürliche Weise aus der Umsetzung der Bellman-Gleichungen in Iterationsverfahren und dienen als Ausgangspunkt für Ideen zu Reinforcement Learning-Verfahren, die ohne Kenntnis der Systemdynamik auskommen.

Policy-Evaluation

Die *Policy-Evaluation*, also eine Bewertung einer Politik, löst das *Vorhersage-Problem* zu einer gegebenen Strategie oder Politik π die State-Value-Funktion $V^\pi(s)$ zu finden. Interpretiert man die Bellman-Gleichung (2.21) für $V^\pi(s)$ als Fixpunktgleichung, so folgt daraus das Iterationsverfahren

$$V_{k+1} = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V_k(s') \right). \tag{2.26}$$

Die Konvergenz $V_k \longrightarrow V^\pi$ für $k \to \infty$ ist garantiert, sofern V^π existiert.

Policy-Verbesserung

Der Begriff *Policy-Verbesserung* oder *policy improvement* bezeichnet die Konstruktion einer neuen Politik π' auf Grundlage einer anderen Politik π, indem erstere so gewählt wird, dass sie greedy bezüglich der State-Value-Funktion V^π von π ist (hier für deterministische Politiken π und π'):

$$\pi'(s) = \arg\max_a Q^*(s,a) \tag{2.27}$$

$$= \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right) \tag{2.28}$$

Auf diese Weise ist sichergestellt, dass die neue Politik π' mindestens so gut ist wie π, wie in (2.11) definiert. Ein solcher Verbesserungs-Prozess ist essentiell für alle Reinforcement Learning-Verfahren, die nach einer optimalen Politik suchen und bei denen der Agent von dem unvollkommenen Wissen ausgehen muss, das ihm gerade zur Verfügung steht. Er kann explizit geschehen, wie bei der *Policy-Iteration* oder implizit wie bei der *Value-Iteration*. Beide Verfahren werden in den folgenden zwei Abschnitten in Kurzform dargestellt.

(Allgemeine) Policy-Iteration

Die *Policy-Iteration* ist ein DP-Algorithmus, der die Policy-Evaluation und die Policy-Verbesserung abwechselnd sukzessive durchführt, bis ein zufriedenstellendes Ergebnis erreicht ist. Dabei entsteht eine Folge der Form (nach Sutton und Barto 1998)

$$\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{V}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{V}} \pi_2 \xrightarrow{\text{E}} \ldots \xrightarrow{\text{V}} \pi^* \xrightarrow{\text{E}} V^*, \tag{2.29}$$

ausgehend von einer beliebigen Anfangsstrategie π_0. Neben der optimalen State-Value-Funktion V^* wird also zusätzlich eine optimale Politik π^* ermittelt. Damit ist die Policy-Iteration ein Verfahren zur Lösung des *Kontroll-Problems*, das allerdings die Kenntnis der Größen $\mathcal{P}^a_{ss'}$ und $\mathcal{R}^a_{ss'}$ zur Berechnung benötigt.

Während der Policy-Iteration werden Policy-Evaluation und Policy-Improvement explizit abwechselnd durchgeführt. Es gibt aber auch Verfahren, bei denen diese beiden Teilschritte enger miteinander verwoben sind, wie z.B. bei der in dem nachfolgenden Abschnitt beschriebenen Value-Iteration. Im Prinzip enthalten aber praktisch alle Reinforcement Learning-Algorithmen, auch diejenigen, die keine Kenntnis der Systemdynamik benötigen, die beiden Konzepte der Policy-Evaluation und -Verbesserung, die allerdings auf

unterschiedlichen Ebenen miteinander verwoben sind. Die Allgemeinheit dieser Reinforcement Learning-Verfahren wird von Sutton und Barto (1998) mit dem Begriff *Allgemeine Policy-Iteration* oder engl. *generalized policy iteration* (GPI) bezeichnet.

Neben den Algorithmen der dynamischen Programmierung sind die prominentesten Vertreter GPI-Verfahren der Q(λ)-Algorithmus von Watkins und Dayan (1992) oder auch der Sarsa(λ)-Algorithmus (z.B. Sutton 1996). Letzterer wird zusammen mit einer neuen Erweiterung für kontinuierliche Aktionen in dieser Arbeit verwendet und in Abschnitt 2.7 näher erläutert. Beiden Algorithmen ist gemein, dass das Lernen und die Anwendung des Argmax-Operators eng miteinander verschränkt ist.

Bei Verfahren, in denen beide Schritte separat erkennbar sind, spricht man häufig von einer *Actor-Critic-Architektur*. Bezogen auf den Agenten innerhalb des Reinforcement Learning-Problems wird dabei derjenige Teil, der die aktuelle Politik verbessert *Actor* genannt und der *Critic* ist der Teil des Agenten, der diese Politik bewertet. Neuere Vertreter dieser Algorithmen, die das Prinzip der Policy-Iteration im Namen tragen sind die *Least-Squares Policy Iteration* (LSPI) von Lagoudakis und Parr (2003) oder die *Kernel-based Least-Squares Policy Iteration* (KLSPI) von Xu et al. (2007). Letzteres stellt insofern eine Weiterentwicklung von LSPI dar, dass kein Vorwissen bei der Generalisierung des Zustandsraums eingeht.

Value-Iteration

Das Verfahren der *Value-Iteration* gehört auch zur Klasse der GPI-Algorithmen und führt die Politik-Verbesserung nicht wie die (spezielle) Policy-Iteration explizit sondern implizit durch und iteriert nur über die Value-Funktionen. Es ergibt sich durch eine Übertragung der Bellman-Optimalitätsgleichung (2.24) in die Iterationsvorschrift

$$V_{k+1}(s) = \max_a E\left\{ r_{t+1} + \gamma V_k(s_{t+1}) \,\middle|\, s_t = s, a_t = a \right\} \tag{2.30}$$

$$= \max_a \sum_{s'} \mathcal{P}^{a(s)}_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V_k(s') \right) \tag{2.31}$$

für alle $s \in \mathcal{S}$. Dies ist analog zur Bestimmung eines Fixpunkts $x = f(x)$ durch eine Iteration der Form $x_{n+1} = f(x_n)$ mit $n = 1, 2, 3, \ldots$ zu sehen. Die Werte $V_0(s)\,\forall s$ sind dabei beliebig. Die Durchführung der Iteration resultiert in einer Folge

$$V^{\pi_0} \longrightarrow V^{\pi_1} \longrightarrow V^{\pi_2} \longrightarrow \ldots \longrightarrow V^*, \tag{2.32}$$

die abgebrochen wird, sobald die erzielte Genauigkeit ausreicht. Sutton und Barto (1998) beschreiben einen Algorithmus, bei dem der Abbruch erfolgt, wenn die maximale Änderung von $V_k(s)$ über alle Zustände s zwischen zwei Iterationsschritten eine vorgegebene kleine Zahl unterschreitet.

Hat man somit eine hinreichend gute Näherung $\tilde{V}^*$ für V^* ermittelt, so kann über eine direkte Suche in der Form

$$\tilde{\pi}^*(s) = \arg\max_a \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \tilde{V}^*(s') \right] \tag{2.33}$$

eine Näherung für eine deterministische optimale Politik bestimmt werden. Wie schon bei der Policy-Iteration geht auch an dieser Stelle die Kenntnis über die Systemdynamik und die Vergabe der Belohnungen in den Lösungsalgorithmus mit ein.

2.5 Temporal-Difference Learning

Die im vorangegangen Abschnitt 2.4 beschriebenen Lösungsmethoden verwenden explizit Kenntnisse über die Systemdynamik. Wie in der Einleitung beschrieben, versucht man im Reinforcement Learning Verfahren zu finden, die eine optimale Politik bestimmen können, ohne dieses Wissen zu haben oder anzuwenden. Somit ist dem Agenten zunächst keine Vorhersage über die Zukunft möglich – dennoch muss er sich für Aktionen entscheiden, welche auf lange Sicht eine maximale Gesamtbelohnung in Aussicht stellen. Wie soll der Agent bewerten, ob eine ausgeführte Aktion a „gut“ war, wenn doch die Folgen dieser Aktion noch nicht abzusehen sind?

Eine Strategie ist es, bis zum Ende einer Episode abzuwarten und im Nachhinein die durchgeführten Aktionen zu bewerten, je nachdem wie zufriedenstellend der Ausgang der Episode ist. Diese Idee macht man sich bei der Anwendung von *Monte-Carlo-Methoden* des Reinforcement Learning zu Nutze (Sutton und Barto 1998). Die Umsetzung ist jedoch im Allgemeinen schwierig für lang andauernde Episoden, da unter Umständen viel Speicher benötigt wird. Die Monte-Carlo-Methoden sind gar nicht anwendbar für andauernde Lernprozesse, die keinen terminalen Zustand haben.

Ein anderer Ansatz zu obiger Frage leitet sich aus der Idee der Value-Iteration ab und verwendet die als Reaktion auf die Aktion a erhaltene Belohnung $r = r(s, a, s')$ und die Schätzung von $V(s)$ und $V(s')$ bzw. $Q(s, a)$ und $Q(s', a')$ (für die auf s' folgende Aktion a') um einen sofortigen Feedback zu berechnen. Diese Art der Algorithmen nennt man *Temporal-Difference-Methoden* (Sutton 1988).

Ein Beispiel für einen solchen Algorithmus ist das folgende *Update* der State-Value-Funktion $V(s)$, die das momentane Wissen des Agenten über die Funktion $V^\pi(s)$ darstellt, wobei π die Politik ist, die der Agent verfolgt:

$$V(s) \leftarrow V(s) + \alpha \left(r + \gamma V(s') - V(s)\right). \tag{2.34}$$

Jeder Besuch des Zustands s wirkt sich dahingehend aus, dass die Schätzung $V(s)$ seines Wertes näher an den Ausdruck $r + \gamma V(s')$ heran rückt, wobei die Lernrate[2] α festlegt, wie groß dieser Schritt ist. Bei Übereinstimmung von $V(s)$ und $r + \gamma V(s')$ ändert das Update nichts mehr. Dies ist analog zur Value-Iteration, die in Abschnitt 2.4 beschrieben ist, mit dem Unterschied, dass die Erfahrungen des Agenten sich aus der Wahrnehmung seiner Umgebung zusammensetzen und nicht aus der Auswertung eines Modells.

Für die Wahl von α gibt es verschiedene Ansätze. Es kann gezeigt werden, dass für diskrete Zustände und Aktionen bei fester Politik π die Anwendung der Update-Regel (2.34)

[2] In der engl. Literatur ist neben dem Begriff *learning rate* auch der Ausdruck *step-size parameter* üblich, siehe auch Sutton und Barto (1998).

zur Konvergenz $V \rightarrow V^*$ führt, sofern die Bedingungen

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \tag{2.35}$$

und

$$\sum_{k=1}^{\infty} \alpha_k^2(a) < \infty \tag{2.36}$$

eingehalten werden. Dabei ist $\alpha_k(a)$ die Lernrate, die für das Update von $V(s)$ beim k-ten Auftreten der Aktion a verwendet wird. Diese Voraussetzungen entsprechen denen der *stochastischen Approximation*, die auf Robbins und Monro (1951) zurückgeht. Zur stochastischen Approximation gehört eine ganze Klasse von Algorithmen, u.a. die Schätzung eines Funktionsarguments aus zufälligen Funktionswerten, für die nur ein Erwartungswert bekannt ist (Honerkamp 1994). Reinforcement Learning-Algorithmen, die auf ein Update unter Verwendung eines TD-Fehlers zurückgreifen, können unter diesem Gesichtspunkt interpretiert werden (Bharath und Borkar 1999). Häufig wird jedoch bewusst auf die Konvergenz von α verzichtet, um in nicht-stationären Umgebungen auf lange Sicht auf Änderungen reagieren zu können. Wie Sutton und Barto (1998) ausführen, wird eine Lernrate, die den Bedingungen (2.35) und (2.36) genügt, eher für theoretische Arbeiten eingesetzt. Even-Dar und Mansour (2003) untersuchen beispielsweise den Einfluss verschiedener Schemata der Verringerung von α auf den Q-Learning-Algorithmus. In dieser Arbeit wird stets ein konstantes α verwendet.

Die Anwendung von (2.34) ist der TD(0)-Algorithmus zur Schätzung der State-Value-Funktion $V^\pi(s)$ zu einer gegebenen Politik π; er ist ein Spezialfall des TD(λ)-Algorithmus, der im folgenden Abschnitt erläutert wird.

2.6 Der TD(λ)-Algorithmus

Viele Reinforcement Learning-Algorithmen können um die so genannten *Eligibility Traces* erweitert werden. Der Begriff *Eligibility-Trace* könnte übersetzt werden mit „Spur der Förderungswürdigen“. Praktisch gesehen ist es eine Markierung kürzlich besuchter Punkte im Zustands- oder auch Aktionsraum, die es möglich macht, einen TD-Fehler denjenigen Zuständen bzw. Aktionen zuzurechnen, die für diesen Fehler verantwortlich sind. Bevor die Traces eingeführt werden, soll deren Grundidee erläutert werden.

2.6.1 Konzept der Eligibility Traces

Zur Einführung der Eligibility Traces sind einige Vorbemerkungen nötig. Ist T die Anzahl der Zeitschritte einer Episode bis zum Erreichen eines terminalen Zustands, so gilt für die Gesamtbelohnung, die der Agent nach (2.8) ab dem Zeitpunkt t erhält

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T. \tag{2.37}$$

Das Update in (2.34) entspricht der allgemeinen Form

$$\text{neue Schätzung} \leftarrow \text{alte Schätzung} + \text{Lernrate} \times (\text{Zielgröße} - \text{alte Schätzung}) \tag{2.38}$$

und kann auch für eine alternative *Zielgröße* v formuliert werden:

$$V(s) \leftarrow V(s) + \alpha(v - V(s)) \tag{2.39}$$

Die Abweichung $v - V(s)$, die mit der Lernrate α gewichtet wird, wird auch der *Temporal-Difference-Fehler* oder *TD-Fehler* genannt (engl. *temporal difference error*).

Bei der Suche nach von $V^*(s_t)$ ist der Erwartungswert von R_t nach (2.37) die Zielgröße, der man durch ein Update von $V(s_t)$ näher kommen möchte. Im Allgemeinen sind dem Agenten die zukünftigen Belohnungen $r_{t+1}, r_{t+2}, \ldots, r_T$ in (2.37) nicht bekannt, daher wird ein Schätzer benötigt. Für die TD(0)-Methode wird dazu

$$R_t^{(1)} := r_{t+1} + \gamma V_t(s_{t+1}) \tag{2.40}$$

verwendet. Dabei ist $V_t(s_{t+1})$ die bis zum Zeitpunkt t ermittelte Schätzung der State-Value-Funktion V^π für den Folgezustand s_{t+1}. Stehen für das Ziel des Updates zwei Belohnungen zur Verfügung, so liefert

$$R_t^{(2)} := r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2}) \tag{2.41}$$

im Allgemeinen einen besseren Schätzer, da mehr Informationen berücksichtigt werden. Auf n Schritte übertragen erhält man den sogenannten *n-Schritt-Ertrag* oder *n-step return* (Sutton und Barto 1998)

$$R_t^{(n)} := r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}). \tag{2.42}$$

Dies ist ein Schätzer, der sich aus den n Belohnungen zusammensetzt, die der Aktion a_t folgen, korrigiert durch die momentane Schätzung der Bewertung des Zustands s_{t+n}. Dieser Schätzer kann verwendet werden, um die Näherung für V^π zu aktualisieren:

$$V_{t+1}(s) = V_t + \alpha\left(R_t^{(n)} - V_t(s_t)\right). \tag{2.43}$$

Die Grundidee der TD(λ)-Algorithmen basiert nun darauf, dass dieses Update nicht nur mit $R_t^{(n)}$ durchgeführt werden kann, sondern auch mit jedem Mittelwert von n-Schritt-Erträgen für verschiedene n. Jeder n-Schritt-Ertrag wird dabei mit einem Gewicht λ^{n-1} mit $0 \leq \lambda \leq 1$ gewichtet. Dadurch erhält man den sogenannten *λ-return* oder *λ-Ertrag*

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} = (1-\lambda)\left[R_t^{(1)} + \lambda R_t^{(2)} + \lambda^2 R_t^{(3)} + \ldots\right]. \tag{2.44}$$

Wegen (2.7) ist die unendliche Summe auch mit endlich langen Episoden vereinbar und mit

$$\sum_{n=1}^{\infty} \lambda^{n-1} = \frac{1}{1-\lambda} \qquad \text{für } 0 < \lambda < 1$$

gewährleistet der Normalisierungsfaktor $(1-\lambda)$, dass die Gewichte der n-Schritt-Erträge sich zu 1 summieren. Für Episoden mit terminalen Zuständen kann man zeigen, dass $\lambda = 1$ dem Einsetzen der tatsächlich auftretenden Gesamtbelohnung R_t entspricht: Wegen $R_t^{(n)} = R_t$ für $n \geq T - 1$ lässt sich der λ-Ertrag durch

$$R_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + (1-\lambda) \sum_{n=T-t}^{\infty} \lambda^{n-1} R_t \tag{2.45}$$

$$= (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + R_t \lambda^{-T-t} \underbrace{(1-\lambda) \sum_{k=0}^{\infty} \lambda^k}_{=1} \tag{2.46}$$

ausdrücken. Es gilt $\lim_{\lambda \to 1} R_t^\lambda = R_t$. Für $\lambda = 0$ ergibt sich (2.40).

Möchte man nun Updates in der Form

$$V(s) \leftarrow V(s) + \alpha \left(R_t^\lambda - V(s)\right) \tag{2.47}$$

explizit durchführen, benötigt man alle zukünftigen Belohnungen. Diese sind dem Agenten zum Zeitpunkt t jedoch nicht zugänglich, insofern ist diese Vorgehensweise akausal und nicht direkt vereinbar mit dem Ziel des Reinforcement Learning.

Allerdings gibt es zu der beschriebenen vorausschauenden Anwendung der λ-Erträge auch eine *rückschauende* Perspektive, die sich unter Verwendung der bereits gemachten Erfahrungen des Agenten implementieren lässt. Dazu wird eine zusätzliche Hilfsgröße eingeführt, der *Eligibility-Trace* $e_t(s) > 0$, eine Zahl, die für alle Zustände in jedem Schritt exponentiell verringert wird bzw. für den gerade besuchten Zustand s_t inkrementiert wird:

$$e_t(s) = \gamma \lambda e_{t-1}(s) + \begin{cases} 1 & \text{falls } s = s_t, \\ 0 & \text{sonst,} \end{cases} \quad \forall s \in \mathcal{S}. \tag{2.48}$$

Der Parameter γ ist der in (2.8) eingeführte Diskontierungsfaktor und ist problemspezifisch bzw. gehört genauer gesagt zur Beschreibung des Steuerungsziels. Der Parameter λ, im englischen *trace decay parameter* genannt, entspricht demjenigen im λ-Ertrag (2.44) und wird dem Lernalgorithmus zugeordnet. Über ihn lässt sich definieren, bis zu welchem Grad der Agent nach Besuch eines Zustands s_t zukünftige Erfahrungen an diesen zurück fließen lässt. Dies geschieht durch eine entsprechende Modifikation des Updates von $V(s)$ und wird für diskrete und kontinuierliche Zustandsräume jeweils unterschiedlich gehandhabt, wie in den folgenden Abschnitten beschrieben wird.

Bei (2.48) handelt es sich um *akkumulierende* Traces, die sich dadurch kennzeichnen, dass ein zeitnaher wiederholter Besuch eines Zustands s zu Werten $e_t(s) > 1$ führt. Neben dieser Strategie Traces aufzubauen, gibt es auch andere Vorgehensweisen, wie beispielsweise das Ersetzen eines Trace durch den Wert 1, sobald dessen Zustand s besucht wird:

$$e_t(s) = \begin{cases} 1 & \text{falls } s = s_t, \\ \gamma \lambda e_{t-1}(s) & \text{sonst,} \end{cases} \quad \forall s \in \mathcal{S}. \tag{2.49}$$

Singh und Sutton (1996) diskutieren den Effekt dieser Vorgehensweise theoretisch und an ausgewählten Beispielen. Für die numerischen Ergebnisse dieser Arbeit wurden ausschließlich die akkumulierten Traces aus (2.48) verwendet.

2.6.2 Diskrete Zustandsräume

Mit dem TD-Fehler

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V(s_t), \tag{2.50}$$

den der Agent nach Durchführung der Aktion a_t und Erhalt von r_{t+1} und s_{t+1} (siehe Abbildung 1.1) bestimmen kann, kann für diskrete Zustände die momentane State-Value-Funktion im sogenannten TD(λ)-Algorithmus durch

$$V_{t+1}(s) = V_t(s) + \alpha\delta_t e_t(s) \; \forall s \tag{2.51}$$

explizit für alle Zustände angepasst werden. Den Einsatz der Eligibility Traces charakterisiert hier gerade, dass der Informationsgewinn aus dem TD-Fehler dabei nicht nur zur Anpassung von $V(s_t)$ für den soeben beobachteten Zustand s_t, sondern potentiell für alle möglichen Zustände $s \in \mathcal{S}$ verwendet wird und zwar mit umso größerem Gewicht, je näher der letzte Besuch dieses Zustands in der Vergangenheit liegt. Sutton und Barto (1998) beschreiben, warum sich die vorausschauende und die rückschauende Interpretation von TD(λ)-Updates entsprechen, da sie letztlich die gleichen Änderungen von $V(s)$ bewirken.

2.6.3 Kontinuierliche Zustandsräume

Für kontinuierliche Zustandsräume ist es nicht mehr möglich, jeden Zustand s ein- oder mehrfach zu besuchen, wie es noch für diskrete Zustandsräume möglich ist und für Gleichung (2.51) benötigt wird. Für kontinuierliche Zustände muss der Agent generalisieren, d.h. er muss aufgrund seiner beschränkten Erfahrung Rückschlüsse bezüglich unbekannte Zustände und Aktionen ziehen. Für Algorithmen, die auf die Auswertung und Entwicklung von Value-Funktionen basieren, bedeutet das, dass diese durch eine Approximation dargestellt werden müssen.

Sei $\boldsymbol{\theta}_t$ ein Parametervektor, durch den die Approximation $\tilde{V}$ der State-Value-Funktion V^π bezüglich der Politik π festgelegt ist. Der mittlere quadratische Fehler (MQF) der Approximation $\tilde{V}^\pi(s;\boldsymbol{\theta}_t)$ im Vergleich mit dem wahren Wert $V^\pi(s)$ zum Zustand s ist

$$\mathrm{MQF}(\boldsymbol{\theta}_t) = \sum_s P(s)\left(V^\pi(s) - \tilde{V}(s;\boldsymbol{\theta}_t)\right)^2, \tag{2.52}$$

dabei steht $P(s)$ für die Verteilung der beobachteten Zustände. Die Berücksichtigung einer solchen Verteilung ist wichtig, weil man im Allgemeinen nicht davon ausgehen kann, dass alle Zustände gleich gut besucht werden und auf diese Weise sich auch der Fehlerbeitrag von selten besuchten Zuständen reduziert.

Die übliche Vorgehensweise bei der Minimierung von (2.52) ist die Anwendung eines *Gradientenabstieg-Verfahrens*. Dabei wird bei jeder neuen Beobachtung s_t der Parametervektor $\boldsymbol{\theta}_t$ ein wenig in diejenige Richtung gelenkt, in der sich der Fehler für den gerade beobachteten Zustand s_t am meisten verändert - also in Richtung des negativen Gradienten:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{1}{2}\alpha\nabla_{\boldsymbol{\theta}_t}\left[V^\pi(s_t) - \tilde{V}(s;\boldsymbol{\theta}_t)\right]^2 \tag{2.53}$$

$$= \boldsymbol{\theta}_t + \alpha\left(V^\pi(s_t) - \tilde{V}(s;\boldsymbol{\theta}_t)\right)\nabla_{\boldsymbol{\theta}_t}\tilde{V}(s;\boldsymbol{\theta}_t). \tag{2.54}$$

Im Rahmen des Reinforcement Learning steht die wahre Lösung $V^\pi(s_t)$ nicht zur Verfügung und man ist wie im Falle von diskreten Zustandsräumen auf einen Schätzer angewiesen. Für den TD(λ)-Algorithmus zur Bestimmung einer guten Schätzung $\tilde{V}(s;\boldsymbol{\theta}_t)$ wird dazu der Ausdruck (2.44) verwendet.

Die neben neuronalen Netzwerken am häufigsten genutzten Methode zur Funktionsapproximation ist die *lineare Approximation*

$$\tilde{V}(s;\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t{}^{\mathrm{T}} \boldsymbol{\phi}^{\mathcal{S}}. \tag{2.55}$$

Dies liegt daran, dass ein gefundenes lokales Optimum gleichzeitig auch ein globales Optimum ist. Der so genannte *Featurevektor* $\boldsymbol{\phi}^{\mathcal{S}}$ als Funktion des Zustands s stellt dabei einen Mechanismus zur Generalisierung von Zuständen bereit und sollte so gewählt werden, dass er die wesentlichen Eigenschaften des Zustands erfasst. In Sutton und Barto (1998) wird anhand von Beispielen diskutiert, wie diese Featurevektoren gewählt werden können.

Die Linearität in $\boldsymbol{\theta}_t$ ermöglicht es, den Gradienten in (2.54) sehr einfach zu berechnen:

$$\nabla_{\boldsymbol{\theta}_t} \tilde{V}(s;\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t. \tag{2.56}$$

Auf diese Weise lässt sich der TD(λ)-Algorithmus zur Schätzung von $V^\pi(s)$ mit den folgenden Gleichungen zusammenfassen:

$$\begin{aligned}
\delta_t &= r_{t+1} + \gamma \tilde{V}(s_{t+1};\boldsymbol{\theta}_t) - \tilde{V}(s_t;\boldsymbol{\theta}_t) && \text{(2.57a)}\\
&= r_{t+1} + \boldsymbol{\theta}_t{}^{\mathrm{T}} \left(\gamma \boldsymbol{\phi}^{\mathcal{S}}(s_{t+1}) - \boldsymbol{\phi}^{\mathcal{S}}(s_t)\right), && \text{(2.57b)}\\
\boldsymbol{e}_t &= \gamma\lambda \boldsymbol{e}_{t-1} + \boldsymbol{\theta}_t, && \text{(2.57c)}\\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha\delta_t \boldsymbol{e}_t. && \text{(2.57d)}
\end{aligned}$$

In Form des Vektors $\boldsymbol{e}_t$ werden hier, wie in Abschnitt 2.6.1 beschrieben, die Eligibility Traces verwaltet. Im Unterschied zum Fall von diskreten Zustandsräumen steht jedoch hier jeder Eintrag nicht für genau einen Zustand, sondern für alle Zustände s, die auf denselben Featurevektor $\boldsymbol{\phi}^{\mathcal{S}}$ abgebildet werden.

Es kann gezeigt werden, dass der TD(λ)-Algorithmus für $\gamma < 1$ in Kombination mit einem linearen Funktionsapproximator und bestimmten Voraussetzungen konvergiert (Tsitsiklis 1997), jedoch nicht zwingend zum Parametervektor $\boldsymbol{\theta}^*$, der den mittleren quadratischen Fehler (2.52) minimiert, jedoch zu einem ähnlichen Parametervektor $\boldsymbol{\theta}_\infty$, dessen Fehler beschränkt ist (siehe auch Sutton und Barto 1998):

$$\mathrm{MQF}(\boldsymbol{\theta}_\infty) \leq \frac{1-\gamma\lambda}{1-\gamma}\, \mathrm{MQF}(\boldsymbol{\theta}^*). \tag{2.58}$$

Das Konzept des TD(λ)-Algorithmus soll im Folgenden sowohl für diskrete als auch kontinuierliche Aktionen erweitert werden.

2.7 Der Sarsa(λ)-Algorithmus

Der TD(λ)-Algorithmus behandelt das so genannte *Prediction-Problem*, also die möglichst gute Vorhersage der Zustandswerte $V^\pi(s)$ zu einer gegebenen Politik π. Der Sarsa(λ)-Algorithmus

verwendet dessen Ideen zur Suche einer Approximation der optimalen Action-Value-Funktion Q^*, aus der mit Hilfe von (2.15) eine optimale Politik abgeleitet werden kann.

Er gehört zur Klasse der *On-Policy-Algorithmen*, d.h. der Agent verbessert gerade diejenige Politik, die er auch zur Selektion seiner tatsächlichen Aktionen verwendet. Das entsprechende *Off-Policy*-Pendant ist der Q-Learning-Algorithmus (Watkins und Dayan 1992), bei dessen Anwendung der Agent eine andere Politik verfolgen kann, während er Q^* sucht. Auf diesen Algorithmus wird an dieser Stelle nicht näher eingegangen.

2.7.1 Diskrete Zustandsräume und diskrete Aktionen

Der Sarsa(λ)-Algorithmus für diskrete Zustandsräume lässt sich mathematisch durch das Update

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \;\forall s,a \tag{2.59}$$

mit

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{2.60}$$

und einer Vorschrift

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + \begin{cases} 1 & \text{falls } s = s_t \text{ und } a = a_t, \\ 0 & \text{sonst,} \end{cases} \quad \forall s \in \mathcal{S},\ a \in \mathcal{A}. \tag{2.61}$$

zur Entwicklung der Traces zusammenfassen (Sutton und Barto 1998). Aufgrund der Möglichkeit, die $Q(s,a)$-Werte in einer Tabelle zusammenzufassen, wird einem solches Verfahren das Attribut *tabular* zugeordnet. Nun fehlt noch die Information, wie neue Aktionen selektiert werden, was letztendlich wesentlicher Bestandteil der Politik-Verbesserung ist (siehe Abschnitt 2.4). Hier gibt es verschiedene Strategien.

Charakteristisch für das Reinforcement Learning ist der Widerstreit zwischen Erforschung neuer Aktionen mit der Möglichkeit, bessere Lösungen zu finden und der Versuchung, die bislang gesammelten Kenntnisse soweit wie möglich auszunutzen (*greedy*, also „gierig“). Soll der Agent eher neugierig sein oder gierig? Zwischen beiden Polen ist eine Balance nötig, um gleichzeitig bessere Strategien zu finden und aus dem bisherigen Wissen Profit zu schlagen.

Häufig wird dazu eine ϵ-greedy-Politik genutzt, d.h. mit der Wahrscheinlichkeit ϵ wird eine zufällige Aktion aus $\mathcal{A}(s)$ gewählt und mit der Wahrscheinlichkeit $(1-\epsilon)$ wird auf $\arg\max_a Q(s,a)$ zurückgegriffen. Auf diese Weise kann erreicht werden, dass der Agent auch neue, unbekannte Wege geht, anstatt sich nur an alten Erfahrungen festzuhalten. Eine häufig angewandte Vorgehensweise ist, ϵ im Laufe des Lernprozesses zu dekrementieren, um nach und nach das erworbene Wissen immer stärker auszunutzen.

Eine andere häufig verwendete Möglichkeit ist die *Softmax-Aktionsselektion* oder *Boltzmann exploration strategy*, die eine Boltzmann-Verteilung nutzt. Dabei wird die Aktion a zum Zeitpunkt t mit der folgenden Wahrscheinlichkeit

$$\pi(s,a;Q_t) = \frac{e^{Q_t(s,a)/\tau}}{\sum_{b\in\mathcal{A}(s)} e^{Q_t(s,b)/\tau}} \tag{2.62}$$

ausgewählt. Der Parameter $\tau > 0$ spielt dabei die Rolle einer Temperatur. Im Grenzfall $\tau \to 0$ entspricht diese Form der Aktionsselektion dem Argmax-Operator, d.h. der Auswertung des Ausdrucks $\arg\max_a Q(s,a)$. In den numerischen Lösungen, die in dieser Arbeit verwendet werden, wird stets direkt der Argmax-Operator angewendet, da der Agent durch die Vorbesetzung von Q dazu angehalten wird, unbekannte Aktionen zu probieren (siehe auch Abschnitt 4.7.2).

Sei die verwendete Politik in Abhängigkeit der momentan bekannten Aktionswerte $Q(s,a)$ als $\pi(s,Q)$ bezeichnet. Damit lässt sich das Sarsa(λ)-Verfahren für diskrete Aktionen wie in Algorithmus 2.1 beschrieben, formulieren (Sutton und Barto 1998)[3].

Algorithmus 2.1 (Tabularer) Sarsa(λ)-Algorithmus für diskrete Zustände und Aktionen

$Q(s,a) = 0 \; \forall s,a$
for jede Episode **do**
  $e(s,a) \leftarrow 0 \; \forall s$
  $s \leftarrow s_0$
  $a \leftarrow \pi(s;Q)$
  while (s nicht terminal) $\wedge$ ($t < T_{\max}$) **do**
    Aktion a durchführen
    neuen Zustand s' und Belohnung $r = r(s,a,s')$ erhalten
    $a' \leftarrow \pi(s';Q)$
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    for all s,a **do**
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    end for
    $s \leftarrow s'$
    $a \leftarrow a'$
  end while
end for

2.7.2 Kontinuierliche Zustandsräume durch lineare Approximation

Bei der üblichen Notation für diskrete Aktionen wird gewöhnlich für jede mögliche Aktion $a \in \{\bar{a}_1, \ldots, \bar{a}_m\}$ ein separater Parametervektor $\boldsymbol{\theta}_a$ vorgesehen und die lineare Approximation $\tilde{Q}(s,a)$ der Action-Value-Funktion für den Zeitpunkt t kann als

$$\tilde{Q}(s,a;\boldsymbol{\theta}_{\bar{a}_1,t},\boldsymbol{\theta}_{\bar{a}_2,t},\ldots,\boldsymbol{\theta}_{\bar{a}_m,t}) = {\boldsymbol{\theta}_{a,t}}^{\mathrm{T}}\boldsymbol{\phi}^{\mathcal{S}}(s) \tag{2.63}$$

definiert werden. Die folgende Beschreibung weicht bewusst von dieser Notation ab und repräsentiert das Tupel (s,a) von einem Zustand s und einer Aktion a durch einen gemeinsamen Feature-Vektor $\boldsymbol{\phi}(s,a)$, um in dem später folgenden Abschnitt 2.7.4 auf natürliche

[3] Die Eligibility Traces werden hier abweichend von Sutton und Barto (1998) zu Beginn jeder Episode neu initialisiert (siehe dazu auch Nissen 2007).

Weise eine Erweiterung auf kontinuierliche Aktionen vorzunehmen:

$$\boldsymbol{\phi}(s,a) = \left(\delta_{\bar{a}_1,a} \cdot \boldsymbol{\phi}^{\mathcal{S}}(s), \ldots, \delta_{\bar{a}_m,a} \cdot \boldsymbol{\phi}^{\mathcal{S}}(s)\right)^{\mathrm{T}} \tag{2.64}$$

mit

$$\delta_{\bar{a}_l,a} = \begin{cases} 1 & \text{für } a = \bar{a}_l \\ 0 & \text{sonst} \end{cases} \tag{2.65}$$

für $l = 1, \ldots, m$. Bei Anwendung einer linearen Funktionsapproximation

$$\tilde{Q}(s,a;\boldsymbol{\theta}_t) = \boldsymbol{\theta}_t{}^{\mathrm{T}} \boldsymbol{\phi}(s,a). \tag{2.66}$$

werden dann nur diejenigen Komponenten des Parametervektors $\boldsymbol{\theta}_t$ verwendet, die der Aktion a zugeordnet sind. Für den Gradienten von (2.66) gilt einfach

$$\nabla_{\boldsymbol{\theta}} \tilde{Q} = \boldsymbol{\phi}(s,a) \tag{2.67}$$

und daher kann die Aktualisierung der Gewichte folgendermaßen zusammengefasst werden:

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma \tilde{Q}(s_{t+1}, a_{t+1}; \boldsymbol{\theta}_t) - \tilde{Q}(s_t, a_t; \boldsymbol{\theta}_t) \\ &= r_{t+1} + \boldsymbol{\theta}_t{}^{\mathrm{T}} \left(\gamma \boldsymbol{\phi}(s_{t+1}, a_{t+1}) - \boldsymbol{\phi}(s_t, a_t)\right), && (2.68) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \delta_t \boldsymbol{e}_t, && (2.69) \\ \boldsymbol{e}_{t+1} &= \gamma \lambda \boldsymbol{e}_t + \boldsymbol{\phi}(s_{t+1}, a_{t+1}). && (2.70) \end{aligned}$$

Der Vektor $\boldsymbol{e}_t$ beschreibt die Eligibility-Traces (siehe Abschnitt 2.6.1) und wird wie $\boldsymbol{\theta}_t$ für $t = 0$ mit Nullen initialisiert. Der Algorithmus 2.2 fasst die Vorgehensweise für den Sarsa(λ)-Algorithmus bei linearer Funktionsapproximation zusammen[4]. Die Strategie $\pi(s;\tilde{Q})$ kann dabei auch explorative Elemente enthalten. Ist die Strategie stets *greedy*, so ist

$$\pi(s) = \arg\max_a \tilde{Q}(s,a). \tag{2.71}$$

Solche Aktionen werden im Folgenden auch als *Greedy-Aktionen* bezüglich $\tilde{Q}$ bzw. einfach als *greedy* bezeichnet. Die Tatsache, dass $\pi(s;\tilde{Q})$ auf die Funktion $\tilde{Q}$ zurückgreift, die durch den Lernprozess verändert wird, macht das Sarsa(λ)-Algorithmus zu einem On-Policy-Verfahren.

2.7.3 Anwendung auf das Mountain-Car-Problem mit diskreten Aktionen

Problembeschreibung

Das Mountain-Car-Problem ist ein vielfach verwendetes anschauliches Problem mit zwei kontinuierlichen Zustandsvariablen und einer diskreten Aktionsvariable. Für dieses Problem

[4] Man beachte zu der Angabe der Algorithmen in Sutton und Barto (1998) auch die Liste der Errata auf der Homepage des ersten Autors: `http://www.cs.ualberta.ca/~sutton/book/errata.html`

Algorithmus 2.2 Sarsa(λ) mit linearer Funktionsapproximation und Gradientenabstiegsverfahren

$\boldsymbol{\theta} = 0$
for jede Episode **do**
  $\boldsymbol{e} \leftarrow 0$
  $R \leftarrow 0$
  $s \leftarrow$ Anfangszustand
  $a \leftarrow \pi(s)$
  $k \leftarrow 0$
  while (s nicht terminal) $\wedge$ ($k < T_{\max}$) **do**
    $\boldsymbol{\phi}(s, a)$ berechnen
    $\boldsymbol{e} \leftarrow \boldsymbol{e} + \boldsymbol{\phi}(s, a)$
    Aktion a durchführen
    neuen Zustand s' und Belohnung $r = r(s, a, s')$ erhalten
    $R \leftarrow R + \gamma^k r$
    $\tilde{Q}_{s,a} \leftarrow \tilde{Q}(s, a)$
    if s' terminal **then**
      $\tilde{Q}_{s',a'} \leftarrow 0$
    else
      $a' = \pi(s')$
      $\tilde{Q}_{s',a'} \leftarrow \tilde{Q}(s', a')$
      $s \leftarrow s'$
      $a \leftarrow a'$
    end if
    $\delta \leftarrow r + \gamma \tilde{Q}_{s',a'} - \tilde{Q}_{s,a}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\boldsymbol{e}$
    $\boldsymbol{e} \leftarrow \gamma\lambda\boldsymbol{e}$
    $k \leftarrow k + 1$
  end while
end for

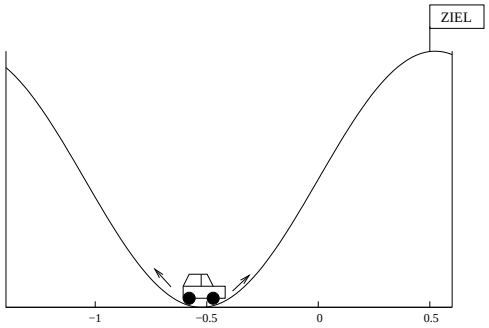


Abbildung 2.1: Das Mountain-Car-Problem. Der Fahrer eines Wagens in einer Mulde möchte sein Fahrzeug über die gekennzeichnete Zielfahne hinaus befördern. Der Motor ist zu schwach, um auf direktem Wege heraus zu fahren. Dem Agenten stehen drei diskrete Aktionen in Form einer Beschleunigung zur Verfügung, die jeweils eine Zeiteinheit lang den Bewegungszustand des Autos ändert: $a \in \{-1, 0, 1\}$. Wird nach einer solchen Aktion das Ziel erreicht, erhält der Agent die „Belohnung“ 0, sonst muss er sich mit -1 begnügen. In Sutton und Barto (1998) wurden für die Position x die Begrenzungen $x_{\min} = -1.4$, $x_{\max} = 0.6$ und für die Geschwindigkeit v entsprechend $v_{\max} = -v_{\min} = 0.07$ gewählt.

lassen sich die Näherungen für die optimale State-Value-Funktion V^*, die durch das Reinforcement Learning gewonnen werden, gut visualisieren. Damit ist es eines der wenigen Standard-Probleme mit kontinuierlichen Zuständen, bei der sich auch der Lernprozess durch Betrachtung von $\max_a \tilde{Q}(s, a)$ beobachten lässt. Als Referenz-System dient meistens jenes von Sutton (1996), welches im Folgenden beschrieben wird.

Die beiden kontinuierlichen Zustandsgrößen sind die Position x und Geschwindigkeit v eines Autos in einer Mulde, das versuchen soll einen bestimmten Punkt mit der Position $x^* = 0.5$ oberhalb der Mulde zu überfahren (siehe Abb. 2.1). Das Problem besteht darin, dass der Motor des Wagens zu schwach ist, um aus eigener Kraft direkt in Richtung Ziel aus der Mulde herauszufahren. Der Agent, der den Wagen steuert, hat zu jedem Zeitpunkt t drei verschiedene Aktionen zur Auswahl:

- vorwärts beschleunigen ($a_t = +1$),
- rückwärts beschleunigen ($a_t = -1$) und
- nichts tun ($a_t = 0$).

Diese Steuerung geht in die diskreten Bewegungsgleichungen des Wagens ein:

$$x_{t+1} = [x_t + v_{t+1}]_x, \tag{2.72a}$$

$$v_{t+1} = [v_t + 0.001 a_t - 0.0025 \cos(3x_t)]_v. \tag{2.72b}$$

Dabei symbolisieren die eckigen Klammern die Beschränkung des Zustandsraums:

$$[x]_x = \begin{cases} x & x_{\min} \leq x \leq x_{\max}, \\ x_{\min} \quad \text{für} & x < x_{\min}, \\ x_{\max} & x > x_{\max} \end{cases} \tag{2.73}$$

und entsprechendes gilt für $[v]_v$. Das System durchläuft eine festgelegte Anzahl von Episoden, die jeweils beendet werden, wenn der Wagen das Ziel erreicht, spätestens aber, wenn der Agent eine vorgegebene maximale Anzahl $T_{\max}$ von Entscheidungen seit Beginn der Episode getroffen hat. Der Wagen beginnt jede Episode stets bei $x = -0.5$, also unten in der Mulde. Ziel ist es nun, eine Strategie zu finden, mit der das Auto in möglichst wenig Schritten aus der Mulde ins Ziel gelangt. Die „Belohnung" für den Agenten ist dabei stets -1 für eine Aktion, die das Ziel nicht erreicht, bzw. 0, wenn das Ziel erreicht wird. Der neue Zustand wird anhand von (2.72) bestimmt.

Generalisierung der Zustände

Zur Verwendung von kontinuierlichen Zustandsvariablen beim Mountain-Car-Problem wird eine Generalisierung in Form eines Feature-Vektors benötigt, mit dessen Hilfe ähnliche Zustände zu Klassen zusammengefasst werden können. Es gibt viele Wege solch einen Feature-Vektor zu berechnen. Eine der bekanntesten Ansätze ist die Verwendung des sogenannten *Tile-Codings* in einem *Cerebellar Model Articulation Controller* (kurz *CMAC*, Albus 1975), wie es u.a. erfolgreich für kontinuierliche Zustände angewandt wurde (Sutton 1996). In der folgenden Anwendung des Sarsa(λ)-Algorithmus wird eine weniger aufwendige Methode verwendet: *radial-basierte* Feature-Vektoren, die aus Gauß-Funktionen zusammengesetzt werden (siehe auch Sutton und Barto 1998). Einen Vergleich von radialbasierten Features mit CMAC für die Funktionsapproximation findet man in Kretchmar und Anderson (1997).

Sei $\hat{n}_x$ die Anzahl der Gauß-Funktionen in x-Richtung und $\hat{n}_v$ die entsprechende Anzahl in v-Richtung. Mit $\sigma_x = \frac{x_{\max} - x_{\min}}{\hat{n}_x}$ und

$$\hat{x}_i = x_{\min} + \left(i - \frac{1}{2}\right)\sigma_x \tag{2.74}$$

und entsprechenden Gleichungen für v erhält man ein gleichförmiges Gitter

$$c_{ij}^{\mathcal{S}} = \left(\hat{x}_i, \hat{v}_j\right) \tag{2.75}$$

mit $i = 1, \ldots, \hat{n}_x$ und $j = 1, \ldots, \hat{n}_v$ im Zustandsraum. Zusammen mit den Größen σ_x und σ_v werden die Komponenten

$$\hat{\phi}_{ij}^{\mathcal{S}}(s) = \exp\left(-\left(\frac{x - \hat{x}_i}{\sigma_x}\right)^2 - \left(\frac{v - \hat{v}_j}{\sigma_v}\right)^2\right) \tag{2.76}$$

gebildet. Durch Aneinanderhängen der Zeilen wird der Vektor $\tilde{\boldsymbol{\phi}}^{\mathcal{S}}(s)$ mit den Komponenten

$$\left(\tilde{\boldsymbol{\phi}}^{\mathcal{S}}(s)\right)_{(i-1)\hat{n}_v + j} = \hat{\phi}_{ij}^{\mathcal{S}}(s) \tag{2.77}$$

mit $i = 1, \ldots, \hat{n}_x$ und $j = 1, \ldots, \hat{n}_v$ gebildet. Dieser wird schließlich normalisiert und man erhält den Feature-Vektor

$$\boldsymbol{\phi}^{\mathcal{S}}_{\mathrm{rb}}(s) = \frac{\tilde{\boldsymbol{\phi}}^{\mathcal{S}}(s)}{\sum_l \tilde{\phi}^{\mathcal{S}}_l(s)}. \tag{2.78}$$

Numerische Lösung

Die Zählung der Episoden beginnt mit Episode 0. Die Abbildung 2.2 zeigt Schnappschüsse des Lernprozesses bei der Anwendung des Sarsa(λ)-Algorithmus unter Verwendung der Parameter $T_{\max} = 1000$, $\alpha = 0.05$, $\lambda = 0.7$ mit den oben beschriebenen Feature-Vektoren mit $\hat{n}_x = \hat{n}_v = 20$. Der Startzustand ist für alle Episoden stets $s_0 = (x_0, v_0) = (-0.5, 0)$, d.h. der Wagen steht zu Beginn ungefähr am tiefsten Punkt der Senke, der sich bei $-\pi/6$ befindet. Die Abbildung 2.2 zeigt die Zeitentwicklung für drei verschiedene Episoden mit den Nummern 800, 2000 und 3200. Nach etwa 3200 Episoden ist das System konvergiert, d.h. jede neue Episode gleicht der vorhergehenden, und es folgt einer Politik, die es ermöglicht, die Mulde innerhalb von 102 Schritten zu verlassen. Diese Politik erhebt nicht den Anspruch insgesamt optimal zu sein, da das System nur einen kleinen Teil der möglichen Punkte im $\mathcal{S} \times \mathcal{A}$-Raum besucht hat. Für eine konkrete Aufgabenstellung wie die gegebene kann eine solche Lösung aber durchaus ausreichend sein. Die dritte Spalte in Abbildung 2.2 macht deutlich, wie sich die Folge der Entscheidungen des Agenten immer weiter verfeinert. Während in der Episode 800 noch viele Beschleunigungswechsel geschehen und häufig die Aktion $a = 0$ gewählt wird, ist das Bild für Episode 3200 viel klarer: Bis auf zwei Wechsel behält der Agent seine Aktion im Wesentlichen bei und das „Nichtstun“ in Form von $a = 0$ wird nicht mehr verwendet. Die zweite Spalte zeigt die Entwicklung von x_t und v_t für die betrachteten Episoden. Deutlich ist zu sehen, wie der Agent immer schneller sein Ziel erreicht und dazu insgesamt immer weniger große Geschwindigkeiten benötigt werden. Die erste Spalte zeigt in Form der Funktion $\max_a \tilde{Q}(x, v, a)$ die Entwicklung einer für das Mountain-Car-Problem typischen spiralförmigen Struktur, die in Form einer Höhenkarte gedacht, den Weg des Mountain-Car vorzeichnet.

Es stellt sich die Frage, wie die erzielten Ergebnisse mit denen verglichen werden können, die mit anderen Methoden erzielt worden sind. Sutton (1996) finden eine Lösung, die ähnlich viele Schritte benötigt, wie die Simulation, die der Abbildung 2.2 zu Grunde liegt, allerdings geht die Anzahl der Schritte die ihre Lösung benötigt, nicht daraus hervor. Der Vergleich mit den Ergebnissen von Kretchmar und Anderson (1997) ist nicht direkt möglich, da sie eine mittlere Schrittweite für Lösungen angeben, die von beliebigen Startzuständen ausgehen. Ein ähnliches Kriterium wird von Smart und Kaelbling (2000) verwendet, allerdings ist die Problembeschreibung eine etwas andere als bei Sutton (1996). Laut Taylor et al. (2008) können die Autoren die bislang besten Ergebnisse für das Mountain-Car-Problem in Verbindung mit dem Sarsa(λ)-Algorithmus erzielen, indem sie eine CMAC-basierte Funktionsapproximation, eine ϵ-greedy-Strategie mit einer Explorationsrate von $\epsilon = 0.1$, die am Ende jeder Episode mit 0.99 multipliziert wird, sowie $\alpha = 0.5$ und $\lambda = 0.95$.

Insgesamt gesehen scheint es sinnvoll, definierte Steuerungsprobleme zu verwenden, die einen Vergleich verschiedener Methoden ermöglichen. Wird dieser Vergleich durch eine bekannte optimale Lösung unterstützt, ist zu hoffen, dass die Wahrscheinlichkeit größer ist,

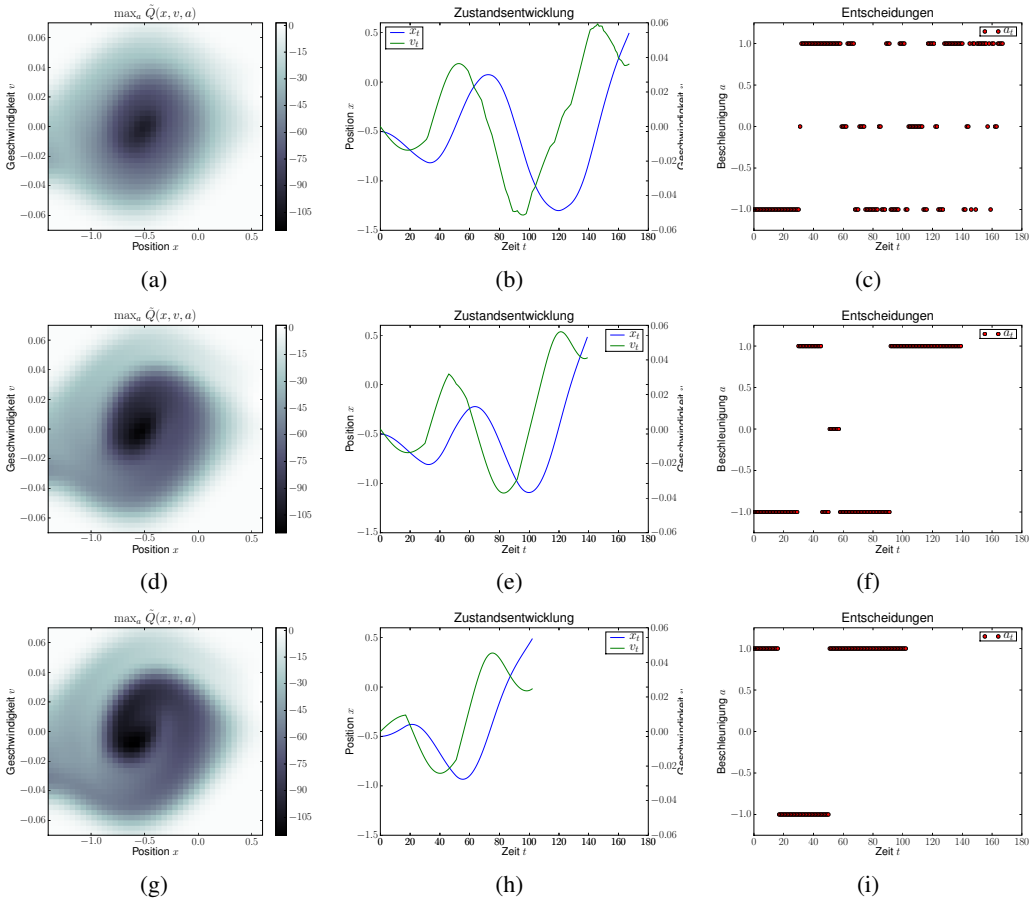


Abbildung 2.2: Lernprozess für das Mountain-Car-Problem. Der Startzustand für alle Episoden ist stets $s_0 = (x_0, v_0) = (-0.5, 0)$. Die Grafiken in der ersten Zeile beziehen sich auf die Episode 800, die Grafiken in der zweiten Zeile auf die Episode 2000 und die Abbildungen in der dritten Zeile schließlich auf die Episode 3200. *Erste Spalte:* Funktion $\max_a \tilde{Q}(x, v, a)$ nach Ablauf der jeweiligen Episode. *Zweite Spalte:* Zustandsentwicklung der Position x_t und der Geschwindigkeit v_t innerhalb der Episode (Skala von v auf rechten Achse). Der Wagen kommt im Laufe der Episoden immer schneller ans Ziel. Zu Beginn des Lernprozesses probiert der Agent zunächst zurück zu fahren, später fährt er zunächst vorwärts. *Dritte Spalte:* Die Entscheidungen des Agenten. Die Entscheidung $a = 0$, also das Abwarten, wird am Ende nicht mehr verwendet, es finden nur noch zwei Beschleunigungswechsel statt. Parameter: Lernrate $\alpha = 0.05$, $\lambda = 0.7$, 20×20 radiale Basisfunktionen für die Zustandsfeatures nach (2.78).

dass in verschiedenen Veröffentlichungen wirklich Systeme mit gleichen Parametern untersucht werden. Aus diesem Grund wird in Kapitel 4 ein Steuerungsproblem mit optimaler Lösung vorgeschlagen, dass schließlich für diskrete und kontinuierliche Aktionen gelöst wird. Als Ergänzung ist im Anhang in Abschnitt B.1 ein weiteres Steuerungsproblem diskutiert, dessen bekannte Lösung mit dem Sarsa(λ)-Algorithmus aus Abschnitt 2.7 gefunden werden konnte.

Bislang wurde nur die Verwendung diskreter Aktionen diskutiert. Im Folgenden wird eine neue Erweiterung beschrieben, die die Verwendung von kontinuierlichen Aktionen möglich macht.

2.7.4 Erweiterung für kontinuierliche Aktionen bei linearer Approximation

Bei der Suche nach einer Steuerung für Probleme mit einer kleinen endlichen Anzahl von Aktionen (*diskrete* Aktionen) besteht die Möglichkeit, einer jeden Aktion $a \in \mathcal{A}$ eine eigene Approximation $\tilde{Q}(s,a) \equiv \tilde{Q}_a(s)$ zuzuordnen. Bei der Verwendung einer linearen Approximation wird dieses häufig implementiert, indem jedem Zustand s ein fester Feature-Vektor $\boldsymbol{\phi}(s)$ zugeordnet wird, welcher unabhängig von a ist. Jede Aktion erhält dabei einen separaten Gewichtsvektor $\boldsymbol{\theta}_a$:

$$\tilde{Q}_a(s) = \boldsymbol{\theta}_a{}^{\mathrm{T}} \boldsymbol{\phi}(s). \tag{2.79}$$

Für eine große Anzahl vom möglichen Aktionen, z.B. wenn die Aktion als Tupel von dicht liegenden reellen Zahlen aus einem Intervall gewählt werden kann (*kontinuierliche* Aktionen), stößt dieser Ansatz an seine Grenzen. Zusätzlich wird die Operation $\max_{a'} Q(s,a')$ sehr aufwendig, wenn sie als lineare Suche über alle möglichen Aktionen realisiert wird, und kann nicht mehr in vernünftiger Zeit durchgeführt werden.

Der in dieser Arbeit verfolgte Ansatz, der mehr einen Baustein für RL-Algorithmen darstellt als eine eigene Methode, verwendet Featurevektoren nicht nur für die Generalisierung von Zuständen sondern auch für die von Aktionen. Zustands- und Aktionsfeatures werden auf geeignete Weise miteinander verknüpft, so dass jede Kombination von Komponenten von Zustandsfeatures und Aktionsfeatures ein separates Gewicht erhält, das durch den Lernprozess angepasst werden kann.

Mit einem kombinierten Feature-Vektor $\boldsymbol{\phi}(s,a)$ kann die lineare Approximation durch

$$\tilde{Q}(s,a) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\phi}(s,a), \tag{2.80}$$

ausgedrückt werden. Dabei wird jedem Gewicht, also einer Komponente von $\boldsymbol{\theta}$, eine Kombination von bestimmten Mengen von Zuständen und Aktionen zugeordnet und es stellt sich die Frage, wie solch ein kombinierter Feature-Vektor $\boldsymbol{\phi}(s,a)$ zusammengesetzt werden kann. Dazu werden hier zwei Möglichkeiten diskutiert.

Sei $\boldsymbol{\phi}^{\mathcal{S}}(s) \in \mathbb{R}^{n_s}$ ein Vektor mit Zustandsfeatures sowie $\boldsymbol{\phi}^{\mathcal{A}}(a) \in \mathbb{R}^{n_a}$ ein Vektor, der die Aktion a charakterisiert. Haben die Vektoren die Komponenten

$$\boldsymbol{\phi}^{\mathcal{S}}(s) = \left(\phi_1^{\mathcal{S}}(s), \ldots, \phi_{n_s}^{\mathcal{S}}(s)\right) \text{ und} \tag{2.81}$$

$$\boldsymbol{\phi}^{\mathcal{A}}(a) = \left(\phi_1^{\mathcal{A}}(a), \ldots, \phi_{n_a}^{\mathcal{A}}(a)\right), \tag{2.82}$$

so besteht eine Möglichkeit der Kombination zu einem Vektor darin, alle Komponenten direkt aneinander zu hängen (*CC*: concatenate):

$$\boldsymbol{\phi}^{\mathrm{CC}}(s,a) = (\boldsymbol{\phi}^{\mathcal{S}}(s), \boldsymbol{\phi}^{\mathcal{A}}(a)) \tag{2.83}$$

$$= \left(\phi_1^{\mathcal{S}}(s), \ldots, \phi_{n_s}^{\mathcal{S}}(s), \phi_1^{\mathcal{A}}(a), \ldots, \phi_{n_a}^{\mathcal{A}}(a)\right). \tag{2.84}$$

Betreffend der linearen Approximation führt dies zu einer Separation in den Gewichten für Zustände und Aktionen. Dies ist im Allgemeinen für eine Schätzung einer Funktion $Q(s,a)$ nicht geeignet, da die Kopplung von Zuständen und Aktionen verloren geht:

$$\tilde{Q}^{\mathrm{CC}}(s,a) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{\phi}^{\mathrm{CC}}(s,a) = \boldsymbol{\theta}_1{}^{\mathrm{T}}\boldsymbol{\phi}^{\mathcal{S}}(s) + \boldsymbol{\theta}_2{}^{\mathrm{T}}\boldsymbol{\phi}^{\mathcal{A}}(a). \tag{2.85}$$

Alternativ kann man ein Matrix-Produkt (*MP*) der Feature-Vektoren bilden, was in einem Gewicht für jede Kombination von Komponenten der Vektoren $\boldsymbol{\phi}^{\mathcal{S}}(s)$ und $\boldsymbol{\phi}^{\mathcal{A}}(a)$ führt (Röttger und Liehr 2009):

$$\phi^{\mathrm{MP}}_{(i-1)n_a+j}(s,a) = \left(\boldsymbol{\phi}^{\mathcal{S}}(s)\left[\boldsymbol{\phi}^{\mathcal{A}}(a)\right]^{\mathrm{T}}\right)_{i,j}, \tag{2.86}$$

wobei

$$i = 1, \ldots, n_s,\ j = 1, \ldots, n_a.$$

Um der Terminologie Feature*vektor* treu zu bleiben, wird dabei die sich ergebende Matrix in einen Vektor transformiert, indem die Reihen miteinander verbunden werden. Diese Art der Verbindung von Zustands- und Aktionsfeatures wird im Rahmen dieser Arbeit verwendet. Von nun an wird das Kürzel $^{\mathrm{MP}}$ der besseren Lesbarkeit wegen vernachlässigt.

Mit einem solchen kombinierten Feature-Vektor kann der bekannte Fall der diskreten Aktionen mit separaten Gewichten für jede Aktion mit Hilfe von (2.80) ausgedrückt werden. Dazu wird der Vektor mit den Aktionsfeatures als

$$\boldsymbol{\phi}^{\mathcal{A}}(a) = (\delta_{a,\bar{a}_1}, \ldots, \delta_{a,\bar{a}_m})^{\mathrm{T}} \tag{2.87}$$

gewählt, mit $a \in \mathcal{A}(s) \subset \mathcal{A} = \{\bar{a}_1, \ldots, \bar{a}_m\}$ und $\delta_{a,\bar{a}_i} = 1$ für $a = \bar{a}_i$ bzw. 0 sonst. Dies ist äquivalent zur Verwendung von binären Features bei denen nur ein Feature präsent ist. Dieses Feature entspricht der Aktion a.

Für kontinuierliche Aktionen können prinzipiell alle Methoden zur Featureberechnung in Betracht gezogen werden, die auch für Zustände denkbar sind. Dabei stellt sich wie bei den Zuständen heraus, dass eine bessere Anpassung der Features an das zu steuernde System auch ein besseres Steuerungsergebnis liefert. Dies wird in den Abschnitten 4.7 und 4.8.2 demonstriert.

Aktionsselektion Bei der Aktionsselektion für kontinuierliche Aktionen kann bei Vorliegen einer Approximation $\tilde{Q}(s,a)$ prinzipiell genauso vorgegangen werden, wie in Abschnitt 2.7.1 beschrieben, z.B. durch Auswertung des Argmax-Operators entsprechend

$$\pi(s) = \arg\max_a \tilde{Q}(s,a). \tag{2.88}$$

Dies ist jedoch im Falle der Funktionsapproximation schwieriger als im Fall einer tabellarischen Repräsentation der Funktion $Q(s,a)$, bei der lediglich alle Funktionswerte $Q(s,a)$ für alle möglichen Aktionen $a \in \mathcal{A}(s)$ miteinander verglichen werden müssen. Für kontinuierliche Aktionen entspricht die Selektion einer Greedy-Aktion gemäß (2.88) einer Suche nach einem *globalen Maximum* und es sollten im Allgemeinen die folgenden Punkte bedacht werden:

1. Standard-Methoden der Numerik suchen häufig nur nach lokalen und nicht nach globalen Extrema,

2. das globale Maximum könnte am Rand von $\mathcal{A}(s)$ liegen, welcher wiederum nicht durch alle Methoden erfasst wird und

3. die Suchroutine arbeitet nur mit einer Approximation $\tilde{Q}(s,a)$ von $Q(s,a)$.

Für die Ergebnisse in dieser Arbeit wurde der folgende Ansatz gewählt: Die ersten beiden Punkte werden berücksichtigt, indem $\tilde{Q}(x,y,\varphi)$ an 100 gleichförmig verteilten Aktionen ausgewertet wird. Im Fall eines eindimensionalen Aktionsraums wie beim Richtungssucher aus Kapitel 4 werden zusätzlich auch die Funktionswerte am Rand bestimmt. Aus den sich so ergebenden Kandidaten wird der beste ausgewählt, um von dort aus schließlich eine Suche nach dem lokalen Maximum zu starten. Diese Suche wurde mit einer Methode namens *Truncated-Newton method using a conjugate-gradient* (TNC) durchgeführt (Dembo und Steihaug 1983). Im allgemeinen Fall von multidimensionalen Aktionen a könnte das Problem der möglichen globalen Maxima auf dem Rand von $\mathcal{A}(s)$ derart behandelt werden, dass die Optimierungsprozedur rekursiv auf Untermengen von $\mathcal{A}(s)$ angewandt wird, wobei die entsprechenden Komponenten of a an den Grenzen der Untermengen festgehalten werden. Dazu ist u.U. eine zusätzliche Transformation nötig.

3 Informationstechnologische Infrastruktur

Die Verwendung von elektronischen Rechenanlagen ist aus den Wissenschaften und insbesondere den Naturwissenschaften nicht mehr wegzudenken. Spätestens mit der kostengünstigen Verfügbarkeit leistungsfähiger Personalcomputer nimmt die rechnergestützte Verarbeitung und Analyse numerischer Daten einen signifikanten Anteil der täglichen Arbeit von Physikern, Chemikern und vielen Mathematikern ein. Dabei liegt das Hauptinteresse des Wissenschaftlers in der wissenschaftlichen Thematik, der Rechner ist dabei Mittel zum Zweck. Die Beschäftigung mit dessen Details sollte, wenn nötig, möglichst langfristig zweckdienlich sein. Dazu gehört auch die Art und Weise, wie wissenschaftliche Daten verarbeitet werden. In diesem Kapitel wird zunächst erläutert, welche Probleme im Umgang mit wissenschaftlichen Daten typisch sind und wie man ihnen konzeptionell begegnen kann. Am Beispiel des Reinforcement Learning wird schließlich eine Umsetzung dieser Konzepte erläutert.

3.1 Konzepte für den Umgang mit wissenschaftlichen Daten

3.1.1 Unerwünschte Kopplungen

Bei der Beobachtung des Umgangs von Wissenschaftlern mit Computern zur Verarbeitung ihrer Daten stellt man fest, dass es immer wieder zu einer wiederholten Auseinandersetzung mit denselben technischen Details kommt, die ermüdend ist und von der eigentlichen wissenschaftlichen Arbeit ablenkt. Wie kommt es dazu? Ein Großteil dieser Probleme entsteht sicher durch die speziellen Fragestellungen, aber auch durch die Komplexität heutiger Betriebssysteme und Softwarepakete, bei denen oft eine maßgeschneiderte Anpassung an die persönlichen Bedürfnisse schwierig oder unmöglich ist, oder aber gesonderte Kenntnisse oder Einarbeitungszeit erfordert. Einige Schwierigkeiten oder Verzögerungen entstehen jedoch auch durch die Art und Weise, wie Daten verarbeitet werden. Häufig sind dabei verschiedene Aspekte unnötig eng aneinander gekoppelt, wodurch sich Abhängigkeiten ergeben. Im Folgenden werden einige typische Kopplungen dieser Art beschrieben.

Daten und ihre physikalische Repräsentation Binäre Dateiformate, die Zahlen in Bitmustern kodieren anstatt als lesbaren Text, haben den Vorteil platzsparend zu sein. In der Vergangenheit wählten viele Wissenschaftler aus diesem Grund den Weg, ihre Simulationsergebnisse in einem undokumentierten Binärformat abzuspeichern, das in der Regel programmiersprachen- und hardwareabhängig ist. Sie können nur erneut verarbeitet werden, wenn man auf gleicher Hardware das gleiche Programm oder zumindest die gleiche

Programmiersprache nutzt bzw. detaillierte Kenntnisse darüber hat, auf welche Art und Weise die Daten kodiert wurden. Eine Weitergabe der Daten ist unter diesen Bedingungen sehr erschwert und jede Änderung der Rechnerumgebung kann Datenverlust zur Folge haben. Einfachste Datentypen, wie beispielsweise ganze Zahlen oder Gleitkommazahlen, können auf einer anderen Hardware eine völlig andere Repräsentation in Form der Bitreihenfolge haben (z.B. *Little* oder *Big Endian*).

Ähnliches gilt für die Speicherung in Datenformaten, die nicht *offen* gelegt, also *proprietär* sind: Ihr Aufbau ist nicht für jedermann einsehbar und unabhängig implementierbar und folgt keinem unabhängigen Standard. In diesem Fall ist man von einer speziellen Software abhängig und gezwungen, neu erscheinende Versionen zu erwerben. Wird dieses versäumt oder ist dies nicht mehr möglich, sind die Daten auf Dauer verloren. Dies geschieht möglicherweise in einer Geschwindigkeit, die mit der Empfehlung der Deutschen Forschungsgemeinschaft (DFG), die Daten zu Veröffentlichungen 10 Jahre lang aufzubewahren um sie reproduzieren zu können, nicht vereinbar ist (DFG 1998). Zudem ist der Austausch mit anderen Wissenschaftlern erschwert, man ist an ein bestimmtes Betriebssystem und möglicherweise deswegen auch an eine bestimmte Hardware gebunden.

Daten und Wissen über Struktur und Umstände Selbst wenn die im vorherigen Abschnitt beschriebene Abhängigkeit nicht oder nur in einem erträglichen Maß besteht, gibt es weitere Fallstricke bei der Datenspeicherung. Oft geht aus dem verwendeten Datenformat das Datenmodell nicht hervor und die problemspezifische Struktur kann nur mit externer Hilfe rekonstruiert werden. Ohne die Struktur sind die Daten wertlos. Das gleiche gilt für wichtige Eingabeparameter oder äußere Bedingungen, die zur Interpretation der Daten notwendig sind. Nicht nur die Daten selbst sind wichtig, sondern auch das Wissen *über* die Daten. In diesem Punkt ist man davon abhängig, dass diese Informationen zusammen mit den Daten weitergegeben werden und bei Bedarf auffindbar sind.

Simulation, Analyse und Visualisierung Gerade um die Frage der Datenspeicherung zu umgehen, wird häufig die Simulation ohne Zwischenschritt direkt mit der Analyse oder Präsentation der Simulationsergebnisse verknüpft. Was schnell und praktikabel für einfache Berechnungen und Analysen mit kurzer Laufzeit ist, kann sich schnell zu einem Hemmschuh für die weitere Arbeit herausstellen. Was ist, wenn eine andere Analyse durchgeführt werden soll, die Berechnung aber zu lange dauert? Oder eine andere Form der Visualisierung der Daten benötigt wird? Eine Vermeidung einer unnötigen Abhängigkeit ermöglicht auch eine Arbeitsteilung im Team oder in der Zusammenarbeit mit anderen Gruppen.

Implementierung von Problemstellungen und Lösungsmethoden Die Mathematik bildet einen Formalismus, der es erlaubt Lösungen für eine ganze Klasse von Problemen zu formulieren, unabhängig davon, in welchem Kontext ein Repräsentant dieser Klasse auftaucht. In ähnlicher Weise sollten auch numerische Verfahren auf elektronischen Rechenanlagen möglichst so implementiert werden, dass sie sich direkt ohne Anpassung auf verschiedene Aufgabenstellungen der gleichen Klasse anwenden lassen.

Auf diese Weise wird es zudem erst möglich, eine Implementierung zunächst an einfachen Aufgaben zu testen, bevor sie auf schwierigere Probleme angewendet wird. Letzteres kann direkt durchgeführt werden, ohne dass sich bei Neuimplementierung Fehler einschleichen können. Auf der anderen Seite kann es nützlich sein, die gleiche Problemstellung mit unterschiedlichen Methoden zu behandeln, z.B. um diese miteinander zu vergleichen. Gerade für komplexe Aufgabenstellungen ist es hilfreich, deren maschinelle Beschreibung in verschiedenen Kontexten wiederverwenden zu können. Eine Entkopplung ist hier durch eine separate Implementierung für die Problembeschreibung, als auch für die Lösungsmethoden möglich, wie im Fall des Reinforcement Learning in Abschnitt 3.2.5 beschrieben ist.

3.1.2 Nachhaltige Datenspeicherung

Wie können solche unerwünschten Kopplungen im Umgang mit wissenschaftlichen Daten möglichst vermieden werden? Ein wichtiges Hilfsmittel dazu ist eine *nachhaltige Datenspeicherung*, welche folgende Charakteristika aufweist:

Plattformunabhängigkeit Es spielt (möglichst) keine Rolle, unter welchem Betriebssystem, mit welcher Programmiersprache oder auf welcher Hardware ein Datensatz erzeugt wurde oder wieder eingelesen werden soll. Durch diese Entkopplung wird Flexibilität und Sicherheit erreicht. Die Datensätze sollten möglichst auch nicht an das Vorhandensein einer bestimmten Anwendung geknüpft werden.

Selbstbeschreibung Die Daten dokumentieren sich insofern selbst, als dass aus ihnen auch die Umstände hervorgehen, unter denen sie erzeugt worden sind, bzw. dass auch scheinbar selbstverständliche Informationen enthalten sind. Diese *Metadaten*, also Informationen über die Daten, können beispielsweise neben sämtlichen Eingabeparametern auch physikalische Einheiten, den Typ der verwendeten Messgeräte oder den Namen des messenden Wissenschaftlers enthalten. Dadurch werden die Daten von den Kenntnissen des Nutzers über Struktur und Umstände der Erzeugung weitgehend entkoppelt.

Erweiterbarkeit Es ist auch zu einem späteren Zeitpunkt möglich, weitere Daten wie z.B. die Ergebnisse einer Analyse, auf natürliche Weise den zuvor gespeicherten Informationen hinzuzufügen oder zumindest zuzuordnen. Dies ist ein wichtiger Punkt für die Automatisierung, da ohne interaktiven Eingriff Beziehungen zwischen Datensätzen genutzt werden können, beispielsweise die Visualisierung von Simulationsergebnissen. Es besteht weiterhin die Möglichkeit, die Struktur zu speichernder Simulationsdaten zu erweitern ohne befürchten zu müssen, dass auch Werkzeuge der Analyse und der Visualisierung angepasst werden müssen.

Alle diese Eigenschaften tragen zur Entkopplung von Simulation bzw. Messung, Analyse und Visualisierung bei, da diese Schritte auf unterschiedlichen Plattformen zu unterschiedlichen Zeitpunkten ohne Rückgriff auf erläuternde Notizen durchgeführt werden können.

Wird zudem die Speicherung unabhängig von den Lösungsalgorithmen und von der Problemstellung definiert, können verschiedene Kombinationen ohne zusätzlichen Aufwand leicht miteinander verglichen werden.

Weiter ergibt sich die Möglichkeit je nach Bedarf einzelne Verarbeitungsschritte wahlweise *interaktiv* oder *automatisiert* durchzuführen. Dies gilt auch für die Generierung von Dokumentation als automatisierten Zwischenschritt, wie z.B. die einer Übersicht über die vorhandenen Daten zusammen mit grundlegenden Analysen.

Eine nachhaltige Speicherung erfordert einen zusätzlichen initialen Aufwand, sie fördert jedoch langfristig die Fokussierung auf die wissenschaftliche Thematik. Ihre Umsetzung für das Reinforcement Learning im Rahmen dieser Arbeit wird in Abschnitt 3.2.1 erläutert.

3.1.3 Interaktiv oder automatisiert?

Wann immer es möglich ist, sollte eine Simulation, eine Analyse oder Visualisierung sowohl *interaktiv* als auch *automatisiert* bzw. im *Batch-Betrieb* ablaufen können.

Die interaktive Arbeit mit dem Rechner ist vor allen Dingen für die Probephase einer neuen Idee geeignet, um bewusst nach Fehlern zu suchen und sicherzugehen, dass das System sich wie gewünscht verhält. Auf Dauer ist eine wiederholte, gleichartige, interaktive Arbeit jedoch unnötig zeitaufwendig, ermüdend und fehleranfällig. So ist z.B. bei der Visualisierung möglicherweise entschieden, wie und was visualisiert werden soll, nur soll dies jetzt für viele unterschiedliche Simulationsläufe geschehen. Dabei möchte man die resultierenden Grafiken direkt miteinander vergleichen ohne darüber nachzudenken, wie sie einzeln erzeugt werden können bzw. ohne diesen Schritt mehrfach bewusst zu wiederholen.

An dieser Stelle ist die Automatisierung eines Vorgangs vorteilhaft: Sie ermöglicht z.B. die Verwendung nicht-lokaler Rechenressourcen über ein Netzwerk, ohne eine ständige Verbindung zu erfordern. Bei nachhaltiger Speicherung der anfallenden Daten fördert sie die Konsistenz und Rückverfolgbarkeit der wissenschaftlichen Arbeit. Zudem ermöglicht die Umsetzung einer automatisierten Datenverarbeitung ohne interaktiven Eingriff mehrerer Komponenten eine weitere Automatisierung auf höheren Abstraktionsstufen. So kann beispielsweise der Wissenschaftler eine Übersicht über seine Daten erhalten, indem für neu anfallende Daten automatisch Analysen durchgeführt werden. Ein Nachteil der Automatisierung ist: Sie suggeriert leicht eine hohe persönliche Produktivität, auch wenn die Umsetzung fehlerhaft sein sollte. Deshalb ist darauf zu achten, sie nicht verfrüht einzusetzen[1]. Hier können automatisierte Testverfahren anhand von bekannten Ergebnissen Abhilfe schaffen, um immer wieder ohne Aufwand die erwartete Funktionalität sicher zu stellen.

Generell kann Folgendes als Richtlinie herangezogen werden: Wird die Arbeit mit numerischen Daten ermüdend oder uninteressant, so steigt die Fehleranfälligkeit und der Wunsch seine Ressourcen effektiver einzusetzen. An einem solchen Punkt ist die Sinnhaftigkeit und Machbarkeit einer Automatisierung zu prüfen, um diese gegebenenfalls umzusetzen.

[1] Oder wie es Knuth (1974), eine der zentralen Persönlichkeiten in der Geschichte der Informatik, bezüglich der Optimierung von Algorithmen bemerkte:

> We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

Der Einsatz einer Komponente, die zugleich interaktiv als auch automatisiert zu nutzen ist, wird in Abschnitt 3.2.2 anhand des Visualisierungwerkzeugs demonstriert, das für die Ansicht der Daten, die beim Ablauf der Lernprozesse entstehen, entwickelt wurde. Zudem wird in Abschnitt 3.2.3 diskutiert, wie sich solche Werkzeuge nutzen lassen, um automatisiert eine Übersicht über laufende und bereits durchgeführte Rechnungen zu generieren.

3.2 Umsetzung für das Reinforcement Learning

3.2.1 Nachhaltige Speicherung

Das Dateiformat HDF5

Das Dateiformat *HDF5* wurde vom *National Center for Supercomputing Applications* (NCSA) der Universität von Illinois, USA, entwickelt. Die Einrichtung weist zwanzig Jahre Erfahrungen auf dem Gebiet der wissenschaftlichen Datenformate auf. Ein wesentliches Ergebnis der Einrichtung ist das *HDF5*-Format, das im Rahmen dieser Arbeit zur Speicherung der Simulationsdaten verwendet wurde. Das Kürzel *HDF* steht dabei für *Hierarchical Data Format*. Das Format bietet dem Wissenschaftler viele Vorteile, von denen hier einige aufgezählt werden sollen:

Es ist *hierarchisch*. Die wesentlichen Elemente im HDF5 sind *Tabellen*, *Felder*, *Attribute* sowie *Gruppen*, die wiederum alle diese Elemente enthalten können. *Tabellen* selbst sind als Menge von n-Tupeln zu sehen, wobei die Elemente dieser Tupel u.a. einfache Zahlen aber auch HDF5-Felder sein können. *Felder* sind Felder im Sinne der Informatik und können beispielsweise Vektoren, Matrizen oder n-dimensionale Tensoren speichern. Die Menge der Dimensionen hat dabei für praktische Anwendungen keine nennenswerte Beschränkung. Die *Attribute* dienen zur Annotation von Datenelementen und können z.B. Metadaten festhalten. Neben den erwähnten Elementen gibt es weitere spezielle Elemente wie z.B. Bilder oder Referenzen, die Redundanzen vermeiden helfen. Auf diese Weise können sehr komplexe wissenschaftliche Datensätze von theoretisch unbegrenzter Größe zusammen mit beschreibenden Metadaten auf effiziente und nachhaltige Weise abgespeichert werden.

Das Format ist *plattformunabhängig*, wobei der Begriff „Plattform“ hier mehrere Deutungen besitzt. Zum Einen bezieht er sich auf die Hardware, auf der die Daten erzeugt wurden bzw. wieder gelesen werden sollen. Desweiteren ist man jedoch auch unabhängig vom Betriebssystem und den verwendeten Programmiersprachen und Bibliotheken. Dies ist darin begründet, dass sowohl das Datenmodell, die Umsetzung in ein konkretes Dateiformat, die Schnittstellen für die Softwareentwickler, ausführliche Dokumentation als auch zusätzliche Werkzeuge für den Umgang mit den Daten im Quellcode offen gelegt und zudem kostenlos zugänglich sind (The HDF Group 2009). Trotz der Unabhängigkeit von der verwendeten Hardware handelt es sich um ein platzsparendes Binärformat das zudem maßgeschneiderte Kompressionsverfahren für einzelne Datensätze nutzt. Der Zugriff auf die Daten ist selektiv und effizient möglich, ebenso eine spätere Erweiterung oder Umstrukturierung der Daten. Zudem ist die Verwendung in verteilten Systemen mit parallelem Zugriff vorgesehen.

Diesen Vorteilen steht die Anfangshürde gegenüber, sich mit den entsprechenden Bibliotheken zum Zugriff auf HDF5-Dateien zu beschäftigen und Zeit in die Planung und

Implementierung der Ein- und Ausgaberoutinen seiner eigenen Daten zu investieren. Dabei ist jedoch zu bedenken, dass dieser Aufwand sich mit hoher Wahrscheinlichkeit schon durch die Nutzung eines Bruchteils der Möglichkeiten, die HDF5 insgesamt bietet, schnell amortisiert.

Im Rahmen dieser Arbeit konnte durch die Verwendung von HDF5 als Speicherungs- und Austauschformat eine konzeptionelle Trennung zwischen Simulation, Analyse und Visualisierung erreicht werden.

Alternativen zum HDF5-Format für den Umgang mit wissenschaftlichen Daten ist z.B. das ähnliche Format *netCDF* (Unidata Community 2009), das in seiner aktuellen Version 4 auf HDF5 basiert, sich gerade für Zeitreihen eignet und einfacher zu verwenden ist als HDF5. Ein weiteres, neues Datenformat ist das am Freiburger Materialforschungszentrum entwickelte *FMF*-Format (Riede et al. 2009), das sich auf kleine tabellarische Datenmengen spezialisiert, sich bei Bedarf auch ohne eine Bibliothek verwenden lässt und die Beschreibung mit physikalischen Einheiten vorsieht.

Verwendung von HDF5 zur Speicherung der Lernprozesse

Möchte man bei der Beschäftigung mit Reinforcement Learning-Algorithmen und entsprechenden Steuerungsaufgaben eine Vergleichbarkeit zwischen verschiedenen Verfahren zur Approximation oder zur Suche nach optimalen Politiken ermöglichen, steht man vor einem Problem: Die entsprechenden Lernprozesse benötigen auf dem Computer möglicherweise unbequem lange Zeit (von Stunden bis zu mehreren Tagen). So kommt es beispielsweise vor, dass Simulationen miteinander verglichen werden sollen, die zu unterschiedlichen Zeitpunkten entstanden sind. Dies ist durch eine nachhaltige und effiziente Speicherung der Simulationsergebnisse möglich, falls diese auf einer gemeinsamen Basis der Lernprozesse beruht.

Ein offensichtlicher Bestandteil einer solchen Basis sind die Episoden mit der Folge

$$s_0 \overset{a_0}{\to} (r_1, s_1) \overset{a_1}{\to} (r_2, s_2) \overset{a_2}{\to} \dots \overset{a_{T-1}}{\to} (r_T, s_T)$$

der Zustände, Aktionen und Belohnungen. Er entspricht der grundsätzlichen Basis aller Reinforcement Learning-Probleme (siehe auch Abbildung 1.1). Damit ist jedoch nur festgehalten, was außerhalb des Agenten passiert.

Zur Speicherung des Wissens des Agenten wird folgender Weg gewählt: Der Nutzer entscheidet sich für eine Menge $\mathcal{S}_{\log}$ von Beispiel-Zuständen bzw. Sample-Zuständen, sowie für eine Menge $\mathcal{A}_{\log}$ von Beispielaktionen bzw. Sample-Aktionen. Für die Simulation, die der Abbildung 4.10 zugrunde liegt, sind das die Mengen

$$\mathcal{S}_{\log} = \left\{ \left(\frac{2i-1}{16}, \frac{2j-1}{16} \right) \middle| \, i, j = 1, \dots, 40 \right\} \tag{3.1a}$$

$$\mathcal{A}_{\log} = \left\{ \frac{2l-1}{16}\pi \middle| \, l = 1, \dots, 16 \right\}. \tag{3.1b}$$

Konkret geschieht die Selektion von $\mathcal{S}_{\log}$ und $\mathcal{A}_{\log}$ für die Simulationen im Rahmen dieser Arbeit durch die Angabe von Auflösungen für jede Dimension von $s \in \mathcal{S}$ und $a \in \mathcal{A}$,

mit denen $\mathcal{S}_{\log}$ und $\mathcal{A}_{\log}$ als gleichmäßige Gitter innerhalb der beschränkten Mengen $\mathcal{S}$ und $\mathcal{A}$ gebildet werden. Im obigen Beispiel also durch die Angaben `40x40` bzw. `16`. Im Allgemeinen werden zu einer Größe $x \in [x_{\min}, x_{\max}]$ die $n_{\log,x}$ Punkte

$$x_{\log,i} = x_{\min} + \frac{2i-1}{2i}\,(x_{\max} - x_{\min}) \quad \text{mit } i = 1, \ldots, n_{\log,x} \tag{3.2}$$

auf dem Gitter gewählt. Mit $n_{\log,x} = n_{\log,y} = 40$, $x_{\max} = y_{\max} = 5$ und $x_{\min} = y_{\min} = 0$ bzw. $n_{\log,\varphi} = 16$, $\varphi_{\max} = 2\pi$ und $\varphi_{\min} = 0$ ergibt sich (3.1). Prinzipiell können problemspezifisch andere Verteilungen von Punkten anstelle der gleichförmigen Gitter ausgewählt werden, falls diese eine bessere Vergleichbarkeit erlauben.

Die Tabelle 3.1 zeigt die oberste Hierachieebene von Datenelementen in der HDF5-Datei bei der Speicherung der Ergebnisse eines Lernprozesses. Die Mengen $\mathcal{S}_{\log}$ und $\mathcal{A}_{\log}$ sind jeweils als Matrizen abgelegt (siehe die ersten beiden Einträge der Tabelle): In den Zeilen stehen die einzelnen Elemente der Menge, über die Spalten erfolgt der Zugriff auf die Dimension. Für das Beispiel in (3.1) wird $\mathcal{S}_{\log}$ durch eine $(1600x2)$-Matrix als zweidimensionales HDF5-Feld und $\mathcal{A}_{\log}$ durch eine $(16x1)$-Matrix als eindimensionales HDF-Feld repräsentiert. Auf diese Weise können Punkte zu beliebigen Zustands- und Aktionsräumen festgelegt werden, sofern sie sich jeweils als Tupel reeller Zahlen darstellen lassen. Die Punkte haben durch die Matrizenform eine festgelegte Reihenfolge, die es möglich macht, die Werte der Datenelemente, die die Value-Funktionen speichern, den jeweiligen Zuständen und Aktionen zuzuordnen.

Mit der Festlegung auf $\mathcal{S}_{\log}$ und $\mathcal{A}_{\log}$ lassen sich nun viele weitere Merkmale des Lernprozesses beschreiben, wie auch die optimale Lösung in Form der optimalen Aktionen $\pi^*(\mathcal{S}_{\log})$ zu jedem Samplezustand, die optimalen Werte der Value-Funktionen $V^*(\mathcal{S}_{\log})$ und $Q^*(\mathcal{S}_{\log}, \mathcal{A}_{\log})$ oder auch die Werte $Q^*(\mathcal{S}_{\log}, \pi^*(\mathcal{S}_{\log}))$ der optimalen Aktionen (siehe dazu auch Tabelle 3.1). Die Speicherung der Start- und Beendigungszeiten einer Simulation erlauben es den Zeitbedarf zukünftiger Simulationen auf der gleichen Hardware abzuschätzen.

Die oberste Hierachieebene enthält zwei weitere wesentliche Elemente, welche vom HDF5-Typ *Tabelle* sind. Das erste, genannt `interactions`, enthält den Ablauf jeder gespeicherten Episode (siehe Tabelle 3.2). Die zweite Tabelle (`summaries`) speichert eine Zusammenfassung jeder Episode in Form einer zum Vergleich geeigneten Repräsentation des Wissen des Agenten am Ende der Episode. Deren Struktur ist in Tabelle 3.3 dargestellt. Die Zusammenfassung enthält als Vektoren und Matrizen die Werte der Approximationen $\tilde{V}$ und $\tilde{Q}$ für die Sample-Zustände und Aktionen aus $\mathcal{S}_{\log}$ bzw. $\mathcal{A}_{\log}$ sowie die sich daraus ergebenden Greedy-Aktionen $\arg\max_a \tilde{Q}(\mathcal{S}_{\log}, a)$, zusammen mit ihren Werten $\max_a \tilde{Q}(\mathcal{S}_{\log}, a)$. Die Zuordnung zu den Punkten aus $\mathcal{S}_{\log}$ bzw. $\mathcal{A}_{\log}$ erfolgt über die Zeilen bzw. Spalten. Durch die gleiche Form der Speicherung, wie sie auch für die optimale Lösung gewählt wurde, ist ein direkter Vergleich mit dieser möglich. Beispielsweise kann für die Samplezustände die Funktion $(V^* - \max_a \tilde{Q}(\cdot, a))$ direkt durch die Differenz der HDF5-Elemente `optimal_state_values` und `summaries.greedy_action_values` der interessanten Episoden ausgewertet werden.

Neben den Daten zu den einzelnen Episoden werden weitere wichtige Metadaten abgelegt, die die Simulation als Ganzes betreffen. Dies ist zum Einen eine Beschreibung des

HDF5-Element	Typ	Symbol	Bemerkung
`sampling_states`	F	$\mathcal{S}_{\log}$	Sample-Zustände für die Ausgabe
`sampling_actions`	F	$\mathcal{A}_{\log}$	Sample-Aktionen für die Ausgabe
`optimal_actions`	F	$\pi^*(\mathcal{S}_{\log})$	Auswertung einer optimalen Politik
`optimal_state_values`	F	$V^*(\mathcal{S}_{\log})$	
`optimal_action_values`	F	$Q^*(\mathcal{S}_{\log}, \mathcal{A}_{\log})$	
`optimal_action_values_for_optimal_actions`	F	$Q^*(\mathcal{S}_{\log}, \pi^*(\mathcal{S}_{\log}))$	
`final_theta_Q`	F	$\boldsymbol{\theta}_Q^{\mathrm{f}}$	Parameter für $\tilde{Q}(s, a; \boldsymbol{\theta}_Q)$ am Ende des Lernprozesses
`final_theta_V`	F	$\boldsymbol{\theta}_V^{\mathrm{f}}$	Parameter für $\tilde{V}(s; \boldsymbol{\theta}_V)$ am Ende des Lernprozesses
`initial_theta_Q`	F	$\boldsymbol{\theta}_Q^{\mathrm{i}}$	Parameter für $\tilde{Q}(s, a; \boldsymbol{\theta}_Q)$ zu Beginn des Lernprozesses
`initial_theta_V`	F	$\boldsymbol{\theta}_V^{\mathrm{i}}$	Parameter für $\tilde{V}(s; \boldsymbol{\theta}_V)$ zu Beginn des Lernprozesses
`interactions`	T		siehe Tabelle 3.2
`summaries`	T		siehe Tabelle 3.3
`purpose`	A		Absicht der Simulation
`description`	A		Beschreibung Steuerungsproblem, Algorithmus, Eingabeparameter
`timestamp_logging_start`	A		Zeitpunkt des Beginns der Simulation
`timestamp_logging_end`	A		Zeitpunkt des Simulationsendes

Tabelle 3.1: Inhalt der HDF5-Datei. Unabhängig vom Steuerungsproblem, von der Dimensionalität von Zustands- und Aktionsräumen sowie von dem verwendeten Verfahren wird dieselbe Struktur verwendet. Typen der HDF-Elemente: Tabelle (T), Feld (F) und Attribut (A). Die Symbole stellen den Bezug der HDF5-Elemente zur mathematischen Notation her.

Spaltenname	Symbol	Bemerkung
`episode`	e	Episodennummer
`time`	t^e	Zeitpunkt innerhalb der Episode e
`state`	s_t^e	Zustand zum Zeitpunkt t in Episode e
`action`	a_t^e	Aktion im Zustand s_t^e
`next_state`	s_{t+1}^e	Folgezustand nach Aktion a_t^e
`reward`	r_{t+1}^e	Belohnung nach Aktion a_t^e
`total_reward`	R_t^e	Gesamtbelohnung nach a_t^e
`state_values`	$\tilde{V}(\mathcal{S}_{\log})$	(optional)
`action_values`	$\tilde{Q}(\mathcal{S}_{\log}, \mathcal{A}_{\log})$	(optional)
`greedy_actions`	$\arg\max_a \tilde{Q}(\mathcal{S}_{\log}, a)$	(optional)
`greedy_action_values`	$\max_a \tilde{Q}(\mathcal{S}_{\log}, a)$	(optional)

Tabelle 3.2: Inhalt der HDF5-Tabelle `interactions` mit dem Ablauf der in der HDF5-Datei gespeicherten Episoden. Die als optional gekennzeichneten Elemente speichern Werte der Value-Funktionen für jeden Zeitpunkt jeder Episode. Sie benötigen enorm viel Zeit und Platz zur Speicherung und werden daher nur bei Bedarf eingesetzt. Zur Hierarchie der gespeicherten Daten siehe auch Tabelle 3.1.

Spaltenname	Symbol	Bemerkung
`episode`	e	Episodennummer
`initial_state`	s_0^e	Anfangszustand der Episode e
`end_time`	T^e	letzter Zeitpunkt in Episode e
`total_reward`	$R_{T^e}^e$	(diskontierte) Gesamtbelohnung am Ende der Episode e
`exp_opt_reward`	$V^*(s_0^e)$	erwartete diskontierte Gesamtbelohnung
`state_values`	$\tilde{V}(\mathcal{S}_{\log})$	
`action_values`	$\tilde{Q}(\mathcal{S}_{\log}, \mathcal{A}_{\log})$	
`greedy_actions`	$\arg\max_a \tilde{Q}(\mathcal{S}_{\log}, a)$	
`greedy_action_values`	$\max_a \tilde{Q}(\mathcal{S}_{\log}, a)$	

Tabelle 3.3: Inhalt der HDF5-Tabelle `summaries` mit einer Zusammenfassung der in der HDF5-Datei gespeicherten Episoden. Die Mengen $\mathcal{S}_{\log}$ und $\mathcal{A}_{\log}$ mit Samplepunkten im Zustands- und Aktionsraum sind ebenfalls in der HDF-Datei definiert. Die Größen $R_{T^e}^e$ und $V^*(s_0^e)$ können direkt zur Berechnung des Umwegskoeffizienten eingesetzt werden (siehe Abschnitt 4.6). Zur Hierarchie der gespeicherten Daten siehe auch Tabelle 3.1.

zu lösenden Steuerproblems, die automatisch aus dem definierenden Quellcode extrahiert wird, zusammen mit den Eingabeparametern und den Optionen, mit denen der Simulationslauf gestartet wurde. Zum Anderen besteht die Möglichkeit für den Nutzer seine *Absicht* beim Start der Berechnungen zusammen mit den Ergebnissen in Textform abzulegen (HDF5-Element `purpose`). Dadurch besteht auch nach längerer Zeit die Möglichkeit, die Daten in einen größeren Kontext einzuordnen oder beim Erzeugen automatischer Übersichten Zusatzinformationen anzubieten (siehe dazu auch Abschnitt 3.2.3).

Die Implementierung der Speicherung der Daten ist bewusst sowohl von der Implementierung der Lernalgorithmen und der verschiedenen Verfahren zur Featureberechnung, als auch von der Beschreibung der Steuerungsaufgaben getrennt. Durch diese Entkopplung wird erreicht, dass sich die Ergebnisse von Lernprozessen zu verschiedenen Algorithmen oder zu verschiedenen Steuerungsproblemen relativ einfach miteinander vergleichen lassen.

Automatische Wahl eindeutiger Simulationsnamen

Werden während der wissenschaftlichen Arbeit über einen längeren Zeitraum eine Vielzahl von verschiedenen Simulationen, Messungen oder numerische Analysen durchgeführt, so ist es erforderlich, diese eindeutig zu benennen um in Notizen, Dokumenten oder nachfolgenden Daten auf die Datenquelle verweisen zu können. Häufig wird die Wahl dieser Namen der Fantasie des ausführenden Wissenschaftlers überlassen – dabei ist es üblich, Metadaten in Dateinamen zu kodieren[2]. Die manuelle Wahl von Dateinamen findet dort seine Grenzen, wo Berechnungen und Analysen automatisiert werden sollen und hat außerdem den Nachteil, auf die Disziplin des Namensgebers angewiesen zu sein. Folgen diese Namen nicht einem festgelegten Schema, fällt möglicherweise die Zuordnung später schwer, vor allen Dingen wenn ein neuer Mitarbeiter innerhalb einer Arbeitsgruppe die Daten zum ersten Mal einsieht. Zudem kann es sein, dass Berechnungen wiederholt werden müssen und der Vergleich mit einer alten Version weiterhin erwünscht ist, jedoch keine Daten überschrieben werden sollen. Erschwerend kommt hinzu, dass im Laufe der Zeit Parameter hinzukommen können, die zunächst noch konstant waren. Besteht der Anspruch, alle notwendigen Metadaten in den Dateinamen zu kodieren, so ist dies mitunter sehr aufwendig und fehleranfällig.

Diese Problematik lässt sich leicht umgehen, indem man *eindeutige*, *automatisch generierte* Namen für Simulationen, Analysen oder Messungen verwendet. Die Eindeutigkeit lässt sich auf verschiedene Arten erreichen. Ein möglicher Weg ist die Wahl einer laufenden Nummer, geschrieben mit führenden Nullen mit einer festen Anzahl von Ziffern. Dieser Ansatz ist geeignet, solange noch eine manuelle Kontrolle oder eine anderweitige Sicherheit darüber besteht, dass niemals zwei Berechnungen gleichzeitig gestartet werden, z.B. weil eine zentrale Instanz existiert, die die Nummern exklusiv vergibt. Ist eine manuelle Wahl erforderlich, widerspricht dies der Idee der Automatisierung und ist fehleranfällig. Andere Mechanismen erfordern u.U. einen zusätzlichen Aufwand, z.B. die Anfrage einer zentralen Instanz über das Netzwerk oder die Untersuchung bereits vorhandener Dateinamen, sofern ein gemeinsamer Datenbereich existiert. Ein Vorteil einer Eindeutigkeit durch laufende Nummern ist, dass dies eine knappe Notation ermöglicht.

[2] Es ist generell ratsam, die Metadaten *in* Dateien zu speichern und zusammen mit den Primärdaten abzulegen (wie z.B. beim HDF5-Format in derselben Datei möglich, siehe Abschnitt 3.2.1).

Im Rahmen der Arbeit mit Reinforcement Learning und anderer Projekte am Freiburger Materialforschungszentrum (vgl. Kühne und Liehr 2009) hat sich folgendes Schema als geeignet erwiesen um eindeutige Namen durch Simulations- oder Analyseprogramme automatisch zu generieren:

JJJJMMTT`-`*hhmmss*`-`*Rechnername*`[-`*Stichwort*`]`*

Dabei bezeichnet *JJJJMMTT* das aktuelle Datum, *hhmmss* die momentane Zeit und es können beliebig viele Stichworte mit einem führenden Bindestrich angefügt werden, um Metadaten sichtbar zu machen oder eine schnelle Zuordnung zu ermöglichen. Die Angabe des Rechnernamens verhindert bei Nutzung vieler Ressourcen die Verwendung gleicher Namen zum gleichen Zeitpunkt. So trägt beispielsweise die Simulation, die der Abbildung 4.9 zu Grunde liegt, den Namen

`20090125-195507-neptun-pathplanner`.

Das Kürzel `pathplanner` wurde systemintern als Stichwort für den Richtungssucher verwendet.

Für den Rechnernamen sowie die Stichworte sollte möglichst nur ein eingeschränkter Zeichensatz (7-Bit-ASCII) ohne Umlaute und ohne Leerzeichen verwendet werden. Dies verhindert erfahrungsgemäß Probleme im Umgang mit verschiedenen Datei- und Betriebssystemen und Programmiersprachen. Innerhalb der Daten selbst können auch Metadaten problemlos mit erweitertem Zeichensatz verwendet werden, sofern dies durch das verwendete Datenformat abgesichert ist.

Ein Simulationsname kann bei der Arbeit mit den Daten an verschiedensten Stellen eingesetzt werden, um die Daten zu referenzieren. So werden die während der Simulation entstehenden HDF5-Dateien gemäß dem Schema

Simulationsname`.h5`

benannt. Entsteht während der Arbeit die Notwendigkeit zeitweise alternative Namen für Simulationen zu verwenden, so ist zu prüfen, inwieweit sich Mechanismen des verwendeten Dateisystems nutzen lassen um so genannte *Links* oder *Verknüpfungen* anzulegen. Dies ist sinnvoller, als durch eine Kopie Daten zu duplizieren.

Dateien, die bei weiterführenden Analysen entstehen, wie zum Beispiel andere HDF5-Dateien oder Grafiken lassen sich durch Anhängen weiterer Stichwörter bzw. passender Dateiendungen eindeutig zuordnen. So ist z.B. die Grafik einer Standard-Perspektive der Ergebnisse der letzten Episode zur Simulation mit dem Namen `20090125-195507-neptun-pathplanner` als

`20090125-195507-neptun-pathplanner-last-episode.png`

gespeichert.

3.2.2 Flexible Visualisierung

Um eine Trennung der Visualisierung von der Simulation zu erreichen wurde im Rahmen der Arbeit ein dediziertes Werkzeug (`rl-view`) entwickelt, um Simulationsergebnisse sichtbar zu machen. Es arbeitet sowohl interaktiv, als auch ohne Bedienung und Ansicht durch den Benutzer.

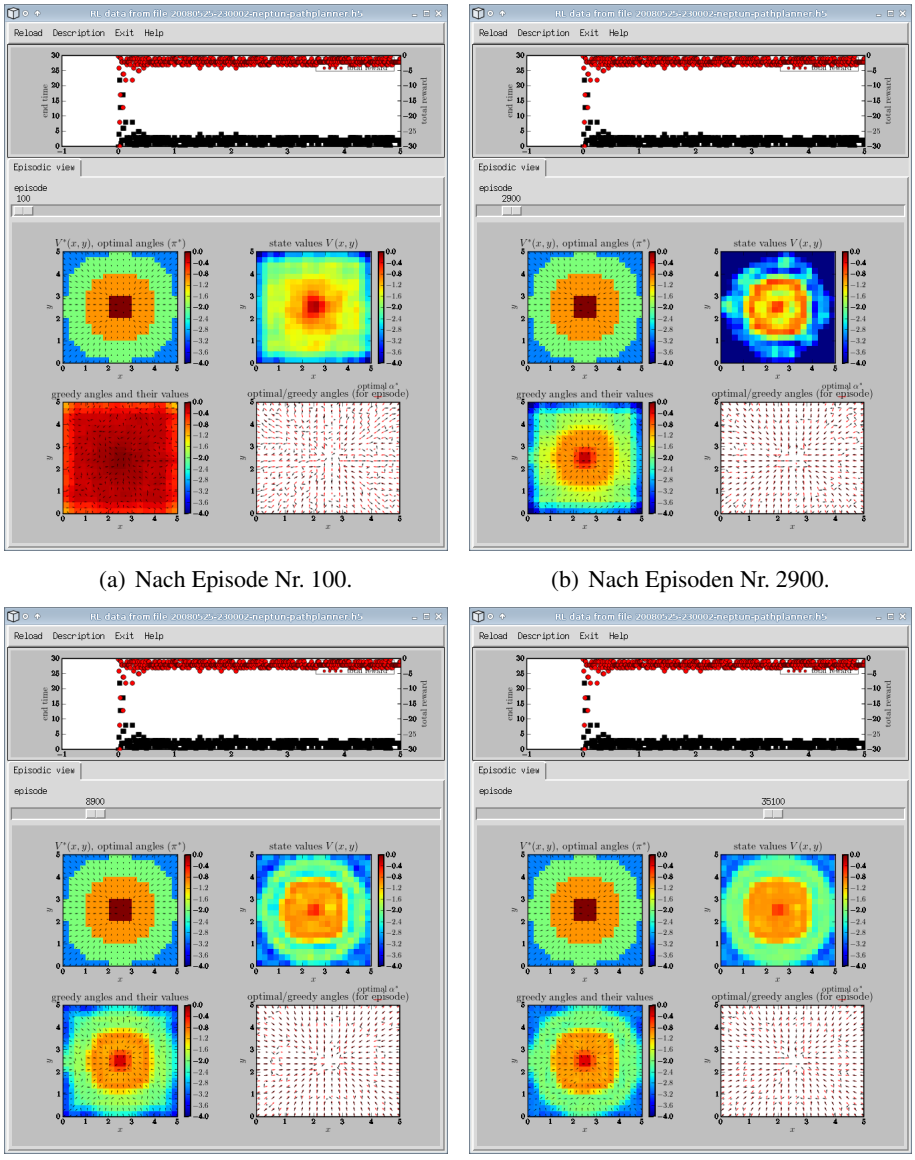
Im *interaktiven* Modus bietet es dem Nutzer eine Übersicht über R^e und T^e für alle gespeicherten Episoden und ermöglicht es, sich einen visuellen Eindruck über den Lernprozess zu machen, indem verschiedene Episoden gezielt angewählt werden (vgl. Abbildung 3.1). Darüber hinaus besteht die Möglichkeit, Ausschnitte einzelner Grafiken zu betrachten und in eine Datei zu exportieren. Hervorzuheben ist an dieser Stelle, dass durch die HDF5-Eigenschaft der parallelen Ein- und Ausgabe möglich ist, einen Einblick in *laufende* Simulationen zu bekommen, ohne diese zu beeinflussen. Damit kann eine Rechnung bei Bedarf frühzeitig abgebrochen werden, um eine alternative Simulation zu starten. Oft werden Simulation und Visualisierung miteinander verschränkt, um diese Funktionalität zu erhalten, dies ist jedoch bei der Verwendung eines Datenformats wie HDF5 nicht nötig. Zudem ist es möglich, die Simulation auf einem anderen Rechner im Netzwerk zu betreiben als auf dem Rechner, auf dem das Visualisierungswerkzeug `rl-view` gestartet wurde.

Unterbindet man beim Start des Werkzeugs `rl-view` den interaktiven Modus, so lassen sich *automatisiert* Bilder und Bilderserien erzeugen. Dies ist nützlich, um eine Übersicht über viele Simulationen zu erhalten und Prozesse sichtbar zu machen (wie z.B. in der Abbildung 2.2). Dabei können die Bilder auch automatisch mit dem Simulationsnamen (siehe Abschnitt 3.2.1) und der Nummer der Episode, auf die sich das Bild bezieht, beschriftet werden. Hierdurch wird eine spätere Zuordnung von Grafiken erleichtert, die manuell durchgeführt zeitaufwendig und fehleranfällig wäre.

Neben der Automatisierung bietet das Werkzeug dem Nutzer eine Übersicht über Parameter des Lernprozesses und des Steuerungsproblems. Darüber hinaus ist das Werkzeug so konstruiert, dass es unabhängig vom Steuerungsproblem ist und praktisch unabhängig von der Struktur der HDF5-Elemente, die während der Simulation geschrieben werden. Hier wird lediglich das Vorhandensein einer bestimmten HDF5-Gruppe, sowie das der Daten für R^e und T^e vorausgesetzt. Die tatsächlich generierte Ansicht stützt sich auf die maschinelle Beschreibung einer *Perspektive* der Daten in Form einer Zuordnung von grafischen Elementen, wie z.B. Funktionsgraphen und Kontur-Plots, zu HDF5-Elementen zusammen mit Achsenbeschriftungen. Damit können die Daten auf verschiedene Art und Weise betrachtet werden ohne den Quellcode des Visualisierungswerkzeugs zu ändern. Dabei bietet es sich an, zunächst mit Hilfe des interaktiven Modus die Perspektive zu testen, bevor sie automatisiert genutzt wird um mehrere Grafiken zu erzeugen. Die Abbildung 3.2 zeigt ein Beispiel für eine derartige Beschreibung zusammen mit den erzeugten Grafiken für die Simulation, die der Abbildung 2.2 zu Grunde liegt.

3.2.3 Automatisch generierte Übersicht über Lernprozesse

Während der wissenschaftlichen Nutzung von Rechenanlagen trifft man häufig auf Fragestellungen, für die es gilt verschiedene Konzepte, Parametersätze oder auch Verfahren nu-



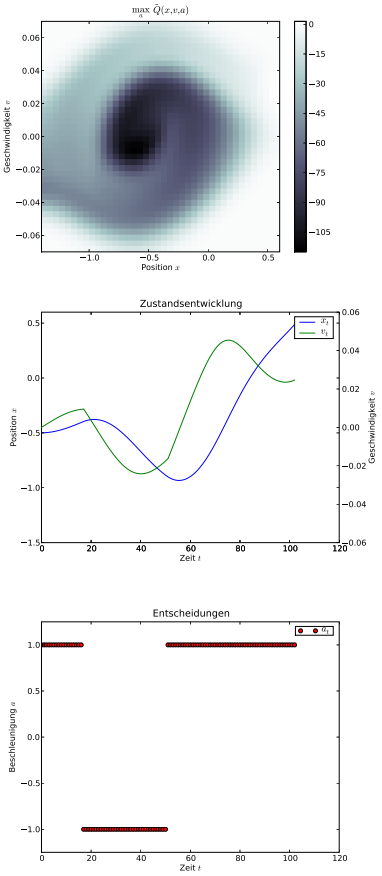
(a) Nach Episode Nr. 100.

(b) Nach Episoden Nr. 2900.

(c) Nach Episode Nr. 8900.

(d) Nach Episode Nr. 35100.

Abbildung 3.1: Interaktive Verwendung des Visualisierungswerkzeugs zum Vergleich verschiedener Stadien des Lernprozesses. Über einen Schieberegler bzw. über die Tastatur kann die zu untersuchende Episode eingestellt werden. Der Plot oberhalb des Schiebereglers zeigt jeweils die Gesamtbelohnung R_e und den Endzeitpunkt T_e aller Episoden. Abhängig von der eingestellten Episode sind drei der vier Darstellungen unterhalb des Reglers: So zeigt jeweils die erste Spalte die Funktionen $V^*(x,y)$ und darunter $\max_\varphi \tilde{Q}(x,y,\varphi)$. Die zweite Funktion wird der ersten im Laufe des Lernprozesses ähnlicher.




```
-
 type:  image plot
 title: $\max_a\, \tilde{Q}(x,v,a)$
 xlabel: Position $x$
 xypoints: sampling_states
 ylabel: Geschwindigkeit $v$
 Zdata:  summaries.greedy_action_values{episode}
 Zmin: -115
-
  type:  twin line plot
  title: Zustandsentwicklung
  xdata:  interactions.time{episode}
  xlabel: Zeit $t$
  ydata:
   locator: interactions.state{episode}[:,0]
   label: $x_t$
   style: b-
  y2data:
   locator: interactions.state{episode}[:,1]
   label: $v_t$
   style: g-
  ylabel: Position $x$
  y2label: Geschwindigkeit $v$
  ymin: -1.5
  ymax: 0.6
  y2min: -0.06
  y2max: 0.06
  xmin: 0
  xmax: 120
-
  type:  line plot
  title: Entscheidungen
  xdata:  interactions.time{episode}
  xlabel: Zeit $t$
  ydata:
   locator: interactions.action{episode}
   label: $a_t$
   style: ro
  ylabel: Beschleunigung $a$
  ymin: -1.25
  ymax: 1.25
  xmin: 0
  xmax: 120
```

Abbildung 3.2: Beispiel für die Beschreibung einer Perspektive auf eine Mountain Car-Simulation zusammen mit den erzeugten Grafiken. Die Daten sind der Episode 20000 entnommen, welche praktisch identisch zur Episode 3200 ist. Für jede Grafik ist eine Reihe von Visualisierungsparametern angegeben, wie z.B. `title` für die Überschrift der Grafik oder `ydata` für ein HDF5-Element, aus denen die Daten gelesen werden, die der y-Achse zuzuordnen sind. Für die Zuordnung zu den HDF5-Elementen wird die Notation `<`*`Tabelle`*`>.<`*`Spalte`*`>{episode}[<`*`Feld-Indizes`*`>]` verwendet. Der Ausdruck `{episode}` selektiert dabei diejenigen Tabellenzeilen, die zur ausgewählten Episode gehören, z.B. interaktiv, wie in Abbildung 3.1 demonstriert. Zusammen mit den Tabellen 3.1, 3.2 und 3.3 kann die Zuordnung zu den HDF5-Elementen hergestellt werden. Zur Syntax: Es wird die Sprache *Yet Ain't Markup Language* (YAML) verwendet, mit der sich in einer für den Menschen leicht lesbaren Form kleine Datensätze mit komplexer Struktur plattformunabhängig beschreiben und serialisieren lassen (Evans 2009).

merisch miteinander zu vergleichen. Sind dabei die einzelnen Simulationsläufe oder Analysen sehr zeitaufwendig, hemmt dies eine interaktive Durchführung der wissenschaftlichen Arbeit. Man wird daher, wann immer möglich, parallel mehrere verteilte Rechenanlagen im Batchbetrieb nutzen.

An dieser Stelle sieht sich der Wissenschaftler mit dem Problem konfrontiert, die Übersicht über die laufenden Rechnungen zu bewahren und zu kontrollieren, welche Ideen schon realisiert wurden um Wiederholungen zu vermeiden. Dies trifft auch für die Arbeit mit unterschiedlichen Reinforcement Learning-Algorithmen, -Parametern und -Steuerungsproblemen zu. Diese bringen aufgrund der Komplexität des Reinforcement Learning-Problems und dem Anspruch, dieses möglichst ohne Einsatz der Kenntnis der Systemdynamik zu lösen, mitunter lange Wartezeiten mit sich. Eine interaktive Arbeit kann dadurch behindert werden.

Im Rahmen dieser Arbeit wurde zur Lösung dieses Umstands ein System entwickelt (`rl-html`), das die Ergebnisse bereits durchgeführter Simulationen automatisch voranalysiert. Der Zugriff auf diese Ergebnisse erfolgt über eine Übersicht, die den Nutzer darüber informiert, ob die Ergebnisse bereits vorliegen oder noch nicht. Es basiert darauf, dass die Daten aller an der Rechnung beteiligten Maschinen ihre Ergebnisdaten in Form von HDF5-Dateien über das Netzwerk an einer zentralen Stelle ablegen. Das System greift im Minutentakt auf alle vorhandenen HDF5-Dateien zu und analysiert diese. Sofern noch nicht vorhanden, wird zu jeder abgeschlossenen Simulation eine Übersichtsseite erstellt, die über einen HTML-Browser betrachtet werden kann. Die Abbildung 3.3 gibt einen Eindruck einer solchen Seite.

Zusätzlich wird eine Gesamtübersicht generiert, die zu jeder HDF5-Datei einen Eintrag enthält, wobei die Sortierung gemäß der Stichworte im Dateinamen erfolgt (siehe Abschnitt 3.2.1). Jeder Eintrag besteht aus dem eindeutigen Simulationsnamen als Link auf die Übersichtsseite zu dieser Simulation, sofern diese bereits fertig gestellt ist. Daneben ist die Absicht notiert, die mit dem Start der Simulation verknüpft war (HDF5-Element `purpose`, siehe Tabelle 3.1). Diese Gesamtübersicht erlaubt also einen Zugriff auf Details einer Simulation und stellt gleichzeitig eine Erinnerungshilfe für die zu Grunde liegenden Fragestellungen dar.

Dieses Konzept lässt sich prinzipiell auch auf andere Situationen der wissenschaftlichen Informationsverarbeitung übertragen, beispielsweise wenn in einer Arbeitsgruppe regelmäßig Messdaten aus Experimenten gespeichert werden. Der Ansatz diese Messungen automatisch einer Voranalyse zu unterziehen und an einer zentraler Stelle übersichtlich zu präsentieren, kann die Zusammenarbeit in der Gruppe unterstützen und zu einem früheren Zeitpunkt erworbene Ergebnisse dauerhaft und personenunabhängig verfügbar machen.

3.2.4 Bewertung der Simulationsdaten

Durch die Speicherung der optimalen Lösung, gemeinsam mit den Simulationsdaten, ist es mittels einem Analysewerkzeuge möglich, parallel auf beides zuzugreifen. Das Werkzeug `rl-perf` ermöglicht es beispielsweise, den Umwegskoeffizienten, wie er in Abschnitt 4.6 definiert ist, für eine gegebene Simulation zu berechnen. Ein anderes Werkzeug `rl-extract` berechnet Distanzfunktionen zwischen verschiedenen Value-Funktionen. Als Distanzfunk-

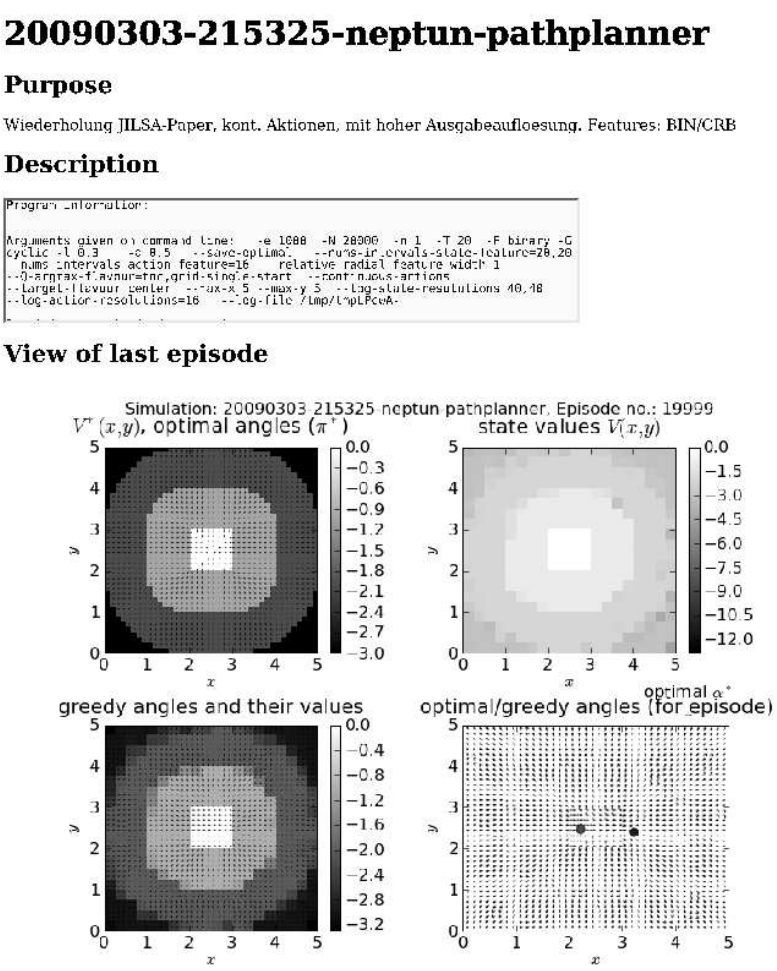


Abbildung 3.3: Beispiel für den Beginn einer automatisch generierten Übersichtsseite (HTML) aus einer Simulationsdatei. Neben einer Notiz zur Absicht (*purpose*), die mit dem Start der Simulation erstellt wird, ist eine Beschreibung des Programmaufrufs sowie eine Grafik mit einer Standardperspektive für Simulationen zum Richtungssucherproblem zu sehen. Unterhalb des Bildes ist ein Pfad zur YAML-Datei notiert, die zur Beschreibung der Perspektive herangezogen wurde (siehe auch die Bemerkung zu Tabelle 4.1). Eine solche Übersicht kann auch in anderen Situationen im Umgang mit wissenschaftlichen Daten nützlich sein, z.B. bei der zentralen Sammlung von Messdaten in einer experimentell arbeitenden Gruppe.

tionen stehen

$$d_1(f,g) = \sum_{x \in X} |f(x) - g(x)|, \tag{3.3}$$

$$d_2(f,g) = \sqrt{\sum_{x \in X} (f(x) - g(x))^2}, \tag{3.4}$$

$$d_3(f,g) = \max_{x \in X} |f(x) - g(x)| \text{ und} \tag{3.5}$$

$$d_4(f,g) = \sqrt{\frac{1}{|X|} \sum_{x \in X} (f(x) - g(x))^2} \tag{3.6}$$

zur Verfügung, die u.a. auf die an den Samplepunkten aus $\mathcal{S}_{\log} \times \mathcal{A}_{\log}$ ausgewerteten Valuefunktionen V^*,Q^*,$\tilde{V}$,$\tilde{Q}$ oder $\max_a \tilde{Q}(\cdot, a)$ angewendet werden können. Die Menge X steht dabei wahlweise für $\mathcal{S}_{\log}$, $\mathcal{A}_{\log}$ oder $\mathcal{S}_{\log} \times \mathcal{A}_{\log}$.

3.2.5 Generische Schnittstelle für Reinforcement Learning-Probleme

Durch die Entwicklung einer generischen Schnittstelle für Reinforcement Learning-Probleme wird eine Trennung von Problemstellung und Lösungsverfahren erzielt. Die Abbildung 3.4 zeigt schematisch in welcher Form diese Trennung konzeptionell erfolgt. Die Lösungsmethoden sind dabei in Form eines Framework implementiert.

Bei einem *Framework* handelt es sich um eine Menge von kooperierenden Klassen im Sinne des objektorientierten Entwurfs, die einen wiederverwendbaren Entwurf für einen bestimmten Anwendungsfall implementieren. Der Nutzer definiert seine Anwendung durch eine Implementierung gegen eine gegebene Schnittstelle. Ein wesentliches Merkmal ist die Anwendung des Hollywood-Prinzips (*Don't call us, we call you!*), bei dem der Code des Anwenders vom Framework aufgerufen wird, anstatt dass der Anwendungscode Bibliotheksfunktionen aufruft, wie bei Programm-Bibliotheken üblich.

Das Steuerungsproblem wird separat zusammen mit dem Zustands- und Aktionsraum beschrieben (siehe Abb. 3.4). Auf diese Art und Weise können die Lösungsmethoden mit einfachen Steuerungsproblemen, wie z.B. dem Richtungssucher, der in Kapitel 4 zusammen mit der bekannten optimalen Lösung beschrieben ist, getestet werden. Dadurch kann das Verfahren, ohne Änderung dessen Implementierung, auf ein komplexeres Problem oder eines ohne bekannte Lösung angewendet werden. Auf diese Weise werden die Lösungsverfahren von den Steuerungsproblemen entkoppelt. Es können sowohl verschiedene Lösungsmethoden am gleichen Steuerungsproblem, als auch verschiedene Steuerungsprobleme mit der gleichen Lösungsmethode kombiniert werden.

Neue Steuerungsprobleme lassen sich relativ leicht hinzufügen, da nur die wesentlichen Aspekte wie der Übergang $s \xrightarrow{a} s'$, die Funktion $r(s,a,s')$ und $\mathcal{S}$ und $\mathcal{A}(s)$ beschrieben werden müssen. Dies wird in den folgenden Abschnitten näher erläutert. Das Framework übernimmt automatisch die Speicherung der Simulationsdaten und Metadaten (wie in Abschnitt 3.2.1 erläutert) zusammen mit einer optimalen Lösung in Form von V^*, Q^* und π^*, sofern diese zur Verfügung steht. Auch die Wahl zwischen diskreten oder kontinuierlichen Aktionen ist direkt möglich.

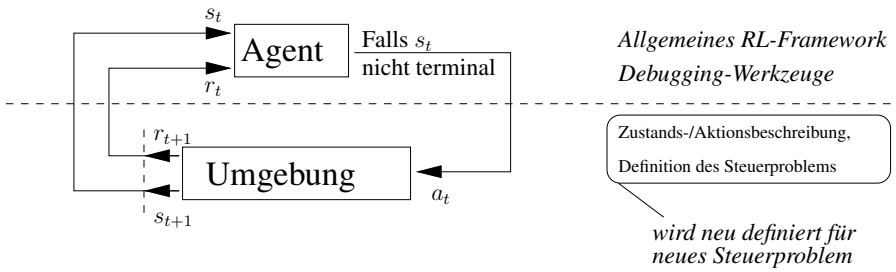


Abbildung 3.4: Trennung zwischen Implementierung der Lösungsmethoden und der Definition des Steuerungsproblems bzw. des Zustands- und Aktionsraums.

Definition des Steuerproblems Das Reinforcement Learning-Problem wird separat von den Lösungsroutinen durch folgende Angaben beschrieben:

- Beschreibung der Aufgabe in Textform zu Dokumentationszwecken. Diese wird automatisch in der HDF5-Datei abgelegt.
- Definition eines Kriteriums, ob ein Zustand terminal ist.
- Optional: Definition eines Kriteriums, ob ein Zustand gültig ist.
- Optional: Definition eines Kriteriums, ob eine Aktion in einem gegebenen Zustand gültig ist.
- Definition des Übergangs $s \xrightarrow{a} s'$, also der Systemdynamik.
- Definition der Belohnung $r(s, a, s')$.
- Optional: Angabe von $V^*(s)$.
- Optional: Angabe von $Q^*(s, a)$.
- Optional: Angabe einer optimalen Politik $\pi^*(s)$.

Die optionale Angabe der optimalen Lösung ermöglicht einen direkten Vergleich mit Lösungen des Reinforcement Learning (vgl. Kapitel 4). Dazu kommt eine Definition des Zustands- und Aktionsraums, wie in Abschnitt 3.2.5 erläutert.

Bei der Kombination des Steuerproblems mit dem Framework übernimmt ersteres alle Optionen und Möglichkeiten, die zur Lösung von Reinforcement Learning-Problemen momentan zur Verfügung stehen. Dies sind beispielsweise Parameter wie die Lernrate α, der Trace-Parameter λ oder die Angabe, welche Arten von Featurevektoren zur Generalisierung von Zuständen und Aktionen ausgewählt werden können. Je nach Steuerproblem kommen möglicherweise spezielle Konfigurationsmöglichkeiten hinzu, welche der Anwender auf Wunsch zusätzlich in das System einbringen kann.

```
sps = PropertySet('state')
sps.add_continuous('position', MIN_X, MAX_X)
sps.add_continuous('velocity', MIN_V, MAX_V)
aps = PropertySet('action')
aps.add_discrete('acceleration', [-1,0,1])
```

Abbildung 3.5: Definition des Zustands- und Aktionsraums für das Mountain-Car-Problem in der verwendeten Programmiersprache *Python*. Eine Instanz der Klasse `PropertySet` definiert ein Tupel von Eigenschaften. Diese Größen sind benannt und können so bei der Fehlersuche leichter zugeordnet werden. Die angegebenen Grenzen und Werte dienen zur automatischen Bereichsüberprüfung, der Bildung von Samplepunkten für die Ausgabe der Value-Funktionen (siehe Tabelle 3.1), sowie im Fall der kontinuierlichen Variablen zur Berechnung von Featurevektoren zur Generalisierung. Konkret für das Mountain-Car-Problem wird hier zunächst ein Zustandsraum mit zwei kontinuierlichen Größen x und v für Ort und Geschwindigkeit definiert, zusammen mit minimalen und maximalen Werten, hier durch symbolische Konstanten repräsentiert. Es schließt sich die Definition des Aktionsraums mit einer diskreten Größe a an, der Beschleunigung, für die es nur die drei Werte -1,0 und 1 geben kann.

Definition von Zustands- und Aktionsräumen Die Definition der Aktions- und Zustandsräume erfolgt jeweils durch die Festlegung einer geordneten Menge von Eigenschaften. Für diskrete Variablen ist das die Menge der möglichen Werte. Ein Beispiel sind die diskreten Aktionen des Richtungssuchers: $\{\rightarrow, \uparrow, \leftarrow, \downarrow\}$. Für kontinuierliche Variablen wird ein minimaler und ein maximaler Wert benötigt. Die Abbildung 3.5 zeigt beispielhaft einen Programmcode in der verwendeten Programmiersprache *Python* mit einer Definition des Zustandsraums mit zwei kontinuierlichen Variablen für das Mountain Car-Problem zusammen mit der Definition der drei diskreten Aktionen *vorwärts beschleunigen*, *nichts tun* und *rückwärts beschleunigen* in Form der Werte -1, 0 und 1. Kontinuierliche Zustands- und Aktionsräume können in Form von beschränkten n-dimensionalen Quadern definiert werden.

Bei Angabe der gewünschten Art von Feature-Vektoren beim Start der Simulation, (z.B. *binär*, *radial-basiert*, oder *zyklisch radial-basiert*, siehe auch die Abschnitte 4.7.1 und 4.8.1) werden die Feature-Vektoren automatisch aus den gegebenen Grenzen berechnet und verwendet.

4 Der Richtungssucher als Steuerproblem mit bekannter Lösung

Die Literatur zum Reinforcement Learning kennt eine Vielzahl von verschiedenen Steuerungsproblemen mit denen Algorithmen getestet werden, jedoch sind darunter nur wenige, für die eine exakte Lösung bekannt ist. In diesem Kapitel wird das *Richtungssucher-Problem* als neues Steuerungsproblem beschrieben und es wird eine analytische optimale Lösung sowohl für diskrete als auch für kontinuierliche Aktionen angegeben. Diese Steuerungsaufgabe zeichnet sich dadurch aus, dass die Wahl zwischen kontinuierlichen Aktionen im Allgemeinen eine bessere Gesamtbelohnung des Agenten ermöglicht als die Beschränkung auf wenige diskrete Aktionen.

Das Richtungssucher-Problem wird mit Hilfe des Sarsa(λ)-Algorithmus gelöst, sowohl für diskrete als auch kontinuierliche Aktionen, sowie für verschiedene Arten der Generalisierung der kontinuierlichen Variablen. Zum Vergleich der verschiedenen Lösungen wird ein Maß eingeführt, das sich auch für die Bewertung von Verfahren bezüglich anderer Steuerungsaufgaben eignet: Der so genannte *Umwegskoeffizient*. Es stellt sich heraus, dass sich durch die Nutzung problemangepasster Featurevektoren im Vergleich mit der optimalen Lösung bessere Strategien finden lassen. Für kontinuierliche Aktionen wird dazu die Periodizität der Action-Value-Funktion ausgenutzt.

4.1 Beschreibung der Steuerungsaufgabe

Der *Richtungssucher* ist eine vereinfachte Version des sogenannten *Path-Finding-Problems* (siehe z.B. Millán und Torras 1992, Fukao et al. 1998), welches dahingehend modifiziert wurde, dass keine Hindernisse umgangen werden müssen. Der Agent beginnt seine Episode an einer beliebigen Position auf einem rechteckigen Grundgebiet:

$$\mathcal{S} = \left\{ (x, y) \,\middle|\, 0 \leq x \leq x_{\max} \wedge 0 \leq y \leq y_{\max} \right\}.$$

Seine reellwertige Position $s = (x, y)$ ist der aktuelle Zustand des Systems (vgl. Abbildung 4.1). Das Ziel des Agenten ist es, mit möglichst wenig Schritten eine Zielregion zu betreten. Diese Zielregion ist ein Quadrat mit dem Mittelpunkt (c_x, c_y) und der Seitenbreite $2d$ inklusive Randlinie. Die Größe d ist der Abstand zur Zielregion, der unterschritten werden muss, um eine Episode zu beenden. Jede Position in der Zielregion, also auch diejenigen auf dem Rand, entsprechen einem terminalen Zustand:

$$(x, y) \text{ terminal } :\Leftrightarrow \; |x - c_x| \leq d \;\wedge\; |y - c_y| \leq d. \tag{4.1}$$

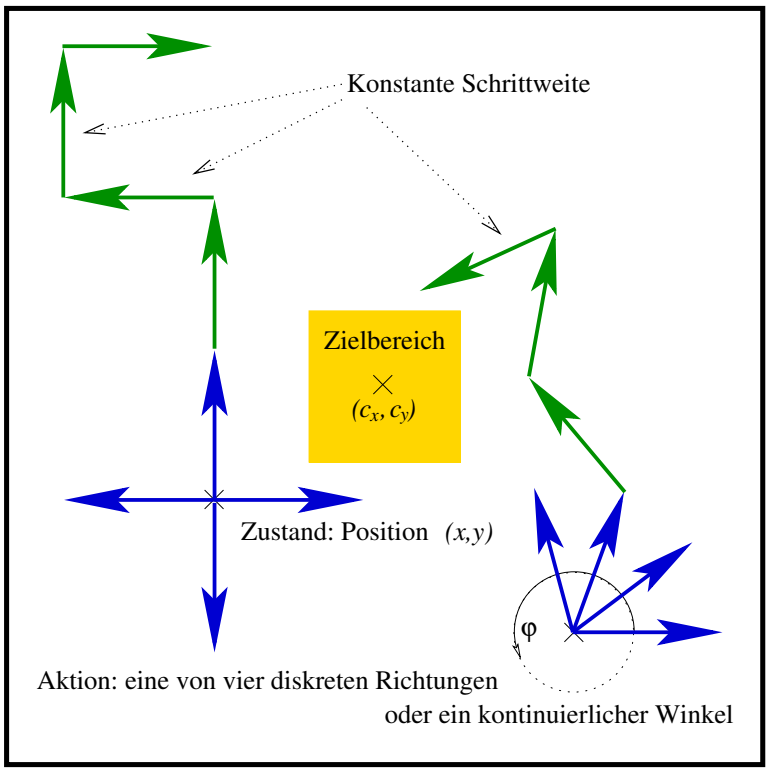


Abbildung 4.1: Zusammenfassung der Steuerungsaufgabe für den Richtungssucher. Dargestellt ist das Grundgebiet, in dem sich der Agent bewegt. Der Zustand des Systems ist durch die reellwertigen (x, y)-Koordinaten des Agenten gegeben. Der Agent soll mit möglichst wenig Schritten in die markierte Zielregion in der Mitte gelangen; dazu stehen ihm Schritte mit einer festen Schrittweite zur Verfügung. Vor jedem Schritt muss der Agent die Entscheidung treffen, in welche Richtung er geht. Würde der Schritt über das Grundgebiet hinaus führen, gilt die Entscheidung als getroffen, der Agent bleibt aber stehen. Betritt der Agent die Zielregion (inklusive des Randes), so ist eine Episode beendet. Für jeden Schritt erhält der Agent eine Belohnung von -1. Die Maximierung der Gesamtbelohnung entspricht daher einer Minimierung der Schrittzahl. Es werden zwei Ausprägungen des Systems behandelt: Zum einen hat der Agent vier bestimmte Winkel zur Verfügung ($\leftarrow, \rightarrow, \uparrow, \downarrow$; diskrete Aktionen) oder aber beliebige Winkel zwischen 0 und 2π (kontinuierliche Aktionen).

Um einen solchen zu erreichen, geht der Agent einen oder mehrere Schritte mit einer festen Schrittweite von 1. Vor jedem Schritt muss er sich entscheiden, welche Richtung er sich bewegen möchte. Die Richtung wird in Form eines Winkels φ gemäß positiver mathematischer Notation gewählt. Würde die Aktion den Agenten aus dem Grundgebiet heraus führen, so wird der Schritt nicht durchgeführt und die Position des Agenten ändert sich nicht. Bei gegebenem Zustand $s = (x, y)$ und Aktion $a = \varphi$ und unter Verwendung der Abkürzung

$$(\tilde{x}, \tilde{y}) = (x + \cos\varphi, y + \sin\varphi), \tag{4.2}$$

berechnet sich der Folgeszustand $s' = (x', y')$ durch

$$(x', y') = \begin{cases} (\tilde{x}, \tilde{y}) & \text{falls } (0 \leq \tilde{x} \leq x_{\max} \ \wedge \ 0 \leq \tilde{y} \leq y_{\max}), \\ (x, y) & \text{sonst.} \end{cases} \tag{4.3}$$

Als Belohnung wird hier, unabhängig von dem Ausgang einer Aktion, stets der Wert $r \equiv -1$ verwendet. Wegen $\gamma = 1$ ist die gesamte Belohnung gleich der negativen Anzahl der Schritte bzw. Aktionen, die nötig sind um die Zielregion zu erreichen.

Die Abbildung 4.1 fasst das Problem zusammen. Für die folgende Darstellung der optimalen Lösung, sowie für die Verwendung dieses Steuerungsproblems im weiteren Verlauf der Arbeit wird

$$x_{\max} = y_{\max} = 5, \ d = \frac{1}{2} \text{ und } c_x = c_y = 2.5 \tag{4.4}$$

gewählt.

Die optimale Lösung für dieses Problem hat zwei besondere Eigenschaften:

- Im Allgemeinen erlaubt die Verwendung von kontinuierlichen Winkeln als Aktionen eine größere Gesamtbelohnung als in Fällen, in denen man sich auf endlich viele Winkel beschränkt.

- Die Value-Funktionen V^* und Q^* können analytisch ausgedrückt werden, sodass Zwischenergebnisse von RL-Algorithmen auf einfache Weise mit beliebiger Genauigkeit verglichen werden können.

Beide Eigenschaften macht das Richtungssucher-Problem auch für didaktische Zwecke im Bereich des Reinforcement Learning interessant.

Boyan und Moore (1995) beschreiben ein dem Richtungssucher sehr ähnliches Problem und geben eine optimale Value-Funktion für vier diskrete Richtungen an. Das Problem ist wird allerdings ungenau spezifiziert – so ist nicht klar, wie groß der Zielbereich eigentlich ist und wie das System reagiert, wenn der Agent auf einen Rand des Grundgebiets stößt. Daher kann für dieses System die Lösung nicht vollständig angegeben werden. Kontinuierliche Richtungen werden nicht betrachtet.

Im Folgenden werden die Lösungen des Richtungssuchers für diskrete und kontinuierliche Aktionen beschrieben.

4.2 Optimale Lösung für diskrete Aktionen

Für eine endliche Menge von vier diskreten Aktionen $\{\rightarrow, \uparrow, \leftarrow, \downarrow\}$, die in Radiant als $\left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right\}$ ausgedrückt werden können, kann der Wert der optimalen State-Value-Funktion $V^*(x, y)$ für einen beliebigen Zustand (x, y) durch

$$V^*_{\text{discr}}(x, y) = -\lceil h(g_x(x))\rceil - \lceil h(g_y(y))\rceil. \tag{4.5}$$

bestimmt werden. Das Symbol $\lceil\cdot\rceil$ steht für die Aufrundungsfunktion, d.h.

$$\lceil x\rceil := \min_{k\in\mathbb{Z}, k\geq x}(k), \tag{4.6}$$

also die kleinste ganze Zahl, die größer gleich x ist. Die Bedeutung der Ausdrücke $h(g_x(x))$ und $h(g_y(y))$ ist wie folgt gegeben:

$$g_x(x) = |x - c_x| - d, \tag{4.7a}$$

$$g_y(y) = |y - c_y| - d, \tag{4.7b}$$

$$h(z) = \begin{cases} 0 & \text{für} \quad z \leq 0, \\ z & \text{für} \quad z > 0. \end{cases} \tag{4.7c}$$

Der Wert eines terminalen Zustands ist 0, da kein Schritt benötigt wird. Die Abbildung 4.2 zeigt die optimale State-Value-Funktion V^* für diskrete Aktionen für die Parameter (4.4). Die Funktion ist unstetig und man erkennt ein schachbrettartiges Muster. Die Zielregion in der Mitte ist als der Bereich erkennbar, für den $V^*(x, y) = 0$ ist, wobei die Randlinie dazu gehört. Wegen der begrenzten Anzahl von Richtungen benötigt der Agent vier Schritte, wenn er in der Nähe einer Ecke des Grundgebiets beginnt.

4.3 Optimale Lösung für kontinuierliche Aktionen

Falls für den Agenten beliebige Winkel $\varphi \in [0, 2\pi[$, d.h. kontinuierliche Aktionen, wählbar sind, berechnet sich der bestmögliche Wert eines Zustands durch:

$$V^*_{\text{cont}}(x, y) = -\left\lceil \sqrt{h^2(g_x(x)) + h^2(g_y(y))} \right\rceil. \tag{4.8}$$

Wie auch für diskrete Aktionen, wurden die Hilfsfunktionen in (4.7), sowie die Aufrundungsfunktion (4.6) verwendet. Die Abbildung 4.3 zeigt die optimale State-Value-Funktion V^* für kontinuierliche Aktionen bei Verwendung der Parameter (4.4). Im Gegensatz zum Fall mit diskreten Aktionen werden maximal drei Schritte benötigt, um die Zielregion zu erreichen. Zwar ist die Funktion auch hier unstetig, jedoch ist statt eines schachbrettartiges Musters (vgl. Abb. 4.2) eine ringförmige Struktur zu sehen. Diese wird ausgehend von der quadratischen Zielregion in der Mitte nach außen hin immer runder. Abgesehen von der erweiterten Menge der möglichen Aktionen ist die Definition des Steuerproblems gleich der für diskrete Aktionen.

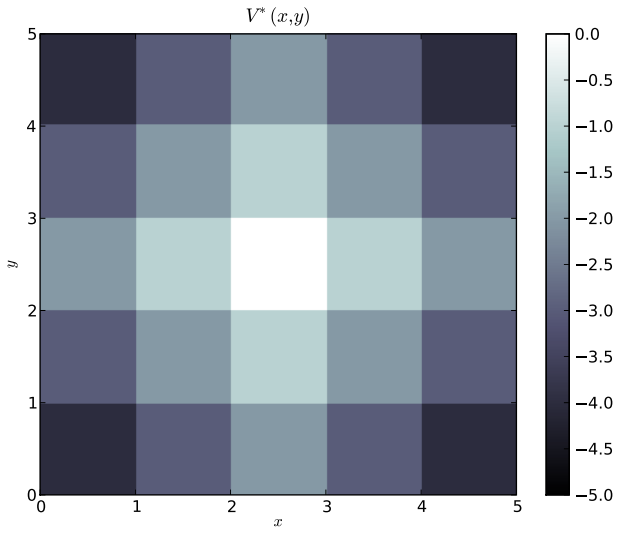


Abbildung 4.2: Optimale State-Value-Funktion $V^*(x, y)$ für das Richtungssucher-Problem mit den vier diskreten Aktionen $\rightarrow$, $\uparrow$,$\leftarrow$ und $\downarrow$. Aufgrund von $\gamma = 1$ ist $V^*(x, y)$ gleich der negativen Anzahl der Schritte, die benötigt werden, um die quadratische Zielregion in der Mitte (hier ausgezeichnet durch den Funktionswert 0) zu erreichen. Es gilt $V^*(x, y) \in \{-4, -3, -2, -1, 0\}$.

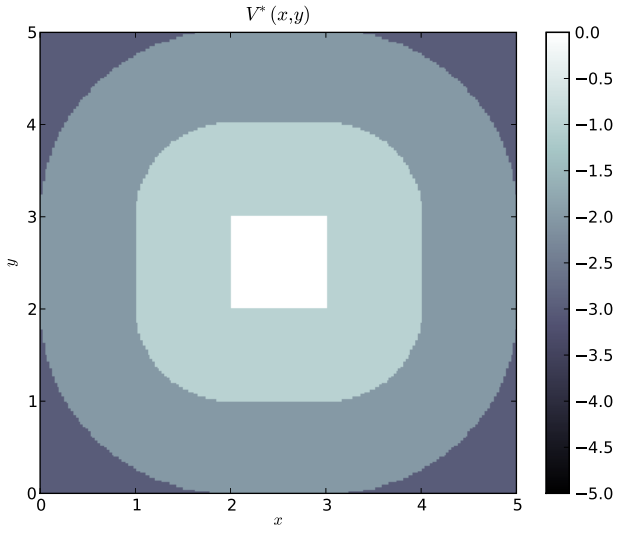


Abbildung 4.3: Optimale State-Value-Funktion $V^*(x, y)$ für das Richtungssucher-Problem mit kontinuierlichen Aktionen, d.h. reellwertigen Winkel $\varphi \in [0, 2\pi[$. Die auftretenden Werte sind $V^*(x, y) \in \{-3, -2, -1, 0\}$. Im Vergleich zu vier diskreten Aktionen (Abb. 4.2) werden maximal drei statt vier Schritte benötigt.

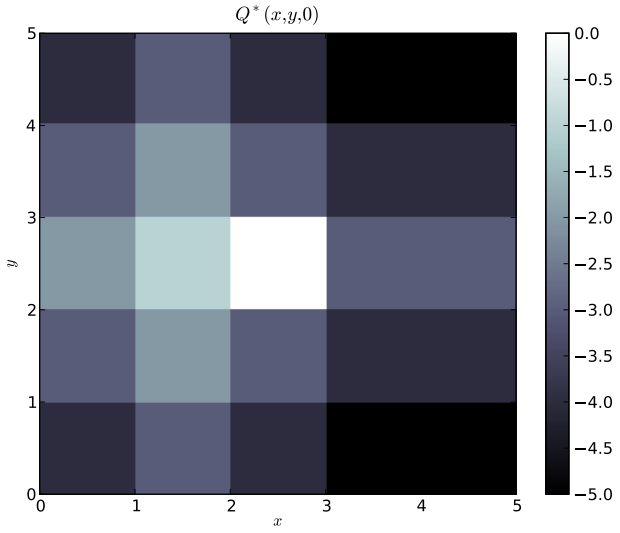


Abbildung 4.4: Optimale Action-Value-Funktion Q^* für alle Zustände (x, y) und die Aktion $\rightarrow$ (bzw. $\varphi = 0$) für den Fall vier diskreter Aktionen $\rightarrow, \uparrow, \leftarrow$ und $\downarrow$. Abzulesen für eine beliebige Startposition ist die Anzahl der Schritte, die der Agent benötigt, wählt er zunächst die Aktion $\rightarrow$, um dann einer optimalen Politik, der die optimale State-Value-Funktion V^* in Abbildung 4.2 zugeordnet ist, zu folgen. Es gilt $Q^*(x, y, 0) \in \{-5, -4, \ldots, 1, 0\}$.

4.4 Optimale Action-Value-Funktion Q^*

Das Problem des Richtungssuchers ist deterministisch, d.h. sowohl der Übergang von einem Zustand zum nächsten als auch die Belohnung sind deterministisch. Mit gegebenem V^* und $\gamma = 1$ ist es in diesem Fall sehr einfach, die optimale Action-Value-Funktion $Q^*(x, y, \varphi)$ zu berechnen. Man bestimmt dazu mittels (4.2) und (4.3) den Wert $V^*(x', y')$ des auf (x, y, φ) folgenden Zustands, berücksichtigt die konstante Belohnung bzw. betrachtet terminale Zustände gesondert:

$$Q^*(x, y, \varphi) = \begin{cases} 0 & \text{falls } (x, y) \text{ terminal ist,} \\ V^*(x', y') - 1 & \text{sonst.} \end{cases} \tag{4.9}$$

Dabei ist der Folgezustand (x', y') definiert wie in (4.3). Der Wert $V^*(x', y')$ wird im Fall von diskreten Aktionen durch (4.5) und im Fall kontinuierlicher Aktionen durch (4.8) bestimmt.

Für die Richtungswahl $\rightarrow$ bzw. $\varphi = 0$, ist die Funktion Q^* in den Abbildungen 4.4 (für diskrete Aktionen) bzw. 4.5 (für kontinuierliche Winkel) zu sehen. Lässt man die Zielregion außer Acht, lassen sich beide Darstellungen qualitativ in Form einer Konstruktion beschreiben: Als einen Plot der Funktion $V^*(x, y) - 1$, verschoben um 1 nach links, mit einer anschließenden Kopie der Werte von $x \in]3, 4]$ nach $x \in]4, 5]$. Die Funktion $Q^*(x, y, \varphi)$ lässt sich für beliebige Richtungen auswerten. Die Abbildung 4.6 zeigt für kontinuierliche Rich-

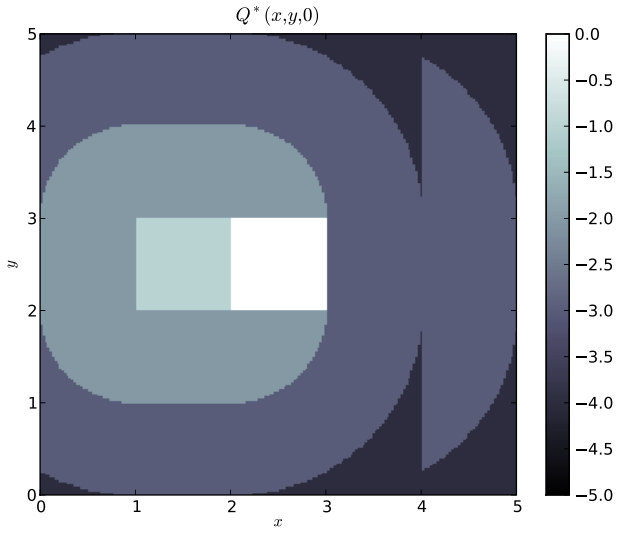


Abbildung 4.5: Optimale Action-Value-Funktion Q^* für alle Zustände (x, y) und die Aktion $\rightarrow$ (bzw. $\varphi = 0$) für kontinuierliche Aktionen, d.h. beliebiger Richtungswahl. Die entsprechende optimale State-Value-Funktion V^* ist in Abbildung 4.3 zu sehen. Es gilt $Q^*(x, y, 0) \in \{-4, 3, \ldots, 1, 0\}$.

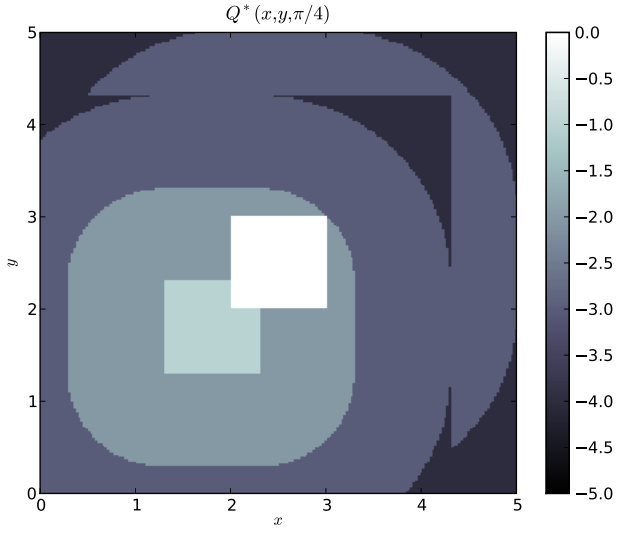


Abbildung 4.6: Optimale Action-Value-Funktion Q^* für alle Zustände (x, y) und die Aktion $\varphi = \pi/4$ bei prinzipiell beliebiger Richtungswahl. Vergleiche auch die Abbildungen 4.5 bzw. 4.3.

tungen die Funktion $Q^*(x, y, \frac{\pi}{4})$, die sich ebenfalls durch eine entsprechende Verschiebung von $V^*(x, y) - 1$ darstellen lässt.

4.5 Verfahren zur Berechnung einer optimalen Politik π^*

Um zum Vergleich eine optimale kontinuierliche Richtung $\varphi^*(s)$ für einen beliebigen nichtterminalen Zustand s und eine beliebige Lage des Zielbereichs zu bestimmen, wurde folgende Vorgehensweise implementiert:

1. Bestimme diejenigen Punkte auf dem Rand des Zielgebiets, die sich als Schnittpunkte des Lots von $s = (x, y)$ auf die vier Geraden, welche durch die vier Eckpunkte des Zielgebiets gebildet werden, ergeben. Verwirf davon alle Schnittpunkte, die nicht auf dem Rand des Zielgebiets liegen.

2. Die aus im ersten Schritt übrigen bilden zusammen mit den Eckpunkten des Zielgebiets eine Menge von „Kandidatenpunkten“. Bestimme aus dieser Menge denjenigen Punkt $(\hat{x}, \hat{y})$, der dem Zustand $s = (x, y)$ geometrisch am nächsten liegt. Diesen Punkt direkt anzustreben ist Teil einer optimalen Strategie.

3. Wähle denjenigen Winkel als Aktion $\pi^*(s) = \varphi^*(s)$, der vom Zustand s direkt auf dem gewählten Punkt $(\hat{x}, \hat{y})$ zeigt.

Eine solche Politik ist in Abbildung 4.7 für die verwendeten Parameter aus (4.4) in Form von Pfeilen dargestellt. Man erkennt, dass die dargestellte Strategie stets darin besteht, sich auf direktem Wege auf den nächstliegenden Punkt der Zielregion hinzubewegen. Dies ist eine Ecke des Zielgebiets für alle Punkte, die sich in einem der Quadranten befinden, die jeweils durch eine solche Ecke und der gegenüberliegenden Ecke des Grundgebiets gebildet werden. Für alle anderen Positionen ist ein Punkt auf einer Seitenlinie des Zielgebiets der nächstliegende Punkt. Die dargestellte Strategie ist nicht eindeutig – deshalb können Variationen in einzelnen Richtungen immer noch zu einer gleich hohen Gesamtbelohnung führen.

Aus einer solchen optimalen Politik für kontinuierliche Aktionen lässt sich auch eine optimale Politik für diskrete Aktionen ableiten, indem man zu jedem kontinuierlichen Winkel φ^* jeweils den ähnlichsten Winkel aus der Menge $\{\rightarrow, \uparrow, \leftarrow, \downarrow\}$ sucht (siehe Abbildung 4.8). Auch hier besteht die Strategie darin, sich so schnell wie möglich auf den nächstliegenden Punkt der Zielregion hinzubewegen. Für die Positionen (x, y), für die dieser nächstliegende Punkt ein Eckpunkt des Zielgebiets ist, kann das erreicht werden, indem der Agent eine der Diagonalen durch das Grundgebiet anstrebt.

4.6 Evaluation durch den Umwegskoeffizienten

Zur Bewertung der Ergebnisse wird ein Maß benötigt, das es ermöglicht den Grad der Zielerreichung gemessen an dem Ziel einer optimalen Lösung bzw. der am besten bekannten

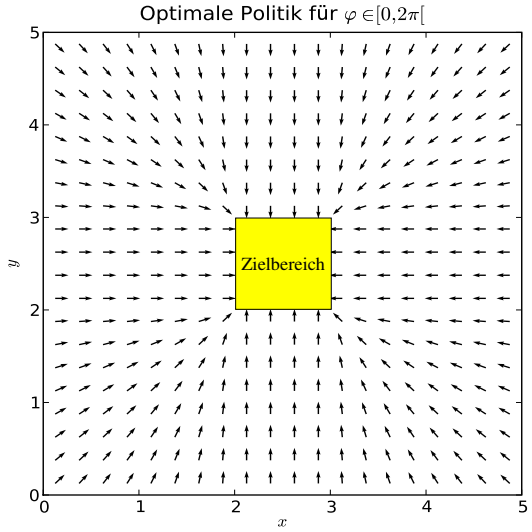


Abbildung 4.7: Beispiel für eine (offensichtliche) optimale Politik des Richtungssuchers für kontinuierliche Aktionen, also beliebige Winkel $\varphi \in [0, 2\pi[$. Dargestellt sind Aktionen $\pi^*(s)$ für Beispielzustände auf einem Gitter mit 20×20 Punkten, jeweils symbolisiert durch einen Pfeil mit dem Ursprung in s. Alle Zustände im Zielbereich sind terminal.

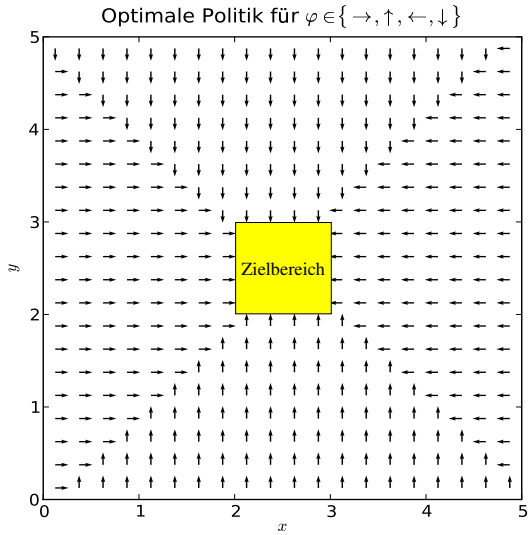


Abbildung 4.8: Beispiel für eine optimale Politik des Richtungssuchers für die vier Richtungen $\varphi \in \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$. Diese kann leicht aus einer optimalen Politik für beliebige Winkel (siehe Abbildung 4.7) abgeleitet werden, indem man sich auf die vier möglichen Richtungen beschränkt.

Lösung zu beurteilen. Dies ist gerade für den Richtungssucher möglich, da eine optimale Lösung bekannt ist. Im Folgenden wird ein solches Maß vorgestellt, welches auch auf andere Steuerungsprobleme mit bekannten optimalen Lösungen übertragbar ist.

Sei $\mathcal{E}$ eine Menge von Episoden, die „nicht-trivial“ in dem Sinne sind, dass sie mit einem nicht-terminalen Zustand starten, so dass mindestens eine Aktion ausgeführt werden muss. Zu dieser Menge wird ein Maß $\eta(\mathcal{E})$ definiert, welches im Folgenden als *Umwegskoeffizient* bezeichnet wird:

$$\eta(\mathcal{E}) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \left(V^*(s_0^e) - R^e \right) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \eta_e. \tag{4.10}$$

Dabei ist R^e die (diskontierte) Gesamtbelohnung, die der Agent innerhalb der Episode e erwirbt und s_0^e ist der entsprechende Startzustand dieser Episode. In dem Beispiel des Richtungssuchers ist der Ausdruck

$$\eta_e := V^*(s_0^e) - R^e \tag{4.11}$$

gleich der Anzahl der Schritte, die der Agent in der Episode e im Vergleich zur optimalen Lösung zusätzlich benötigt, um die Zielregion zu erreichen. Die Größe $|\mathcal{E}|$ bezeichnet die Anzahl der Episoden in $\mathcal{E}$. Die Kenntnis der optimalen Lösung des Problems geht über die optimale State-Value-Funktion $V^*(s)$ ein, die für den Startzustand ausgewertet wird. Je kleiner $\eta(\mathcal{E})$ ist, als desto effektiver kann ein Lernprozess betrachtet werden. Es gilt stets $\eta(\mathcal{E}) \geq 0$, da die optimale Lösung nicht übertroffen werden kann.

Der Umwegskoeffizient hängt von der zufälligen Auswahl der Startzustände der Episoden $e \in \mathcal{E}$ ab, daher wird hier $\eta(\mathcal{E})$ als Schätzer betrachtet, dem mit

$$s_\eta(\mathcal{E}) = \sqrt{\frac{1}{|\mathcal{E}|\,(|\mathcal{E}|-1)} \sum_{e \in \mathcal{E}} (\eta_e - \eta(\mathcal{E}))^2} \tag{4.12}$$

ein Standardfehler zugeordnet wird. Für die numerischen Resultate in dieser Arbeit werden beim Vergleich von verschiedenen Verfahren oder Parametern stets Episoden mit gleichen Startzuständen verwendet.

4.7 Numerische Lösung für diskrete Richtungen

4.7.1 Berechnung der Featurevektoren für Zustände und diskrete Aktionen

Das Richtungssucher-Problem, wie es in Abschnitt 4.1 beschrieben ist, hat einen Zustand, der sich aus zwei kontinuierlichen Variablen zusammensetzt. Demnach wird eine Generalisierung benötigt, die durch Feature-Vektoren umgesetzt wird, wie schon im Fall des Mountain-Car-Problems (vgl. Abschnitt 2.7.3). Eine der einfachsten Feature-Typen sind *binäre* Features, bei denen ein Vektor zugeordnet wird, der lediglich die Werte 0 oder 1 in seinen Komponenten hat. Ein derartiger Vektor kann gebildet werden, indem man den zu generalisierenden Raum in Intervalle einteilt und bei Zugehörigkeit eine 1 zuordnet und sonst eine 0. Zur Generalisierung der Zustände des Richtungssuchers mit Hilfe von binären

Features wurden für die x- und y-Richtung jeweils $n_x = n_y = 20$ quadratische Intervalle der Form

$$\mathcal{I}_{i,j} = \left[\frac{i-1}{n_x} x_{\max}, \frac{i}{n_x} x_{\max}\right] \times \left[\frac{j-1}{n_y} y_{\max}, \frac{j}{n_y} y_{\max}\right] \tag{4.13}$$

$$= \left[\frac{i-1}{4}, \frac{i}{4}\right] \times \left[\frac{j-1}{4}, \frac{j}{4}\right] \tag{4.14}$$

mit $i, \ldots, n_x$ und $j = 1, \ldots, n_y$ verwendet. Die Zentren $c_{ij}^{\mathcal{S}}$ dieser Intervalle sind die Punkte

$$c_{ij}^{\mathcal{S}} = \left(\frac{2i-1}{2n_x} x_{\max}, \frac{2j-1}{2n_y} y_{\max}\right) = \left(\frac{2i-1}{8}, \frac{2j-1}{8}\right). \tag{4.15}$$

Für binäre Zustandsfeatures wird

$$\phi_{ij}^{\mathcal{S}}(s) = \begin{cases} 1 & \text{falls } s \in \mathcal{I}_{i,j}, \\ 0 & \text{sonst} \end{cases} \tag{4.16}$$

ausgewertet. Hierbei steht der Zustand s für das Tupel (x, y). Durch das Aneinanderhängen der Zeilen werden die Komponenten von (4.16) zu einem Featurevektor $\boldsymbol{\phi}_{\text{bin}}^{\mathcal{S}}(s)$ mit den $n_x \cdot n_y$ Komponenten

$$\left(\boldsymbol{\phi}_{\text{bin}}^{\mathcal{S}}(s)\right)_{(i-1)n_y+j} = \phi_{ij}^{\mathcal{S}}(s) \tag{4.17}$$

für $i = 1, \ldots, n_x$ und $j = 1, \ldots, n_y$ zusammengesetzt.

Eine Erweiterung der binären Features sind *radial-basierte* Featurevektoren, bei denen nicht nur die Werte 0 und 1, sondern auch alle Zwischenwerte erreicht werden können. Dazu platziert man Gauß-Funktionen in die Mitte der Intervalle (4.14) (Sutton und Barto 1998). Im Vergleich zu binären Features erlauben sie es Approximationsfunktionen zu konstruieren, die differenzierbar sind. Im Fall des Richtungssuchers wurden für radial-basierte Zustandsfeatures die Ausdrücke

$$\tilde{\phi}_{ij}^{\mathcal{S}}(s) = \exp\left(-\left(\frac{x - \left(c_{ij}^{\mathcal{S}}\right)_x}{x_{\max}/n_x}\right)^2 - \left(\frac{y - \left(c_{ij}^{\mathcal{S}}\right)_y}{y_{\max}/n_y}\right)^2\right) \tag{4.18}$$

$$= \exp\left(-16(s - c_{ij}^{\mathcal{S}})^2\right) \tag{4.19}$$

verwendet. Dabei ist $s = (x, y)$ und $\left(c_{ij}^{\mathcal{S}}\right)_x$ bzw. $\left(c_{ij}^{\mathcal{S}}\right)_y$ die Komponenten von (4.15). Analog den binären Features werden wiederum die Komponenten (4.19) zu einem Featurevektor $\tilde{\boldsymbol{\phi}}^{\mathcal{S}}(s)$ zusammengesetzt, welcher anschließend noch zusätzlich normalisiert wird:

$$\boldsymbol{\phi}_{\text{rb}}^{\mathcal{S}}(s) = \frac{\tilde{\boldsymbol{\phi}}^{\mathcal{S}}(s)}{\sum_l \tilde{\phi}_l^{\mathcal{S}}(s)}. \tag{4.20}$$

Die beiden Ansätze (4.16) bzw. (4.17) und (4.19) wurden innerhalb dieser Arbeit auch für die Zustände innerhalb der Simulationen mit kontinuierliche Aktionen verwendet (siehe Abschnitt 4.8).

Nun fehlt noch die Angabe eines Feature-Vektors für diskrete Aktionen, um die allgemeine Notation (2.86) für einen kombinierten Featurevektor nutzen zu können. Für den Satz $\mathcal{A} = \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$ von diskreten Aktionen ergibt die Anwendung von (2.87) einen binären Feature-Vektor

$$\boldsymbol{\phi}^{\mathcal{A}}(\varphi) = (\delta_{\varphi,\rightarrow}, \delta_{\varphi,\uparrow}, \delta_{\varphi,\leftarrow}, \delta_{\varphi,\downarrow})^{\mathrm{T}}. \tag{4.21}$$

4.7.2 Ergebnisse

Als Feature-Vektoren $\boldsymbol{\phi}^{S}(s)$ wurden sowohl die binären Features nach (4.16) als auch normalisierte radial-basierte Features entsprechend (4.20) verwendet. Für $\boldsymbol{\phi}^{\mathcal{A}}(a)$ wurde (4.21) benutzt.

Für die optimale Lösung des Richtungssuchers gilt, mit Ausnahme der terminalen Zustände, die die Zielregion bilden, stets $V^*(s) < 0$. Die lineare Funktionsapproximation wird, wie in Algorithmus 2.2 auf Seite 38 beschrieben, durch $\boldsymbol{\theta} = 0$ initialisiert. Dies entspricht der Approximation $\tilde{Q}(x, y, \varphi) = 0$ für alle Zustände $s = (x, y)$ und alle Aktionen $a = \varphi$. Dieser Ansatz ermuntert den Agenten dazu solange wie möglich neue Erfahrungen zu machen, statt alte Erfahrungen auszunutzen (*optimistic initial values* nach Sutton und Barto 1998). Ist der Aktions-Zustandsraum hinreichend gleichmäßig besucht worden, tritt der Lernprozess von selbst aus der Explorationsphase in eine Optimierungsphase ein. Diese Vorgehensweise wird auch als *optimism in the face of uncertainty* bezeichnet (Kaelbling et al. 1996).

Um die Fähigkeit des Algorithmus eine Lösung ohne Vorwissen zu finden zu testen, wurden 10000 Episoden durchlaufen. Die weiteren Parameter waren: Lernrate $\alpha = 0.5$, Trace-Parameter $\lambda = 0.3$ und eine maximale Zeit von $T_{\max} = 20$. Nach zwanzig Aktionen wird eine Episode abgebrochen, unabhängig davon, ob ein terminaler Zustand erreicht wurde oder nicht.

Im Falle von vier diskreten Richtungen ist die Aktionsselektion einfach: Die Action-Value-Funktion $\tilde{Q}(x, y, \varphi)$ wird für eine gegebene Position $s = (x, y)$ und für alle vier möglichen Aktionen φ mit $\varphi \in \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$, ausgewertet. Der maximale Wert gibt den Ausschlag, bei Gleichheit gilt der Vorrang nach genannter Reihenfolge. Gemessen an den letzten 1000 Episoden ergab sich hier für *binäre* Features $\eta(\mathcal{E}) = 0 \pm 0$ und für *radial-basierte* $\eta(\mathcal{E}) = 0.012 \pm 0.005$.

Die zugehörige Approximation $\tilde{Q}(x, y, \varphi)$ nach 10000 Episoden für radial-basierte Zustandsfeatures ist in Abbildung 4.9(a) für ausgewählte Zustände zusammen mit den zugehörigen besten Richtungen dargestellt. Die gefundene Strategie kann man in Worten folgendermaßen beschreiben:

> *Bewege dich auf einem der schnellsten Wege zur einem Punkt* (x, y) *mit* $x \in [2, 3]$ *oder* $y \in [2, 3]$. *Gehe dann gradlinig horizontal bzw. vertikal bis zur Zielregion.*

Diese Strategie unterscheidet sich von derjenigen, die in Abschnitt 4.5 für diskrete Richtungen diskutiert wurde und in Abbildung 4.8 dargestellt wird, ist ihr aber gleichwertig. Dies zeigt anschaulich, dass eine optimale Strategie nicht eindeutig sein muss.

Nach 10000 Episoden der Simulation mit radial-basierten Zustandsfeatures ist für einige vereinzelte Zustände noch keine optimale Richtung gefunden worden, wie z.B. für

$(x, y) = (4.0625, 0.9375)$: Hier verweist die Strategie nach unten in Richtung des Randes des Grundgebiets. Aufgrund solcher Punkte kommt das Ergebnis $\eta(\mathcal{E}) > 0$ für radial-basierte Zustandsfeatures zu Stande. Es ist jedoch nicht ausgeschlossen, dass bei einem länger andauernden Lernprozess auch zu diesen Punkten eine optimale Richtung gefunden wird.

Zur Darstellung von $\max_\varphi \tilde{Q}(x, y, \varphi)$ in Abbildung 4.9(a) zeigt die Abbildung 4.9(b) die Differenz $V^*(x, y) - \max_\varphi \tilde{Q}(x, y, \varphi)$. Wegen (2.23) kann eine solche Darstellung einen Hinweis geben, wo die Approximation ungenau ist. Hier fallen besonders die Unstetigkeitsstellen von V^* auf. Für das Ergebnis mit binären Zustandsfeatures ist diese Differenz überall gleich 0.

Es zeigt sich, dass die binären Features für Zustände sehr gut geeignet sind, um das Richtungssucher-Problem für diskrete Richtungen zu lösen. Dies war zu erwarten, denn die binären Features sind optimal an den Verlauf der optimalen State-Value-Funktion V^* angepasst, wie im Vergleich der Abbildung 4.2 mit Gleichung (4.16) deutlich wird. An dieser Stelle zeigt sich auch, wie das Einbringen von Vorwissen die Suche nach einer optimalen Strategie begünstigen kann.

4.8 Numerische Lösung für beliebige Richtungen

Der Richtungssucher mit beliebigen Richtungen $\varphi \in [0, 2\pi[$ dient im Folgenden dazu, vier verschiedene Varianten des Sarsa(λ)-Algorithmus mit Hilfe des schrittweisen Beurteilungsverfahrens zu testen. Bei den Varianten handelt es sich um die folgenden Kombinationen von Zustands- und Aktionsfeatures:

- binäre Zustandsfeatures, radial-basierte Aktionsfeatures
- binäre Zustandsfeatures, zyklisch-radial-basierte Aktionsfeatures
- radial-basierte Zustandsfeatures, radial-basierte Aktionsfeatures
- radial-basierte Zustandsfeatures, zyklisch-radial-basierte Aktionsfeatures

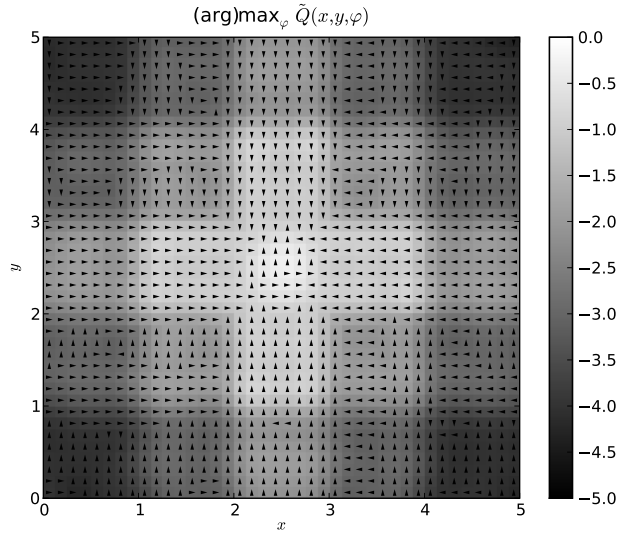
Die Berechnung der binären und radial-basierten Zustandsfeatures ist in Kapitel 4.7.1 beschrieben. Die Berechnung der Featurevektoren für kontinuierliche Richtungen wird im folgenden Abschnitt erläutert.

4.8.1 Berechnung der Featurevektoren für kontinuierliche Aktionen

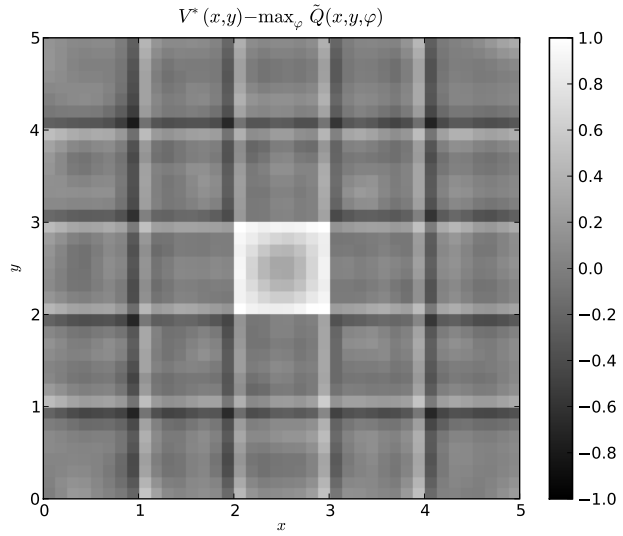
Für kontinuierliche Aktionen, d.h. für beliebige Winkel $\varphi \in [0, 2\pi[$, wurden die $n_\varphi = 16$ Komponenten

$$\tilde{\phi}_l^{\mathcal{A}}(\varphi) = \exp\left(-\left(\frac{\varphi - c_l^{\mathcal{A}}}{2\pi/n_\varphi}\right)^2\right) \tag{4.22}$$

$$= \exp\left(-\frac{64}{\pi^2}\left(\varphi - c_l^{\mathcal{A}}\right)^2\right) \tag{4.23}$$



(a)



(b)

Abbildung 4.9: Ergebnis des Lernens mit diskreten Aktionen, nach Durchführung von 10000 Episoden unter Verwendung von radial-basierten Zustandsfeatures nach (4.19). Mögliche Richtungen: $\varphi \in \{\rightarrow, \uparrow, \leftarrow, \downarrow\}$. Parameter: $\alpha = 0.5$, $\lambda = 0.3$ und $T_{\max} = 20$. Für diese Darstellungen wurden die Value-Funktionen und die Politik an einem gleichförmigen Gitter im Zustandsraum mit 40x40 Punkten ausgewertet. (a) Diskrete Greedy-Aktionen $\varphi_{\mathrm{g}} = \arg\max_\varphi \tilde{Q}(x, y, \varphi)$ und die entsprechenden Werte $\tilde{Q}(x, y, \varphi_{\mathrm{g}})$ der Action-Value-Funktion . Die Winkel, die den Greedy-Aktionen entsprechen, werden durch Pfeilspitzen symbolisiert. (b) Differenz zwischen $V^*(x, y)$ und $\max_\varphi \tilde{Q}(x, y, \varphi)$. Die größten Abweichungen sind an den Unstetigkeitsstellen von $V^*(x, y)$ zu finden. Siehe auch Abbildung 4.2.

mit $l = 1, \ldots, n_\varphi$ und $c_l^{\mathcal{A}} = \frac{2l-1}{16}\pi$ benutzt. Diese Gaußkurven sind deutlich lokalisiert und können im Abstand von 2π von deren Mitten $c_l^{\mathcal{A}}$ vernachlässigt werden:

$$\tilde{\phi}_l^{\mathcal{A}}(\varphi \pm 2n\pi) \approx 0 \;\; \forall l,\, \varphi \in [0, 2\pi],\, n = 1, 2. \tag{4.24}$$

Diese Eigenschaft wird im Folgenden dazu verwendet, Feature-Vektoren zu konstruieren, die die Periodizität

$$Q^*(x, y, \varphi) = Q^*(x, y, \varphi \pm 2\pi), \tag{4.25}$$

berücksichtigen. Dahinter steht die Idee, dass diese auch für die Approximation $\tilde{Q}(x, y, \varphi)$ näherungsweise erfüllt sein sollte. Um dies zu erreichen, werden die Komponenten aus (4.23) entsprechend

$$\hat{\phi}_l^{\mathcal{A}}(\varphi) = \tilde{\phi}_l^{\mathcal{A}}(\varphi - 2\pi) \; + \tilde{\phi}_l^{\mathcal{A}}(\varphi) + \tilde{\phi}_l^{\mathcal{A}}(\varphi + 2\pi) \tag{4.26}$$

zusammengesetzt und normalisiert:

$$\boldsymbol{\phi}_{\text{crb}}^{\mathcal{A}}(\varphi) = \frac{\hat{\boldsymbol{\phi}}^{\mathcal{A}}(\varphi)}{\sum_l \hat{\phi}_l^{\mathcal{A}}(\varphi)}. \tag{4.27}$$

Diese Form des Feature-Vektors wird schließlich in (2.86) für $\boldsymbol{\phi}^{\mathcal{A}}(a)$ verwendet.

Unter der Berücksichtigung der Gleichungen (4.24) und (4.26) folgt näherungsweise die Periodizität für $\tilde{Q}$:

$$\begin{aligned}
& \hat{\phi}_l^{\mathcal{A}}(\varphi) = \hat{\phi}_l^{\mathcal{A}}(\varphi \pm 2\pi) \\
\overset{(4.27)}{\Longrightarrow} \quad & \boldsymbol{\phi}_{\text{crb}}^{\mathcal{A}}(\varphi) = \boldsymbol{\phi}_{\text{crb}}^{\mathcal{A}}(\varphi \pm 2\pi) \\
\Longrightarrow \quad & \boldsymbol{\phi}^{\mathcal{S}}(x, y)(\boldsymbol{\phi}_{\text{crb}}^{\mathcal{A}}(\varphi))^{\mathrm{T}} = \boldsymbol{\phi}^{\mathcal{S}}(x, y)(\boldsymbol{\phi}_{\text{crb}}^{\mathcal{A}}(\varphi \pm 2\pi))^{\mathrm{T}} \\
\overset{(2.86)}{\Longrightarrow} \quad & \boldsymbol{\phi}(x, y, \varphi) = \boldsymbol{\phi}(x, y, \varphi \pm 2\pi) \\
\overset{(2.80)}{\Longrightarrow} \quad & \tilde{Q}(x, y, \varphi) = \tilde{Q}(x, y, \varphi \pm 2\pi).
\end{aligned} \tag{4.28}$$

Featurevektoren für kontinuierliche Winkel, wie sie mit Hilfe von (4.26) und (4.27) gebildet werden, sollen von nun an als *zyklisch radial-basiert* (crb) bezeichnet werden. Um eine weitere Form eines Featurevektors für kontinuierliche Aktionen zum Vergleich zur Verfügung zu haben, werden auch die Komponenten $\tilde{\phi}_l^{\mathcal{A}}(\varphi)$ aus (4.23) zu einem Vektor zusammengefasst und äquivalent wie in (4.27) normalisiert. Diese Features werden im weiteren Verlauf *radial-basiert* (rb) genannt und sind nicht zyklisch.

4.8.2 Ergebnisse

Der Parametervektor $\boldsymbol{\theta}$ der Approximation $\tilde{Q}$ wird, wie auch schon bei den diskreten Aktionen (siehe Kapitel 4.7.2), mit Nullen initialisiert. Die Ergebnisse der Suche nach einer optimalen Politik für den Richtungssucher unter Verwendung von beliebigen Winkeln und gemessen an der optimalen Lösung, sind in Tabelle 4.1 zusammengefasst. Für alle Simulationen wurde die folgende Konfiguration verwendet: konstante Lernrate von $\alpha = 0.5$,

Zustands-Features	Aktions-Features	$\eta(\mathcal{E})$	$s_\eta(\mathcal{E})$
bin	rb	0.109	0.011
bin	crb	0.115	0.013
rb	rb	0.090	0.011
rb	crb	0.063	0.009

Tabelle 4.1: Ergebnisse der Suche nach einer optimalen Politik für den Richtungssucher mit kontinuierlichen Aktionen für verschiedene Approximationsmethoden. Berechnung von $\eta(\mathcal{E})$ mit den letzten 1000 nicht-trivialen Episoden des Lernprozesses, der insgesamt 20000 Episoden hat. Es wurden vier verschiedene Kombinationen von Feature-Vektoren untersucht. Die Abkürzungen stehen für: *binär* (bin), *radial-basiert* (rb) und *zyklisch radial-basiert* (crb). Diese Ergebnisse wurden im Rahmen dieser Arbeit veröffentlicht (Röttger und Liehr 2009).

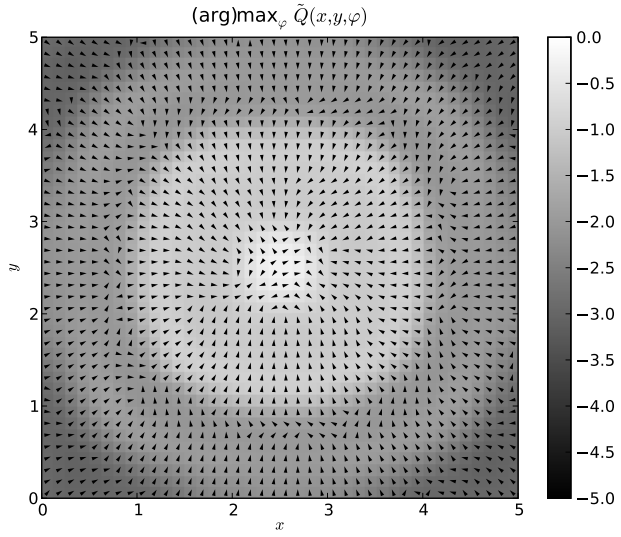
Trace-Parameter $\lambda = 0.3$, maximale Simulationszeit $T_{\max} = 20$ und 20000 Episoden. Zur Berechnung des Umwegskoeffizienten nach (4.10) und (4.12) wurden wie schon bei den Ergebnissen zu diskreten Aktionen in Kapitel 4.8 die letzten 1000 nicht-trivialen Episoden verwendet. Dies sind Episoden, die mit einem nicht-terminalen Zustand beginnen. Aus dem Vergleich in Tabelle 4.1 der vier Kombinationen von Featurevektoren für kontinuierliche Aktionen ergeben sich die Aussagen:

- Radial-basierte Features sind zur Generalisierung der Zustände besser geeignet als binäre Features.
- Bei der Wahl von radial-basierten Featurevektoren für Zustände sind zyklische Features zur Darstellung beliebiger Richtungen besser geeignet als einfache radial-basierte Features, die die Periodizität nicht berücksichtigen.

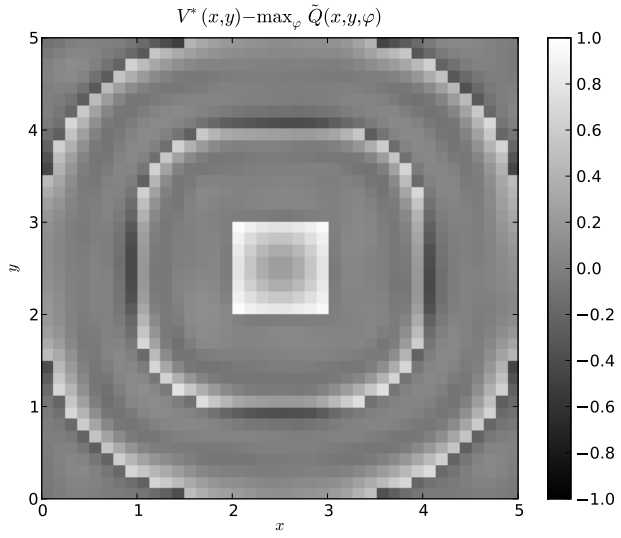
Die Abbildung 4.10 zeigt die Funktion $\max_\varphi \tilde{Q}(x, y, \varphi)$ und ihren Vergleich mit $V^*(x, y)$ für radial-basierte Zustandsfeatures nach (4.19) und zyklisch radial-basierten Aktionsfeatures nach (4.27) nach 20000 Episoden. In der Darstellung der Differenz in Abbildung 4.10(b) treten, wie schon für diskrete Aktionen, gerade die Zustände in den Vordergrund, die in der Nähe der Unstetigkeiten von $V^*(x, y)$ liegen.

Beim Vergleich der optimalen Politik in Abbildung 4.7 mit der Darstellung des Ausdrucks $\arg\max_\varphi \tilde{Q}(x, y, \varphi)$ in Abbildung 4.10(a) fällt auf: Die numerische Lösung folgt prinzipiell der Strategie den nächstliegenden Randpunkt des Zielgebiets zu erreichen. Gerade in der Nähe der Unstetigkeitsstellen von V^* kommt es jedoch zu Abweichungen von dieser Vorgehensweise. Dies ist nicht optimal aber vertretbar, da in den 1000 untersuchten Episoden mit insgesamt 1869 Schritten nur 63 Schritte mehr gegangen werden als benötigt (siehe auch Tabelle 4.1).

Um einen zusätzlichen Eindruck vom Ablauf des Lernverfahrens zu erhalten, ist es möglich, die Entwicklung einer Schätzung für V^* durch Beobachtung der Approximation $\tilde{V}$ zu erhalten. Diese Schätzung man z.B. durch Anwendung des TD(λ)-Algorithmus gewinnen (siehe Kapitel 2.6). Dieser läuft parallel und unabhängig vom angewandten Lernverfahren



(a)



(b)

Abbildung 4.10: Ergebnis des Lernens mit kontinuierlichen Aktionen nach Durchführung von 20000 Episoden unter Verwendung von radial-basierten Zustandsfeatures nach (4.19) und zyklisch radial-basierten Aktionsfeatures nach (4.27). Beliebige Richtungen sind hier möglich: $\varphi \in [0, 2\pi[$. Parameter siehe Text in Kapitel 4.8. Wie auch für Abbildung 4.9 wurde ein gleichförmiges Gitter von 40x40 Zuständen verwendet, um Werte der Funktion $\max_\varphi Q(x, y, \varphi)$ zu berechnen. (a) Greedy-Aktionen $\varphi_\mathrm{g} = \arg\max_\varphi \tilde{Q}(x, y, \varphi)$ und deren Werte $\tilde{Q}(x, y, \varphi_\mathrm{g})$. Die Winkel, die den Greedy-Aktionen entsprechen, sind durch Pfeilspitzen symbolisiert. (b) Differenz zwischen $V^*(x, y)$ und $\max_\varphi \tilde{Q}(x, y, \varphi)$. Vergleiche mit Abb. 4.3.

und erlaubt eine Einschätzung, in wie weit der Zustandsraum durch die gemachten Erfahrungen abgedeckt ist.

5 Zusammenfassung und Ausblick

Das *Reinforcement Learning (RL)* ist ein Gebiet mit hohen Ansprüchen: Nur mit wenig oder gar keinen Vorgaben soll ein Agent allein durch die Wahrnehmung numerischer Werte, den Belohnungen, eine Strategie bestimmen, mit der sich eine optimale oder zumindest relativ gute Gesamtbelohnung ergibt. Um diese Ansprüche zu erfüllen, braucht es ausgefeilte Methoden und relativ viel Rechenzeit, um aus den gemachten Erfahrungen das nötige Wissen zu kondensieren ohne auf weitere Kenntnisse über das zu steuernde System zurückzugreifen. Für verschiedene Steuerungsaufgaben gibt es unterschiedliche Methoden, die am besten zur Suche nach einer optimalen Politik geeignet sind, sodass sich bei neuen Steuerungsaufgaben die Frage stellt, welcher Algorithmus angewendet werden soll. In dieser Arbeit wurden Konzepte vorgestellt, die eine systematische Suche nach der Antwort auf diese Frage unterstützen und die sich prinzipiell auf den allgemeinen Umgang mit wissenschaftlichen Daten übertragen lassen. Darüber hinaus wurde in neuer Algorithmus entwickelt, um RL-Probleme mit kontinuierlichen Aktionen zu lösen. In diesem Kapitel werden Ideen zur Erweiterung dieser Konzepte vorgestellt.

5.1 Motivation und Ergebnisse

In der Einleitung dieser Arbeit wurde anhand der in der Literatur am häufigsten verwendeten Steuerungsaufgaben deutlich gemacht, dass für Steuerungsprobleme mit kontinuierlichen Zustandsräumen bislang keine geeignete Basis zur Verfügung stand, um die Effektivität von Algorithmen systematisch zu untersuchen. Eine solche Basis kann zum Einen durch Steuerungsaufgaben gebildet werden, für die eine optimale Lösung bekannt ist. Zum Anderen kann sie auch geeignete Methoden beinhalten, die dazu dienen verschiedene Algorithmen zu vergleichen. Eine bekannte Lösung ist auch hilfreich für den Fall, dass eine Implementierung eines Algorithmus nicht den gewünschten Lerneffekt zeigt. In einer solchen Situation ist es nützlich, über möglichst viele Informationen der gesuchten Lösung zu verfügen, um gezielte numerische Vergleiche während der Ausführung der Algorithmen zu ermöglichen. Ziel ist es, durch Nutzung dieses Wissens die Neuentwicklung von Algorithmen zu begleiten um eventuelle Kinderkrankheiten frühzeitig erkennen zu können. Das Potential der Weiterentwicklung des Reinforcement Learning erschöpft sich nicht allein in der Entwicklung neuer Algorithmen zur Suche nach optimalen Politiken; es gilt auch, die *systematische Entwicklung* von Reinforcement Learning-Algorithmen zu fördern.

Um hier Beiträge zu leisten, wurde im Rahmen dieser Arbeit ein Steuerungsproblem entwickelt, für das die optimale Lösung sowohl für diskrete als auch für kontinuierliche Aktionen bekannt ist, das *Richtungssucher-Problem* (siehe Kapitel 4). Hierbei wurden die optimalen Valuefunktionen V^* und Q^* in analytischer Form zusammen mit einem Verfahren

zur Bestimmung einer optimalen Politik angegeben. Die Steuerungsaufgabe zeichnet sich außerdem dadurch aus, dass die Wahl von kontinuierlichen Aktionen im Allgemeinen eine höhere Gesamtbelohnung ermöglicht als die Verwendung von diskreten Aktionen. Diese Eigenschaften machen das Problem nicht nur zum Vergleich verschiedener Algorithmen, sondern auch für didaktische Zwecke interessant.

Um schließlich einen Vergleich verschiedener Algorithmen zu ermöglichen, wurde in Abschnitt 4.6 ein Maß eingeführt, das die Kenntnis der optimalen State-Value-Funktion V^* ausnutzt: Der *Umwegskoeffizient*. Er ist ein statistisches Maß zur Quantifizierung des Umwegs, den der Agent im Richtungssucher im Vergleich zur optimalen Lösung geht; er eignet sich jedoch prinzipiell für alle Steuerungsprobleme, für die die Funktion V^* bekannt ist.

Der Umwegskoeffizient wurde schließlich in Abschnitt 4.7 dazu benutzt, numerische Lösungen zur Suche nach einer optimalen Politik für das Richtungssucherproblem zu bewerten. Dabei konnte quantitativ die Einschätzung bestätigt werden, dass Featurevektoren bei der Approximation der Action-Value-Funktion umso bessere Resultate erzielen, je besser sie an die gegebenen Zustands- und Aktionsräume angepasst sind. Für das Richtungssucherproblem mit diskreten Aktionen ergibt sich auch numerisch die optimale Lösung, da die Generalisierung der Zustände mit einfachen Mitteln optimal möglich war.

Die Suche nach optimalen Strategien für kontinuierliche Aktionsräume mit Methoden des RL ist Gegenstand aktueller Forschung. Die Diskussion in Abschnitt 1.3.3 machte deutlich, dass in der Literatur eine Vielzahl von Methoden existiert, welche teilsweise sehr unterschiedlichen Konzepten folgen. Dabei werden oft spezielle Steuerungsprobleme eingeführt, die keinen direkten Vergleich der Methoden erlauben.

Im Rahmen dieser Arbeit wird eines weiteres Verfahren zur Suche nach einer optimaler Politik für kontinuierliche Aktionen beigesteuert. Diese ist eine Modifikation des bekannten Sarsa(λ)-Algorithmus für diskrete Aktionen, mit dem Unterschied, dass die kontinuierlichen Aktionen wie auch schon die Zustände durch Featurevektoren generalisiert werden (siehe Abschnitt 2.7.4). Die dabei verwendete Notation ermöglicht gleichzeitig weiterhin die Nutzung diskreter Aktionen. Diese Art und Weise mit kontinuierlichen Aktionen umzugehen ist leicht zu implementieren und kann einen Baustein darstellen, um kontinuierliche Aktionen auch in anderen Algorithmen einzuführen. Die Wirkungsweise der Methode wurde in Abschnitt 4.8 anhand des Richtungssucherproblems demonstriert.

Bei der Lösung des Richtungssucherproblems für kontinuierliche Aktionen wurde eine weitere Neuerung eingeführt: Die Bildung von Featurevektoren für Aktionen, bei der näherungsweise die Periodizität der zu approximierenden Action-Value-Funktion berücksichtigt werden kann (Abschnitt 4.8.1). Dieses Vorgehen hat einen signifikant positiven Einfluss auf die Qualität der gefundenen Lösung, wie in Abschnitt 4.8.2 diskutiert wurde.

Die Untersuchung der Reinforcement Learning-Algorithmen und des Richtungssucherproblems wurde mit selbstentwickelter Software durchgeführt, die sich aus einem Framework zur Lösung von RL-Problemen, der Definition der Steuerungsprobleme und einigen Werkzeugen zur Visualisierung und Analyse der Daten aus den Lernprozessen zusammensetzt. Diese Software wurde unter dem Gesichtspunkt von Konzepten entwickelt, deren Berücksichtigung allgemein im Umgang mit wissenschaftlichen Daten auf elektronischen Rechnenanlagen zu einer effizienten Arbeitsweise führt. Dies wurde im Abschnitt 3.1 dis-

kutiert. Dabei stellte sich heraus, dass unnötige Abhängigkeiten zu einer Behinderung der wissenschaftlichen Arbeit führen können. Solche Abhängigkeiten können beispielsweise durch die Art der physikalischen Repräsentation von Daten entstehen oder dadurch, dass die Daten nicht selbsterklärend sind. Eine *nachhaltige Datenspeicherung* zusammen mit der Möglichkeit, verschiedene Arbeitsschritte wahlweise automatisiert oder interaktiv durchführen zu können, löst im Allgemeinen dieses Problem. Die Umsetzung dieser Konzepte im Rahmen dieser Arbeit wurde in Abschnitt 3.2 erläutert. Dazu gehört u.a. die Nutzung des wissenschaftlichen Dateiformats *HDF5* (stellvertretend für andere Formate mit ähnlichen Eigenschaften), die konzeptionelle Trennung von Problem und Lösungsmethode, aber auch die automatische Analyse von Simulationsergebnissen. Letztere lässt sich prinzpiell auch in einen experimentellen Kontext übertragen und kann auf diese Weise den Informationsaustausch in einer Arbeitsgruppe vereinfachen und sichern. Zum Zeitpunkt der Abgabe dieser Arbeit treten Ideen zur gemeinsamen Nutzen wissenschaftlicher Daten immer mehr in den Vordergrund und werden deshalb bewusst durch die Deutsche Forschungsgemeinschaft gefördert (DFG 2006, 2008, 2009).

5.2 Direkte Erweiterungen

Viele Lösungsverfahren für Reinforcement Learning-Probleme sind auf die Verbesserung einer Politik angewiesen, welche sich auf die (näherungsweise) Auswertung eines Argmax-Operators, angewendet auf eine Action-Value-Funktion, stützen. Dies gilt auch für das in dieser Arbeit für kontinuierliche Aktionen vorgestellte Verfahren. Die im Abschnitt 2.7.4 beschriebene Methode zur Aktionsselektion ist relativ aufwendig und nutzt das Wissen über die Linearität der Action-Value-Funktion $\tilde{Q}$ und über die Berechnung der Feature-Vektoren nicht aus. Diesbezüglich liegt es nahe, ein Optimierungsverfahren zu suchen, das bei gleicher Genauigkeit schneller arbeitet. Zudem kann man dabei auch explorative Elemente einschließen, z.B. bei mehreren ähnlich guten Kandidaten nicht immer nur die beste Aktion gewinnen zu lassen, um den Lernprozess nicht zu schnell einfahren zu lassen. Über eine Größe wie den Umwegskoeffizienten aus Abschnitt 4.6 zusammen mit einer Steuerungsaufgabe mit bekannter Lösung wie den Richtungssucher aus Kapitel 4 können verschiedene Vorgehensweisen zur Aktionsselektion miteinander verglichen werden. Dazu sind eventuell auch einige Ideen aus dem Gebiet der *globalen Optimierung* interessant, welche von (Neumaier 2004) kategorisiert und zusammengefasst wurden.

Methodisch gibt es neben der Wahl anderer Methoden zur Aktionsselektion noch viele Möglichkeiten, das Richtungssucherproblem zusammen mit dem Umwegskoeffizienten zum Vergleichen zu nutzen. So wurde in dieser Arbeit lediglich der Sarsa(λ)-Algorithmus als Lernalgorithmus verwendet. Denkbar sind detaillierte Vergleiche z.B. mit dem verwandten Q(λ)-Algorithmus (Watkins und Dayan 1992) oder mit dem ganz verschiedenen KLSPI-Verfahren (Xu et al. 2007).

Auch für die informationstechnologische Infrastruktur aus Abschnitt 3.2 lassen sich einige direkte Erweiterungen formulieren. So kann durch ein einfaches Auszählen die Frequenz, mit dem einzelne Zustände und Aktionen, die den Samplepunkten aus $\mathcal{S}_{\log}$ und $\mathcal{A}_{\log}$ ähnlich sind, gespeichert werden. Diese Information kann dann bei der Bewertung einer Simu-

lation einfließen, beispielsweise als Gewichtung bei der Berechnung eines Abstandsmaßes zwischen Value-Funktionen. Für den Richtungssucher war die Berechnung eines solchen Abstandsmaßes bislang auf Grund der Unstetigkeiten in V^* wenig hilfreich, jedoch können solche Maße möglicherweise für andere Steuerungsprobleme Gewinn bringend eingesetzt werden.

Sinnvolle Erweiterungen der Nutzung des Formats *HDF5* für das Speichern von Lernprozessen zum Reinforcement Learning wären beispielsweise:

- Nachträgliche Speicherung eines Fazits in Form einer Bemerkung, die das Ergebnis anhand von möglichen Erwartungen diskutiert, sowie die
- Speicherung von Referenzen auf Simulationen (z.B. eindeutige Simulationsnamen), deren Ergebnis Ausschlag gebend für die Initiierung der betrachteten Simulation sind.

Auf diese Weise wäre es möglich, den Ablauf der wissenschaftlichen Arbeit systematisch zusammen mit Erwartungen und Ergebnissen zu dokumentieren und in Form eines gerichteten Graphen darzustellen, der die Übersicht bei großer Variabilität in Parametern und Methoden erleichtert, Schlussfolgerungen leichter nachvollziehbar macht und überflüssige Wiederholungen zu vermeiden hilft. Eine solche Vorgehensweise ist wiederum auch auf viele andere wissenschaftliche Disziplinen, bei denen frühere Messungen oder Simulationen Anlass zu neuen Untersuchungen geben, anwendbar, sofern die Daten zentral gespeichert werden.

Eine Erweiterung, betrifft die Granularität der Speicherung der Eingabeparameter, die bisher lediglich zusammengefasst in einer menschenlesbaren Textform abgelegt sind. Hier erlauben es die hierachischen und beschreibenden Möglichkeiten von HDF5, trotz variierender Menge und Art der Parameter diese so speichern, dass ein direkter maschineller Zugriff möglich wird.

5.3 Qualitätssicherung für RL-Algorithmen

Im Rahmen dieser Arbeit wurde die Kenntnis einer optimalen Lösung für den Richtungssucher verwendet, um verschiedene Formen der Generalisierung von Zuständen und Aktionen mit Hilfe des Umwegskoeffizienten miteinander zu vergleichen (siehe Abschnitte 4.7.2 und 4.8.2). Die Kenntnis einer optimalen Lösung eines Steuerungsproblems zusammen mit einem Vergleichsmaß, das diese Lösung nutzt, erlaubt jedoch prinzipiell eine viel weitergehendere Unterstützung bei der Entwicklung von neuen Reinforcement Learning-Algorithmen. In diesem Abschnitt soll die Idee eines Verfahrens zur *Qualitätssicherung* diskutiert werden. Es ist anwendbar, wenn der zu prüfende Algorithmus sich auf eine Approximation $\tilde{Q}$ der Action-Value-Funktion, sowie auf die Auswertung von $\arg\max_a \tilde{Q}(s,a)$ stützt.

Die im Folgenden beschriebene Vorgehensweise erlaubt es sicherzustellen, dass ein solcher Reinforcement Learning-Algorithmus einige grundlegende Fähigkeiten besitzt, die benötigt werden, um komplexe Steuerungsaufgaben zu bewältigen. Dies ist gerade beim Einsatz einer Funktionsapproximation, wie sie für kontinuierliche Zustandsräume erforderlich

ist, eine wertvolle Information. So hat sich herausgestellt, dass selbst wenn ein Algorithmus eine gute Approximation der optimalen Action-Value-Funktion Q^* gefunden hat, es immer noch sein kann, dass die gefundene Politik unerwartet schlechte Ergebnisse liefert (siehe z.B. Sabes 1994).

Um das Verfahren zur Qualitätssicherung zu verwenden, wird ein konkreter Algorithmus und ein Steuerungsproblem mit bekannter optimaler Lösung in Form von V^* und Q^* benötigt. Dazu kann das in Kapitel 4 beschriebene und gelöste Richtungssucherproblem oder aber auch jedes andere Steuerungsproblem mit bekannter optimaler Lösung sein, sofern es zu dem zu testenden Algorithmus passt. Weitere Steuerungsprobleme mit bekannter optimaler Lösung, die im Rahmen dieser Arbeit entwickelt wurden, sind im Anhang B zu finden.

Das Verfahren besteht aus vier einzelnen Schritten, die aufeinander aufbauen:

1. Prüfung der Aktions-Selektion,
2. Prüfung der Approximation,
3. Prüfung auf Stabilität und
4. Prüfung auf Eigenständigkeit in Form der Suche nach einer optimalen Politik ohne Vorwissen.

Die Abbildung 5.1 fasst das vorgeschlagene Verfahren zur Qualitätssicherung bzw. -kontrolle eines Reinforcement Learning-Algorithmus, der auf einer Approximation der Action-Value-Funktion beruht, zusammen. Es wird in den folgenden Abschnitten zusammen mit den zu Grunde liegenden Ideen erläutert. Für den zweiten und dritten Schritt wird ein Vorverarbeitungsschritt benötigt, der in Anhang A beschrieben ist. Dabei geht Vorwissen in Form der optimalen Lösung ein – es handelt sich um das Vorbesetzen der Approximationsfunktion $\tilde{Q}$ durch Q^*.

Prüfen der Aktions-Selektion Die Aufgabe der Aktions-Selektion ist es, aus einer vorliegenden Action-Value-Funktion eine sinnvolle Politik abzuleiten. Die in dieser Arbeit verwendete Methode zur Aktionsselektion sind in Abschnitt 2.7.4 beschrieben. Beginnt der Agent seine Suche ohne jegliches Vorwissen, so können nicht von Anfang an sinnvolle Aktionen in dem Sinne erwartet werden, dass die Gesamtbelohnung maximiert wird. Ist der Lernprozess jedoch schon weit fortgeschritten, so sollte es möglich sein, für den Agenten günstige Aktionen zu wählen. Der Extremfall ist, dass der Agent die optimale Action-Value-Funktion vollständig kennt und darauf seine Entscheidungen gründet. Man erwartet an dieser Stelle, dass der Agent stets eine sehr gute Gesamtbelohnung erzielen kann, sollte er es schaffen, dieses Wissen auszunutzen.

Ist Q^* bekannt, so kann durch die Auswertung von $\arg\max_a Q^*(s,a)$, also durch die direkte Verwendung der optimalen Action-Value-Funktion Q^*, die Wirkung des verwendeten Argmax-Operators getestet werden, ohne dass bereits ein Einfluss der Approximation zu berücksichtigen ist. Dazu ist es sinnvoll eine gewisse Anzahl von Testepisoden für zufällig gewählte Startzustände zu erzeugen, bei denen der Agent Zugriff auf eine numerische Auswertung von Q^* hat. Durch die Auswertung des Umwegskoeffizienten nach (4.10) wird das

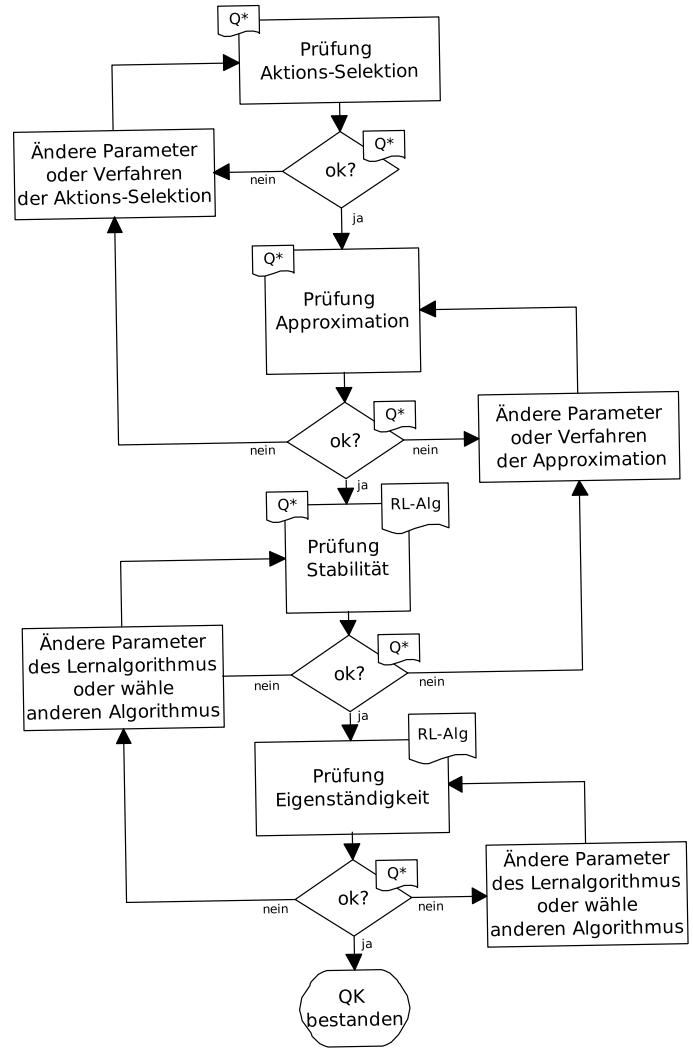


Abbildung 5.1: Verfahren zur Qualitätskontrolle für RL-Algorithmen. An den Prüfungsschritten ist jeweils vermerkt, ob die optimale Lösung und/oder der Lernalgorithmus in die Prüfung eingeht. Falls ein Schritt nicht zufriedenstellend ist, besteht prinzipiell auch immer die Möglichkeit, zum jeweils davor liegenden Schritt zurückzukehren. Mit dem Kürzel „Q^*“ ist markiert, wo überall die Kenntnis über die optimale Lösung eingesetzt wird, das Kürzel „RL-Alg“ zeigt an, wo der Lernalgorithmus eingeht.

Ergebnis in Form von η_{AS} zusammen mit einer Unsicherheit festgehalten. Ist der Wert von η_{AS} zu groß, dann sind die Parameter der verwendeten Methode zur Aktions-Selektion zu überprüfen bzw. gegebenenfalls ist ein anderes Verfahren zu wählen.

Prüfung der Approximation Ist die Prüfung der Aktions-Selektion zufriedenstellend verlaufen, dann ist in einem nächsten Schritt die Approximation $\tilde{Q}$ von Q hinzuzunehmen, sofern eine solche verwendet wird. Dies ist z.B. dann der Fall, wenn kontinuierliche Zustände bzw. Aktionen zur Anwendung kommen und eine tabellarische Repräsentation von Q nicht mehr ausreicht.

Die Grundidee für diesem Schritt stellt sich wie folgt dar: Soll der Agent bei der Suche nach einer optimalen Lösung imstande sein, diese mit den gegebenen Mitteln zur Approximation darzustellen, so sollte eine möglichst gute Approximation der optimalen Lösung sich auch wie eine solche verhalten.

Dazu wird die Approximation $\tilde{Q}$ mittels der Methode der kleinsten Quadrate entsprechend der Beschreibung im Anhang A vorbesetzt. Der Agent absolviert nun wieder eine vorgegebene Anzahl von Testepisoden ausgehend von zufälligen Startzuständen. Die Politik, die er dabei verfolgt, ist diesmal das Ergebnis eines Zusammenwirkens von Approximation *und* Aktions-Selektion; daher sollte die Prüfung der Aktions-Selektion vorgeschaltet werden. Wichtig ist dabei, dass an dieser Stelle noch kein Lernalgorithmus zur Anwendung kommt, dieser ist erst Gegenstand der nachfolgenden Prüfungsschritte. Die Funktion $\tilde{Q}$ bleibt also konstant. Die Bewertung der Testepisoden wird in einem Umwegskoeffizienten η_{Ap} festgehalten. Ist der berechnete Wert hinreichend klein, so gilt die verwendete Kombination aus Approximation und Aktionsselektion als ausreichend um den nächsten Schritt des Verfahrens anzugehen.

Falls der Umwegskoeffizient zu groß ist, sollen die Grundidee und Parameter der Approximation bzw. der Aktionsselektion so lange überdacht werden, bis ein hinreichend guter Wert in diesem Schritt erzielt werden kann oder aber die gewählte Kombination verworfen werden kann. Gegebenenfalls ist die Prüfung der Aktions-Selektion aus dem vorherigen Abschnitt zu wiederholen.

Prüfung auf Stabilität In diesem Punkt kommt innerhalb des Verfahrens zur Qualitätskontrolle erstmalig der Lernalgorithmus zur Anwendung. Die Prüfung auf Stabilität basiert auf dem Gedanken, dass die Repräsentation der optimalen Lösung ein stabiler Fixpunkt bei der Anwendung des Lernalgorithmus ist – sofern der Agent prinzipiell in der Lage ist, diesen selbständig zu finden. Ist der Agent einmal dort angelangt, soll er auch dort verbleiben. Anders gesagt: Ist die Repräsentation der optimalen Lösung durch das gewählte Approximationsverfahren kein stabiler Fixpunkt unter Anwendung des Lernverfahrens, so besteht wenig Aussicht, dass der Agent diese Lösung ohne Vorwissen finden kann.

Wie auch bei der Prüfung der Approximation wird zunächst $\tilde{Q}$ möglichst gut an Q^* angepasst, wie in Anhang A beschrieben. Die anschließenden Testepisoden werden dann mit dem zu testenden Lernalgorithmus durchgeführt, das heißt es werden Änderungen des Parametervektors $\boldsymbol{\theta}$ bzw. allgemein von $\tilde{Q}$ zugelassen. Das Ergebnis wird in einem Umwegskoeffizienten η_{St} zusammengefasst. Ist die Kombination aus Approximation und Aktionsse-

lektion hinreichend gut, so erwartet man von einem brauchbaren Lernalgorithmus, dass η_{St} unter dessen Anwendung des Lernverfahrens kaum größer ist als η_{Ap}. In diesem Fall ist die Kombination aus Approximation, Aktionsselektion und Lernverfahren *stabil*. Ist dies nicht der Fall, können die Parameter des Lernverfahrens geändert werden. Gelingt es auf diese Weise nicht eine stabile Konfiguration zu finden, so ist entweder der Algorithmus fehlerhaft implementiert oder ungeeignet bzw. dessen Kombination mit der verwendeten Approximationsmethode bzw. dem Verfahren zur Aktions-Selektion ist nicht brauchbar. Soll in diesem Fall an dem Lernalgorithmus festgehalten werden, ist gegebenenfalls die Approximation zu überdenken, sodass der vorherige Schritt neu durchlaufen werden muss.

Suche einer Lösung ohne Vorwissen Der letzte Schritt besteht schließlich in der Anwendung des zu testenden Lernalgorithmus ohne Vorwissen, also der eigentlichen Aufgabe des Algorithmus. Da der Agent keine Vorabinformation erhält, werden die Parameter $\boldsymbol{\theta}$ der Approximation $\tilde{Q}$ beliebig gesetzt bzw. derart, dass der Agent dazu ermuntert wird, neue Erfahrungen zu machen (siehe dazu Abschnitt 4.7.2).

Daraufhin wird der zu testende Lernalgorithmus durchlaufen, währenddessen sich $\tilde{Q}$ entwickelt und es wird ein Umwegskoeffizient η_{RL} bestimmt. Falls durch die vorherige Prüfung eine ausreichende Stabilität gegeben ist, erwartet man, dass η_{RL} bei Konvergenz einen ähnlichen Wert wie η_{St} hat. In diesem Fall ist soll die Qualitätskontrolle als bestanden betrachtet werden und der Algorithmus als geeignet, um das betrachtete Problem zu lösen.

Die in dieser Arbeiten präsentierten Ergebnisse konnten ohne Qualitätskontrolle erarbeitet werden. Generell ist davon auszugehen, dass die Anwendung eines solchen Verfahrens die systematische Entwicklung von Reinforcement Learning-Algorithmen fördert.

5.4 Entwicklung einer Testumgebung für RL-Algorithmen

Der in Abschnitt 5.3 besprochene Ansatz zur Qualitätskontrolle kann in der zukünftigen Arbeit zu einer *Testumgebung* für RL-Algorithmen erweitert werden. Dazu gehören die Aufnahme weiterer Steuerungsprobleme mit bekannten Lösungen, die weitere Aspekte abdecken oder einfach komplexere Lösungen haben. Dies können zum Einen weitere deterministische Systeme sein, wie die im Anhang B besprochenen. Zum Anderen sind dafür aber auch Systeme mit stochastischer Systemdynamik interessant. Das *Hopworld-Problem* (Boyan 1999, Xu et al. 2002, 2005, Xu 2006) kann hier als Testproblem für diskrete Zustandsräume dienen - möglicherweise kann man auch den Richtungssucher hier erweitern, sodass auch für kontinuierliche Zustandsräume ein solches System zur Verfügung steht. Ein weiterer Aspekt, den eine Testumgebung für RL-Algorithmen berücksichtigen kann, sind die POMDPs, d.h. *Partially Observable Markov Decision Processes* (Aberdeen 2003, Braziunas 2003, Kreutz 2003). Dies sind Probleme, bei denen der Systemzustand nicht direkt beobachtbar ist. Ein klassisches Beispiel hierfür ist das Tiger-Problem, für das eine optimale Lösung durch Value-Iteration berechenbar ist (siehe z.B. Hoey und Poupart 2005).

Neben der Sammlung von Steuerproblemen mit bekannten optimalen Lösungen ist es auch denkbar, weitere in der Literatur wichtige Steuerprobleme darin aufzunehmen, deren optimale Lösungen man bisher nicht bestimmen konnte, für die aber jedoch gute Lösun-

gen bekannt sind. Ein Beispiel ist das Mountain-Car-Problem, das in Abschnitt 2.7.3 dieser Arbeit beschrieben ist. Die Einführung eines Vergleichmaßstabs, der statt von einer optimalen Lösung von der bisher besten bekannten Lösung Gebrauch macht, kann eine Basis bilden, um auch hier vorhandene Algorithmen besser miteinander vergleichen zu können als es bisher in der Literatur der Fall war.

Langfristig kann es sich als sinnvoll erweisen, *Algorithmen mit zugehörigen Parametersätzen* aufzunehmen, die die Qualitätskontrolle aus Abschnitt 5.3 bestehen. Auf diese Weise kann eine Sammlung aufgebaut werden, die zum Einen Anwendern einen Pool von Methoden bietet, deren Qualität anhand des Anwendungsfalls numerisch bemessen werden kann. Zum Anderen ermöglicht eine solche Sammlung Wissenschaftlern, die sich mit der Entwicklung von neuen Algorithmen beschäftigen, den direkten Vergleich mit bereits bekannten Methoden. So werden Fortschritte direkt sichtbar.

Die Bedeutung eines solchen Ansatzes wird an der Tatsache deutlich, dass die *RL-Community* (The RL Community 2009a) kurz vor Abschluss dieser Arbeit das Projekt *RL-Logbook* ins Leben gerufen hat (The RL Community 2009c). Es handelt sich dabei um ein ein elektronisches, gemeinschaftlich genutzes Laborbuch, in dem zentral zu verschiedenen Algorithmen und Steuerungsproblemen Ergebnisse gesammelt werden sollen. Interessant sind in diesem Zusammenhang die diskutierten Ideen zum Aufbau und Nutzen einer solchen Datensammlung. Eine davon ist die Verwendung des *Cloud-Computing*-Konzepts (Hayes 2008), um brachliegende Rechenleistung zur systematischen Lösung von Reinforcement Learning-Problemen zu nutzen.

Während der Erstellung dieser Arbeit hat sich innerhalb der *RL-Community* parallel ein Framework entwickelt, das große Unterstützung durch namhafte Wissenschaftler im Bereich des Reinforcement Learning erfährt: *RL-Glue* (The RL Community 2009b). Es bildet ähnlich wie das im Rahmen dieser Arbeit entwickelte Framework eine einheitliche Schnittstelle zwischen RL-Algorithmen und Steuerungsproblemen. Es sieht zum Zeitpunkt der Erstellung der Arbeit jedoch noch keine definierte Schnittstelle für bekannte optimale Lösungen vor. Interessant an RL-Glue ist die dazugehörige standardisierte Form der Beschreibung der wichtigsten Eigenschaften eines Steuerungsproblems, wie z.B. ob das Problem episodisch ist oder nicht und wie die Zustands- und Aktionsräume beschaffen sind. Durch eine Erweiterung von RL-Glue mit den in dieser Arbeit vorgestellten Konzepten kann das RL-Logbook direkt von Steuerungsproblemen mit bekannten optimalen Lösungen profitieren. Auf diese Weise können Wissenschaftler mit relativ wenig Aufwand neue Algorithmen mit einem Verfahren zur Qualitätskontrolle, wie es in Abschnitt 5.3 beschrieben ist, automatisch auf Grundfunktionalitäten prüfen lassen um so konzeptionelle Fehler frühzeitig entdecken und beheben zu können.

A Vorbesetzen von $\tilde{Q}$ durch Q^*

Ist die optimale Lösung zu einem Steuerungsproblem bekannt, so kann diese zur Vorbereitung des zweiten und dritten Schrittes der Qualitätskontrolle (siehe Abschnitt 5.3) genutzt werden, um die Approximation $\tilde{Q}$ möglichst „gut“ an Q^* anzupassen. Dies kann mit Hilfe der *Methode der kleinsten Quadrate* durchgeführt werden und wird im Folgenden beschrieben.

Sei $\tilde{Q}(s, a; \boldsymbol{\theta})$ eine Näherung der optimalen Action-Value-Funktion Q^*, die durch den Parametervektor $\boldsymbol{\theta}$ bestimmt ist. Sei $\mathcal{S}_{\chi^2}$ eine ausgewählte Menge von Zuständen sowie $\mathcal{A}_{\chi^2}$ eine ausgewählte Menge von Aktionen zu denen Werte von Q^* vorgegeben werden sollen. Die Verteilung und Anzahl $|\mathcal{S}_{\chi^2}|$ bzw. $|\mathcal{A}_{\chi^2}|$ der Elemente von $\mathcal{S}_{\chi^2}$ bzw. $\mathcal{A}_{\chi^2}$ legen fest, wie gut die Approximation an die optimale Lösung angepasst werden kann. Mit

$$M' = |\mathcal{S}_{\chi^2}| \cdot |\mathcal{A}_{\chi^2}|, \; M = M' + |\mathcal{S}_{\chi^2}| \tag{A.1}$$

wird nun die optimale Action-Value-Funktion Q^* verwendet, um die M Trippel

$$(s_k, a_l, Q^*(s_k, a_l)), k = 1, \ldots, |\mathcal{S}_{\chi^2}|, \; l = 1, \ldots, |\mathcal{A}_{\chi^2}|, \tag{A.2}$$

$$(s_k, \pi^*(s_k), Q^*(s_k, \pi^*(s_k))), k = 1, \ldots, |\mathcal{S}_{\chi^2}|, \tag{A.3}$$

zu berechnen. Die Aktion $\pi^*(s_k) = \arg\max_a Q^*(s_k, a)$ steht dabei für eine der möglichen optimalen Aktionen im Zustand s_k. Um die Notation zu vereinfachen werden diese M Trippel als $(\hat{s}_i, \hat{a}_i, \hat{Q}_i^*)$ mit $i = 1, \ldots, M$ bezeichnet.

Mit der Vorgabe dieser Werte der optimalen Action-Value-Funktion und gegebenenfalls M Gewichten $w_i > 0$ kann nun für jeden Parametervektor $\boldsymbol{\theta}$ die Abweichung

$$\chi^2(\boldsymbol{\theta}) = \sum_{i=1}^{M} w_i \left(\tilde{Q}(\hat{s}_i, \hat{a}_i; \boldsymbol{\theta}) - \hat{Q}_i^* \right)^2 \tag{A.4}$$

bestimmt werden. Hier kann es sinnvoll sein, als Gewichte w_i jeweils eine Schätzung der Wahrscheinlichkeit einzufügen, mit der die Kombination (s_i, a_i) auftaucht. Durch Anwendung eines Minimierungsverfahren wie z.B. des von Levenberg und Marquardt (siehe z.B. Press et al. 1996) kann dann ein Parametervektor $\boldsymbol{\theta}^*$ zur Bestimmung einer bestmöglichen Näherung $\tilde{Q}(s, a; \boldsymbol{\theta}^*)$ an Q^* gesucht werden:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \chi^2(\boldsymbol{\theta}). \tag{A.5}$$

Im Fall einer linearen Approximation wie in (2.66) mit einem Feature-Vektor $\boldsymbol{\phi}(s, a)$ kann der zugehörige Parametervektor $\boldsymbol{\theta}$ durch Lösung eines linearen Gleichungssystems

bestimmt werden. Sind die Trippel $(\hat{s}_i, \hat{a}_i, \hat{Q}_i^*)$, $i = 1, \ldots, M$ vorgegeben, definiert man zunächst die Matrix

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{\phi}(\hat{s}_1, \hat{a}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\hat{s}_2, \hat{a}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\hat{s}_M, \hat{a}_M)^{\mathrm{T}} \end{pmatrix} \tag{A.6}$$

und einen Vektor

$$\boldsymbol{b} = \begin{pmatrix} \hat{Q}_1^* \\ \hat{Q}_2^* \\ \vdots \\ \hat{Q}_M^* \end{pmatrix}. \tag{A.7}$$

Sei $\boldsymbol{W}$ eine Diagonalmatrix mit den Elementen $W_{ii} = w_i$. Der optimale Parametervektor $\boldsymbol{\theta}^*$ errechnet dann sich durch Lösung der Normalgleichungen

$$\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{A}\right)\boldsymbol{\theta}^* = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}\,\boldsymbol{b}. \tag{A.8}$$

Die Matrix $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{A}$ ist symmetrisch und positiv definit. Zur numerischen Lösung des Systems (A.8) existieren Verfahren, die diese besonderen Eigenschaften ausnutzen (siehe auch Press et al. 1996).

B Weitere Steuerungsprobleme mit bekannten Lösungen

Für die Durchführung der ersten drei Schritte der Qualitätskontrolle bzw. für die Berechnung des Umwegskoeffizient wird ein Steuerproblem benötigt, für das die optimale Lösung bekannt ist. Die Art des Steuerungsproblems richtet sich nach den Fähigkeiten des zu testenden Algorithmus.

In Kapitel 4 wurde der Richtungssucher diskutiert, für den eine optimale Lösung bekannt ist. Der Zustand hat dabei zwei kontinuierliche Variablen, die Koordinaten x und y. Die Aktion, die Wahl einer Richtung, ist entweder diskret oder kontinuierlich.

Dieses sind bestimmte Bedingungen für einen RL-Algorithmus und es ist u.U. sinnvoll, diesen auch unter einfacheren oder schwierigeren Bedingungen zu testen. Im Rahmen dieser Arbeit wurden dazu noch zwei weitere Steuerungsprobleme mit optimaler Lösung entworfen, die in den folgenden Kapiteln beschrieben und diskutiert werden.

B.1 Der Mittensucher

Das *Mittensucher-Problem* ist dem Richtungssucher sehr ähnlich, es beschränkt sich jedoch auf einen eindimensionalen Zustandsraum und ist damit konzeptionell einfacher. Der Zustandsraum kann diskret oder kontinuierlich gewählt werden, der Aktionsraum wird auf diskrete Aktionen beschränkt.

Der Agent befindet sich beim Mittensucher-Problem an einer Position s auf der reellen Achse, d.h. der Zustand wird einfach durch eine reelle Zahl beschrieben. Die Aufgabe des Agenten besteht darin, durch Bewegung nach rechts oder links die Position 0 (die "Mitte") zu erreichen bzw. eine bestimmte Distanz ϵ zu unterschreiten. Die Menge der Positionen, die der Agent einnehmen kann, ist entweder diskret, wobei die 0 enthalten sein soll, oder kontinuierlich, begrenzt durch eine maximale und eine minimale Position.

Die Aktion a ist wiederrum eine eindimensionale Größe, die einfach die Weite des Schritts bestimmt, die der Agent vorwärts ($a > 0$) oder rückwärts geht ($a < 0$). Die neue Position des Agenten ergibt sich also schlicht durch

$$s' = s + a.$$

Die Menge der möglichen Aktionen kann dabei prinzipiell wiederrum diskret oder kontinuierlich sein, die folgenden Betrachtungen beschränken sich zunächst auf diskrete Aktionen.

Der Agent startet bei irgendeiner Position s_0. Die Episode endet, wenn er der Position 0 genügend nahe kommt, d.h. $|s_T| \leq \epsilon$ mit einem Abstand ϵ, der kleiner oder gleich der typischen Schrittweite ist. Das System hat einen *terminalen* Zustand erreicht. Für jede Aktion

erhält der Agent eine "Belohnung", die in den meisten Fällen eine Bestrafung ist:

$$r(s,a,s') = \begin{cases} 1 & \text{für } s' \text{ terminal,} \\ -1 & \text{sonst.} \end{cases} \tag{B.1}$$

Für diskrete Zustände ist die Abbruchbedingung für eine Episode gleichbedeutend mit dem genauen Erreichen der Position 0.

Diskrete Zustände und diskrete Aktionen

Sei $s \in \{-6,-5,\ldots,0,\ldots,6\}$ und $a \in \{-1,0,1\}$. Unter Berücksichtigung der Belohungsfunktion (B.1) ist die (optimale) Strategie π^* naheliegend:

$$a^*(s) = \begin{cases} +1 & \\ 0 & \text{für} \\ -1 & \end{cases} \begin{array}{l} s<0, \\ s=0, \\ s>0. \end{array} .$$

Die Bewegungsgleichungen seien weiterhin so modifiziert, dass stets $|s| \leq 6$ gewährt bleibt, d.h. der Agent bleibt bei $s = -6$ bzw. $s = 6$ stehen, sollte ein Schritt ihn über den Rand hinaus führen.

Ausgehend von der Position s_0 benötigt der Agent mit dieser Vorgehensweise s_0 Schritte, bis er die Position 0 erreicht. Verzichtet man auf ein Discounting, d.h. es gilt $\gamma = 1$, so ist die zu erwartende Gesamtbelohnung gleich der optimalen State-Value-Funktion $V^*(s_0)$:

$$V^*(s_0) = \begin{cases} +1 & \text{für } |s_0| \leq 1, \\ 2-|s_0| & \text{sonst.} \end{cases} \tag{B.2}$$

Es soll hier stets mindestens eine Aktion ausgeführt werden, also auch für $s_0 = 0$ – hier ist die optimale Aktion $a_0 = 0$.

Zu den anderen Werten für s_0: Startet der Agent beispielsweise bei $s_0 = -5$, so benötigt er vier Schritte bis zur Position -1 mit der entsprechenden Bestrafung $4 \cdot (-1)$, bis er schließlich vom Schritt von -1 auf 0 eine Belohnung von $+1$ erhält. Insgesamt ergibt sich ein Gesamtreward von $2 - |-5| = -3$.

Die zugehörige Action-Value-Funktion bestimmt sich zu

$$Q^*(s,a) = r(s,a,s') + \begin{cases} 0 & \text{für } s' \text{ terminal,} \\ V^*(s') & \text{sonst.} \end{cases} ,$$

da der Discountfaktor γ gleich 1 ist. Dabei wird $V^*(s')$ entsprechend (B.2) ausgewertet. Führt man immer mindestens eine Aktion aus und beendet die Episode gegebenenfalls, falls $s' = 0$, so hat $Q^*(s,a)$ die Funktionswerte wie in Tab. B.1 aufgelistet.

Die Aktionswertfunktion ist für eine *beliebige* Strategie bei Wahl von $\gamma = 1$ nur dann endlich, wenn man die maximale Episodendauer $T_{\max}$ begrenzt. Ansonsten wäre es beispielsweise denkbar, an einer nicht-terminalen Position zu verharren, indem stets die Aktion $a = 0$ gewählt wird, wobei sich die Belohnungen unbegrenzt aufsummierten.

s	$V^*(s)$	$Q^*(s,-1)$	$Q^*(s,0)$	$Q^*(s,+1)$
-6	-4	-5	-5	-4
-5	-3	-5	-4	-3
-4	-2	-4	-3	-2
-3	-1	-3	-2	-1
-2	0	-2	-1	0
-1	1	-1	0	1
0	1	0	1	0
1	1	1	0	-1
2	0	0	-1	-2
3	-1	-1	-2	-3
4	-2	-2	-3	-4
5	-3	-3	-4	-5
6	-4	-4	-5	-5

Tabelle B.1: Werte der optimalen Valuefunktionen $V^*(s)$ und $Q^*(s,a)$ für das Mittensucher-Problem mit diskreten Zuständen und diskreten Aktionen, d.h. $s \in \{-6,\ldots,0,\ldots,6\}$ und $a \in \{-1,0,1\}$.

Kontinuierliche Zustände und diskrete Aktionen

Die Lösung des Mittensucher-Problems kann auch für kontinuierliche Zustände formuliert werden, in diesem Fall ist die Position x eine reelle Zahl aus dem Intervall $[-6;6]$. Die Aktionen sollen weiterhin diskret und auf die Möglichkeiten $a \in \{-1,0,1\}$ beschränkt sein. Es sei $\epsilon = 0.5$ der terminale Abstand, der zur Beendigung einer Episode führt. Auf diese Weise kann von einer beliebigen Startposition ein terminaler Zustand erreicht werden.

Die Tabelle B.2 zeigt die Werte der optimalen Value-Funktionen V^* und Q^* zusammen mit optimalen Aktionen. Es gibt mehrere gleichwertige optimale Strategien, was in der Wahl der terminalen Positionen begründet ist.

An den Stellen $x = 0.5$ und $x = -0.5$ ist die optimale Aktion mehrdeutig.

Diskussion

Das Mittensucherproblem eignet sich wie der Richtungssucher ebenfalls zu didaktischen Zwecken und zum Test von Algorithmen für diskrete Aktionen. Es wurden im Rahmen dieser Arbeit auch Versuche mit dem Mittensucher und kontinuierlichen Aktionen durchgeführt, bei denen die Schrittweite eine reelle Zahl zwischen 0 und 1 ist. Die sich ergebenden optimalen Politiken sind jedoch praktisch identisch mit denen für diskrete Aktionen, daher ist das Mittensucher-Problem für kontinuierliche Aktionen weniger interessant als das Richtungssucher-Problem. Für einen ersten Test eines Algorithmus zur Suche einer optimalen Strategie mit diskreten Aktionen ist es aber durchaus geeignet.

s	$V^*(s)$	$Q^*(s,-1)$	$Q^*(s,0)$	$Q^*(s,+1)$	$a^*(s)$
$[-6;-5.5[$	-4	-5	-5	-4	+1
$[-5.5;-4.5[$	-3	-5	-4	-3	+1
$[-4.5;-3.5[$	-2	-4	-3	-2	+1
$[-3.5;-2.5[$	-1	-3	-2	-1	+1
$[-2.5;-1.5[$	0	-2	-1	0	+1
$[-1.5;-0.5[$	1	-1	0	1	+1
-0.5	1	0	1	1	0,+1
$]-0.5;0.5[$	1	0	1	0	0
0.5	1	1	1	0	0,-1
$]0.5;1.5]$	1	1	0	-1	-1
$]1.5;2.5]$	0	0	-1	-2	-1
$]2.5;3.5]$	-1	-1	-2	-3	-1
$]3.5;4.5]$	-2	-2	-3	-4	-1
$]4.5;5.5]$	-3	-3	-4	-5	-1
$]5.5;6]$	-4	-4	-5	-5	-1

Tabelle B.2: Werte der optimalen Valuefunktionen $V^*(s,a)$ und $Q^*(s,a)$ und die optimalen Aktionen für den Mittensucher mit kontinuierlichen Zuständen und diskreten Aktionen, d.h. für $s \in [-6;6]$ und $a \in \{-1,0,1\}$. Zur Festlegung der terminalen Zustände wurde $\epsilon = 0.5$ gewählt, damit von jeder Startposition eine Zielposition erreicht werden kann. An den Stellen $x = 0.5$ und $x = -0.5$ ist die optimale Aktion mehrdeutig.

B.2 Der Dispatcher

Der *Dispatcher* ist ein einfaches Resourcenverteilungsproblem, dass als ein klassisches Problem der optimalen Steuerung für kontinuierliche Zeiten formuliert ist und mit Hilfe des Pontryagin'schen Maximumsprinzips (Bryson und Ho 1975) gelöst werden kann. Das Problem weist drei kontinuierliche Zustandsvariablen und zwei kontinuierliche Aktionsvariablen auf. Die Übertragung des Problems auf diskrete Zeiten machen es für RL-Algorithmen zugänglich.

Beschreibung der Steuerungsaufgabe

Der Agent ist hier eine entscheidene Instanz in einem Computernetzwerk, das nur über zwei verschiedene Arten von Rechnern verfügt: Eine erste Gruppe mit der Rechengeschwindigkeit 1 und eine zweite Gruppe mit der doppelten Geschwindigkeit 2. Dies sind die *Ressourcen*.

Es soll eine festgelegte Anzahl von Aufgaben (*Jobs*) durch diese Ressourcen bearbeitet werden. Die Ressourcen sind begrenzt, d.h. wenn alle Rechner ausgelastet sind, können gleichzeitig keine zusätzlichen Aufgaben mehr parallel abgearbeitet werden. Der Agent muss zu jedem Zeitpunkt eine Entscheidung treffen, die beinhaltet, wieviele der noch zu erledigenden Aufgaben jeweils den verschiedenen Typen von Ressourcen zugeordnet werden.

In der Literatur zur klassischen optimalen Steuerung ist es üblich, den Zustand mit dem Symbol x und die Aktion mit u zu bezeichnen. Der Zustand

$$x(t) = (j(t), p_1(t), p_2(t))$$

setzt sich zusammen aus der Mengenangabe $j(t) \in \mathbb{R}$ der momentan zum Zeitpunkt t noch zu erledigenden Jobs und der Mengenangaben $p_i(t) \in \mathbb{R},\ i = 1, 2$ der momentan arbeitenden Ressourcen vom Typ i. Die Mengenangaben seien so skaliert, dass gilt: $0 \leq j$, $0 \leq p_1 \leq R_1$ und $0 \leq p_2 \leq R_2$. Dabei sind R_1 und R_2 Konstanten, die eine obere Begrenzung der verfügbaren Ressourcen markieren.

Ressourcen vom Typ 2 sind doppelt so schnell wie Ressourcen vom Typ 1. Ausgehend von einem Anfangszustand

$$(j_0, p_{10}, p_{20}) = (j(0), p_1(0), p_2(0))$$

entwickelt sich das System streng deterministisch gemäß

$$\frac{dj}{dt} = -p_1(t) - 2p_2(t), \tag{B.3a}$$

$$\frac{dp_1}{dt} = d_1(t), \tag{B.3b}$$

$$\frac{dp_2}{dt} = d_2(t), \tag{B.3c}$$

wobei $d_1(t)$ und $d_2(t)$ die Kontrollgrößen sind und zusammen die Aktion $u = (d_1, d_2)$ des Agenten bilden. Eine Episode endet zum Zeitpunkt T, in dem alle Jobs abgearbeitet sind: $j(T) = 0$.

Die Kontrollvariablen $d_1(t)$ und $d_2(t)$ sollen so gewählt werden, dass die Kosten

$$J(T) = \int_0^T dt\, L(t), \text{ mit } L(t) = 1 + d_1^2(t) + d_2^2(t), \tag{B.4}$$

minimiert werden unter den Nebenbedingungen

$$j(T) = 0, \tag{B.5a}$$
$$d_1(T) = d_2(T) = 0, \tag{B.5b}$$
$$0 \leq p_i(t) \leq 1 \ \forall t. \tag{B.5c}$$

Diese hier als *Dispatcher-Problem* bezeichnete Steuerungsaufgabe kann mit Mitteln der optimalen Steuerung gelöst werden. Dabei kommt das *Pontryagin'sche Maximumsprinzip* zur Anwendung (siehe z.B. Bryson und Ho 1975, Paulus 1992). Im folgenden Abschnitt wird die Lösung präsentiert, ohne auf die Berechnung weiter einzugehen.

Lösung des zeitkontinuierlichen Systems

Für einen gegebenen Anfangszustand (j_0, p_{10}, p_{20}) und eine gegebene Endzeit T erhält man die Systemdynamik

$$p_1(t) = -\frac{1}{4}\lambda_1 t^2 + \frac{1}{2}\lambda_1 T t + p_{10}, \tag{B.6a}$$
$$p_2(t) = -\frac{1}{2}\lambda_1 t^2 + \lambda_1 T t + p_{20}, \tag{B.6b}$$
$$j(t) = \frac{5}{12}\lambda_1 t^3 - \frac{5}{4}\lambda_1 T t^2 + (p_{10} + 2p_{20})t + j_0 \tag{B.6c}$$

wobei zur Abkürzung die Hilfsgröße

$$\lambda_1 = \frac{6}{5}\frac{j_0 - (p_{10} + 2p_{20})T}{T^3} \tag{B.7}$$

verwendet wurde. Die zugehörige optimale Steuerung bei Endzeit T ist

$$d_1(t) = \frac{\lambda_1}{2}(T - t)\,,\ d_2(t) = 2d_1(t). \tag{B.8}$$

Ausgehend von einem allgemeinen Anfangszustand (j_0, p_{10}, p_{20}) berechnen sich für eine gegebene Endzeit T die Kosten zu

$$J(T, j_0, p_{10}, p_{20}) = T + \frac{5}{12}\lambda_1^2 T^3 = T + \frac{3}{5T^3}\left(j_0 - (p_{10} + 2p_{20})T\right)^2. \tag{B.9}$$

Nun stellt sich noch die Frage, für welche Endzeit die Kosten insgesamt minimal werden. Sei dazu $\beta = p_{10} + 2p_{20}$. Die optimale Endzeit T^*, mit der minimale Kosten erzielt werden, kann im allgemeinen nur numerisch bestimmt werden. Sie muss notwendigerweise

$$\frac{dJ}{dT}(T^*) = 0 \iff \frac{5}{3}(T^*)^4 = \beta^2 (T^*)^2 - 4 j_0 \beta T^* + 3 j_0^2$$

erfüllen. Hinreichend für ein Minimum ist zusätzlich

$$\frac{d^2 J}{dT^2}(T^*) > 0.$$

Für $\beta = 0$ und $j_0 = 1$ ist dies stets erfüllt; für $\beta > 0$ und $j_0 > 0$ muss allgemein

$$T^* > \frac{j_0}{\beta}(3 + \sqrt{3}) \ \vee \ T^* < \frac{j_0}{\beta}(3 - \sqrt{3})$$

gelten. Mit den speziellen Startbedingungen $\beta = 0$, also $p_{10} = p_{20} = 0$, und $j_0 = 1$ führt das auf

$$T^* = \sqrt[4]{9/5} \approx 1.1583.$$

Die minimalen Kosten belaufen sich dann auf $J^* \approx 1.5444$. Weiterhin gilt dann

$$\lambda_1 = \frac{6}{5\,(T^*)^3}, \ d_2(0) = 2\,d_1(0) = \frac{2}{\sqrt{5}}.$$

Diese Werte eignen sich zur Kontrolle bei der Implementierung des Dispatchers bzw. bei der Suche nach einer optimalen Steuerung ausgehend von der Anfangsbedingung $p_{10} = p_{20} = 0$ und $j_0 = 1$. In Abbildung B.1 ist für diesen Fall die optimale Steuerung zusammen mit der resultierenden Entwicklung der Zustandsvariablen grafisch dargestellt.

Die optimale State-Value-Funktion kann durch die negativen Kosten ausgedrückt werden (siehe auch (B.9)):

$$V^*(s) = V^*(j, p_1, p_2) = -J(T^*(j, p_1, p_2), j, p_1, p_2) = -T^* - \frac{3}{5(T^*)^3}\left(j - (p_1 + 2p_2)T^*\right)^2.$$

Für die optimale Action-Value-Funktion gilt dann

$$Q^*(s, a) = r(s, a, s') + \gamma V^*(s').$$

Die Lösung des Dispatcher-Problems kann nicht vollständig analytisch ausgedrückt werden, da zur Bestimmung der Zeit T^* im Allgemeinen nur eine numerische Lösung existiert.

Diskussion

Eine zeitdiskrete Version des Dispatcher-Problems erhält man durch Ersatz von (B.3) durch die Verwendung der Systemdynamik

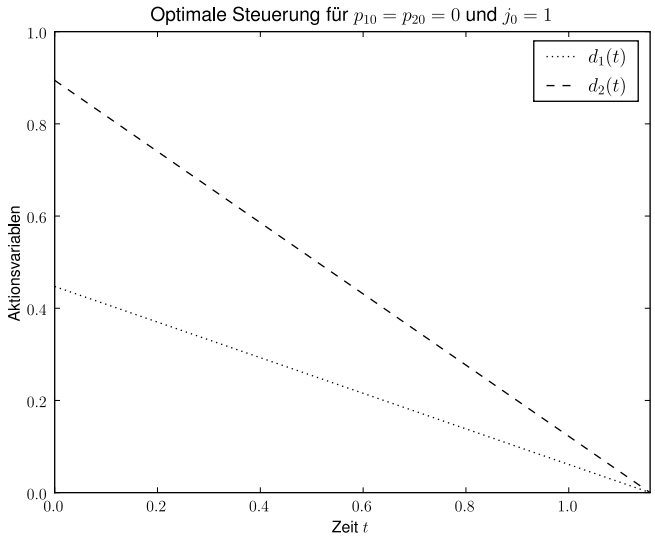
$$p_1(t+1) = p_1(t) + \Delta t\, d_1(t), \tag{B.10a}$$

$$p_2(t+1) = p_2(t) + \Delta t\, d_2(t), \tag{B.10b}$$

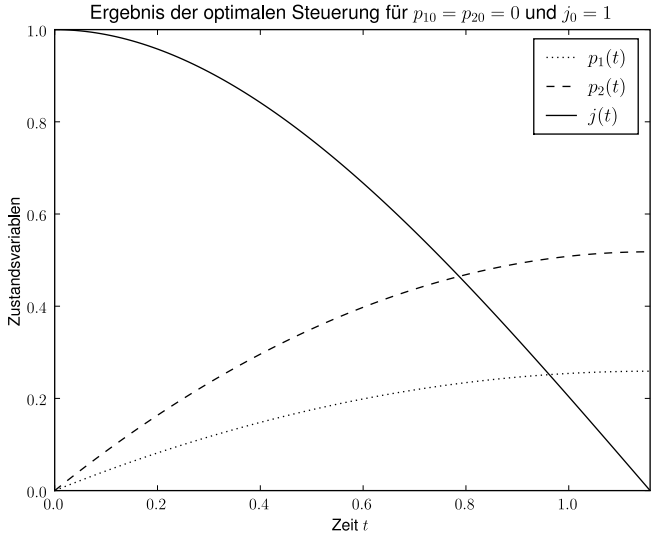
$$j(t+1) = j(t) - \Delta t\ (p_1(t) + 2p_2(t)) - (\Delta t)^2 \left(\frac{d_1(t)}{2} + d_2(t)\right). \tag{B.10c}$$

Als Belohnungsfunktion wurde im Wesentlichen

$$r(j, p_1, p_2, d_1, d_2, j', p_1', p_2') = r(d_1, d_2) = -\Delta t \left(1 + d_1^2 + d_2^2\right) \tag{B.11}$$



(a)



(b)

Abbildung B.1: Systemdynamik und optimale Steuerung für den Dispatcher mit $p_{10} = p_{20} = 0$ und $j_0 = 1$. (a) Die optimale Steuerung als zeitliche Entwicklung der Aktionsvariablen $d_1(t)$ und $d_2(t)$. Sie erfüllt $d_1(T^*) = d_2(T^*) = 0$ mit $T^* = \sqrt[4]{9/5} \approx 1.1583$. (b) Die zeitliche Entwicklung der Zustandsvariablen $p_1(t)$, $p_2(t)$ und $j(t)$. Es gilt $j(T^*) = 0$.

genutzt, vgl. mit der Kostenfunktion (B.4).

Die zeitdiskrete Form des Dispatcher-Problems konnte bislang mit dem in dieser Arbeit beschriebenen Sarsa(λ)-Algorithmus weder für diskrete, noch für kontinuierliche Aktionen zufriedenstellend gelöst werden. Vermutlich ist dies darin begründet, dass für dieses Problem sehr viele gute Lösungen dicht an der optimalen Lösung liegen, sodass es sich schwierig gestaltet ohne Kenntnis der Systemdynamik unter allen lokalen Minima das globale Optimum zu finden. Dadurch wird der Dispatcher zu einem interessanten Steuerungsproblem, wenn es darum geht, mehrere Algorithmen unter diesem Gesichtspunkt miteinander zu vergleichen.

Literaturverzeichnis

[Aberdeen 2003] Aberdeen, Douglas: A (Revised) Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes / National ICT Australia. 2003. – Forschungsbericht

[Albus 1975] Albus, J S.: A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC). In: *Journal of Dynamic Systems, Measurement and Control* 97 (1975), S. 220–233

[Aler et al. 2009] Aler, Ricardo ; Valls, Jose M. ; Camacho, David ; Lopez, Alberto: Programming Robosoccer agents by modeling human behavior. In: *Expert Systems with Applications* 36 (2009), Nr. 2, Part 1, S. 1850 – 1859. – URL http://www.sciencedirect.com/science/article/B6V03-4RF459T-6/2/7553664599460422f84b889363366fe5. – ISSN 0957-4174

[Anderson et al. 1997] Anderson, Charles W. ; Hittle, Douglas C. ; Katz, Alon D. ; Kretchmar, R. M.: Synthesis of reinforcement learning, neural networks and PI control applied to a simulated heating coil. In: *Artificial Intelligence in Engineering* 11 (1997), Oktober, Nr. 4, S. 421–429. – URL http://www.sciencedirect.com/science/article/B6V1X-3SP7D48-C/2/62d9b99362b40b9b2c65fe4b94d0c47e

[Arie et al. 2006] Arie, H. ; Namikawa, J. ; Ogata, T. ; Tani, J. ; Sugano, S.: Reinforcement learning algorithm with CTRNN in continuous action space. In: *Neural Information Processing, Pt 1, Proceedings* 4232 (2006), S. 387–396

[Baird und Klopf 1993] Baird, L.C. ; Klopf, A. H.: Reinforcement learning with high-dimensional, continuous actions / Wright-Patterson Air Force Base Ohio: Wright Laboratory. 1993 (WL-TR-93-1147). – Techn. Rep.

[Bellman 1957] Bellman, R E.: Markovian decision processes. In: *Journal of Mathematics and Mechanics* 38 (1957), S. 716–719

[Bertoluzzo und Corazza 2007] Bertoluzzo, Francesco ; Corazza, Marco: Making Financial Trading by Recurrent Reinforcement Learning. In: *Knowledge-Based Intelligent Information and Engineering Systems* (2007), S. 619–626. – URL http://dx.doi.org/10.1007/978-3-540-74827-4_78

[Bharath und Borkar 1999] Bharath, B. ; Borkar, V. S.: Stochastic approximation algorithms: Overview and recent trends. In: *Sādhanā* 24 (1999), Aug & Oct, S. 425–452. – Part 4 & 5

[Bonarini et al. 2006] Bonarini, A. ; Montrone, F. ; Restelli, M.: Reinforcement distribution in continuous state action space fuzzy Q-learning: A novel approach. In: *Fuzzy Logic And Applications* 3849 (2006), S. 40–45

[Boyan 1999] Boyan, Justin A.: Least-squares temporal difference learning. In: *Machine Learning: Proceedings of the Sixteenth International Conference*, Morgan Kaufmann, 1999, S. 49–56

[Boyan und Moore 1995] Boyan, Justin A. ; Moore, Andrew W.: Generalization in reinforcement learning: Safely approximating the value function. In: *Advances in Neural Information Processing Systems 7*, MIT Press, 1995, S. 369–376

[Braziunas 2003] Braziunas, Darius: POMDP solution methods: A survey / Department of Computer Science, University of Toronto. 2003. – Forschungsbericht

[Bryson und Ho 1975] Bryson, Arthur E. ; Ho, Yu-Chi: *Applied Optimal Control*. Hemissphere Publishing, 1975

[Bryson Jr. 1996] Bryson Jr., A.E.: Optimal control – 1950 to 1985. In: *Control Systems Magazine, IEEE* 16 (1996), Jun, Nr. 3, S. 26–33

[Cziko 1995] Cziko, Gary: *Without Miracles: Universal Selection Theory and the Second Darwinian Revolution*. Cambridge, Massachusetts and London, England : A Bradford Book, The MIT Press, 1995. – URL `http://faculty.ed.uiuc.edu/g-cziko/wm/`

[Daley et al. 2005] Daley, R. A. ; Immer, E. A. ; Fortner, B. I. ; Weiss, M. B.: Scientific resource access system: A concept for getting "living with a star" information to do science. In: *Johns Hopkins Apl Technical Digest* 26 (2005), Nr. 1, S. 22–35

[De La Hoz et al. 2008] De La Hoz, S.G. ; Ruiz, L.M. ; Liko, D.: First experience and adaptation of existing tools to ATLAS distributed analysis. In: *The European Physical Journal C - Particles and Fields* 53 (2008), Februar, Nr. 3, S. 467–471. – URL `http://dx.doi.org/10.1140/epjc/s10052-007-0499-9`

[Dembo und Steihaug 1983] Dembo, Ron ; Steihaug, Trond: Truncated-Newton algorithms for large-scale unconstrained optimization. In: *Mathematical Programming* 26 (1983), Juni, Nr. 2, S. 190–212. – URL `http://dx.doi.org/10.1007/BF02592055`

[Deng und Er 2003] Deng, Chang ; Er, Meng J.: Efficient implementation of dynamic fuzzy Q-learning. In: *Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on* 3 (2003), Dec, S. 1854–1858 vol.3

[DFG 1998] DFG: *Gute Wissenschaftliche Praxis*. 1998. – URL `http://www.dfg.de/antragstellung/gwp/index.html`

[DFG 2006] DFG: *Wissenschaftliche Literaturversorgungs- und Informationssysteme: Schwerpunkte der Förderung bis 2015*. online. 2006. – URL `http:`

//www.dfg.de/forschungsfoerderung/wissenschaftliche_infrastruktur/lis/download/positionspapier.pdf. – Positionspapier

[DFG 2008] DFG: *Forderung im Umgang mit Forschungsprimärdaten und dessen Förderung*. online. 2008. – URL http://www.dfg.de/forschungsfoerderung/wissenschaftliche_infrastruktur/lis/projektfoerderung/initiative_digitale_information/primaerdaten.html

[DFG 2009] DFG: *Ausschreibung „Repositorien – Ausbau und Entwicklung Informationsdienstleistungen für die Wissenschaft“*. 2009. – URL http://www.dfg.de/forschungsfoerderung/wissenschaftliche_infrastruktur/lis/aktuelles/download/ausschreibung_massnahme_repositorien_081212.pdf

[Duan et al. 2007] Duan, Yong ; Liu, Qiang ; Xu, XinHe: Application of reinforcement learning in robot soccer. In: *Engineering Applications of Artificial Intelligence* 20 (2007), Nr. 7, S. 936 – 950. – URL http://www.sciencedirect.com/science/article/B6V2M-4PKP4B5-1/2/a61f958f55b3cd7dc7787a25867ffec7. – ISSN 0952-1976

[Epshteyn und DeJong 2006] Epshteyn, Arkady ; DeJong, Gerald: Qualitative reinforcement learning. In: *ICML '06: Proceedings of the 23rd international conference on Machine learning*. New York, NY, USA : ACM, 2006, S. 305–312. – ISBN 1-59593-383-2

[Evans 2009] Evans, Clark C.: *YAML Ain't Markup Language*. 2009. – URL http://www.yaml.org/

[Even-Dar und Mansour 2003] Even-Dar, E. ; Mansour, Y.: Learning rates for Q-learning. In: *Journal Of Machine Learning Research* 5 (2003), Dezember, S. 1–25

[Fakulät für Maschinenbau, Universität Ilmenau 2009] Fakulät für Maschinenbau, Universität Ilmenau: *Verbrennungsprozessoptimierung mit lernenden Neuronalen Netzen*. 2009. – URL http://www.tu-ilmenau.de/fakmb/Verbrennungsprozesso.1874.0.html

[Flentge 2006] Flentge, F.: Learning to approach a moving ball with a simulated two-wheeled robot. In: *Robocup 2005: Robot Soccer World Cup IX* 4020 (2006), S. 106–117

[Framling 2005] Framling, K.: Dual memory model for using pre-existing knowledge in reinforcement learning tasks. In: *Artificial Neural Networks: Formal Models And Their Applications - Icann 2005, Pt 2, Proceedings* 3697 (2005), S. 203–208

[Fukao et al. 1998] Fukao, T. ; Sumitomo, T. ; Ineyama, N. ; Adachi, N.: Q-learning based on regularization theory to treat the continuous states and actions. In: *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on* Bd. 2, 1998, S. 1057–1062 vol.2

[Galstyan et al. 2005] Galstyan, A. ; Czajkowski, K. ; Lerman, K.: Resource allocation in the Grid with learning agents. In: *Journal of Grid Computing* 3 (2005), S. 91–100. – URL http://www.scopus.com/scopus/inward/record.url?eid=2-s2.0-28844499442\&partnerID=40\&rel=R5.6.0

[Garcia 2003] Garcia, P: A new way to introduce knowledge into reinforcement learning. In: *Machine Learning: ECML 2003* 2837 (2003), S. 157–168. – ISSN 0302-9743

[Gaskett et al. 1999] Gaskett, Chris ; Wettergreen, David ; Zelinsky, Alexander: Q-Learning in Continuous State and Action Spaces. In: *Australian Joint Conference on Artificial Intelligence*, 1999, S. 417–428

[Gray et al. 2005] Gray, Jim ; Liu, David T. ; Nieto-Santisteban, Maria ; Szalay, Alexander S. ; DeWitt, David ; Heber, Gerd: Scientific Data Management in the Coming Decade / Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052. Jan 2005 (MSR-TR-2005-10). – Technical Report

[Hailu und Sommer 1999] Hailu, G. ; Sommer, G.: On amount and quality of bias in reinforcement learning. In: *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on* 2 (1999), S. 728–733 vol.2

[van Hasselt und Wiering 2007] Hasselt, H. van ; Wiering, M. A.: Reinforcement Learning in Continuous Action Spaces. In: *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, IEEE, 2007, S. 272–279. – URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4220844

[Hayes 2008] Hayes, Brian: Cloud computing. In: *Commun. ACM* 51 (2008), Nr. 7, S. 9–11. – ISSN 0001-0782

[Hoey und Poupart 2005] Hoey, Jesse ; Poupart, Pascal: Solving POMDPs with Continuous or Large Discrete Observation Spaces. In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), Edinburgh, July 2005*, 2005

[Honerkamp 1994] Honerkamp, Josef: *Stochastic Dynamical Systems*. 1. VCH Publishers, Inc., 1994. – ISBN 1-56081-563-9

[Howell et al. 1997] Howell, M. N. ; Frost, G. P. ; Gordon, T. J. ; Wu, Q. H.: Continuous action reinforcement learning applied to vehicle suspension control. In: *Mechatronics* 7 (1997), April, Nr. 3, S. 263–276

[Howell und Gordon 2001] Howell, M. N. ; Gordon, T. J.: Continuous action reinforcement learning automata and their application to adaptive digital filter design. In: *Engineering Applications Of Artificial Intelligence* 14 (2001), Oktober, Nr. 5, S. 549–561

[International Council for Science 2009] International Council for Science: *Data Science Journal*. http://dsj.codataweb.org/. 2009. – URL http://dsj.codataweb.org/

[Jodogne und Piater 2006] Jodogne, S. ; Piater, J. H.: Task-driven discretization of the joint space of visual percepts and continuous actions. In: *Machine Learning: Ecml 2006, Proceedings* 4212 (2006), S. 222–233

[Jouffe 1998] Jouffe, L: Fuzzy inference system learning by reinforcement methods. In: *IEEE Transactions On Systems Man And Cybernetics Part C–Applications And Reviews* 28 (1998), AUG, Nr. 3, S. 338–355. – ISSN 1094-6977

[Kaelbling et al. 1996] Kaelbling, Leslie P. ; Littman, Michael L. ; Moore, Andrew W.: Reinforcement learning: a survey. In: *Journal of Artificial Intelligence Research* 4 (1996), S. 237–285

[Keerthi und Ravindran 1995] Keerthi, S S. ; Ravindran, B: A Tutorial Survey of Reinforcement Learning, 1995

[Kim 2002] Kim, Jang-Ho R.: *Optimierungsmethoden und Sensitivitätsanalyse für optimale bang-bang Steuerungen mit Anwendungen in der Nichtlinearen Optik*, Westfälische Wilhems-Universität Münster, Dissertation, 2002

[Kimura 2007] Kimura, Hajime: Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and Gibbs sampling. In: *2007 IEEE/RSJ International Conference On Intelligent Robots And Systems, Vols 1-9*. 345 E 47TH ST, NEW YORK, NY 10017 USA : IEEE, 2007, S. 88–95. – IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, Oct 29-Nov 02, 2007. – ISBN 978-1-4244-0911-2

[Knuth 1974] Knuth, Donald: Structured Programming with go to Statements. In: *Computing Surveys* 6 (1974), Dec, Nr. 4, S. 261–301

[Koller und Parr 2000] Koller, Daphne ; Parr, Ronald: Policy Iteration for Factored MDPs. In: Boutilier, Craig (Hrsg.) ; Goldszmit, Moisés (Hrsg.): *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, Morgan Kaufmann Publishers, 2000, S. 326–334. – URL `citeseer.ist.psu.edu/article/koller00policy.html`

[Kretchmar und Anderson 1997] Kretchmar, R. M. ; Anderson, Charles W.: Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In: *In International Conference on Neural Networks*, 1997, S. 834–837

[Kreutz und Honerkamp 2003] Kreutz, C. ; Honerkamp, J.: Controlling the CPAP using Partial Observable Markov Decision Processes. In: *Proceedings of International Conference on High Performance Scientific Computing, Hanoi Vietnam, 2003*, 2003

[Kreutz 2003] Kreutz, Clemens: *Steuerung stochastischer Systeme*, Albert-Ludwigs-Universität Freiburg, Diplomarbeit, 2003

[Kühne und Liehr 2009] Kühne, Martin ; Liehr, Andreas W.: Improving the Traditional Information Management in Natural Sciences. In: *Data Science Journal* 8 (2009), S. 18–26

[Kwok und Fox 2004] Kwok, C ; Fox, D: Reinforcement learning for sensing strategies. In: *Intelligent Robots and Systems (IROS-2004)* (2004). – URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1389903

[Lagoudakis und Parr 2003] Lagoudakis, M. G. ; Parr, R.: Least-squares policy iteration. In: *Journal Of Machine Learning Research* 4 (2003), aug, Nr. 6, S. 1107–1149

[Lazaric et al. 2008] Lazaric, A. ; Restelli, M. ; Bonarini, A.: Reinforcement Learning in Continuous Action Spaces through Sequential Monte Carlo Methods. In: Platt, J.C. (Hrsg.) ; Koller, D. (Hrsg.) ; Singer, Y. (Hrsg.) ; Roweis, S. (Hrsg.): *Advances in Neural Information Processing Systems 20*. Cambridge, MA : MIT Press, 2008, S. 833–840

[Li und Dagli 2003] Li, H. L. ; Dagli, C. H.: Hybrid Least-Squares methods for reinforcement learning. In: *Developments In Applied Artificial Intelligence* 2718 (2003), S. 471–480

[Madden und Howley 2004] Madden, MG ; Howley, T: Transfer of experience between reinforcement learning environments with progressive difficulty. In: *Artificial Intelligence Review* 21 (2004), JUN, Nr. 3-4, S. 375–398. – ISSN 0269-2821

[Martin und Arroyo 2004] Martin, Kimberly N. ; Arroyo, Ivon: AgentX: Using Reinforcement Learning to Improve the Effectiveness of Intelligent Tutoring Systems. In: al., J. C. L. et (Hrsg.): *Intelligent Tutoring Systems 2004*, Springer, 2004, S. 564–572. – URL http://www.springerlink.com/content/v6lu50384facqalc

[Mataric 1991] Mataric, Maja J.: A Comparative Analysis of Reinforcement Learning Methods / M.I.T. AI Lab. 1991. – Forschungsbericht

[Millán et al. 2002] Millán, J. D. R. ; Posenato, D. ; Dedieu, E.: Continuous-action Q-learning. In: *Machine Learning* 49 (2002), Nov, Nr. 23, S. 247–265. – ISSN 0885-6125

[Millán und Torras 1992] Millán, José Del R. ; Torras, Carme: A Reinforcement Connectionist Approach to Robot Path Finding in Non-Maze-Like Environments. In: *Machine Learning* 8 (1992), May, Nr. 3-4, S. 363–395. – ISSN 0885-6125

[Moore und Atkeson 1995] Moore, Andrew W. ; Atkeson, Christopher G.: The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In: *Machine Learning*, Morgan Kaufmann, 1995, S. 711–718

[Moore 1990] Moore, Andrew W.: Efficient memory-based learning for robot control / Computer Laboratory, University of Cambridge. URL http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-209.html, 1990 (UCAM-CL-TR-209). – Forschungsbericht

[Neumaier 2004] Neumaier, Arnold: *Acta Numerica*. Bd. 13. Kap. Complete Search in Continuous Global Optimization and Constraint Satisfaction, S. 271–369, Cambridge University Press, 2004

[Nissen 2007] Nissen, Steffen: *Large Scale Reinforcement Learning using Q-SARSA(λ) and Cascading Neural Networks*, Department of Computer Science, University of Copenhagen, Denmark, Diplomarbeit, 2007

[Oden et al. 2003] Oden, JT ; Browne, JC ; Babuska, I ; Liechti, KM ; Demkowicz, LF: A computational infrastructure for reliable computer simulations. In: Sloot, PMA and Abramson, D and Bogdanov, AV and Dongarra, JJ and Zomaya, AY and Gorbachev, YE (Hrsg.): *Computational Science - ICSS 2003, PT IV, Proceedings* Bd. 2660. HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY : Springer-Verlag Berlin, 2003, S. 385–390. – ISBN 3-540-40197-0

[Ozawa und Shiraga 2003] Ozawa, S. ; Shiraga, N.: Reinforcement learning using RBF networks with memory mechanism. In: *Knowledge-Based Intelligent Information And Engineering Systems, Pt 1, Proceedings* 2773 (2003), S. 1149–1156

[Paulus 1992] Paulus, Michael: *Deterministische Dynamische Optimalitätsprinzipien mit Anwendungen aus der chemischen Prozeßtechnik*, Albert-Ludwigs-Universität Freiburg im Breisgau, Dissertation, Januar 1992

[Perkins und Precup 1999] Perkins, T. J. ; Precup, D.: Using Options for Knowledge Transfer in Reinforcement Learning / Department of Computer Science, University of Massachusetts. URL citeseer.ist.psu.edu/perkins99using.html, , 1999 (UM-CS-1999-034). – Forschungsbericht

[Press et al. 1996] Press, William H. ; Flannery, Brian P. ; Teukolsky, Saul A. ; Vetterling, William T.: *Numerical Recipes in C. The Art of Scientific Computing. Second Edition*. Cambridge : Cambridge University Press, 1996

[Ren et al. 2007] Ren, Fang-Chin ; Chang, Chung-Ju ; Chen, Yih-Shen ; Chen, Jian-An: *Q-learning-based multi-rate transmission control (MRTC) scheme for RRC in WCDMA systems*. 2007. – URL http://www.patentstorm.us/patents/7286484/description.html. – Patent

[Riede et al. 2009] Riede, Moritz K. ; Sylvester-Hvid, Kristian O. ; Kühne, Martin ; Röttger, M. C. ; Zimmermann, Klaus ; Liehr, Andreas. W.: On the Communication of Scientific Results: The Full-Metadata Format / Freiburg Materials Research Center. 2009 (20090302a). – Scientific Information. DOI: 10.1594/fmf.SI20090302a

[Robbins und Monro 1951] Robbins, Herbert ; Monro, Sutton: A Stochastic Approximation Method. In: *Annals of Mathematical Statistics* 22 (1951), Sep, Nr. 3, S. 400–407

[Rohanimanesh und Mahadevan 2001] Rohanimanesh, Khashayar ; Mahadevan, Sridhar: Decision-Theoretic Planning with Concurrent Temporally Extended Actions. In: *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, URL http://citeseer.ist.psu.edu/rohanimanesh01decisiontheoretic.html, 2001, S. 472–479

[Röttger et al. 2005] Röttger, M. C. ; Liehr, A. W. ; Honerkamp, J.: Anwendung der Optimalen Steuerung zur Nutzung paralleler Resourcen. In: *Titisee-Neustadt 6.-7.10.2005* 13. internes FMF-Kolloquium (Veranst.), 2005. – Vortrag

[Röttger und Liehr 2009] Röttger, Michael C. ; Liehr, Andreas W.: Control task for Reinforcement Learning with known optimal solution for discrete and continuous actions. In: *Journal of Intelligent Learning Systems and Applications* (2009). – (accepted for publication)

[Sabes 1994] Sabes, P. N.: Approximating Q-values with basis function representations. In: *1993 Connectionist Models Summer School*, Lawrence Erlbaum Assoc. Inc., Hillsdale, NJ, 1994

[Sahba et al. 2008] Sahba, Farhang ; Tizhoosh, Hamid ; Salama, Magdy: Application of reinforcement learning for segmentation of transrectal ultrasound images. In: *BMC Medical Imaging* 8 (2008), Nr. 1, S. 8. – URL http://www.biomedcentral.com/1471-2342/8/8. – ISSN 1471-2342

[Santamaría et al. 1997] Santamaría, Juan C. ; Sutton, Richard S. ; Ram, Ashwin: Experiments with reinforcement learning in problems with continuous state and action spaces. In: *Adapt. Behav.* 6 (1997), Nr. 2, S. 163–217. – ISSN 1059-7123

[Satoh 2006] Satoh, H.: Reinforcement learning for continuous stochastic actions - An approximation of probability density function by orthogonal wave function expansion. In: *Ieice Transactions On Fundamentals Of Electronics Communications And Computer Sciences* E89A (2006), August, Nr. 8, S. 2173–2180

[Schultz et al. 1997] Schultz, Wolfram ; Dayan, Peter ; Montague, P. R.: A Neural Substrate of Prediction and Reward. In: *Science* 275 (1997), Nr. 5306, S. 1593–1599. – URL http://www.sciencemag.org/cgi/content/abstract/275/5306/1593

[Senat der Albert-Ludwigs-Universität Freiburg 2007] Senat der Albert-Ludwigs-Universität Freiburg: *Satzung der Albert-Ludwigs-Universität Freiburg zur Sicherung der Selbstverantwortung in der Forschung und zum Umgang mit wissenschaftlichem Fehlverhalten vom 27.10.2004 (Amtl. Bek. v. 10.11.2004, S. 384-399) in der Fassung der 1. Änderungssatzung vom 19.09.2007 (Amtl. Bek. v. 05.10.2007, S. 231).* 09 2007. – URL http://www.uni-freiburg.de/de/forschung/satzung_sicherung_selbstverantwortung_2007-09-19.pdf

[Sherstov und Stone 2004] Sherstov, Alexander A. ; Stone, Peter: *On Continuous-Action Q-Learning Via Tile Coding Function Approximation*. 2004. – URL http://citeseer.ist.psu.edu/637893.html

[Sherstov und Stone 2005] Sherstov, Alexander A. ; Stone, Peter: Improving Action Selection in MDP's via Knowledge Transfer. In: *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005

[Singh und Sutton 1996] Singh, Satinder P. ; Sutton, Richard S.: Reinforcement learning with replacing eligibility traces. In: *Mach. Learn.* 22 (1996), Nr. 1-3, S. 123–158. – ISSN 0885-6125

[Smart und Kaelbling 2000] Smart, William D. ; Kaelbling, Leslie P.: Practical Reinforcement Learning in Continuous Spaces. In: *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2000, S. 903–910. – ISBN 1-55860-707-2

[Sutton 1988] Sutton, Richard S.: Learning to predict by the methods of temporal differences. In: *Machine Learning* Bd. 3, Springer, 1988, S. 9–44

[Sutton 1996] Sutton, Richard S.: Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In: Touretzky, David S. (Hrsg.) ; Mozer, Michael C. (Hrsg.) ; Hasselmo, Michael E. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 8, The MIT Press, 1996, S. 1038–1044

[Sutton und Barto 1998] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning: An Introduction*. The MIT Press, March 1998 (Adaptive Computation and Machine Learning). – ISBN 0262193981

[Syafiie et al. 2007] Syafiie, S. ; Tadeo, F. ; Martinez, E.: Model-free learning control of neutralization processes using reinforcement learning. In: *Engineering Applications Of Artificial Intelligence* 20 (2007), SEP, Nr. 6, S. 767–782. – ISSN 0952-1976

[Taylor et al. 2008] Taylor, Matthew E. ; Kuhlmann, Gregory ; Stone, Peter: Autonomous transfer for reinforcement learning. In: *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC : International Foundation for Autonomous Agents and Multiagent Systems, 2008, S. 283–290. – URL `http://portal.acm.org/citation.cfm?id=1402383.1402427`. – ISBN 978-0-9817381-0-9

[ten Hagen und Kröse 2000] ten Hagen, Stephan ; Kröse, Ben: Q-Learning for systems with continuous state and action spaces. In: *BENELEARN 2000, 10th Belgian-Dutch Conference on Machine Learning*, URL `citeseer.ist.psu.edu/502060.html`, 2000

[Tesauro et al. 2006] Tesauro, G. ; Jong, N. K. ; Das, R. ; Bennani, M. N.: Improvement of systems management policies using hybrid reinforcement learning. In: *Machine Learning: ECML 2006, Proceedings* 4212 (2006), Nr. 10, S. 783–791. – Times Cited: 0. – ISSN 0302-9743

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

[The HDF Group 2009] The HDF Group: *HDF5 Home Page*. 2009. – URL `http://www.hdfgroup.org/HDF5/`

[The RL Community 2009a] The RL Community: *RL-Community.org*. 2009. – URL http://www.rl-community.org/

[The RL Community 2009b] The RL Community: *RL-Glue Home Page*. 2009. – URL http://glue.rl-community.org/

[The RL Community 2009c] The RL Community: *The RL-Logbook*. 2009. – URL http://recordbook.rl-community.org/

[The RoboCup Federation 2009] The RoboCup Federation: *RoboCup Official Site*. 2009. – URL http://www.robocup.org

[Theocharous et al. 2001] Theocharous, G. ; Rohanimanesh, K. ; Mahadevan, S.: *Learning hierarchical partially observable markov decision processes for robot navigation*. 2001. – URL citeseer.ist.psu.edu/theocharous01learning.html

[Tsitsiklis 1997] Tsitsiklis, B.: An analysis of temporal-difference learning with function approximation. In: *Automatic Control, IEEE Transactions on* 42 (1997), May, Nr. 5, S. 674–690. – ISSN 0018-9286

[Unidata Community 2009] Unidata Community: *NetCDF Home Page*. 2009. – URL http://www.unidata.ucar.edu/software/netcdf/. – NetCDF (network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.

[Vengerov 2005] Vengerov, David: *AI 2005: Advances in Artificial Intelligence*. Kap. Adaptive Utility-Based Scheduling in Resource-Constrained Systems, S. 477–488, Springer Berlin / Heidelberg, 2005. – URL http://dx.doi.org/10.1007/11589990_50. – 10.1007/11589990_50

[Vengerov 2007] Vengerov, David: A reinforcement learning approach to dynamic resource allocation. In: *Engineering Applications of Artificial Intelligence* 20 (2007), Nr. 3, S. 383 – 390. – URL http://www.sciencedirect.com/science/article/B6V2M-4M27WXH-1/2/b284e54a17286549d4a5c7b8f5f0d80e. – ISSN 0952-1976

[Wang und Usher 2004] Wang, Yi-Chi ; Usher, John M.: Learning policies for single machine job dispatching. In: *Robotics and Computer-Integrated Manufacturing* 20 (2004), Dezember, Nr. 6, S. 553–562. – URL http://www.sciencedirect.com/science/article/B6V4P-4D1DC62-1/2/00eb0fa406aba1eee802298e147f85fd

[Watkins 1989] Watkins, C. J. C. H.: *Leaning from Delayed Rewards*, King's College, Dissertation, 1989

[Watkins und Dayan 1992] Watkins, C. J. C. H. ; Dayan, P.: Q-Learning. In: *Machine Learning* 8 (1992), Mai, Nr. 3-4, S. 279–292

[Wawrzyński und Pacut 2004] Wawrzyński, Paweł ; Pacut, Andrzej: Model-free off-policy reinforcement learning in continuous environment. In: *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, 2004, S. 1091–1096

[Whitehead und Lin 1995] Whitehead, Steven D. ; Lin, Long-Ji: Reinforcement learning of non-Markov decision processes. In: *Artificial Intelligence* 73 (1995), Nr. 1-2, S. 271–306. – URL http://www.sciencedirect.com/science/article/B6TYF-4002FNS-9/2/ec8ff59b411b176045a62fff3e08d13a. – Computational Research on Interaction and Agency, Part 2. – ISSN 0004-3702

[Whiteson und Stone 2006] Whiteson, S. ; Stone, P.: Evolutionary function approximation for reinforcement learning. In: *Journal Of Machine Learning Research* 7 (2006), Mai, S. 877–917

[Widnall 1971] Widnall, William S.: Lunar Module Digital Autopilot. In: *Journal of Spacecrafts and Rockets* 8 (1971), S. 56–62

[Xu et al. 2002] Xu, X. ; He, H.G. ; Hu, D.W.: Efficient reinforcement learning using recursive least-squares methods. In: *Journal Of Artificial Intelligence Research* 16 (2002), S. 259–292. – ISSN 1076-9757

[Xu 2006] Xu, Xin: A Sparse Kernel-Based Least-Squares Temporal Difference Algorithm for Reinforcement Learning. In: *Advances in Natural Computation* Bd. Volume 4221/2006, Springer, 2006, S. 47–56. – URL http://dx.doi.org/10.1007/11881070_8

[Xu et al. 2007] Xu, Xin ; Hu, Dewen ; Lu, Xicheng: Kernel-Based Least Squares Policy Iteration for Reinforcement Learning. In: *Neural Networks, IEEE Transactions on* 18 (2007), July, Nr. 4, S. 973–992. – ISSN 1045-9227

[Xu et al. 2005] Xu, Xin ; Xie, T. ; Hu, D. W. ; Lu, X. C.: Kernel least-squares temporal difference learning. In: *International Journal of Information Technology* 11 (2005), Nr. 9, S. 54–63

[Young et al. 2003] Young, Peter M. ; Anderson, Charles ; Hittle, Douglas C. ; Kretchmar, Matthew: *Control system and technique employing reinforcement learning having stability and learning phases*. 2003. – URL http://www.patentstorm.us/patents/6665651/fulltext.html. – Patent

[Zhang und Dietterich 1995] Zhang, Wei ; Dietterich, Thomas G.: Value function approximations and job-shop scheduling. In: *Proceedings of the Workshop on Value Function Approximation. Carnegie-Mellon University, School of Computer Science*, 1995

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagung

Meiner erster Dank gilt Herrn Prof. Dr. Honerkamp für die Möglichkeit zur Anfertigung meiner Doktorarbeit, die Gelegenheit des regelmäßigen Austauschs und die vielen Anregungen zur Ordnung meiner Ergebnisse.

Ganz besonders möchte ich Herrn Dr. Andreas W. Liehr für seine unermüdliche Gesprächsbereitschaft und die Hilfestellungen bei der Einschätzung meiner Ergebnisse danken. Sein Optimismus und seine Ideen haben mir immer wieder neue Perspektiven eröffnet.

Herrn Prof. Dr. Xin Xu danke ich für den freundlichen E-Mail-Kontakt und die Beantwortung meiner Fragen zum KLSPI-Algorithmus.

Meiner Kollegin Frau Dr. Lioudmila Belenkaia danke ich für die angenehme Arbeitsatmossphäre im Büro, ihr Interesse an meinen Fortschritten und ihr Obst und Gebäck, das sie immer gern mit mir geteilt hat.

Den Mitarbeitern der Servicegruppe *Rheologie* des Freiburger Materialforschungszentrums (FMF) danke ich für die vielen geselligen Runden bei Tee und Kuchen. Ein besonderer Dank gilt dabei Herrn Dr. Martin Kühne für sein offenes Ohr und unzählige interessante Anregungen.

Den Herren Dr. Uli Würfel und Dr. Bernd Huber danke ich für die Beantwortung meiner Fragen zu praktischen Belangen des Promotionsverfahrens.

Sehr angenehm war die stets pragmatische Unterstützung durch die Verwaltung des FMF. Dafür möchte ich Frau Dr. Stefanie Meisen, Herrn Kurt Ruf, Frau Nicola Weis und Frau Stefanie Kuhl danken. Stefanie Kuhl danke ich außerdem für die vielen gemeinsamen Mittagspausen.

Sehr dankbar bin ich außerdem Stefanie Kröger und Nicole Streitz für ihre große Hilfe beim Korrekturlesen meiner Arbeit.

Erklärung

Hiermit erkläre ich, die vorgelegte Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Freiburg, den 31.03.2009

(Michael Röttger)