Reinforcement-Learning in der Domäne von PL-1
AW-1 Ausarbeitung

David Olszowka

HAW Hamburg
Email: david.olszowka@haw-hamburg.de

I. Motivation

Ich habe mich im Master bereits in der Vorlesung TT2 mit Reinforcement-Learning praktisch beschäftigt. Dabei habe ich mehrere Variationen von Q-Learning Agenten implementiert, die das Spiel 4-Gewinnt lernen sollten. Das Ergebnis diese Projekts war, dass der Zustandsraum für gewöhnliches Q-Learning zu groß war. Bei einem Spielfeld von 6 Zeilen und 7 Spalten mit je 3 möglichen Belegungen gibt es ca. $3^{6 \cdot 7} \approx 2^{66}$ Zustände. Diese Menge an Zuständen ist zu groß, um den Zustandsraum vollständig zu durchlaufen und jeden Zustand zu bewerten.

Deshalb möchte ich im Rahmen des Projekt-1 und später meiner Master-Arbeit untersuchen, inwiefern sich eine Zustandsbeschreibung durch PL-1 Prädikate eignet, um den Zustandsraum so weit wie nötig zu reduzieren, ohne dem Agenten dadurch zu wenig Informationen zum Lösen der Aufgabe zu geben. In der Literatur zu Reinforcement-Learning werden die Algorithmen häufig nur auf Problemstellungen mit relativ kleinem Zustandsraum, wie TicTacToe [1], das Cliff oder die Blocks-World angewandt [2]. Ich will mit dieser Untersuchung prüfen, ob sich Reinforcement-Learning möglicher Weise für Probleme dieser Größenordnung gar nicht eignet.

II. Einleitung

A. Reinforcement Learning

Reinforcement-Learning (dt. verstärkendes Lernen) ist eine Lernmethode, die ihren Ursprung in der Psychologie und Neurowissenschaft hat und ähnliches Lernverhalten zeigt. [3] Das besondere am Reinforcement-Learning ist das Lernen von Verhalten ausschließlich durch die Verteilung von Belohnungen und Bestrafungen. [4]

Das Lernmodell von Reinforcement-Learning ist in Figur 1 abgebildet. Der Agent erhält eine skalare Zustandsbeschreibung und eine skalare Belohnung für die zuletzt gewählte Aktion. Anhand dieser Information und der bisherigen Erfahrung, muss der Agent eine Nachfolgeaktion wählen. Das Ziel ist es, durch ausreichend viele Versuche, eine optimale Strategie π^* zur Lösung der Aufgabe zu finden.

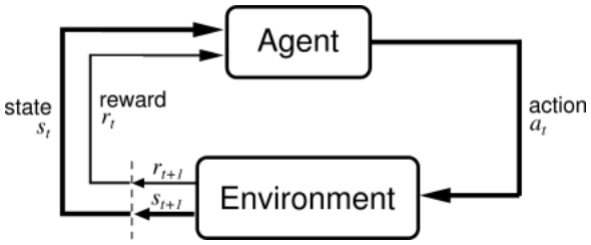

Fig. 1. Interaktion zwischen Agent und Umwelt [5]

Algorithmen für Reinforcement-Learning werden in drei Kategorien unterteilt, die nachfolgend kurz beschrieben werden.

- Dynamische Programmierung
- Monte-Carlo Methoden
- Temporal Difference Learning

1) Dynamische Programmierung: In der dynamischen Programmierung wird davon ausgegangen, dass alle Zustände, alle möglichen Zustandsübergänge und alle dafür erhaltenen Belohnungen (also das Modell der Problemstellung) von Anfang an bekannt sind. Diese Algorithmen gehören genau genommen zu Planungsalgorithmen und nicht zu Lernalgorithmen. Die Algorithmen der dynamischen Programmierung lernen eine Abbildung von Zuständen auf die erwartete Belohnung bis zum Ziel, die V-Funktion ($V : S \to \mathbb{R}$). Durch die Auswertung aller möglichen Aktionen von einem Zustand aus, wird schrittweise versucht die V-Funktion zu berechnen.

$$V_{t+1}(s) := V_t(s) + \alpha \cdot (r + \gamma V_t(s') - V_t(s)) \qquad (1)$$

Dabei ist $V_t(s)$ der V-Wert eines Zustands im Durchlauf t, α die Lernrate, γ der Abschwächungsfaktor für nicht episodische Anwendungen und s' der Nachfolgezustand von s.

Dynamische Programmierung findet immer die optimale Strategie unter der Voraussetzung, dass das Modell vollständig bekannt und der Zustandsraum nicht zu groß ist. Da in diesem Ansatz jeder Zustand in mehreren Iterationen durchgerechnet wird, ist ein großer Zustandsraum das größte Problem für dynamische Programmierung. [4]

2) Monte-Carlo Methoden: Monte-Carlo beschreibt eine Art von Algorithmen, die kein vollständiges Modell der Umwelt benötigen und auf episodische Probleme beschränkt sind. Da ohne Modell der Umwelt unklar ist, welche Aktion, zu welchem Folgezustand führt, verwendet Monte-Carlo statt der V-Funktion, eine Q-Funktion ($Q : S \times A \rightarrow \mathbb{R}$) da der Agent nun lernen muss, welche Aktion in welchem Zustand die beste ist.

Monte-Carlo lernt die Q-Werte indem einzelne Episoden nach der aktuell besten bekannten Strategie durchlaufen werden und zu jedem Zustand-Aktion-Paar gespeichert, wie viel Aufsummierte Belohnung der Agent nach diesem Zustand bis zum Ziel erhalten hat. Der Q-Wert entspricht wird aus dem Durchschnitt dieser Werte über alle Episoden.

3) Temporal Difference Learning: Diese Klasse von Algorithmen benötigt ebenfalls kein vollständiges Modell der Umwelt und lernt dies durch Interaktionen mit der Umwelt. Im Gegensatz zu Monte-Carlo wird die gelernte Strategie nicht nach jeder Episode angepasst, sondern nach jedem einzelnen Schritt. Zudem müssen die Probleme nicht episodisch sein.
Die iterative Berechnung der Q-Werte geschieht analog zur V-Wert-Berechnung.

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \cdot (r + \gamma Q_t(s',a') - Q_t(s,a)) \quad (2)$$

Für die Interpretation von a' gibt es zwei Ansätze. Der *On-Policy*-Algorithmus SARSA wählt für a' die in dem Nachfolgezustand gewählte Aktion. Der *Off-Policy*-Algorithmus Q-Learning wählt stattdessen die am besten bewertete Aktion als a'.
Dadurch spielt bei SARSA eine gute Explorationsstrategie eine wichtige Rolle beim Lernen, denn häufiges Explorieren von schlecht bewerteten Aktionen kann den Lernvorgang verlangsamen und sogar verhindern, dass eine optimale Strategie gefunden wird. Q-Learning hat dieses Problem nicht und lernt unabhängig von der Explorationsstrategie immer die optimale Strategie. Eine schlechte Explorationsstrategie führt bei Q-Learning lediglich dazu, dass die optimale Strategie langsamer gefunden wird. [4]

B. Probleme bei Reinforcement Learning

1) Temporal Discredit Problem: Agenten aus dem Bereich Temporal-Difference-Learning erwarten die Bewertung einer Aktion im Idealfall als direkte Rückmeldung der Umwelt. Für ein direktes Feedback muss die (modellierte)Umwelt in jedem Zustand die optimale Aktion kennen, oder zumindest eine ungefähre Abschätzung besitzen, welche Aktion die wertvollste ist. Diese Information ist im Normalfall nicht vorhanden, denn die Kenntnis der optimalen Aktion in jedem Zustand entspricht der Kenntnis der optimalen Strategie. In diesem Fall wäre das Lernen überflüssig.
Bei vielen Anwendungen(z.B. 4-Gewinnt, Schach) lässt sich keine sinnvolle Regel vorgeben, die einzelne Schritte bewertet. Bei Schach könnte beispielsweise eine große Belohnung für das Schlagen der gegnerischen Dame vergeben werden, jedoch riskiert man damit, dass der Agent durch die hohe und relativ frühe Belohnung, dieses Zwischenziel so hoch priorisiert, dass er in Hinblick auf das Hauptziel nicht optimal spielt und dieses möglicher Weise nicht mehr erreichen kann, da er sich durch das Schlagen der Dame in eine ungünstige Spiellage versetzt hat (Beispiel: Boden-Matt). Um garantiert eine optimale Strategie zu lernen, wird der Agent nur bei Sieg oder Niederlage bewertet. Dadurch hat der Agent jedoch keine Informationen über die Bewertung der einzelnen Zwischenschritte, weshalb der Agent bei einem großen Zustandsraum nur sehr langsam lernt.
Eine Lösung für dieses Problem stellen die $TD(\lambda)$-Algorithmen dar [4]. Diese Algorithmen gehen davon aus, dass alle vorherigen Aktionen einen Teil zu der Bewertung der aktuellen Aktion beigetragen haben und verteilen die Bewertung in abgeschwächter Form auf die Vorgängerzustände. Dadurch können $TD(\lambda)$-Agenten Aufgaben, in denen es nur selten Belohnungen gibt, schneller lernen.

2) Zustandsraumexplosion: Die Zustandsraumexplosion (auch *curse of dimensionality* [6]) ist ein weiteres Problem, welches daraus resultiert, dass der Agent zu jeder Aktion in jedem besuchten Zustand eine Bewertung speichern muss. Für das Spiel 4-Gewinnt sind das bereits ca. 2^{66} Zustände, zu denen es noch jeweils bis zu sieben Aktionen gibt. Dieser große Zustandsraum ist zu groß, um jeden Zustand auf einem 64-Bit System im virtuellen Speicher zu adressieren. Unter der Annahme, dass ein Zustand in nur einem CPU-Takt berechnet werden kann, würde eine 1GHz-CPU über 2000 Jahre benötigen, um jeden Zustand genau einmal zu besuchen. Da für eine optimale Strategie jeder Zustand mehrfach besucht werden muss, ist es nicht realistisch, eine optimale 4-Gewinnt-Strategie auf diese Weise zu lernen.
Dieses Problem kann nur dadurch gelöst werden, dass entweder Teile des Zustandsraums bei der Modellierung weggelassen werden, oder ähnliche Zustände in einem Zustand vereint werden (Feature-Vektoren, PL-1 Zustände). Dabei muss darauf geachtet werden, dass keine Zustandsinformationen weggelassen werden, die für die Lösung der Aufgabe relevant sind. Ansonsten kann der Agent keine gute Strategie mehr lernen.

3) Abstraktion: Da der Agent die Zustände der Umwelt nur als Zahlen erhält, kann er keine Annahmen über die Eigenschaften dieser Zustände treffen, um den Lernvorgang zu beschleunigen. Die Zustandsbeschreibung, die der Agent erhält, dient lediglich als Identifikator mit dem er die Zustände voneinander unterscheiden kann. Jeder Zustand ist für den Agenten somit ein gänzlich neuer Zustand, über den er keine Information verfügt.
Menschen nehmen ihre Welt jedoch anders wahr, nämlich als eine Menge von Objekten und deren Eigenschaften und Beziehungen zueinander. Dadurch können sie auch ähnliche Zustände erkennen, die sich nur durch eine kleine Änderung dieser Eigenschaften voneinander unterscheiden. Gelerntes Wissen kann so auf ähnliche Zustände übertragen

werden, ohne die Situation vollständig neu zu erlernen. Abstraktion von Problemen ist in Reinforcement-Learning erst möglich, wenn die Zustandsbeschreibung selbst eine Struktur enthält, die der Agent nutzen kann, um Aussagen über den Zustand in Relation zu anderen Zuständen zu treffen.

C. Reinforcement-Learning in PL-1

Eine mögliche Lösung für die letzten beiden Probleme des Reinforcement-Learnings stellt die Modellierung der Zustände mithilfe von Prädikaten dar. Ein Schritt in diese Richtung ist das Reinforcement-Learning mit Feature-Vektoren. Statt einer Zahl als Zustandsbeschreibung erhält der Agent einen Vektor von Eigenschaften, die in diesem Zustand erfüllt sind. Dieser Vektor wird so vorgegeben, dass der Agent nur die Zustandseigenschaften erhält, die für die Lösung der Aufgabe notwendig sind. Dadurch kann zum einen die Menge an Zuständen reduziert werden und zum anderen hat der Agent ein Ähnlichkeitsmaß für Zustände. Dies kann der Agent nutzen, um Erfahrungen aus bekannten Zuständen auf ähnliche, neue Zustände zu übertragen.

Verwendet man statt Features zur Zustandsbeschreibung Prädikate, dann fallen viele Features zu einem einzigen Prädikat zusammen, wie in [7] erklärt wird. Erhält ein 4-Gewinnt-Agent beispielsweise ein Feature $gegnerGewinnt$, dann kann der Agent mit der Information, dass der Gegner im nächsten Zug gewinnen kann, nichts anfangen. Damit der Agent einen Mehrwert durch das Feature hat, muss der Agent wissen, in welche Spalte er setzen muss, um dies zu verhindern. Dazu wären bei sieben Spalten die folgenden Features notwendig $gegnerGewinnt1$, $gegnerGewinnt2$, $gegnerGewinnt3$, $gegnerGewinnt4$, $gegnerGewinnt5$, $gegnerGewinnt6$ und $gegnerGewinnt7$.
Da die Features $gegnerGewinnt1$ und $gegnerGewinnt2$ für den Agenten gänzlich verschiedene Features sind, muss der Agent für jedes Feature lernen, welche Aktion die beste ist. In Prädikatenlogik lässt sich diese Eigenschaft des Zustands durch das Prädikat $GegnerGewinnt(X)$ beschreiben. X wird dabei an die entsprechende Spalte gebunden. Jetzt muss der Agent nur lernen, dass er in Zustand $GegnerGewinnt(X)$ die Spalte X wählen muss, um den Gegner am Sieg zu hindern. Damit verringert die Zustandsbeschreibung durch Prädikate zum einen den Zustandsraum und zum anderen erlaubt sie es durch das Binden von Variablen, die Zustände auf eine abstrakte Art zu beschreiben.

III. Eigene Lösungsansätze

Bei der Modellierung eines 4-Gewinnt-Agenten mit Reinforcement-Learning spielen alle in II-B genannten Probleme eine Rolle.
Sowohl die Zustandsraumexplosion, als auch die fehlende Abstraktion sollen durch eine PL-1 Beschreibung der Zustände gelöst werden. Die Zustandsbeschreibung ist dabei so gewählt, dass sie der Herangehensweise eines menschlichen Spielers ähnelt. Ein menschlicher Spieler sucht das Spielfeld nach Möglichkeiten ab, eine eigene Linie zu verlängern, oder eine gegnerische Linie zu blockieren. Der gelbe Spieler würde die Situation in Figur 2 dadurch beschreiben, dass er durch Wahl von Spalte 1 eine Viererkette des Gegners verhindern kann und durch Wahl von Spalte 4 selbst eine Viererkette erreichen und damit das Spiel gewinnen kann. Die äquivalente Zustandsbeschreibung in PL-1 für den Agenten würde wie folgt aussehen:

$$4erblock(1) \wedge 4erkette(4) \tag{3}$$

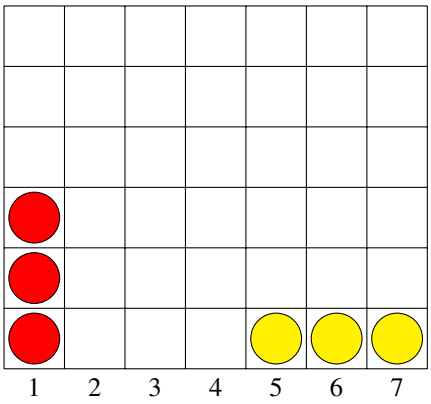

Fig. 2. Eine Spielfeldkonfiguration in welcher der gelbe Spieler im nächsten Zug in Spalte 4 setzen sollte, um zu gewinnen.

Alle Zustände des Spiels würden dann aus den folgenden Grundprädikaten zusammengesetzt werden: *4erblock/1*, *3erblock/1*, *4erkette/1*, *3erkette/1*. Dabei bedeutet ein Prädikat $Xerblock(Y)$, dass eine gegnerische Kette der Länge X verhindert werden kann, wenn der Agent seinen Spielstein in Spalte Y wirft. Analog dazu bedeutet $Xerkette(Y)$, dass der Agent eine Kette der Länge X erhält, wenn er seinen nächsten Spielstein in die Spalte Y wirft. Zweierketten werden dabei nicht als strategisch relevant angesehen, da in nahezu jeder Spalte eine Zweierkette erzeugt oder blockiert werden kann. Der gleiche Zustand hätte durch die Aufnahme von Zweierketten folgende Beschreibung:

$$\begin{aligned} &4erblock(1) \wedge 4erkette(4) \wedge 2erblock(2) \wedge \\ &2erkette(5) \wedge 2erkette(6) \wedge 2erkette(7) \end{aligned} \tag{4}$$

Die Zweierketten würden nur dazu führen, dass der Agent zwischen Zuständen unterscheiden muss, deren Unterschiede in Hinblick auf das Spielziel nicht relevant sind. Als Ergebnis würde der Agent mehr Zustände lernen müssen und dadurch würde der Lernprozess verlangsamt werden.

Als zugrunde liegenden Lernalgorithmus für die Umsetzung werde ich Q-Learning wählen. Q-Learning konvergiert unabhängig von der gewählten Explorationsstrategie zur optimalen Strategie [4]. Dies schließt zumindest eine mögliche Fehlerquelle von vornherein aus. Des weiteren

verwendet die bereits vorhandene Implementierung Q-Learning und es sind bereits einige Agenten zum Vergleich vorhanden. Bei Testspielen kann so also ausgewertet werden, ob Q-Learning mit PL-1 für 4-Gewinnt eine Verbesserung gegenüber normalem Q-Learning bringt.

Die tatsächlichen Zustände der Umwelt werden im Agenten sogenannten abstrakten Zuständen zugeordnet. Diese abstrakten Zustände werden, wie in CARCASS [2] in einer Entscheidungsliste geführt. Die erste abstrakte Zustandsbeschreibung, die mit dem aktuellen Zustand unifiziert werden kann, repräsentiert den Zustand, in dem sich der Agent befindet und dessen Q-Werte der Agent in diesem Schritt lernt. Zu dem Zustand, der in Gleichung 3 beschrieben wird, könnte ein abstrakter Zustand im Agenten wie folgt aussehen:

$$S_1 = 4erblock(X) \wedge 4erkette(Y) \tag{5}$$

In diesem Zustand hat der Agent die möglichen Aktionen X und Y für welche er die Q-Werte lernen muss. In diesem Fall ist $X = 1$ und $Y = 4$. Dieser Zustand beschreibt alle Zustände, in denen der Spieler eine Viererkette verhindern und gleichzeitig selbst erzeugen kann. Dabei spielt die Lage der Ketten für den Agenten keine Rolle und er kann so lernen, welches der Prädikate für den Spielverlauf wichtiger ist und dementsprechend welche Aktion wichtiger ist.
Die Zielsetzung des Agenten ist, diese Zustände autonom zu lernen anhand der Spielsituationen, die der Agent in der Lernphase vorfindet, die Zustände in der Entscheidungsliste so anzuordnen, dass der Zustand, in dem sich der Agent befindet, korrekt erkannt wird und die Bewertungen der Aktionen der einzelnen Prädikate zu erlernen.

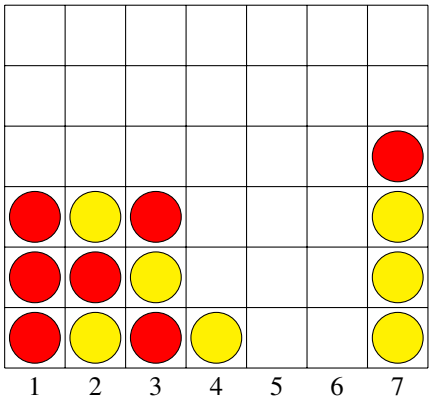

Fig. 3. Eine Spielsituation, in der nur das Setzen in die erste Spalte ein sinnvoller Zug ist

Betrachtet man die Spielsituation in Figur 3, erkennt man, dass auch diese Spielfeldkonfiguration durch den eben genannten abstrakten Zustand S_1 abgedeckt wird, indem die Variablen mit $X = Y = 1$ unifiziert werden. Dennoch hat die Spielsituation eine andere Bedeutung. Sie lässt nur eine Aktion zu, die gleich zwei Prädikate erfüllt. Dies ist für den Spieler eine sehr vorteilhafte Situation, denn er muss nicht zwischen zwei sinnvollen Aktionen abwägen. Damit der Agent lernen kann, dass solche Zustände für die Lösung der Aufgabe besser sind, muss er dafür einen neuen abstrakten Zustand lernen.

$$S_2 = 4erblock(X) \wedge 4erkette(X) \tag{6}$$

Hier stellt sich natürlich die Frage, wann der Agent erkennt, dass er einen neuen Zustand lernen muss, und wann er einen bereits vorhandenen Zustand verwenden kann. Im Projekt-1 möchte ich mich mit dieser Fragestellung noch nicht praktisch beschäftigen, jedoch könnte ich mir vorstellen, dass der Agent bei der Unifikation explizit auf gleiche Werte prüft und so speziellere Prädikate mit weniger Variablen den allgemeineren vorzieht. Sorgt man nun dafür, dass S_2 immer vor S_1 in der Zustandsliste aufgeführt wird, dann wird der Agent auf lange Sicht lernen, Zustände wie S_2 zu erreichen.

Eine Überlegung zur Beschleunigung des Lernvorgangs ist es, kleine Zwischenbelohnungen für das erfüllen einzelner Prädikate zu vergeben. Dabei würde die Bewertung der Prädikate mit deren Spielrelevanz zunehmen.

Prädikat	**Belohnung**
$spielGewonnen$	100
$4erkette(X)$	10
$4erblock(X)$	5
$3erkette(X)$	2
$3erblock(X)$	1

Die Zwischenbelohnungen würden zum einen dazu führen, dass der Agent schneller lernt, da er regelmäßig Belohnungen erhält und so auch lernt, welche Prädikate für den Spielverlauf wertvoller sind und dass Aktionen, die zwei Prädikate gleichzeitig erfüllen, umso wertvoller sind. Problematisch an dem Ansatz ist, dass der Agent dadurch lernt, was ich über das Spiel vermute. Möglicher Weise ist es wichtiger, Dreierketten zu blockieren, anstatt selbst Dreierketten zu setzen. Durch die vorgegebene Belohnungsverteilung würde ich dem Agenten das selbstständige erlernen solcher Zusammenhänge erschweren und dadurch kann der Agent schließlich keine optimale Strategie mehr lernen. Es wird generell empfohlen einem Agenten so wenig eigenes Wissen mitzugeben, wie möglich. [8]
Um solche Effekte zu vermeiden, werde ich die Bewertung einzelner Prädikate erst in Betracht ziehen, wenn der Agent zu langsam lernt.

A. *Risiken*

Möglicher Weise wird trotz des kleinen Zustandsraums das *temporal discredit problem* dazu führen, dass der Agent nicht lernt, oder dies sehr langsam tut. Das würde dazu führen, dass ich den Agenten nicht gegen bereits bestehende Q-Learning Agenten testen kann, um eine Bewertung zu der Verwendung

von PL-1 mit Q-Learning abgeben zu können.
Das Problem lässt sich auf mit zwei Arten lösen. Zum einen durch Belohnungen, die für das Erfüllen einzelner Prädikate des Zustands vergeben werden. Die Belohnung kann so verteilt werden, wie in der Tabelle im letzten Abschnitt dargestellt. Wie bereits erwähnt wurde, hat diese Lösung den Nachteil, dass dem Agenten die Strategie durch die Gewichtung der Prädikate bereits vorgegeben wird und bessere Strategien abseits dieser Vorgabe möglicher Weise nicht gefunden werden. Sollte das *temporal discredit problem* tatsächlich dazu führen, dass der Agent langsam lernt, wird zuerst versucht, das Problem mit dem Einsatz von $TD(\lambda)$-Algorithmen (konkret mit $Q(\lambda)$) zu lösen.

Ein weiteres Risiko stellt die mangelnde Beschreibung der initialen Zustände dar. Während der ersten Spielzüge lassen sich keine der vier vorgegebenen Prädikate auf den Zustand anwenden. Somit würde sich der Agent in diesen Fällen immer im gleichen Zustand befinden und keinerlei Information über das Spielfeld besitzen. Möglicher Weise spielt die fehlende Information zu Beginn des Spiels keine entscheidende Rolle für den Gesamterfolg des Agenten. Sollte der Mangel an Informationen in der Anfangsphase ein ernsthaftes Problem darstellen, dann kann die Modellierung des Spiels wie folgt angepasst werden, um dem Agenten zusätzliche Zutandsinformationen zu Beginn des Spiels mitzugeben. Sollte sich das Spielfeld in einer Situation befinden, in der keines der vier Grundprädikate anwendbar ist, dann (und nur dann) werden zusätzlich die Prädikate *2erblock/1* und *2erkette/1* verwendet, um den Zustand zu beschreiben.
Dadurch würde sich der Zustandsraum zwar vergrößern, jedoch hätte der Agent zu jedem Zustand die notwendigen Informationen, eine optimale Strategie zu lernen. Durch Hinzunahme der Zweierketten, ist der Initialzustand der einzige Zustand, der durch keine Prädikate beschrieben wird.

IV. Verwandte Arbeiten

In [9] wird das TAMER-Framework vorgeschlagen, um das *temporal discredit problem* zu lösen. Die Autoren haben festgestellt, dass Menschen, die eine Problemdomäne kennen, häufig eine gute Abschätzung treffen können, ob eine Aktion gut war, oder nicht. Sie haben das Problem darauf zurückgeführt, diese menschliche Einschätzung in das Modell der Umgebung algorithmisch zu integrieren.
Der Lösungsansatz sieht vor, dass ein menschlicher "Trainer" den Agenten beim Lernen beobachtet und ihn per Knopfdruck belohnen oder bestrafen kann. Kennt sich der Trainer gut in der Problemdomäne aus, dann erhält der Agent so zeitnah Feedback zu den gewählten Aktionen und lernt bereits nach sehr wenigen Episoden gute Strategien. Eine passable Tetris Strategie wurde in deren Messungen nach nur drei Episoden gelernt, während autonome Algorithmen deutlich länger gebraucht haben.
Mit TAMER lässt sich das *temporal discredit problem* für die Anfangsphase des Lernprozesses durch menschliches eingreifen lösen. Die Autoren haben ebenfalls beschrieben, dass TAMER nach sehr vielen Lernzyklen schlechter abschneidet, als autonome Agenten. Die Verwendung von TAMER schließt außerdem schnelles *offline-learning* aus, da Episoden so langsam durchgespielt werden müssen, dass ein menschlicher Trainer eine Chance hat, rechtzeitig einzugreifen. Deswegen schlagen die Autoren den Einsatz von TAMER hauptsächlich für solche Agenten vor, die in der echten Welt lernen müssen, weil es für ihr Anwendungsgebiet keine geeignete Simulation gibt. Bei solchen Agenten ist es besonders wichtig, dass sie bereits nach einer kurzen Lernphase keine groben Fehler machen und ein akzeptables Verhalten zeigen.

Da die Bewertung von Zuständen in 4-Gewinnt ebenfalls nicht trivial ist, bietet sich TAMER als Alternative an, falls die von mir gewählte, einfache Bewertungsfunktion zu keinen guten Ergebnissen führt.

In [6] wird Q-Learning auf eine Objektorientierte Problemstellung übertragen. Dazu wird die Umwelt als eine Menge von Objekten modelliert, die wiederum in Klassen von ähnlichen Objekten unterteilt sind. Die Zustandsraumexplosion bei einer großen Menge an Objekten wird dadurch verhindert, dass analog zum Menschen der Agent nur eine kleine Anzahl von Objekten zur Zeit fokussiert. Der Agent lernt dabei für jede Klasse von Objekten eine eigene Strategie. Die eigentliche Herausforderung ist dabei zu lernen, in welchem Zustand sich der Agent auf welche Klasse von Objekten fokussieren muss. Dieser Ansatz ist besonders für Agenten geeignet, deren Umwelt hauptsächlich aus Objekten besteht und deren Aufgabe es ist, intelligent mit diesen Objekt zu interagieren.

In [7] wird *temporal difference learning* im verteilten Kontext untersucht. Dabei sollten mehrere Agenten eine Aufgabe kooperativ bewältigen und durch gegenseitigen Informationsaustausch schneller lernen, als selbstständige Agenten. Die Umwelt der Agenten wurde mit PL-1 modelliert. Als Lernverfahren verwendeten die Agenten *linear value approximation* (LVA). LVA wird auch häufig verwendet, um Reinforcement-Learning mit Feature-Vektoren zu beschleunigen. Bei LVA wird im Gegensatz zum Q-Learning mit abstrakten Zuständen keine Q-Funktion über Q-Werte approximiert. Stattdessen werden zu jedem Prädikat oder Feature ein Gewichtungsfaktor gelernt und so versucht eine gute Annäherung an die tatsächliche Q-Funktion oder V-Funktion zu erhalten.

$$Q(s,a) \approx \sum_i w_i p_i(s,a) \tag{7}$$

$$V(s) \approx \sum_i w_i p_i(s) \tag{8}$$

Wobei p_i ein Feature oder Prädikat in dem Zustand s (und der gewählten Aktion a) repräsentiert und die Werte 0 oder 1 für Features oder Elemente aus $\mathbb{N}_0$(Anzahl der im Zustand s möglichen Instanzen des Prädikats) für Prädikate annehmen kann. w_i ist die für jedes Feature/Prädikat zu lernende Gewichtung.
Ein großer Vorteil dieser Lernmethode ist die kleine Anzahl zu lernender Werte im Vergleich zum Lernen von vollständigen Q-Funktionen, bzw. V-Funktionen und ist damit eine mögliche Lösung für das Problem der Zustandsraumexplosion. Als Nachteil wird dafür in Kauf genommen, dass nur eine Annäherung zur gesuchten Funktion bestimmt werden kann und eine optimale Strategie möglicher Weise nicht gefunden wird. [10]

In [11], einem etwas älteren Paper, wird ein weiterer Algorithmus vorgestellt, um Q-Learning mit prädikatenlogischen Beschreibungen von Zuständen zu verbinden. Der Algorithmus *rQ-Learning* basiert ebenfalls auf abstrakten Zuständen, die mehrere tatsächliche Zustände abdecken können. Im Gegensatz zu CARCASS, müssen diese abstrakten Zustände explizit vorgegeben werden und können nicht vom Agenten selbstständig gelernt werden. Dazu fordert *rQ-Learning* eine eindeutige Abbildung von einem tatsächlichen Zustand auf einen abstrakten Zustand. CARCASS kann diese Abbildung durch Veränderung der Reihenfolge der Zustandsliste verändern. Zudem müssen in *rQ-Learning* sinnvolle Aktionsmengen für jeden Zustand vorgegeben werden. Eine Aktionsmenge ist definiert über Vorbedingungen und Nachbedingungen, die jeweils vor- und nach der ausgeführten Aktion gelten müssen. Der Agent lernt dann Q-Werte für diese Aktionsmengen.
In dem Entwurf, den ich in Projekt-1 verfolgen will, gebe ich ebenfalls sinnvolle Aktionen vor. Dies geschieht implizit über die Variablen der Zustandsprädikate. Im Gegensatz zu *rQ-Learning* sind diese Variablen in jedem Zustand mit genau einer Aktion unifiziert. Zudem ist meine Zielsetzung für Projekt-2, dass der Agent mit einer leeren Zustandsliste beginnt und neue abstrakte Zustände und deren prädikatenlogische Beschreibung selbstständig lernt, sobald er auf neue Zustände im Spiel stößt.

V. Arbeitsplan Projekt-1

Das in TT2 für Q-Learning entwickelte 4-Gewinnt Spiel werde ich in Projekt-1 so erweitern, dass der Agent die Zustände des Spielfelds als konjunktive Verknüpfungen der genannten vier Prädikate wahrnimmt. Die Zustands-Aktions-Liste des Agenten werde ich in dieser Phase selbst vorgeben, indem ich in Testläufen prüfe, welche Zustände erreicht werden. Die Zustände werden anschließend in Bezug auf ihre strategische Relevanz abgeschätzt und dementsprechend in der Zustandsliste angeordnet. Die Zustandsliste könnte beispielsweise wie folgt aussehen:

Zustand	**Aktionen**
$4erkette(X) \wedge 4erblock(Y)$	X, Y
$4erkette(X)$	X
$4erblock(X)$	X
$3erkette(X) \wedge 3erblock(Y)$	X, Y
...	
$true$	1,2,3,4,5,6,7

Da bei der manuellen Vorgabe der Listen unmögliche alle möglichen Kombinationen von Prädikaten aufgenommen werden können, wird der tatsächliche Zustand wie eine Wissensdatenbank in Prolog interpretiert. Bisher wurde in den Beispielen die gesamte abstrakte Zustandsbeschreibung mit der gesamten tatsächlichen Zustandsbeschreibung unifiziert. Betrachtet man den Zustand als Wissensbasis, dann würde ein Zustand:

$$3erkette(1) \wedge 3erkette(2) \wedge 4erblock(2) \tag{9}$$

Durch jeden der folgenden abstrakten Zustände abgedeckt werden.

$$S_3 = 3erkette(X) \tag{10}$$

$$S_4 = 4erblock(X) \tag{11}$$

$$S_5 = 3erkette(X) \wedge 4erblock(X) \tag{12}$$

Aus diesem Grund spielt die sinnvolle Anordnung der Zustände in der Aktionsliste eine wichtige Rolle.
Um abschließend prüfen zu können, ob der später selbstständig lernende Agent mehr gelernt hat, als die Strategie, die halbwegs durch die Prädikate vorgegeben ist, werde ich in Projekt-1 noch einen Referenzagenten trainieren, der folgende feste Aktionsliste besitzt.

Zustand	**Aktionen**
$4erkette(X)$	X
$4erblock(X)$	X
$3erkette(X)$	X
$3erblock(X)$	X
$2erkette(X)$	X
$2erblock(X)$	X
$true$	1,2,3,4,5,6,7

Dieser Agent wird solange trainiert, bis keine Verbesserung der Werte sichtbar ist und dient dann als Referenzagent. Die Strategie, die dieser Agent ”lernt”, entspricht dem vorgegebenen Wissen aus der Tabelle aus Abschnitt III. Erwartungsgemäß sollte der Agent, der im Zuge von Projekt-2 entwickelt wird und selbstständig abstrakte Zustände lernt, besser spielen, als dieser Referenzagent. Anderenfalls würde dies den Schluss nahelegen, dass eine Zustandsbeschreibung in PL-1 zusammen mit Q-Learning keine Verbesserung gegenüber gewöhnlichem Q-Learning bringt, da der Agent lediglich nach einer vorgegebenen Strategie spielt.
Erst, wenn ein Agent signifikant besser spielt als dieser Referenzagent, kann die Aussage gemacht werden, dass der Agent einen spielerischen Vorteil aus der Zustandsbeschreibung ziehen kann.

REFERENCES

[1] N. Asgharbeygi, D. Stracuzzi, and P. Langley, "Relational temporal difference learning," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 49–56. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143851

[2] M. van Otterlo, *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty in First-Order and Relational Domains*. IOS Press, 2009, vol. 192.

[3] R. S. Sutton and A. G. Barto, "Time-derivative models of pavlovian reinforcement." 1990.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, pp. 237–285, 1996.

[5] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[6] L. C. Cobo, C. L. Isbell, and A. L. Thomaz, "Object focused q-learning for autonomous agents," in *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '13. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1061–1068. [Online]. Available: http://dl.acm.org/citation.cfm?id=2484920.2485087

[7] Q. P. Lau, M. L. Lee, and W. Hsu, "Distributed relational temporal difference learning," in *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '13. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1077–1084. [Online]. Available: http://dl.acm.org/citation.cfm?id=2484920.2485090

[8] P. Tadepalli, R. Givan, and K. Driessens, "Relational Reinforcement Learning: An Overview," in *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning*, 2004, pp. 1–9, uRL: http://www.cs.kuleuven.ac.be/cgi-bin-dtai/publ_info.pl?id=41343.

[9] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The tamer framework," in *Proceedings of the Fifth International Conference on Knowledge Capture*, ser. K-CAP '09. New York, NY, USA: ACM, 2009, pp. 9–16. [Online]. Available: http://doi.acm.org/10.1145/1597735.1597738

[10] S. Sanner and C. Boutilier, "Practical solution techniques for first-order MDPs," *Artificial Intelligence*, vol. 173, no. 5–6, pp. 748 – 788, 2009, advances in Automated Plan Generation. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370208001884

[11] E. F. Morales, "Scaling up reinforcement learning with a relational representation," in *In Proc. of the Workshop on Adaptability in Multi-agent Systems*, 2003, pp. 15–26.