FFHS
Fernfachhochschule Schweiz
Zürich | Basel | Bern | Brig
Mitglied der SUPSI

Fernfachhochschule Schweiz
Studiengang INF

Optimierte Lagerverwaltungs-Strategie durch Reinforcement Learning

Bachelor-Thesis im Studiengang INF

von

Benjamin Cirmena

Eingereicht bei:

Dr. Oliver Kamin
Departement Informatik
Departementsleiter

Referent:

Dr. Beat Tödtli
Dozent bei der FHS St.Gallen

Kehrsatz, 22. September 2020

Zusammenfassung

Lagerverwaltungssysteme benötigen durch die steigenden Anforderungen raffiniertere Strategien. Anpassungen in einem solchen System können einen grossen Einfluss auf die Strategie haben und würden daher eine Anpassung der Strategie benötigen. Mit Hilfe von Reinforcement Learning könnte eine solche Strategie selbstständig erlernt werden, welche möglicherweise die heuristische Strategie übertrifft. In dieser Arbeit wird ein erweiterbares Environment aufgebaut, mit dem möglichst unterschiedliche Lagersysteme dargestellt werden können. Anschliessend werden zwei Experimente aufgebaut, um die Performanz der durch Reinforcement Learning erlernten Strategien mit der heuristischen Strategie zu vergleichen. In den Experimenten werden die beiden Reinforcement Learning-Algorithmen Sarsa und Q-Learning verwendet. Dabei wird die erreichte Performanz, der durchschnittlich erreichte Reward, mit der Performanz der heuristischen Strategie verglichen. Die Resultate haben gezeigt, dass die beiden Algorithmen in den beiden Experimenten die heuristische Strategie übertroffen haben.

Abstract

Warehouse management systems require more sophisticated policies due to the increasing requirements. Adjustments in such a system can have a major impact on the policy and would therefore require policy adjustments. With the use of reinforcement learning, such a policy could be learned autonomously, which could possibly outperform the heuristic policy. In this thesis an extensible environment is built, which allows to represent different storage systems. Subsequently, two experiments are set up to compare the performance of the strategies learned by reinforcement learning with the performance of the heuristic strategy. In the experiments the two reinforcement learning algorithms Sarsa and Q-Learning are used. The achieved performance, the average achieved reward, is compared with the performance of the heuristic strategy. The results showed that the two algorithms outperformed the heuristic strategy in both experiments.

Inhaltsverzeichnis

Abkürzungen **v**

1. Einleitung **1**
1.1. Forschungsfragen 2
1.2. Abgrenzung und Vorgehensweise 2
1.3. Struktur der Arbeit 2

2. Reinforcement Learning **3**
2.1. Markov-Prozess 4
2.2. Markov Reward Process 5
2.3. Bellman-Expectation-Equation 6
2.4. Markov-Decision-Process 8
2.5. Bellman-Optimality-Equation 9
2.6. Model-free Prediction 11
2.6.1. Monte-Carlo-Learning 12
2.6.2. Temporal-Difference-Learning 12
2.7. Model-free Control 13
2.7.1. Epsilon-greedy Exploration 13
2.7.2. Sarsa 14
2.7.3. Q-Learning 15

3. Environment **16**
3.1. OpenAI Gym 16
3.1.1. Gym-Methoden 16
3.1.2. Gym-Attribute 17
3.2. WarehouseEnv 17
3.2.1. Environment-Step 18
3.2.2. State-Space 19
3.2.3. Action-Space 21
3.2.4. Orakel 22

3.3. Heuristik 22
3.4. Komplexität des Problems 22

4. Methodik **23**

5. Experiment 1 **25**
5.1. Beschreibung 25
5.1.1. Exploration 26
5.1.2. Discount-Factor 27
5.1.3. Lernrate 27
5.1.4. Messwerte 28
5.2. Resultate 29

6. Experiment 2 **35**
6.1. Beschreibung 35
6.1.1. Parameter 36
6.1.2. Messwerte 36
6.2. Resultate 37

7. Diskussion **43**

Abbildungsverzeichnis **45**

Tabellenverzeichnis **47**

Literaturverzeichnis **48**

A. Anhang **49**
A.1. Berechnung MDP - State-Values 49
A.2. Berechnung MDP - Action-Values 50

Abkürzungen

RL	Reinforcement Learning
MDP	Markov-Decision-Process
MRP	Markov-Reward-Process
MC	Monte-Carlo
TD	Temporal-Difference

1. Einleitung

Durch die Zunahme des Onlineshoppings und durch Next-Day oder sogar Same-Day-Lieferungen, steigen die Anforderungen an die Logistikprozesse im Hintergrund. Solche Logistikprozesse können komplex werden und müssen auch auf diverse Einflüsse reagieren können. Maschinelles Lernen könnte bei der Optimierung solcher Prozesse eine erhebliche Rolle spielen. Es gibt diverse algorithmische Ansätze, wobei in drei Gruppen aufgeteilt wird: überwachtes, unüberwachtes und bestärkendes Lernen. In dieser Thesis wird das bestärkende Lernen, auch Reinforcement Learning genannt, behandelt. Oft wird von Reinforcement Learning in Zusammenhang mit Spielen berichtet – sei es mit konventionellen Brettspielen wie Go oder Videospielen wie StarCraft. In dieser Arbeit wird überprüft, ob sich Reinforcement Learning eignet, um selbständig eine Policy zu erlernen, welche ein vereinfachtes Lager bewirtschaften kann (einlagern, bestellen, ausliefern). Des Weiteren soll aufgezeigt werden, ob sich Reinforcement Learning auch nach einer Skalierung des vereinfachten Beispiels immer noch eignet. Damit soll die Arbeit Aufschluss darüber geben, ob Reinforcement Learning auch in einem reellen Lager mit mehreren Tausend Artikeln funktionieren kann. Jing-Sheng Song et al. [1] haben prognostiziert, dass in Zukunft der Bedarf an Algorithmen, die in absehbarer Zeit die Abläufe einer Supply-Chain optimieren, immer stärker zunimmt. Es soll möglich sein, eine neue Anforderung festzulegen, sodass der Algorithmus selbständig sämtliche Teile der Supply-Chain optimiert. Die Mehrheit der Literatur befasst sich mit der Supply-Chain als Ganzes. So haben S. Kamal Chaharsooghi et al. [2] erfolgreich eine Supply-Chain-Strategie durch Reinforcement Learning erlernt, die das Simulationsspiel «The Beergame App» mit einem zufriedenstellenden Ergebnis spielt. Das Warenhaus an sich wird dabei jedoch kaum betrachtet. In den Warenhäusern ist neben dem Optimieren der Bestell- und Lieferabläufe auch das Optimieren der Lagerung selbst interessant. Um ein Lager zu optimieren, müssen Strategien festgelegt werden, nach denen die Artikel eingelagert werden. Eine neue Paketgrösse kann unter Umständen zu einer Anpassung der gesamten Strategie führen. Des Weiteren wird es mit zunehmenden Faktoren, wie der Lagertemperatur von Artikeln oder der Beliebtheit eines Artikels, auf die die Strategie reagieren soll, immer schwieriger, eine passende Strategie zu definieren.

1.1. Forschungsfragen

In dieser Engineering-Arbeit werden folgende Forschungsfragen mithilfe einer entwickelten Simulationsumgebung, welche im Abschnitt 3.2 definiert ist, untersucht. Diese Umgebung entspricht auch dem Geltungsbereich.

- **Forschungsfrage 1:** *Wie vergleicht sich die Performanz einer heuristischen Lagerverwaltungsstrategie mit einer durch Reinforcement Learning erlernten Strategie im zuvor definierten Environment?*
- **Forschungsfrage 2:** *Wie verhält sich die Performanz bei einer durch Reinforcement Learning erlernten Strategie mit einer zunehmenden Komplexität?*

1.2. Abgrenzung und Vorgehensweise

Diese Arbeit soll eine Grundlage für zukünftige Forschung bieten. Dabei wird der Fokus hauptsächlich auf die Lagerbewirtschaftung gelegt. Es wird ein vereinfachtes Environment erstellt, welches nur Basisfunktionen bietet. Des Weiteren wird in dieser Arbeit aus zeitlichen Gründen das Augenmerk auf grundlegende Algorithmen gerichtet. Um die beiden Forschungsfragen zu beantworten wird jeweils ein Experiment aufgebaut, welches die Performanz der Strategien mit dem durchschnittlich erhaltenen Reward vergleicht.

1.3. Struktur der Arbeit

Diese Arbeit ist wie folgt gegliedert: In Kapitel 2 wird eine Einführung ins Thema Reinforcement Learning gewährt, wobei der Fokus auf die grundlegenden Methoden und Algorithmen gelegt wird. In Kapitel 3 wird die das Environment definiert. In Kapitel 4 wird die angewendete Forschungsmethode beschrieben. In den Kapitel 5 und 6 werden die jeweiligen Experimente beschrieben, durchgeführt und die Resultate werden präsentiert. Anschliessend folgt in Kapitel 7 die Diskussion mit einem Ausblick auf weiterführende Arbeiten.

2. Reinforcement Learning

Dieses Kapitel dient dazu, die Grundlagen von Reinforcement Learning zu erläutern. Dabei wird besonderer Fokus auf die Terminologie sowie auf grundlegende Konzepte und Algorithmen gelegt. Im Gegensatz zu anderen Machine-Learning-Gebieten wird beim Reinforcement Learning nicht zwingend mit verfügbaren Daten gearbeitet. Die benötigten Informationen werden durch ‹Trial-and-Error› generiert. Ein sogenannter Agent führt in einer Simulationsumgebung, dem Environment, Handlungen aus. Diese Handlungen werden Actions genannt.

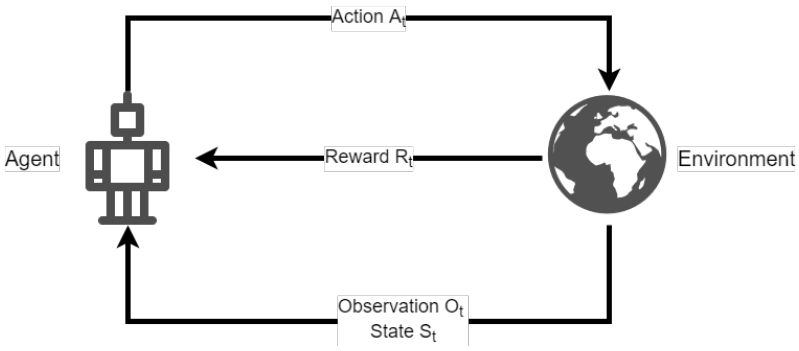

Abbildung 2.1.: RL-Agent und Environment

Nach jeder Action erhält der Agent einen Reward und eine Observation. Der Reward informiert den Agent, wie gut seine Action war. Dabei kann der Reward sowohl positiv als auch negativ sein. Die Wahl des Rewards hängt vom Environment und dem Task ab. Er könnte dem gewonnenen Score in einem Spiel entsprechen, einem Börsen-Kurs oder einem vordefinierten Wert für gewisse Umstände im Environment. Daneben erhält der Agent auch eine Observation. Die Observation enthält Informationen aus dem State des Environments. In einer perfekten Welt umfasst die Observation den gesamten Space des Environments. In diesem Fall würde der State im Agent dem des Environments entsprechen. Wenn in Reinforcement Learning vom State gesprochen wird, ist im Normalfall die Rede vom internen State des Agents. Das Ziel des Agents ist es, den gesamten Reward zu maximieren.

2.1. Markov-Prozess

Reinforcement Learning setzt voraus, dass Entscheidungen in einem Environment nach dem Markov-Decision-Process (MDP) getroffen werden. Jedes Environment handelt nach einem MDP. In den folgenden Kapiteln werden die Bestandteile, welche einen MDP ausmachen, beschrieben. Dadurch soll ein Verständnis aufgebaut werden, wie ein MDP gelöst werden kann. Ein MDP setzt die Markov-Eigenschaft voraus. Diese Eigenschafft ist erfüllt, wenn in einem stochastischen Prozess die Wahrscheinlichkeitsverteilung für zukünftige States nur vom aktuellen State abhängig ist und nicht von der Vergangenheit. Jede Umgebung kann grundsätzlich als MDP definiert werden; dazu müssen nur genügend Informationen im State vorhanden sein, welcher unter Umständen auch die Vergangenheit selbst beinhaltet.

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ \ldots, \ S_t] \tag{2.1}$$

Aufbauend auf dieser Eigenschaft entsteht ein Markov-Prozess. Dieser Prozess ist eine Sequenz von zufälligen States, welche die Markov-Eigenschaft haben. Der Markov-Prozess ist ein Tupel aus $\mathcal{S}$, dem State-Space, einer endlichen Anzahl möglicher States und $\mathcal{P}$, der Übergangsmatrix, welche für jeden State-Übergang die Wahrscheinlichkeit beschreibt. Anhand dieser Tupel kann eine Kopie des Environments erstellt werden, welche genau gleich auf die States reagiert.

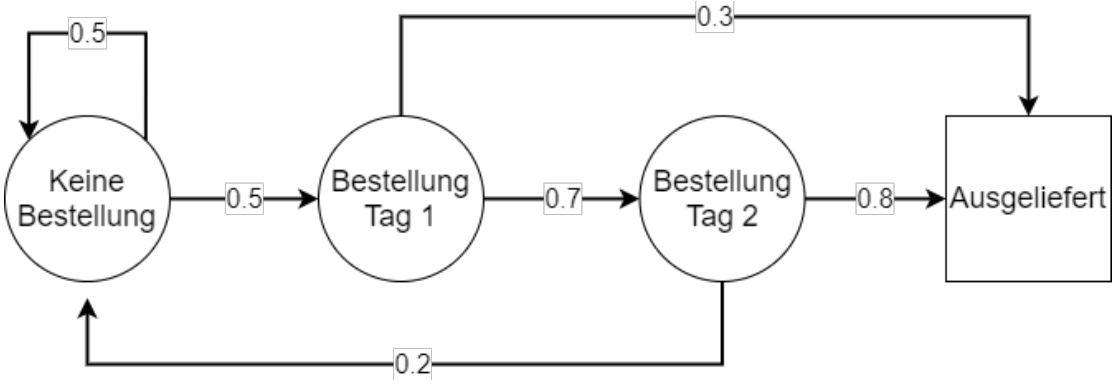

Abbildung 2.2.: Markov-Prozess

Das Beispiel in Abbildung 2.2 zeigt einen solchen Markov-Prozess auf. Im Status «Keine Bestellung» besteht eine Übergangswahrscheinlichkeit von 50 Prozent, dass eine neue Stellung eintrifft. Ist eine Bestellung eingetroffen, besteht die Wahrscheinlichkeit von 30 Prozent, dass der Artikel bereits vorhanden ist und direkt ausgeliefert wird. Wurde

die Bestellung nicht ausgeliefert, so besteht am nächsten Tag erneut eine Wahrscheinlichkeit, dass der Artikel nun vorhanden ist und ausgeliefert wird. Ist das nicht der Fall, annulliert in diesem Beispiel der Kunde die Bestellung.

2.2. Markov Reward Process

Der Markov-Reward-Process (MRP) baut auf dem Markov-Prozess auf und fügt eine Wertung der Übergänge hinzu. Der MRP wird durch die Tupel $\langle S, P, R, \gamma \rangle$ beschrieben. Dabei steht $\mathcal{R}$ für die Reward-Funktion; diese beschreibt, welcher State welchen Reward erhält.

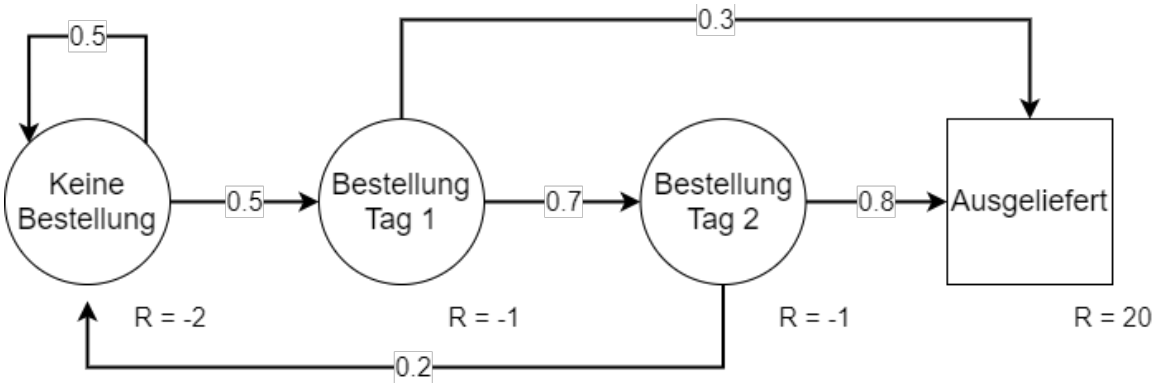

Abbildung 2.3.: Markov-Reward-Process

Der Discount-Factor γ beschreibt, wie stark zukünftige Rewards gewichtet werden. Ein Discount-Factor von 1 beachtet alle zukünftigen Rewards und ein Discount-Factor von 0 betrachtet nur den aktuellen Reward. Das Ziel in einem MRP ist es, den Return G_t, den gesamten diskontierten Reward eines Zeitschritts t, zu maximieren.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \ = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

Um den Return genauer zu veranschaulichen, folgen nun einige mögliche Sequenzen des MRPs aus Abbildung 2.3. Es wird ein Discount-Factor von 0.5 verwendet und die Sequenzen starten beim State «Keine Bestellung».

$$\boldsymbol{KB \rightarrow BT1 \rightarrow A}$$
$$v = -2 + (-1 * \frac{1}{2}) + (20 * \frac{1}{4}) \ = 2.5 \tag{2.3}$$

$$\boldsymbol{KB \rightarrow BT1 \rightarrow BT2 \rightarrow A}$$
$$v = -2 + (-1 * \frac{1}{2}) + (-1 * \frac{1}{4}) + (20 * \frac{1}{8}) \ = -0.25 \tag{2.4}$$

$$\boldsymbol{KB \rightarrow BT1 \rightarrow BT2 \rightarrow KB \rightarrow KB \rightarrow BT1 \rightarrow A}$$
$$v = -2 + (-1 * \frac{1}{2}) + (-1 * \frac{1}{4}) + (-2 * \frac{1}{8}) + (-2 * \frac{1}{16}) + (-1 * \frac{1}{32}) + (20 * \frac{1}{64})$$
$$v = -\frac{91}{32} \approx -2.84 \tag{2.5}$$

2.3. Bellman-Expectation-Equation

Die Value-Function wertet einen gegebenen State aus. Dabei wird der erwartete Return für den entsprechenden State berechnet.

$$\begin{aligned} v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\ v(s) &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ v(s) &= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right] \end{aligned} \tag{2.6}$$

Bei der Value-Function handelt es sich um eine rekursive Gleichung, wobei der Wert der aktuellen States aus dem zu erwartenden Reward und dem diskontierten Wert des folgenden States besteht. Dies entspricht der Bellman-Expectation-Equation; diese kann auch als Matrixrepräsentation dargestellt werden.

$$v = \mathcal{R} + \gamma \mathcal{P} v \tag{2.7}$$

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} \tag{2.8}$$

Diese lineare Gleichung kann direkt gelöst werden:

$$\begin{aligned} v &= \mathcal{R} + \gamma \mathcal{P} v \\ (I - \gamma \mathcal{P})\ v &= \mathcal{R} \\ v &= (I - \gamma \mathcal{P})^{-1}\ \mathcal{R} \end{aligned} \tag{2.9}$$

Dabei ist zu beachten, dass diese Variante nur für kleine MRPs geeignet ist, da die Zeitkomplexität für diese Berechnung $0(n^3)$ beträgt, wobei n für die Anzahl States steht. Für grössere MRPs gibt es erweiterte Methoden, welche noch in den folgenden Kapiteln erläutert werden. Nun kann diese Formel am MRP aus dem vorherigen Beispiel angewendet werden. In diesem Beispiel wird ein Discount-Factor von 1 verwendet, womit alle zukünftigen Returns berücksichtigt werden.

$$\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - 1 \begin{bmatrix} 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.7 & 0.3 \\ 0.2 & 0 & 0 & 0.8 \\ 0 & 0 & 0 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} -2 \\ -1 \\ -1 \\ 20 \end{bmatrix} = \begin{bmatrix} 13.37 \\ 17.37 \\ 17.67 \\ 20 \end{bmatrix} \tag{2.10}$$

In der folgenden Abbildung ist der MRP mit den Values der jeweiligen States ersichtlich.

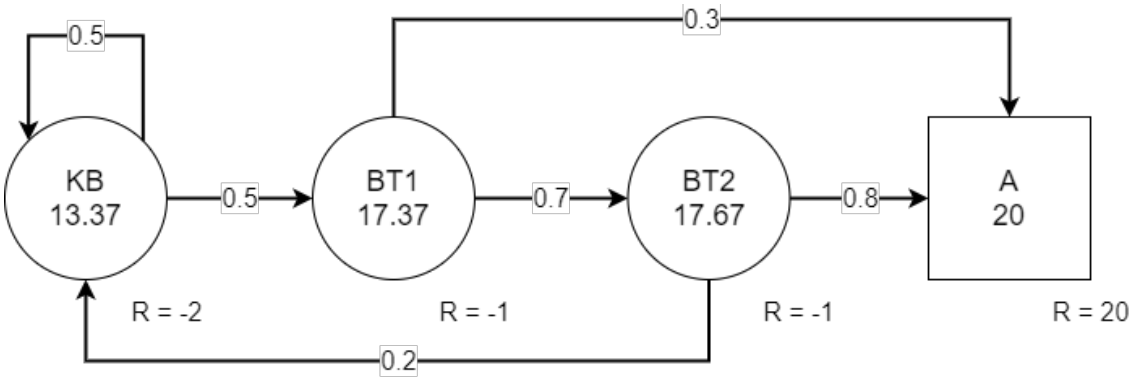

Abbildung 2.4.: MRP mit Values

Ob diese Values stimmen, kann nun problemlos mit einem vorausschauenden Schritt in einem beliebigen State überprüft werden. Die Zahlen wurden auf zwei Kommastellen gerundet.

$$
\begin{aligned}
v(BT2) &= \mathbb{E}[G_t|S_t = s] \\
v(BT2) &= R_{t+1} + \gamma v(KB) * P(KB) + \gamma v(BT1) * P(BT1) \\
&\quad + \gamma v(BT2) * P(BT2) + \gamma v(A) * P(A) \\
v(BT2) &= -1 + 13.37 * 0.2 + 17.37 * 0 + 17.67 * 0 + 20 * 0.8 \\
v(BT2) &= 17.67
\end{aligned} \tag{2.11}
$$

2.4. Markov-Decision-Process

Der Markov-Decision-Process (MDP) erweitert den MRP um $\mathcal{A}$, eine endliche Anzahl möglicher Actions. Somit wird der MDP mit dem folgenden Tupel definiert $\langle S, A, P, R, \gamma \rangle$. Dabei wird die Übergangsmatrix $\mathcal{P}$ mit der jeweiligen Action erweitert und beschreibt somit die Wahrscheinlichkeit der State-Action-Übergänge. Auch die Reward-Funktion $\mathcal{R}$ benötigt zusätzlich eine Action, um den Reward zu berechnen. Da nun die State-Übergänge durch Actions beeinflusst werden, muss die Value-Function angepasst werden. Die Value-Function wird nun so angepasst, dass sie einer Strategie, einer sogenannten Policy π, folgt. Eine solche Policy beschreibt für einen State S, mit welcher Wahrscheinlichkeit die möglichen Actions gewählt werden. Die Value-Function $v_\pi(S)$ beschreibt den Wert eines States – vorausgesetzt, die zukünftigen Actions werden nach der Policy π gewählt.

$$
v_\pi(S) \; = \; \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right] \tag{2.12}
$$

Um einen MDP zu lösen und eine optimale Policy zu finden, muss unterschieden werden, ob das Model bekannt ist, welches das Environment beschriebt – also die kompletten Tupel, die den MDP beschreiben. Oft sind nicht alle Dynamiken eines Environments bekannt oder die Menge an zu verarbeitenden Informationen ist nicht mehr geeignet.

In Abbildung 2.5 wird erneut das Lagerbeispiel verwendet. Die States wurden zudem mit der Information erweitert, ob ein Artikel auf Lager ist oder nicht. Ausserdem verfügt nun der Agent über drei verschiedene Actions: A1 – *Warten*, A2 – *Beim Lieferanten bestellen* und A3 – *Ausliefern*. Die drei Kreuzungen zwischen den States stellen die proba-

bilistischen Ausgänge aus den jeweiligen Actions dar. Falls in diesem Beispiel ein Artikel beim Lieferanten bestellt wird, so ist dieser garantiert am nächsten Tag da. Die Wahrscheinlichkeit, dass am nächsten Tag eine neue Bestellung von einem Kunden eintrifft, ist nur 50 Prozent.

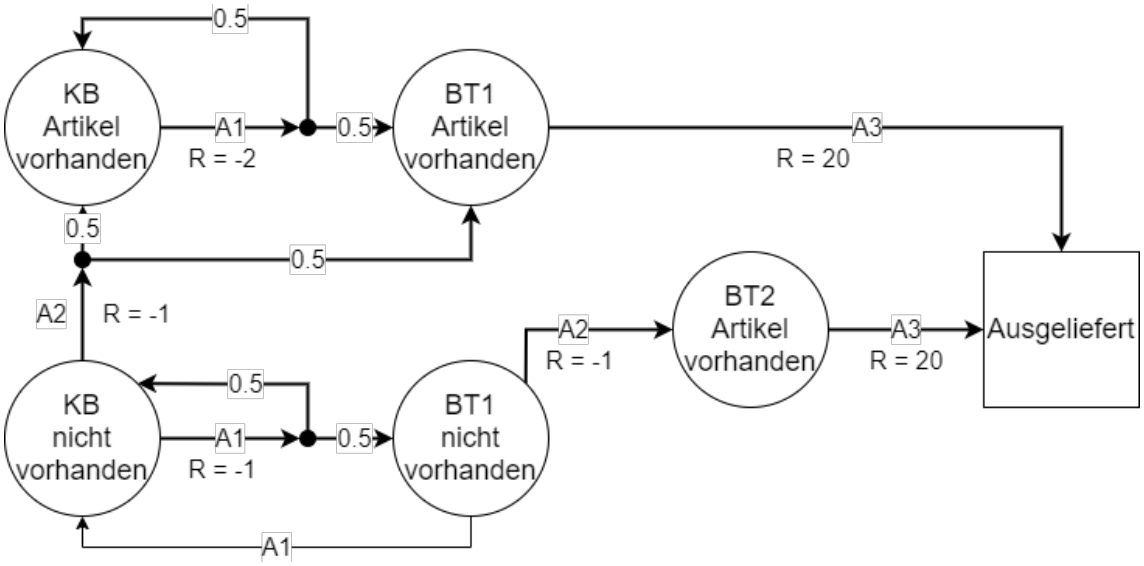

Abbildung 2.5.: Markov-Decision-Process

2.5. Bellman-Optimality-Equation

Anders als bei der Bellman-Expectation-Equation, die bereits zum Evaluieren der Value-Function einer bestimmten Policy benötigt wird, ist die Bellman-Optimality-Equation keine lineare Gleichung. Wenn in der Literatur von der Bellman-Equation die Rede ist und nicht weiter spezifiziert wird, betrifft es meistens die Bellman-Optimality-Equation. Das Ziel dieser Gleichung ist es nicht, den Wert eines States bei einer definierten Policy π zu evaluieren, sondern eine optimale Policy $*$ zu finden.

$$
\begin{aligned}
v_*(s) &= \max_{\pi} v_\pi(s) \\
v_*(s) &= \max_{a} \mathcal{R}(s,a) + \gamma \sum_{S'} \mathcal{P}(s,a,s') v_*(s') \qquad (2.13)
\end{aligned}
$$

Dabei wird bei der optimalen State-Value-Function nicht die optimale Policy gesucht, sondern der maximal mögliche Reward, welcher der Agent aus einem Environment bekommen kann. Im Beispiel des MDP würden bei einem Discount-Factor von 1 folgende Values errechnet (Anhang A.1):

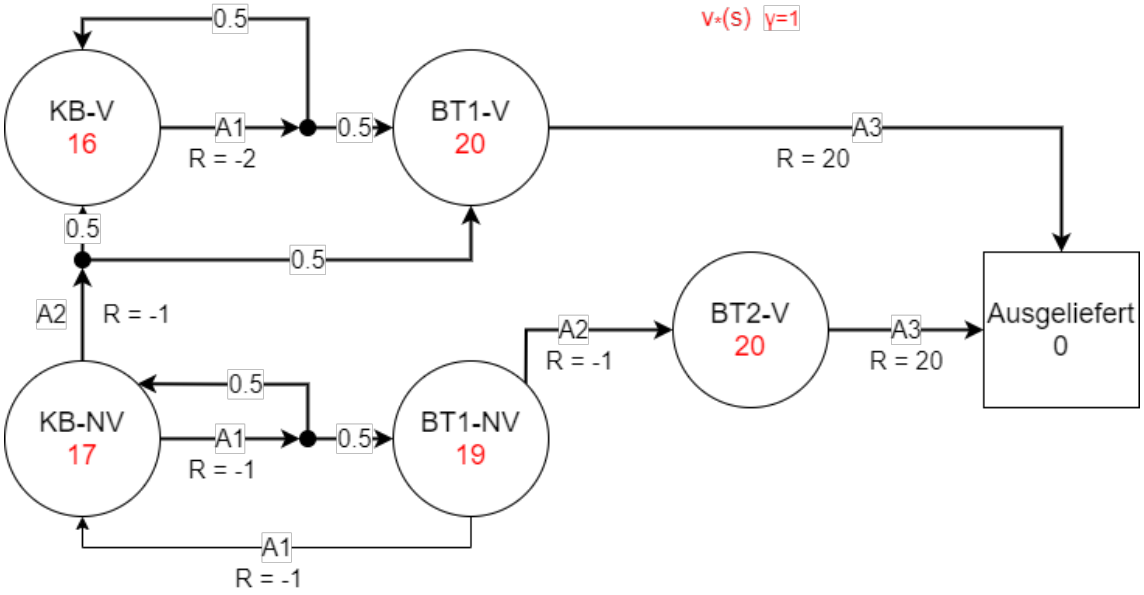

Abbildung 2.6.: MDP-State-Value-Function

Um die optimale Policy zu finden, welche den MDP komplett löst, wird die optimale Action-Value Function benötigt.

$$
\begin{aligned}
q_*(s,a) &= \max_{\pi} q_\pi(s,a) \\
q_*(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{S'} \mathcal{P}(s,a,s') max\ q_{*a'}(s',a')
\end{aligned}
\tag{2.14}
$$

Die optimale Action-State-Function sagt für einen State und eine Action aus, welches der maximale zukünftige Reward für die aktuelle Situation ist. Die entsprechenden Q-Values für diese optimale Policy wurde im Anhang A.2 berechnet.

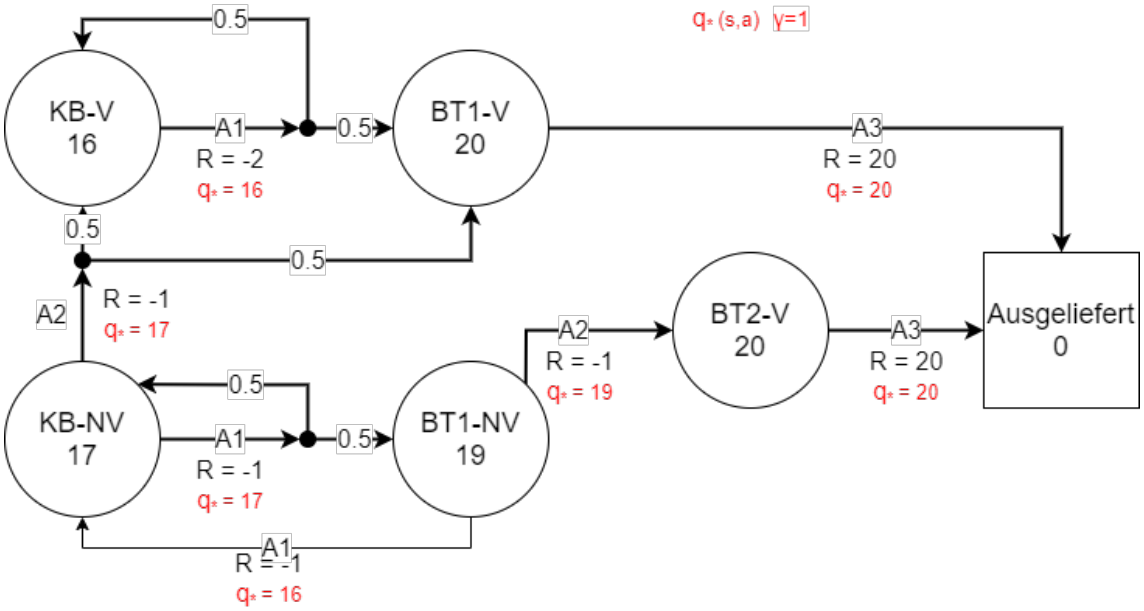

Abbildung 2.7.: MDP-Action-Value-Function

Betrachtet man in diesem Beispiel (Abbildung 2.7) den Start-State «KB-NV», so wird nun klar, dass der maximal mögliche Reward 17 ist, egal ob die Action A1 – *Warten* oder die Action A2 – *Beim Lieferanten bestellen* ausgeführt wird. Mit diesen Q-Values der optimalen Policy können nun die Actions in diesem MDP immer optimal gewählt werden. Ein MDP hat zwingend mindestens eine deterministisch optimale Policy.

2.6. Model-free Prediction

Bei der Model-free Prediction kennt der Agent das Model des MDP nicht. Der Agent sammelt Informationen über das Environment und versucht so, seine aktuelle Policy zu evaluieren. Dazu gibt es diverse Methoden, welche aber im Grundsatz auf den folgenden beiden Methoden aufbauen.

2.6.1. Monte-Carlo-Learning

Die Monte-Carlo(MC)-Methode evaluiert eine Policy anhand der Erfahrung ganzer Episoden. Eine Episode stellt eine Sequenz an Actions dar, die in einem Environment durchgeführt werden. Diese Durchführung von Actions geschieht so lange, bis ein Abbruchkriterium erreicht ist und somit die Episode beendet wurde. Der Agent führt eine bestimmte Sequenz lang Actions nach der Policy durch, bis die Episode beendet wird. Dabei wird eine Anzahl an Episoden festgelegt, die der Agent durchläuft. Bei dieser Methode benötigt der Agent zwingend die kompletten Episoden zum Evaluieren. Der Agent durchläuft dabei die Episoden, nimmt den jeweiligen Return und berechnet den durchschnittlichen Return für alle Episoden. Der dabei berechnete empirische Return konvergiert mit steigender Anzahl der Episoden gegen den zu erwartenden Return. Somit nähert sich auch die Value-Function der tatsächlichen Value-Function an.

2.6.2. Temporal-Difference-Learning

Im Gegensatz zur MC-Methode evaluiert die Temporal-Difference(TD)-Methode eine Policy nicht mit der Erfahrung aus einer kompletten Episode, sondern aus einer Schätzung, welche aus einer Sequenz von Actions durchgeführt wurde. Es wird eine bestimmte Anzahl n Actions durchgeführt, mit welchen anschliessend eine Schätzung für den Return durchgeführt wird. Daraufhin wird nach den nächsten n Actions die Schätzung der Values mit der neuen Schätzung des Returns aktualisiert. Dieses Vorgehen wird Bootstrapping genannt, da eine Schätzung aufgrund einer anderen Schätzung durchgeführt wird.

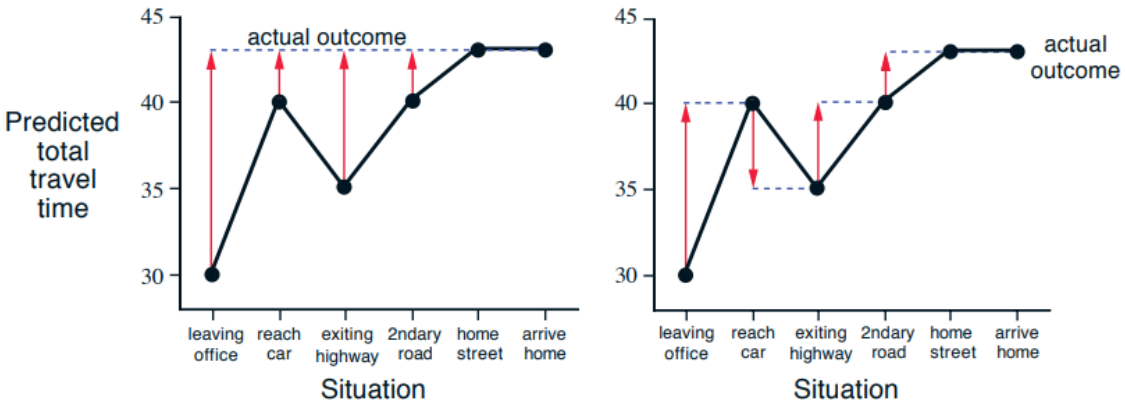

Abbildung 2.8.: Unterschied MC & TD [3]

2.7. Model-free Control

Ist das Model eines MDPs bekannt und der Agent kennt die Übergangswahrscheinlichkeiten sowie die Reward-Funktion des MDPs, so kann mittels Dynamic-Programming und Policy-Iteration die optimale Policy erlernt werden. Oft ist in der Realität bei Problemstellungen, in denen Reinforcement Learning angewendet wird, kein Model bekannt, welches den genauen MDP beschreibt. Daher muss der Agent wie in der Model-free Prediction mithilfe gemachter Erfahrungen die aktuell angewendete Policy evaluieren und verbessern. Zur Verbesserung einer Policy ist ein gewisses Mass an Exploration nötig. Ein Agent, welcher nicht exploriert, sondern nur exploitiert, wählt die aus seiner aktuellen Policy beste Action. Stellt man sich ein minimales Beispiel vor, in dem eine Action im aktuellen State einen Reward von 2 ergibt und die andere einen Reward von 10, wird – sofern die erwarteten Returns mit 0 initialisiert wurden und somit beide einen gleichen erwarteten Return haben – eine zufällige erste Action gewählt. Bei der Wahl einer Action, welche einen Reward von 2 ergibt, wird ohne Exploration nie eine andere Action ausprobiert und der Reward von 10 würde somit nie erreicht werden.

2.7.1. Epsilon-greedy Exploration

Die Epsilon-greedy Exploration ist eine oft verwendete Methode, um sicherzustellen, dass der Agent neue Actions ausprobiert und somit das ganze Environment kennenlernt. So kann er die bestmögliche Policy finden. Dabei wird ein Wert ϵ (Epsilon) zwischen 0 und 1 festgelegt. Vor jeder Action wird mit der Wahrscheinlichkeit 1 - ϵ die beste Action, die Greedy Action ausgeführt. Mit der Wahrscheinlichkeit ϵ wird eine zufällige Action ausgeführt. Würden unendlich viele Schritte durchgeführt, so würde jedes State-Action-Paar unendlich oft besucht werden. Somit kann ausgeschlossen werden, dass eine bestimmte Policy nicht entdeckt wird.

$$\pi(a|s) \begin{cases} 1-\epsilon, & \underset{a \in A}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon, & random(a) \end{cases} \tag{2.15}$$

Ein Nachteil bei der Epsilon-greedy Exploration ist, dass die Policy nie wirklich mit der optimalen Policy konvergiert, da weiterhin zufällig Actions ausgeführt werden. Eine Möglichkeit, dem entgegenzuwirken, besteht darin, ϵ über die Zeit zu verkleinern.

2.7.2. Sarsa

Der Sarsa-Algorithmus [4] basiert auf dem Prinzip der TD. Dabei wird in einem State-Action-Paar gestartet. Der Agent erhält ein Reward und erreicht einen neuen State. Nun nimmt der Agent seine Schätzung für das folgende State-Action-Paar der Policy; dieser diskontierte Q-Value zusammen mit dem Reward muss laut der Bellman-Equation dem vorherigen Q-Value entsprechen. Der Ausgangs-Q-Value $Q(S, A)$ wird anhand der Lernrate α in Richtung TD-Target korrigiert (2.16). Die Differenz zwischen TD-Target und dem Ausgangs-Q-Value ist der TD-Error; dieser muss bei einer Greedy Policy konvergieren, sonst wurde keine optimale Policy gefunden.

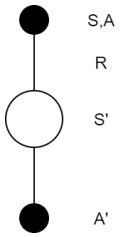

Abbildung 2.9.: Sarsa-Update

$$
\begin{aligned}
Q(S, A) &\leftarrow Q(S, A) + \alpha\left(R + \gamma Q\left(S', A'\right) - Q(S, A)\right) \\
Q(S, A) &\leftarrow Q(S, A) + \alpha\left(TDTarget - Q(S, A)\right) \\
Q(S, A) &\leftarrow Q(S, A) + \alpha\left(TDError\right)
\end{aligned} \tag{2.16}
$$

Algorithm 1 Sarsa-Algorithmus [5] Chapter 6

Algorithm parameters: step size $\alpha \in$ (0, 1], small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+, a \in A(s)$, arbitrarily except that $Q(terminal)$=0
repeat for each episode
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    repeat for each step of episode
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q' (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'; A \leftarrow A';$
    until S is terminal

2.7.3. Q-Learning

Im Gegensatz zu Sarsa ist Q-Learning [6, 7] ein Off-Policy-Algorithmus. Das bedeutet, der Agent wählt zwar – wie bei Sarsa – eine Action nach der Epsilon-greedy Policy, nimmt aber für die Schätzung die Action nach der Greedy Policy. Dieses Vorgehen entspricht dem Prinzip der Bellman-Optimality-Equation.

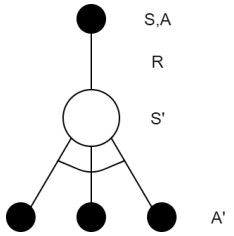

Abbildung 2.10.: Q-Learning-Update

$$Q\left(S,A\right) \leftarrow Q\left(S,A\right) + \alpha\left(R + \gamma \max_{a'} Q\left(S',a'\right) - Q\left(S,A\right)\right) \tag{2.17}$$

Algorithm 2 Q-Learning-Algorithmus [5] Chapter 6

1: Algorithm parameters: step size $\alpha \in$ (0, 1], small $\epsilon > 0$
2: Initialize $Q(s,a)$, for all $s \in S^+, a \in A(s)$, arbitrarily except that $Q(terminal)$=0
3: **repeat** for each episode
4: Initialize S
5: **repeat** for each step of episode
6: Choose A from S using policy derived from Q (e.g., ϵ-greedy)
7: Take action A, observe R, S'
8: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_a Q(S',a) - Q(S,A)]$
9: $S \leftarrow S'$
10: **until** S is terminal

3. Environment

Um die Forschungsfragen dieser Thesis zu beantworten, ist ein Environment nötig, welches auf minimale Weise Prozesse in einem Lager abbildet. Dabei soll das Environment so aufgebaut werden, dass ein Agent lernen muss, möglichst wenige Artikel am Lager zu halten, damit Kosten eingespart werden. Des Weiteren müssen wiederum genügend Artikel gelagert sein, sodass keine Lieferfrist vergeht und somit ein Auftrag storniert wird. Dieses Environment soll dabei möglichst erweiterbar aufgebaut werden und dem gängigen Standard von Reinforcement Learning Environments entsprechen. Dadurch wird sichergestellt, dass ohne grossen Aufwand verschiedene Algorithmen überprüft werden können.

3.1. OpenAI Gym

OpenAI, ein Unternehmen, welches sich mit der Erforschung von künstlicher Intelligenz beschäftigt, hat mit Gym [8] einen Standard für Reinforcement Learning geschaffen. Dabei war es ihr Ziel, eine Library an Environments zur Verfügung zu stellen, um Algorithmen einfacher über verschiedene Situationen zu benchmarken. Hierzu wurde darauf geachtet, dass das Aufsetzen eines Environments ohne grossen Aufwand möglich ist und somit auch die Reproduzierbarkeit veröffentlichter Forschung erleichtert wird.

3.1.1. Gym-Methoden

OpenAI Gym sollte folgende Methoden umfassen:

- **step():** Diese Methode startet einen Zeitschritt im Environment, wobei die auszuführende Action übergeben wird. Der Rückgabe-Wert dieser Funktion enthält immer eine Observation vom Typ «object», ein Reward vom Typ «float», ein Flag vom Typ «boolean», welches bestimmt, ob die aktuelle Sequenz in einem Terminalstatus angekommen ist, sowie zusätzliche Debug-Informationen vom Typ «dict», welche für den Agent nicht sichtbar sein sollten.

- **render():** Diese optionale Methode ermöglicht es, mit dem Gym-integrierten Viewer ein Frame vom Environment zu rendern.
- **reset():** Diese Methode setzt den aktuellen Status im Environment auf den Start zurück. Dabei wird hier auch eine erste Observation zurückgegeben. Reset wird im Normalfall zu Beginn und nach dem «*done*»-Flag nach jeder Sequenz ausgeführt.

3.1.2. Gym-Attribute

Folgende Attribute müssen im Gym enthalten sein:

- **observation_space:** Dieses Attribut widerspiegelt den observierbaren State-Space. Dabei kann es ein Objekt sein, welches dasselbe Format hat wie die Observationen aus den reset()- und step()-Methoden oder eine Instanz der Klasse Space, welche von OpenAI zur Verfügung gestellt wird.
- **action_space:** Dieses Attribut kann ein Array von möglichen Actions sein oder auch eine Instanz der Klasse Space.

3.2. WarehouseEnv

Beim erstellten «WarehouseEnv» handelt es sich um ein Environment, welches einen terminalen State hat. Jede Episode hat genau 100 Zeitschritte, sogenannte Steps.

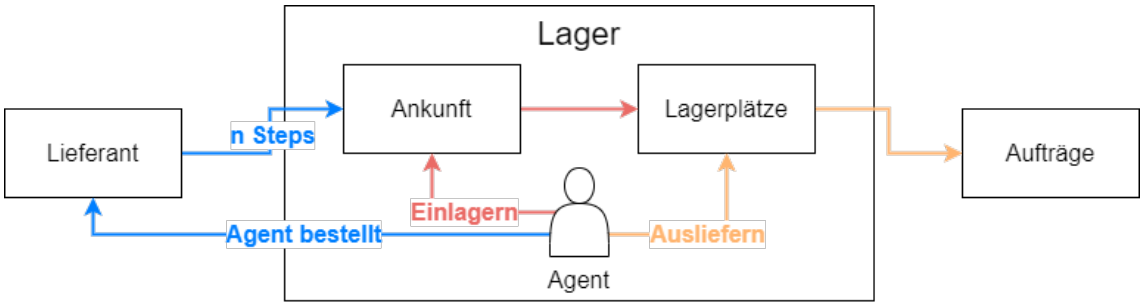

Abbildung 3.1.: WarehouseEnv

3.2.1. Environment-Step

In jedem Step wird zuerst das Environment aktualisiert. Dabei wird überprüft, ob Aufträge storniert wurden und ob eine Bestellung im Wareneingang angekommen ist. Die Rewards dieser beiden Überprüfungen sowie ein Reward für die Lagerkosten werden nun vorgemerkt. Anschliessend findet die eigentliche Action statt; diese wird im Environment durchgeführt. Der Reward, welcher sich aus dem direkten Reward der Action sowie dem Reward der Aktualisierung ergibt, wird berechnet. Der Agent erhält nun eine Observation aus dem State und den Reward des durchgeführten Steps.

Lagerkosten

Reward	Dieser Reward wird vor jeder Aktion verteilt. Er ist abhängig vom genutzten Lagerplatz und soll somit Lagerkosten simulieren. **-10 * Anzahl benutzter Lagerplätze**

Tabelle 3.1.: Lagerkosten

Stornierter Auftrag

Reward	Dieser Reward wird vor jeder Aktion verteilt. Wird ein Auftrag storniert, so wird ein Reward von **-50** hinzugefügt.

Tabelle 3.2.: Stornierter Auftrag

Bestellungseingang

Reward	Der Reward ist davon abhängig, ob im Wareneingang ein Platz frei ist. Artikel angekommen: **0** Artikel verworfen: **-25**

Tabelle 3.3.: Bestellungseingang

3.2.2. State-Space

Der State-Space besteht aus sämtlichen Entitäten und deren möglichen Zuständen. Die Anzahl der Entitäten sind jeweils Konstanten, welche im Environment festgelegt werden.
Lagerplätze

Anzahl	Anzahl der möglichen Lagerplätze	INT (const)	3
Artikel	Pro Lagerplatz kann genau ein Artikel gelagert werden.	Artikel	Instanz

Tabelle 3.4.: Lagerplätze

Artikel

Anzahl	Anzahl der möglichen Artikel	INT (const)	1
Häufigkeit	Die Häufigkeit definiert, mit welcher Wahrscheinlichkeit ein Artikel bestellt wird.	FLOAT	[0-1]

Tabelle 3.5.: Artikel

Aufträge
Sofern nicht die maximale Anzahl an Aufträgen ansteht, wird nach jedem vierten Step ein neuer Auftrag generiert. Dabei wird der Artikel nach der Artikelhäufigkeit gewählt. Ein Auftrag hat ausserdem eine Frist von 5 Steps. Wurde der Auftrag nicht ausgeliefert und somit abgeschlossen, so wird dieser vom Auftraggeber storniert.

Anzahl	Maximale Anzahl anstehender Aufträge	INT (const)	2
Artikel	Jeder Auftrag enthält genau einen Artikel.	Artikel	Instanz

Tabelle 3.6.: Aufträge

Wareneingang
Die Artikel, die beim Hersteller bestellt wurden, erscheinen nach der Lieferzeit von zwei Steps im Wareneingang.

Anzahl	Anzahl der Plätze im Wareneingang	INT (const)	2
Artikel	Jeder Platz im Wareneingang kann genau einen Artikel enthalten.	Artikel	Instanz

Tabelle 3.7.: Wareneingang

Bestellungen
Gibt der Agent eine Bestellung auf, so ist diese hier ersichtlich, bis der Artikel im Wareneingang eingetroffen ist.

Anzahl	Anzahl der gleichzeitigen Bestellungen	INT (const)	2
Artikel	Jede Bestellung enthält genau einen Artikel.	Artikel	Instanz

Tabelle 3.8.: Bestellungen

Eine Observation, welche dem Agent nach jedem Step zur Verfügung steht, besteht aus folgenden Teilen.

State-Teil	Lagerplätze	Aufträge	Wareneingang	Bestellungen
Möglicher Zustand	[0,0,0]	[0,0]	[0,0]	[0,0]

Tabelle 3.9.: Observation

Dabei kann in jedem Element der State 0 für keinen Artikel, 1 für Artikel 1 und 2 für Artikel 2 angenommen werden.

3.2.3. Action-Space

Der Action-Space definiert die möglichen Aktionen, die der Agent ausführen kann:
Lagern

Ablauf	Nimmt einen Artikel aus dem Wareneingang und platziert diesen im Lager
Reward	Der Reward ist abhängig vom Erfolg des Einlagerns: Erfolgreiches einlagern: **0** Besetzter Platz: **-10**

Tabelle 3.10.: Lagern

Ausliefern

Ablauf	Nimmt einen Artikel aus dem Lager und schliesst damit den gewählten Auftrag ab
Reward	Der Reward ist abhängig vom Erfolg der Aktion: Richtiger Artikel abgegeben: **100** Falscher Artikel abgegeben: **-100**

Tabelle 3.11.: Ausliefern

Bestellen

Ablauf	Bestellt einen Artikel beim Lieferanten; der Artikel wird nach zwei weiteren Actions im Wareneingang ankommen. Sind im Wareneingang bereits alle Plätze besetzt, so wird der Artikel verworfen.
Reward	Hier entsteht kein direkter Reward, der Reward wird bei der Ankunft berechnet.

Tabelle 3.12.: Bestellen

Warten

Ablauf	Diese Action ist ein Platzhalter; es wird im Environment der Step durchgeführt, jedoch keine Action.
Reward	Der Reward dieser Action ist immer **0**.

Tabelle 3.13.: Warten

3.2.4. Orakel

Um die Komplexität zu verringern, werden zwei Orakel verwendet. Es handelt sich hierbei um Entscheidungen, welche nach einfachen Bedingungen optimal gelöst werden können.
Lagerplatz wählen:
Dieses Orakel gibt dem Agent einen möglichen Lagerplatz. Damit wird ausgeschlossen, dass ein bereits besetzter Platz gewählt wird.
Artikel für die Lieferung wählen:
Dieses Orakel gibt dem Agent einen Artikel an, welcher geliefert werden kann. Damit kann ausgeschlossen werden, dass ein falscher Artikel ausgeliefert wird. Es wird automatisch der Auftrag gewählt, welcher am längsten offen ist.

3.3. Heuristik

Um die durch Reinforcement Learning erlernte Strategie zu vergleichen, wird eine Heuristik definiert. Ein Agent wird nach dieser Anleitung seine Actions wählen:
Priorität 1 – Ausliefern:
Sofern eine Bestellung ansteht und der entsprechende Artikel im Lager ist, wird dieser Artikel ausgeliefert. Dadurch sollen die Lagerkosten geringgehalten und das Ablaufen einer Bestellung vermieden werden.
Priorität 2 – Einlagern:
Da im Lagereingang nur begrenzt Platz vorhanden ist, sollen Artikel möglichst schnell eingelagert werden.
Priorität 3 – Bestellen:
Es wird ein Artikel bestellt, sofern nicht bereits einer am Lager ist oder bestellt wurde.
Priorität 4 – Warten:
In diesem Fall macht der Agent nichts; es soll vermieden werden, dass unnötige Lagerplätze besetzt werden.

3.4. Komplexität des Problems

Um die für Forschungsfrage 2 relevante Komplexität des Problems zu erhöhen, werden nicht die Orakel entfernt, denn diese haben nur simple Entscheidungen zu fällen. Eine sinnvollere Komplexitätssteigerung wird durch das Hinzufügen eines weiteren Artikels erreicht. Dieser Artikel erhöht damit den State und zwingt den Agent zu bestimmen, welcher Artikel bestellt werden soll.

4. Methodik

Zur Beantwortung der zwei Forschungsfragen werden zwei Experimente aufgebaut. Dabei wird der Design-Science-Ansatz verfolgt, welcher von Hevner et al. in [9] beschrieben wird. Folgende Guideline wird dabei verwendet:

Design as an Artifact:
Das Environment ist dabei das Artefakt, mithilfe dessen die zwei Forschungsfragen beantwortet werden.

Problem-Relevance:
Die Relevanz des Problems wurde in der Einleitung bereits aufgezeigt und besteht darin, dass eine mögliche Optimierung von Lagerverwaltungen durch maschinelles Lernen Kosten und Zeitersparnisse erzielen könnte.

Design-Evaluation:
Zum Evaluieren wird eine experimentelle Methode angewendet; es wird für jede Forschungsfrage ein kontrolliertes Experiment erstellt, in welchem die Forschungsfrage überprüft wird.

Research-Contribution:
Der wissenschaftliche Beitrag besteht neben den beantworteten Forschungsfragen, welche einen ersten Eindruck über die Anwendbarkeit von Reinforcement Learning in Lagerverwaltungen bieten sollen, auch aus dem Environment, welches sich als Artefakt identifiziert. Das Environment soll als erweiterbare Grundlage dienen, um weiterführende Forschung in diesem Bereich durchzuführen.

Research-Rigor
In der Konstruktions- sowie in der Evaluationsphase müssen Methoden angewendet werden, welche sich bereits etabliert haben. Dabei hat sich der Verfasser beim Erstellen des Environments an die Vorgaben von OpenAI [8] gehalten. Zum Evaluieren werden die zwei grundlegenden Algorithmen für Reinforcement Learning verwendet.

Design as a Search-Process:
Die Iterationen, welche in dieser Arbeit erfasst werden, behandeln eine simplifizierte Version des Problems und dessen Überprüfung.

Communication of Research:
Die Erkenntnisse müssen dem technischen Publikum in Form eines reproduzierbaren Artefakts übermittelt werden, welches anschliessend für die Anwendung oder – in diesem Fall – für weiterführende Forschungen zur Verfügung steht. Ausserdem wird dem Publikum aus dem Management-Bereich das Wissen vermittelt, inwiefern das Artefakt in ihrem Problem angewendet werden kann.

Reproduzierbarkeit
Sämtliche Experimente wurden in einem separaten Branch im Repository festgehalten, was die Reproduzierbarkeit der Experimente drastisch verbessern soll. Der entsprechende Link ist jeweils am Ende jedes Experimentes zu finden. Im README.md sind die Abhängigkeiten sowie die Anweisungen zum Reproduzieren des Experiments dokumentiert.

5. Experiment 1

Bei der ersten Forschungsfrage soll überprüft werden, wie sich eine durch Reinforcement Learning erlernte Policy mit der Policy der Heuristik vergleicht, welche in Abschnitt 1.1 beschrieben wurde.

5.1. Beschreibung

Um die erste Forschungsfrage empirisch zu beantworten, bedarf es eines geregelten Vorgehens. Es werden zwei durch unterschiedliche Reinforcement Learning-Algorithmen optimierte Policies mit der Policy der Heuristik verglichen. Diese werden einmal durch Sarsa und einmal durch Q-Learning gelernt. Um zwei optimierte Policies zu erlernen, müssen die Parameter gewählt werden, welche den Reward einer Policy maximieren. Dazu werden verschiedene Durchgänge ausgeführt, bei denen jeweils nur ein Parameter angepasst wurde.

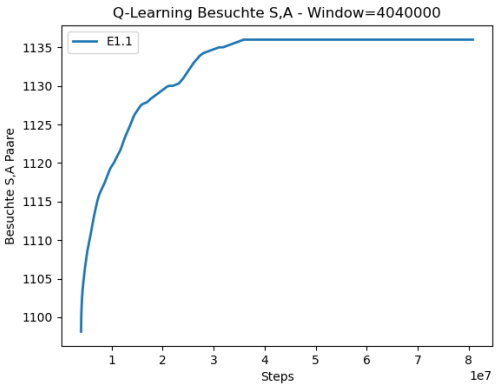

Abbildung 5.1.: SA-Paare – Zufällige Actions 800’000 Episoden

Um sicherzustellen, dass sämtliche möglichen State-Action-Paare besucht werden, wurde ein rein explorativer Agent für 800’000 Episoden aufgeführt. In Abbildung 5.1 ist

ersichtlich, wie nach ungefähr 400'000 Episoden 1136 State-Action-Paare besucht wurden. In der zweiten Hälfte wurden dabei keine weiteren States gefunden, woraus sich schliessen lässt, dass keine weiteren State-Action-Paare erreichbar sind. Um die Parameter zu prüfen, werden jeweils 50'000 Episoden durchgeführt. Diese garantieren nicht, dass jedes State-Action-Paar besucht wurde, da dies bei einem Agent, der zufällige Actions wählt, erst nach ungefähr 400'000 Episoden der Fall ist. Sie geben aber bereits einen hilfreichen Eindruck, welche Parameter besser geeignet sind. Die Policies mit den richtigen Parametern werden aber anschliessend mit 800'000 Episoden erlernt. Ausserdem wird überprüft, ob die 1136 State-Action-Paare erreicht wurden.

5.1.1. Exploration

Als Erstes wird die Exploration untersucht. Der Explorationswert ϵ wird zu Beginn bei 1 festgelegt und über die Zeit mit einem Faktor, dem Epsilon-Verfall, verringert, bis ein Minimum von 0.05 erreicht wird. Dadurch soll sichergestellt werden, dass am Ende immer noch eine 5-prozentige Chance besteht, dass eine zufällige Action gewählt wird. Es werden unterschiedliche Werte für den Epsilon-Verfall gewählt und untersucht. Hierbei wird versucht, möglichst kurz zu explorieren, aber sicherzustellen, dass eine optimale Policy gefunden wird und der Agent nicht bei einem lokalen Optimum verbleibt. Dazu wird verglichen, nach wie vielen Steps wie viele States besucht wurden. Dabei wird im Durchgang 1.5 ein Verfall von 1 gewählt, um so ein rein exploratives Vorgehen zu erzwingen. Ein Parameter möglichst nah an 1, welcher noch genügend Episoden zum Optimieren übriglässt, wäre ideal. Folgende Parameter werden geprüft:

Durchgang	Discount-Factor γ	Lernrate α	ϵ-Verfall
E1.1	0.9	0.5	**0.9985**
E1.2	0.9	0.5	**0.9990**
E1.3	0.9	0.5	**0.9995**
E1.4	0.9	0.5	**0.9999**
E1.5	0.9	0.5	**1**

Tabelle 5.1.: Experiment 1 – Epsilon-Verfall

5.1.2. Discount-Factor

Der Discount-Factor γ definiert, wie stark zukünftige Rewards gewertet werden. Ein Discount-Factor von 0 bedeutet, dass der Agent ausschliesslich auf den aktuellen Reward achtet. Im Gegensatz dazu sagt ein Wert 1 aus, dass sämtliche zukünftigen Rewards berücksichtigt werden. Folgende Parameter werden geprüft:

Durchgang	Discount-Factor γ	Lernrate α	ϵ-Verfall
G1.1	**0.9**	0.5	0.999
G1.2	**0.8**	0.5	0.999
G1.3	**0.7**	0.5	0.999
G1.4	**0.6**	0.5	0.999
G1.5	**0.5**	0.5	0.999

Tabelle 5.2.: Experiment 1 – Discount-Factor

5.1.3. Lernrate

Die Lernrate α definiert, wie schnell die Q-Values auf neue Erkenntnisse angepasst werden. Ein Wert von 0 bedeutet, es wird nichts dazugelernt, und der Wert 1 heisst, dass der komplette TD-Error ausgeglichen wird. Folgende Lernraten werden überprüft:

Durchgang	Discount-Factor γ	Lernrate α	ϵ-Verfall
A1.1	0.9	**0.5**	0.999
A1.2	0.9	**0.6**	0.999
A1.3	0.9	**0.7**	0.999
A1.4	0.9	**0.8**	0.999
A1.5	0.9	**0.9**	0.999

Tabelle 5.3.: Experiment 1 – Lernrate

5.1.4. Messwerte

Der Messwert, mit welchem die Policies verglichen werden, ist der Reward. Dazu wird der mittlere Reward der letzten 50 Episoden verwendet; ausserdem wird dieser Wert noch durch 100 dividiert, um somit den mittleren Step-Reward pro Episode zu erhalten. Der mittlere Step-Reward ist deshalb interessant, da man diesen mit einem optimalen Wert vergleichen kann. Es werden alle 4 Steps Aufträge generiert, und somit wäre ohne Lagerkosten pro Step ein Reward von 25 das Maximum. Zuerst werden in einem direkten Vergleich die beiden Algorithmen gegenübergestellt. Dadurch soll ersichtlich sein, welcher Algorithmus einen höheren Reward erreicht und überprüft werden, ob sämtliche State-Action-Paare besucht wurden. In einem zweiten Schritt wird nun noch die erlernte Policy der Algorithmen direkt auf einen Agent übertragen, welcher jeweils ohne Exploration die beste Action auswählt. Somit kann ein fairer Vergleich mit denselben Bestellungen angestellt werden. Dabei ist zu erwarten, dass die Policies ein konstantes Ergebnis über die Episoden erreichen, da mit nur einem Artikel kein Zufall im Environment vorhanden ist. Die Werte in den Diagrammen werden, um die Übersichtlichkeit zu erhöhen, über eine bestimmte Anzahl an Werten, das sogenannte Window, gemittelt.

Abbildung 5.2.: Experiment 1 – Link

5.2. Resultate

Im ersten Experiment, welches in 5.1 definiert ist, wurde als Erstes die Exploration untersucht. Dabei konnte geprüft werden, welcher Epsilon-Verfall am besten geeignet ist. Der Wert 0.9999 aus Durchgang E1.4 erweist sich bei beiden Algorithmen als gut geeignet, da die State-Action-Paare am schnellsten erreicht werden und bereits ab 30'000 Episoden nur noch mit dem Minimum exploriert wird. Dadurch hat der Agent noch die restlichen Episoden Zeit, die Q-Values der State-Action-Paare zu optimieren. Bei 800'000 Episoden muss das entsprechende Epsilon angepasst werden, was gerundet ungefähr einem Epsilon-Verfall von 0.9999938 entspricht.

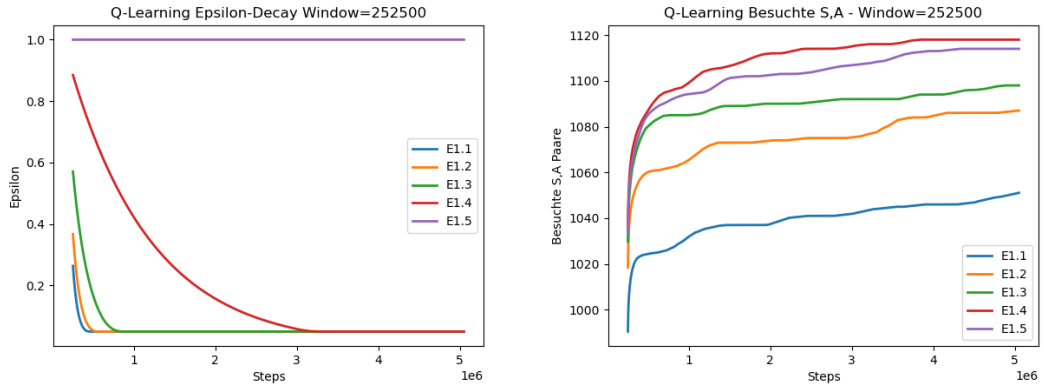

Abbildung 5.3.: Experiment 1 – Exploration Q-Learning

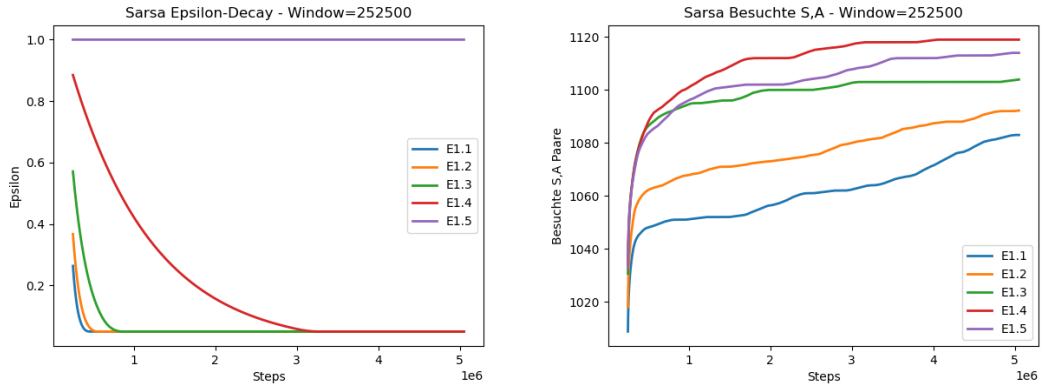

Abbildung 5.4.: Experiment 1 – Exploration Sarsa

In Tabelle 5.4 sind die Ergebnisse aus dem Parametervergleich für die Lernrate ersichtlich. Hierbei wurde jeweils der durchschnittliche Step-Reward der letzten 50 Episoden berechnet. Dabei ist erkennbar, dass die beiden Algorithmen jeweils mit unterschiedlichen Lernraten am besten performen. Für Q-Learning wird beim finalen Trainieren eine Lernrate von 0.9 und bei Sarsa 0.6 verwendet; diese haben in Tabelle 5.4 jeweils den höchsten Step-Reward erzielt.

Durchgang	Lernrate	Q-Learning	Sarsa
A1.1	0.5	16.18	14.71
A1.2	0.6	18.26	**17.22**
A1.3	0.7	18.14	15.16
A1.4	0.8	18.02	13.14
A1.5	0.9	**19.42**	10.84

Tabelle 5.4.: Experiment 1 – Resultate Lernrate

Zuletzt wurde der Discount-Factor überprüft. Dabei wurde auch der durchschnittliche Step-Reward aus den letzten 50 Episoden errechnet. Aus Tabelle 6.1 wird ersichtlich, dass Q-Learning mit einem Gamma von 0.8 und Sarsa mit einem Gamma von 0.6 am besten performen.

Durchgang	Discount-Factor	Q-Learning	Sarsa
G1.1	0.9	16.18	14.71
G1.2	0.8	**17.45**	17.26
G1.3	0.7	16.47	16.01
G1.4	0.6	18.02	**18.43**
G1.5	0.5	15.55	17.93

Tabelle 5.5.: Experiment 1 – Resultate Discount-Factor

Nach den drei Durchgängen, in denen jeweils die besten Parameter für die beiden Algorithmen gewählt wurden, wird nun während 800'000 Episoden versucht, eine optimale Policy zu erlernen. Dabei ist in Abbildung 5.5 zu sehen, dass Q-Learning einen höheren durchschnittlichen Reward erreicht als Sarsa. Der transparente Bereich zeigt die Varianz des Rewards. Der $TDError^2$, welcher aus Abbildung 5.5 hervorgeht, hat bei beiden Algorithmen nach etwa 500'000 Episoden konvergiert. Der Grund dafür ist das Epsilon, welches – wie Abbildung 5.6 zeigt – nach dieser Anzahl an Episoden auf dem Minimum von 0.05 angekommen ist. Ausserdem erkennt man, wie Q-Learning, im Gegensatz zu Sarsa, die Anzahl der erreichbaren State-Action-Kombinationen bereits kurz vor 600'000 Episoden (Abbildung 5.6) erreicht hat.

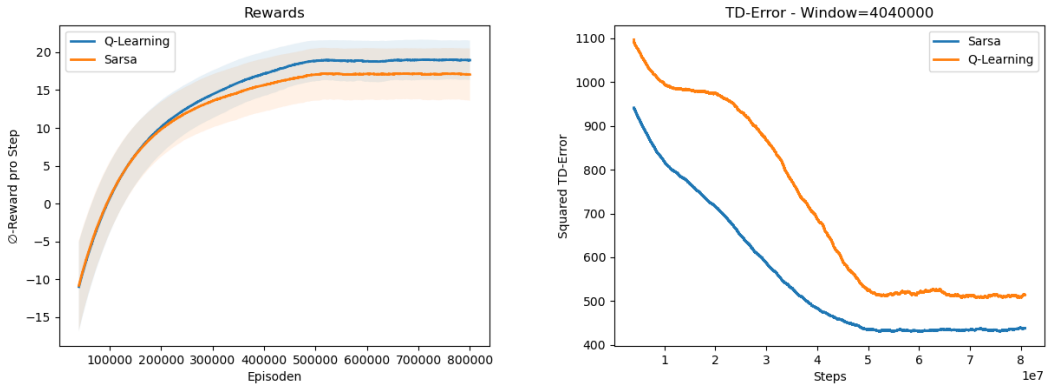

Abbildung 5.5.: Experiment 1 – Reward und TD-Error

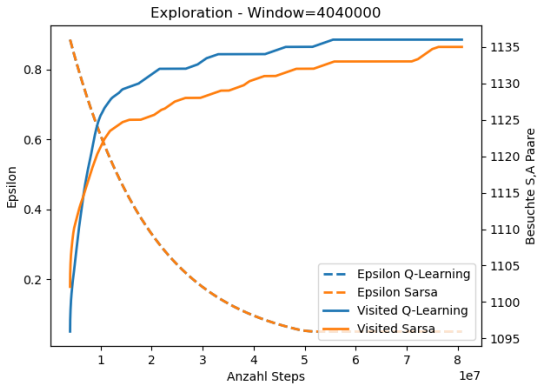

Abbildung 5.6.: Experiment 1 – Exploration

In der folgenden Abbildung 5.7 ist ersichtlich, wie oft ein gewisser Reward in der jeweiligen Episode verteilt wurde. Für negative Rewards wurde die Anzahl invertiert. Dabei ist zu erkennen, wie die Algorithmen die negativen Rewards, wie die für das Stornieren, für eine volle Ankunft oder eine falsche Auslieferung minimieren. Auch die Lagerkosten werden versucht auf das Nötigste zu minimieren. Nur die gelungenen Auslieferungen pro Episode haben mit steigender Anzahl an Episoden zugenommen.

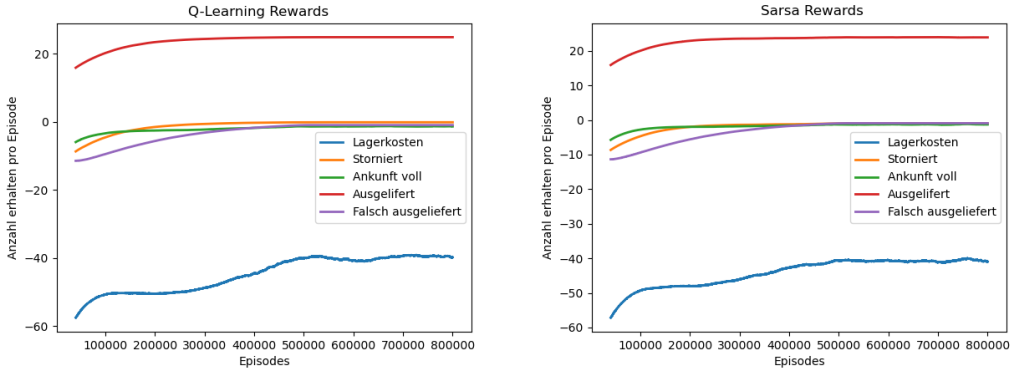

Abbildung 5.7.: Experiment 1 – Vorkommen der Rewards pro Step (Window = 40'400)

Mit dem erfolgreichen Trainieren der Policies steht nun eine Q-Matrix zur Verfügung, die an einen Greedy-Agent übergeben wird. Dieser Agent wählt ausschliesslich die Action, welche den höchsten Q-Value für einen gegebenen State aufweist. Wie erwartet, ist in Abbildung 5.8 zu erkennen, dass die Policies über die 1000 Episoden jeweils denselben Reward erzielen. Das ist korrekt, da in Experiment 1 nur ein Artikel und somit in jeder Episode dasselbe Sample an Aufträgen entsteht.

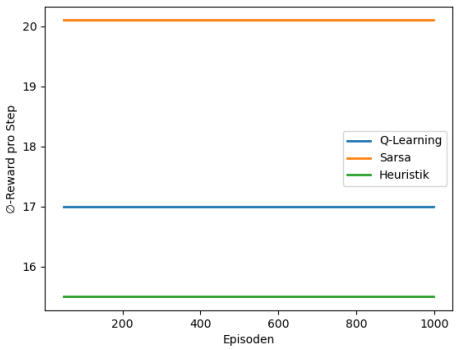

Abbildung 5.8.: Experiment 1 – Vergleich der Policies

In diesem Experiment hat Sarsa den mittleren Step-Reward der heuristischen Policy um 30 Prozent übertroffen (Tabelle 5.6).

Policy	Q-Learning	Sarsa	Heuristik
Ø-Step-Reward	17.0	20.1	15.5
Vergleich zur Heuristik	+9.7%	+29.7%	+0%

Tabelle 5.6.: Experiment 1 – Resultate Policies

In den folgenden Abbildungen 5.9, 5.10 und 5.11 wurde für jeden Agent dieselbe Episode ausgewählt und die erreichten Rewards sowie der State dieser Episode wurden dargestellt. Diese Darstellungen ermöglichen es, die Unterschiede der Agents besser aufzuzeigen. Abbildung 5.9 zeigt die Rewards der Heuristik für eine Episode. Wie man erkennen kann, verpasst diese Policy jede 5. Bestellung. Dieses Resultat ist nachvollziehbar, denn die definierte Heuristik hält nur ein Exemplar pro Artikel auf Lager und das Bestellen bis zum Ausliefern dauert 5 Steps. Es werden jedoch alle 4 Steps Aufträge generiert.

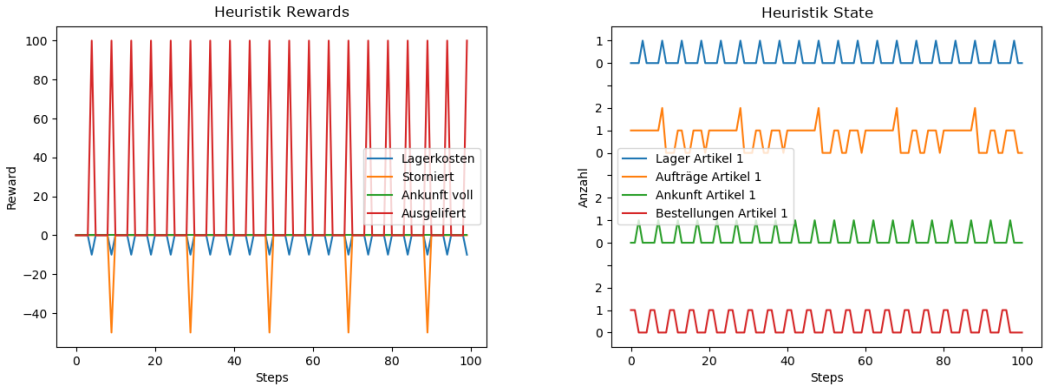

Abbildung 5.9.: Experiment 1 – Rewards und States aus einer Episode Heuristik

Wie in Abbildung 5.10 erkennbar, hat die Q-Learning-Policy versucht, die Lagerplätze durch den Wareneingang zu ersetzen, da dieser im Environment keine Kosten generiert. Da aber zu viele Artikel im Eingang sind, wird trotzdem ein negativer Reward verteilt.

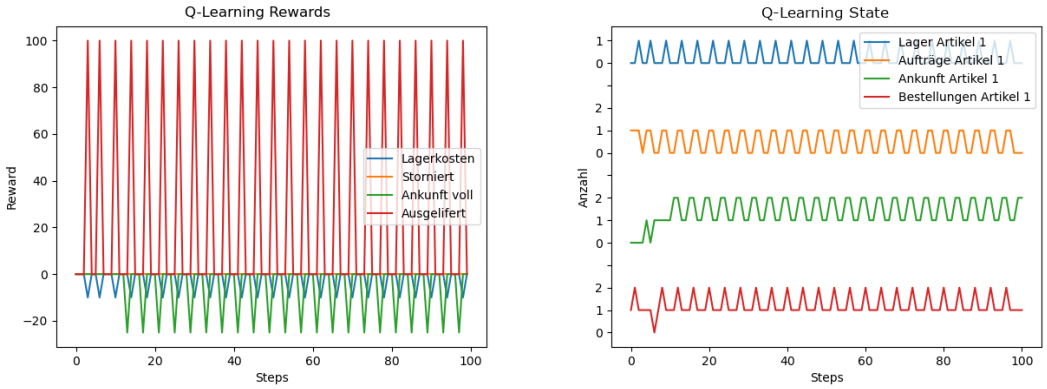

Abbildung 5.10.: Experiment 1 – Rewards und States aus einer Episode Q-Learning

Sarsa hingegen hat gelernt, dass nur zu Beginn zwei Bestellungen nacheinander vorkommen und bestellt diese demensprechend (Abbildung 5.11). Dieses Vorgehen entspricht einem Overfitting, was in diesem Fall das Auswendiglernen einer bestimmten Sequenz bedeutet. Da das Environment aber immer gleich bleibt und keine zufälligen Abläufe stattfinden, ist das in diesem Fall die beste Policy.

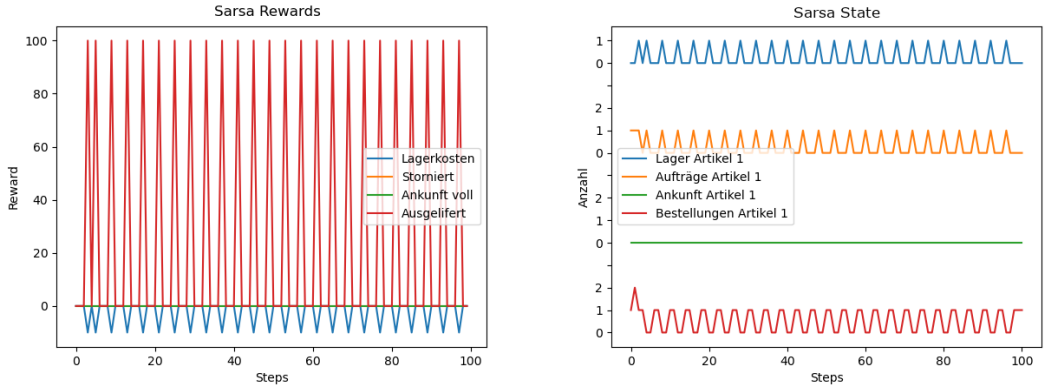

Abbildung 5.11.: Experiment 1 – Rewards und States aus einer Episode Sarsa

6. Experiment 2

Bei der zweiten Forschungsfrage soll das Verhalten der Performanz bei einer steigenden Komplexität im Environment überprüft werden.

6.1. Beschreibung

Um das Verhalten der Performanz bei einer höheren Komplexität zu überprüfen, wird das Environment um einen Artikel erweitert. Diese Erweiterung hört sich nach einer nicht besonders grossen Komplexitätssteigerung an. Um den Unterschied aufzuzeigen, können von beiden Environments die maximal möglichen State-Action-Kombinationen berechnet werden. Eine Observation aus dem State-Space besteht aus den neun in Unterabschnitt 3.2.2 beschriebenen Features, wobei jedes dieser Features 0 für leer, 1 für Artikel 1 und 2 für Artikel 2 annehmen kann.

$$\begin{aligned} States &= (Artikel + 1)^{Features} \\ Actions &= Artikel + 3 \end{aligned} \tag{6.1}$$

Environment	States-Kombinationen	Actions	SA-Kombinationen
1 Artikel	2^9 = 512	5	2'560
2 Artikel	3^9 = 19'683	6	118'098

Tabelle 6.1.: Experiment 1 – Resultate Policies

Diese Kombinationen sind theoretisch möglich; davon sind aber nicht alle erreichbar, da es gewisse Regeln im Environment gibt. Der Agent kennt diese Regeln allerdings nicht. Des Weiteren spielt die Reihenfolge, ob ein Artikel im Lager auf Platz 0, 1 oder 2 vorhanden ist, keine Rolle. Auch diese Information ist dem Agent nicht bewusst und

er müsste alle 3 Varianten kennenlernen. Da in diesem Experiment nun zwei mögliche Artikel verfügbar sind, hat die Häufigkeit der Artikel nun einen Einfluss. Für dieses Experiment wird ein *Artikel 1* mit der Häufigkeit von 0.2 und einer mit der Häufigkeit von 0.8 definiert. Dadurch werden im Durchschnitt vier Mal so viele Aufträge mit dem *Artikel 2* generiert.

6.1.1. Parameter

Die relevanten Parameter für dieses Experiment entsprechen denen aus Experiment 1. Die Anzahl an Episoden ist in diesem Experiment zu Beginn auch auf einen Wert definiert. Es werden pro Durchgang jeweils 50'000 Episoden durchgeführt.
Die Durchgänge werden mit den folgenden Parametern ausgeführt:

Durchgang	Discount-Factor γ	Lernrate α	ϵ-Verfall
E2.1	0.9	0.5	**0.9985**
E2.2	0.9	0.5	**0.9990**
E2.3	0.9	0.5	**0.9995**
E2.4	0.9	0.5	**0.9999**
E2.5	0.9	0.5	**1**
G2.1	**0.9**	0.5	0.999
G2.2	**0.8**	0.5	0.999
G2.3	**0.7**	0.5	0.999
G2.4	**0.6**	0.5	0.999
G2.5	**0.5**	0.5	0.999
A2.1	0.9	**0.5**	0.999
A2.2	0.9	**0.6**	0.999
A2.3	0.9	**0.7**	0.999
A2.4	0.9	**0.8**	0.999
A2.5	0.9	**0.9**	0.999

Tabelle 6.2.: Experiment 2 – Parameter

6.1.2. Messwerte

Auch bei diesem Experiment werden die Resultate dem durchschnittlichen Step-Reward pro Episode gegenübergestellt. Die Algorithmen werden mithilfe desselben Vorgehens

aus Experiment 1 untersucht und miteinander verglichen. Auch hier wird anschliessend wieder mit einem Agent verglichen, der ausschliesslich die Action mit dem grössten Q-Value, der Greedy-Policy, wählt. Dabei sollte eine Varianz in den Resultaten der Policies zu erkennen sein, da nun Aufträge mit zufälligen Artikeln erstellt werden.

Abbildung 6.1.: Experiment 2 – Link

6.2. Resultate

Auch im zweiten Experiment, welches in Abschnitt 6.2 definiert ist, wurde als Erstes die Exploration untersucht. Das Verhalten von Epsilon hat sich zum vorherigen Experiment nicht verändert, da das Epsilon nicht mit der Anzahl an State-Action-Paaren, sondern mit der Anzahl an Episoden verringert wird. Aufgrund der gestiegenen Anzahl möglicher State-Action-Kombinationen werden in derselben Anzahl an Episoden deutlich mehr State-Action-Paare besucht.

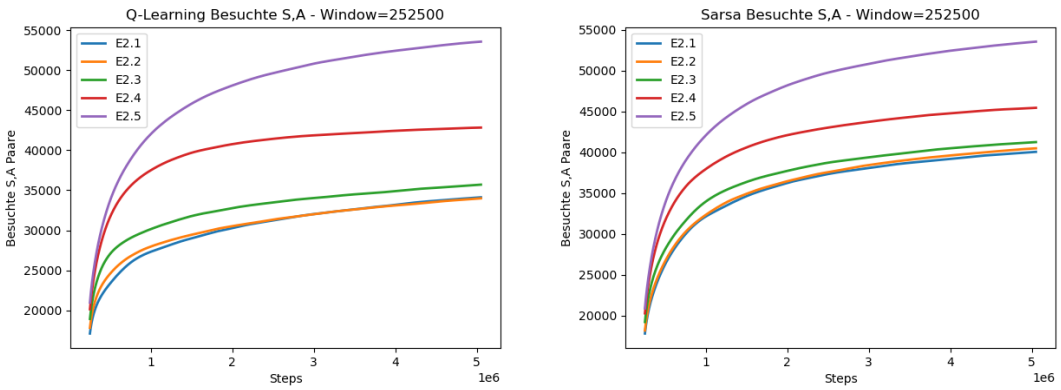

Abbildung 6.2.: Experiment 2 – Besuchte SA-Paare

Als Zweites wurden die Lernraten untersucht. Dabei haben die Agents die in Tabelle 6.3 ersichtlichen Resultate erzielt. Zu sehen ist, dass mit sinkender Lernrate der Reward

weiter gestiegen ist. Aus diesem Grund wurde noch ein zweiter Durchgang gestartet, welcher kleinere Lernraten untersucht.

Durchgang	Lernrate	Q-Learning	Sarsa
A2.1	0.5	**3.72**	**3.05**
A2.2	0.6	-1.16	-2.00
A2.3	0.7	-1.52	-3.27
A2.4	0.8	-4.77	-8.61
A2.5	0.9	-9.35	-14.40

Tabelle 6.3.: Experiment 2 – Resultate Lernrate 1

Beim zweiten Durchgang wurden die folgenden Resultate (Tabelle 6.4) erzielt. Die Annahme wurde bestätigt, wodurch in diesem Durchgang für beide Algorithmen eine bessere Lernrate mit dem Wert 0.1 gefunden werden konnte.

Durchgang	Lernrate	Q-Learning	Sarsa
A2.6	0.4	3.44	0.69
A2.7	0.3	2.61	3.62
A2.8	0.2	6.50	5.46
A2.9	0.1	**6.62**	**6.33**
A2.10	0.01	3.60	2.67

Tabelle 6.4.: Experiment 2 – Resultate Lernrate 2

Als letzter Parameter wurde hier der Discount-Factor untersucht. Wie in Tabelle 6.5 ersichtlich, hat sich ein Gamma von 0.9 für beide Algorithmen als der beste getestete Wert erwiesen.

Durchgang	Discount-Factor	Q-Learning	Sarsa
G2.1	0.9	**3.72**	**3.05**
G2.2	0.8	-0.28	-5.76
G2.3	0.7	-4.84	-13.13
G2.4	0.6	-9.98	-15.34
G2.5	0.5	-10.45	-13.14

Tabelle 6.5.: Experiment 2 – Resultate Discount-Factor

Nach den vier Durchgängen, in welchen die besten Parameter gesucht wurden, werden nun mit beiden Algorithmen erneut die Policies erlernt. In Abbildung 6.3 ist zu erkennen, dass dieses Mal Q-Learning zu einem kleineren TD-Error konvergiert. Die Rewards beim Lernen befinden sich bei beiden Algorithmen in einem ähnlichen Bereich.

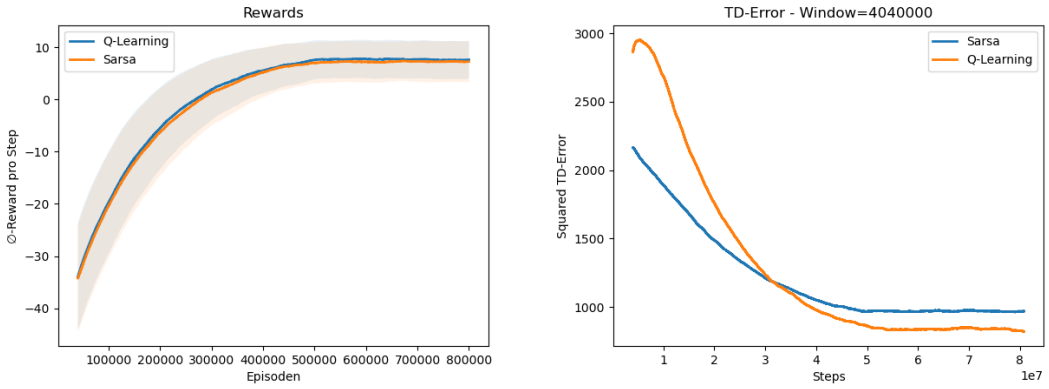

Abbildung 6.3.: Experiment 2 – Reward und TD-Error

Die besuchten State-Action-Paare sind im Verhältnis zur maximalen Anzahl möglicher Kombinationen dem ersten Experiment äußerst ähnlich.

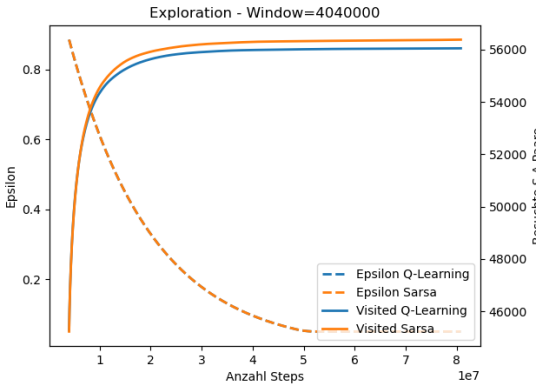

Abbildung 6.4.: Experiment 2 – Exploration

Bei den vorgekommenen Rewards pro Episode (Abbildung 6.5) haben beide Agents erneut gelernt, negative Rewards wie eine stornierte Bestellung und eine volle Ankunft zu vermeiden. Im Vergleich zum ersten Experiment sind aber die Lagerkosten deutlich gestiegen. Dies ist ein gutes Zeichen, denn nun kann der Agent nicht mehr genau einen Step vor der Bestellung den gewünschten Artikel bestellen, da dieser probabilistisch nach der Häufigkeit gewählt wird.

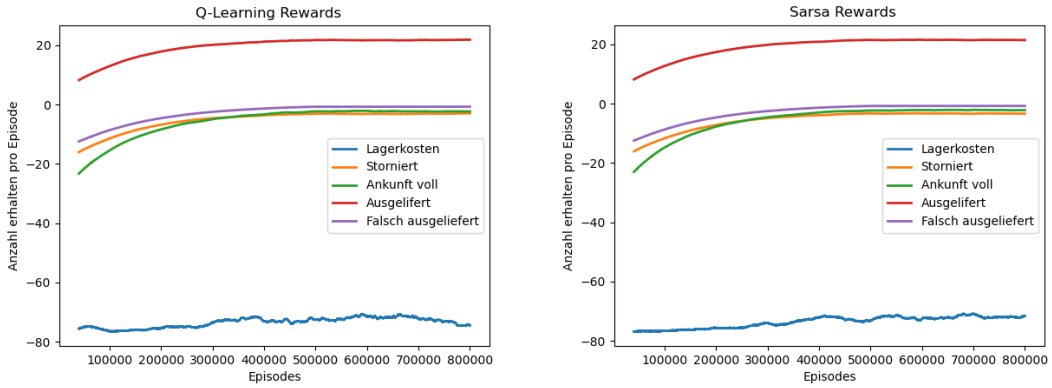

Abbildung 6.5.: Experiment 2 – Vorkommen der Rewards pro Step (Window = 40'400)

Die erlernten Policies konnten im direkten Vergleich mit der Heuristik erneut bessere Resultate erzielen (Abbildung 6.6). Des Weiteren ist nun im Gegensatz zum ersten Experiment eine Varianz ersichtlich, da nun Artikel zufällig sind.

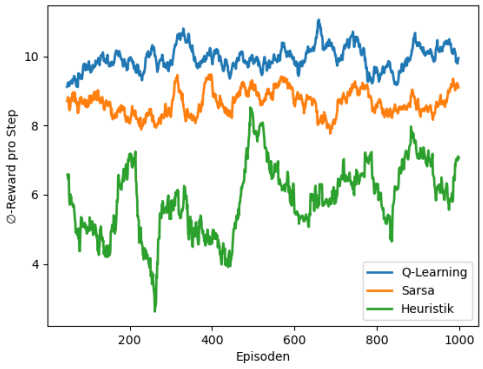

Abbildung 6.6.: Experiment 2 – Vergleich der Policies

Erneut haben die Policies, die durch Reinforcement Learning erlernt wurden, die der Heuristik übertroffen. Q-Learning hat dabei den durchschnittlichen Step-Reward um 67 Prozent übertroffen (Tabelle 6.6).

Policy	Q-Learning	Sarsa	Heuristik
Ø-Step-Reward	9.92	8.66	5.93
Vergleich zur Heuristik	+67.3%	+46.0%	+0%

Tabelle 6.6.: Experiment 2 – Resultate Policies

Die folgenden Abbildungen stellen erneut die Rewards und die States aus einer Episode dar. In der Abbildung 6.7 ist ersichtlich, wie die Policy der Heuristik fast durchgehend einen Artikel 1 auf Lager hat. Das ist nachvollziehbar, da die Heuristik versucht, jeweils mindestens einen Artikel im Lager zu halten, damit keine Lieferfrist für einen Auftrag abläuft. Daher sind die Lagerkosten im Vergleich zu den beiden durch Reinforcement Learning erlernten Strategien höher.

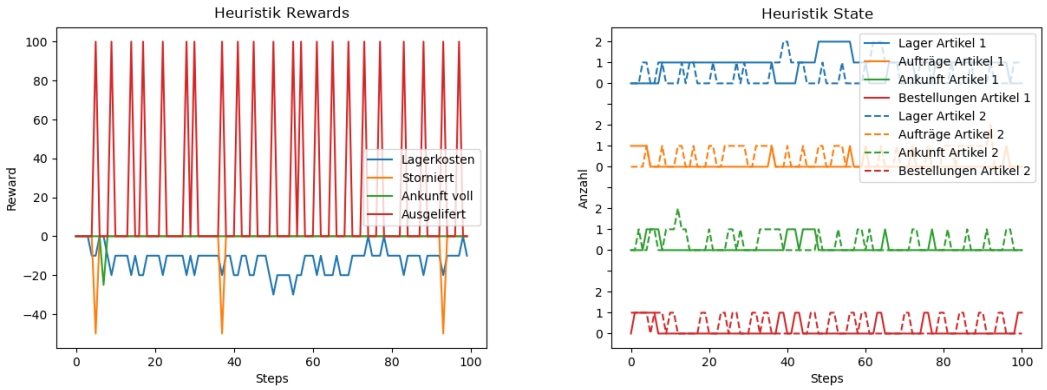

Abbildung 6.7.: Experiment 2 – Rewards und States aus einer Episode Heuristik

Bei Q-Learning ist in Abbildung 6.8 erneut zu erkennen, dass die Ankunft benutzt wird, um Lagerkosten einzusparen. Allerdings gelingt es der Policy nicht, ein Überfüllen der Ankunft zu verhindern und daher bekommt sie drei Mal einen negativen Reward dafür. Dafür sind die Lagerkosten um einiges geringer, da nie drei Artikel im Lager sind.

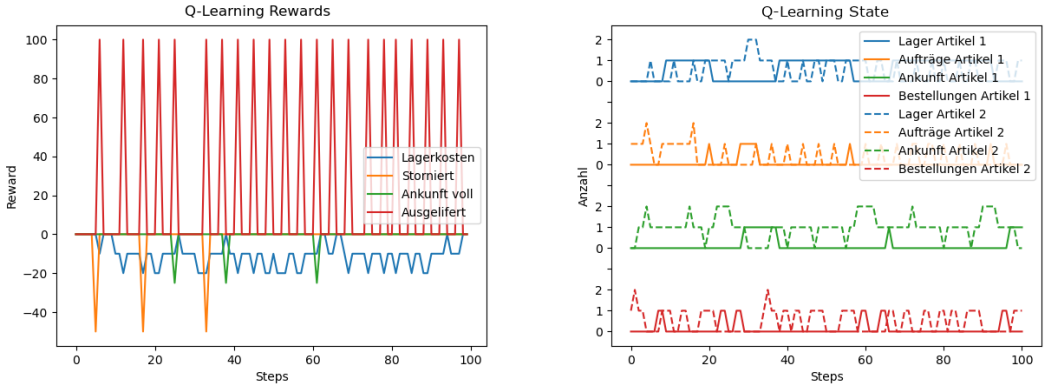

Abbildung 6.8.: Experiment 2 – Rewards und States aus einer Episode Q-Learning

In Abbildung 6.9 sieht man, wie es Sarsa gelingt, die Ankunft sinnvoll zu verwenden, ohne dass die Ankunft ein Mal überfüllt wird.

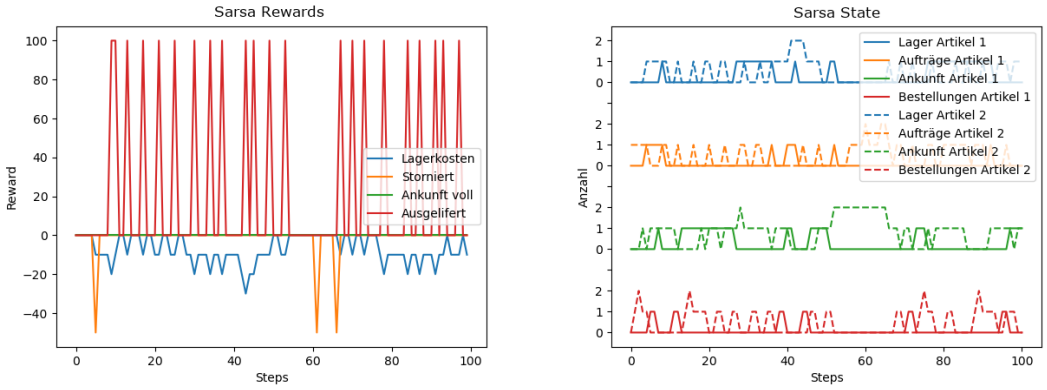

Abbildung 6.9.: Experiment 2 – Rewards und States aus einer Episode Sarsa

7. Diskussion

Die Resultate aus Experiment 1 zeigen, dass es für Reinforcement Learning in dieser minimalen Umgebung durchaus möglich ist, eine gute Policy zu erlernen und die Performanz der heuristisch definierten Policy zu übertreffen. Die durch Sarsa erlernte Policy hat dabei die Resultate der heuristischen Strategie um 30 Prozent übertroffen. Ausserdem hat der Q-Learning-Agent in diesem Experiment eine Policy erlernt, welche beim Erstellen des Environments nicht so eingeplant wurde. Der Agent hat festgestellt, dass Artikel in der Ankunft keine Lagerkosten generieren und dies dementsprechend mit mittlerem Erfolg genutzt. Dabei lässt sich die erste Forschungsfrage mit der Begrenzung auf das untersuchte Environment folgendermassen beantworten: Die Performanz, der durch Reinforcement Learning erlernten Strategie, übertrifft die der heuristischen Strategie. Im zweiten Experiment wurde die Komplexität erhöht. Dabei konnte erneut die Performanz der Heuristik übertroffen werden. Die durch Q-Learning erlernte Policy erreichte einen durchschnittlichen Step-Reward der 67 Prozent höher war als der der Heuristik. Des Weiteren ist aber deutlich zu erkennen, dass durch das Wachstum des State-Action-Spaces eine höhere Anzahl an Episoden nötig war, um alle State-Action-Paare zu besuchen. Die zweite Forschungsfrage kann aufgrund dieser Ergebnisse wie folgt beantwortet werden: Die Performanz, der durch Reinforcement Learning erlernten Strategie, konnte die der heuristischen Strategie übertreffen. Dabei kann jedoch nicht auf ein reelles Lager generalisiert werden – dazu wurden zu wenige Artikel und Lagerplätze initialisiert. Eine Erweiterung um einen Artikel hat eine hohe Auswirkung auf den State. Die Information, welcher Artikel auf Lager ist, wird benötigt, um festzustellen, welche Artikel öfter bestellt werden. Eine Möglichkeit, die Komplexität drastisch zu verringern, wäre diese: Die States vom Environment werden auf 0 und 1 reduziert, wobei 1 bedeutet, es ist ein Artikel im Lager, in der Ankunft oder wurde soeben bestellt. Die Information über die Häufigkeit könnte mit einem Array, welches aus den letzten n bestellten Artikel besteht, gespeichert werden. Die in dieser Arbeit erzielten Ergebnisse dienen zur Grundlage und bieten als Artefakt ein erweiterbares Environment, welches mit geringem Aufwand weiter skaliert werden kann. Die nächste sinnvolle Untersuchung wäre eine Verkleinerung des State-Spaces. Ausserdem sollten weiterführende Algorithmen, welche sich in komplexeren Environments

beweisen können – wie in den Atari-Spielen [10] – verwendet werden. So wird häufig mit Deep Q-Learning ein Neuronales Netz erstellt anstatt einer Q-Matrix, welches besser auf unbesuchte State-Action-Paare generalisieren kann. Ein Einsatz in einem reellen Lager könnte erst nach weiteren Untersuchungen abschliessend abgeschätzt werden.

Abbildungsverzeichnis

2.1. RL-Agent und Environment 3
2.2. Markov-Prozess 4
2.3. Markov-Reward-Process 5
2.4. MRP mit Values 7
2.5. Markov-Decision-Process 9
2.6. MDP-State-Value-Function 10
2.7. MDP-Action-Value-Function 11
2.8. Unterschied MC & TD [3] 12
2.9. Sarsa-Update 14
2.10. Q-Learning-Update 15

3.1. WarehouseEnv 17

5.1. SA-Paare – Zufällige Actions 800'000 Episoden 25
5.2. Experiment 1 – Link 28
5.3. Experiment 1 – Exploration Q-Learning 29
5.4. Experiment 1 – Exploration Sarsa 29
5.5. Experiment 1 – Reward und TD-Error 31
5.6. Experiment 1 – Exploration 31
5.7. Experiment 1 – Vorkommen der Rewards pro Step (Window = 40'400) . . . 32
5.8. Experiment 1 – Vergleich der Policies 32
5.9. Experiment 1 – Rewards und States aus einer Episode Heuristik 33
5.10. Experiment 1 – Rewards und States aus einer Episode Q-Learning 34
5.11. Experiment 1 – Rewards und States aus einer Episode Sarsa 34

6.1. Experiment 2 – Link 37
6.2. Experiment 2 – Besuchte SA-Paare 37
6.3. Experiment 2 – Reward und TD-Error 39
6.4. Experiment 2 – Exploration 39
6.5. Experiment 2 – Vorkommen der Rewards pro Step (Window = 40'400) . . . 40

6.6. Experiment 2 – Vergleich der Policies 40
6.7. Experiment 2 – Rewards und States aus einer Episode Heuristik 41
6.8. Experiment 2 – Rewards und States aus einer Episode Q-Learning 42
6.9. Experiment 2 – Rewards und States aus einer Episode Sarsa 42

Tabellenverzeichnis

3.1. Lagerkosten 18
3.2. Stornierter Auftrag 18
3.3. Bestellungseingang 18
3.4. Lagerplätze 19
3.5. Artikel 19
3.6. Aufträge 19
3.7. Wareneingang 20
3.8. Bestellungen 20
3.9. Observation 20
3.10. Lagern 21
3.11. Ausliefern 21
3.12. Bestellen 21
3.13. Warten 21

5.1. Experiment 1 – Epsilon-Verfall 26
5.2. Experiment 1 – Discount-Factor 27
5.3. Experiment 1 – Lernrate 27
5.4. Experiment 1 – Resultate Lernrate 30
5.5. Experiment 1 – Resultate Discount-Factor 30
5.6. Experiment 1 – Resultate Policies 33

6.1. Experiment 1 – Resultate Policies 35
6.2. Experiment 2 – Parameter 36
6.3. Experiment 2 – Resultate Lernrate 1 38
6.4. Experiment 2 – Resultate Lernrate 2 38
6.5. Experiment 2 – Resultate Discount-Factor 38
6.6. Experiment 2 – Resultate Policies 41

Literaturverzeichnis

[1] J.-S. Song, G.-J. Houtum, and J. Van Mieghem, “Capacity and inventory management: Review, trends, and projections,” *Manufacturing Service Operations Management*, 02 2019.

[2] S. Chaharsooghi, J. Heydari, and S. Zegordi, “A reinforcement learning model for supply chain ordering management: An application to the beer game,” *Decision Support Systems*, vol. 45, pp. 949–959, 11 2008.

[3] R. Kavalov, *Temporal-Difference Learning*, (Zugriff am 06 2020). https://towardsdatascience.com/my-journey-into-reinforcement-learning-part-5-temporal-difference-learning-d0cae79e850.

[4] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cam-bridge, UK, 1994.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. Adaptive computation and machine learning series, Cambridge, Massachusetts: The MIT Press, second edition ed., 2018.

[6] P. Dayan and C. Watkins, “Q-learning,” *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[7] C. J. C. H. Watkins, “Learning from delayed rewards,” 1989.

[8] OpenAI, *A Toolkit for Developing and Comparing Reinforcement Learning Algorithms*, (Zugriff am 03.07.2020). http://gym.openai.com/docs/.

[9] A. Hevner, A. R, S. March, S. T, Park, J. Park, Ram, and Sudha, “Design science in information systems research,” *Management Information Systems Quarterly*, vol. 28, pp. 75–, 03 2004.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

A. Anhang

A.1. Berechnung MDP - State-Values

$\gamma = 1$

KB-NV

$$v_*(KBNV) = \gamma * (P(KBNV) * v_*(KBNV) + P(BT1NV) * v_*(BT1NV)) + R(A1)$$
$$v_*(KBNV) = 1/2x + 9.5 - 1 = x$$
$$v_*(KBNV) = 17 \tag{A.1}$$

KB-V

$$v_*(KBV) = \gamma * (P(KBV) * v_*(KBV) + P(BT1V) * v_*(BT1V)) + R(A1)$$
$$v_*(KBV) = 1/2x + 10 - 2 = x$$
$$v_*(KBV) = 16 \tag{A.2}$$

BT1-V

$$v_*(BT1V) = \gamma * (P(A) * v_*(A)) + R(A3)$$
$$v_*(BT1V) = 0 + 20 = 20 \tag{A.3}$$

BT1-NV

$$v_*(BT1NV) = \gamma * (P(BT2V) * v_*(BT2V)) + R(A1)$$
$$v_*(BT1NV) = 20 - 1 = 19 \tag{A.4}$$

BT2-V

$$v_*(BT2V) = \gamma * (P(A) * v_*(A)) + R(A3)$$
$$v_*(BT2V) = 0 + 20 = 20 \tag{A.5}$$

A.2. Berechnung MDP - Action-Values

$\gamma = 1$

KB-NV

$$q_*(KBNV, A1) = \gamma * (P(KBNV) * v_*(KBNV) + P(BT1NV) * v_*(BT1NV)) + R(A1)$$

$$q_*(KBNV, A1) = 8.5 + 9.5 - 1 = 17 \tag{A.6}$$

$$q_*(KBNV, A2) = \gamma * (P(KBV) * v_*(KBV) + P(BT1V) * v_*(BT1V)) + R(KBNV, A2)$$

$$q_*(KBNV, A2) = 8 + 10 - 1 = 17 \tag{A.7}$$

KB-V

$$q_*(KBV, A1) = \gamma * (P(KBV) * v_*(KBV) + P(BT1V) * v_*(BT1V)) + R(A1)$$

$$q_*(KBV, A1) = 8 + 10 - 2 = 16 \tag{A.8}$$

BT1-V

$$q_*(BT1V, A3) = \gamma * (P(A) * v_*(A)) + R(A3)$$

$$q_*(BT1V, A3) = 0 + 20 = 20 \tag{A.9}$$

BT1-NV

$$q_*(BT1NV, A1) = \gamma * (P(KBNV) * v_*(KBNV)) + R(A1)$$

$$q_*(BT1NV, A1) = 17 - 1 = 16 \tag{A.10}$$

$$q_*(BT1NV, A2) = \gamma * (P(BT2V) * v_*(BT2V)) + R(A2)$$

$$q_*(BT1NV, A2) = 20 - 1 = 19 \tag{A.11}$$

BT2-V

$$q_*(BT2V, A3) = \gamma * (P(A) * v_*(A)) + R(A3)$$

$$q_*(BT2V, A3) = 0 + 20 = 20 \tag{A.12}$$

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich diese Thesis selbständig verfasst und keine andern als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche kenntlich gemacht. Ich versichere zudem, dass ich bisher noch keine wissenschaftliche Arbeit mit gleichem oder ähnlichem Inhalt an der Fernfachhochschule Schweiz oder an einer anderen Hochschule eingereicht habe. Mir ist bekannt, dass andernfalls die Fernfachhochschule Schweiz zum Entzug des aufgrund dieser Thesis verliehenen Titels berechtigt ist.

Ort, Datum, Unterschrift