SAKI SS19 Homework 3

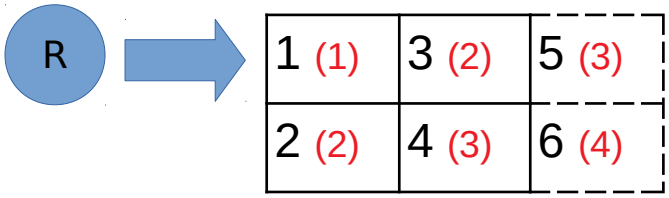
Author: Jannis Wolf

Program code: https://github.com/JannisWolf/smart_warehouse

Summary

Die anhaltende Automatisierung in der Industrie 4.0 birgt nicht nur Möglichkeiten für etablierte Algorithmen, sondern lässt auch mehr und mehr künstliche Intelligenz (KI) in Produktionsabläufe eingreifen.

Im Folgenden wird untersucht, wie effizient ein KI gesteuerter Roboter ein Warenlager verwalten kann, in dem drei verschiedene Produkte (rot, blau und weiß) auf 4 (2x2) bzw. 6 (2x3) Plätzen ein und ausgelagert werden können, siehe die nachstehende Skizze des Lagers.



Der Roboter kann das Lager nur an Position 1 betreten. Die nötigen Schritte um zu einem Feld zu gelangen, kann durch die Manhatten Distanz (rot markiert) angegeben werden, da der Roboter nur direkt angrenzende Felder betreten kann. Das Problem kann als Markov Decision Process (MDP) formuliert werden, ein Modell, bei dem der Nutzen (Reward) eines Agenten von einer Folge von Entscheidungen abhängig ist.

Für die Lösung eines MDP wird ein 4er Tupel (S, A, P_{ij}, R) benötigt. Dieser besteht aus den möglichen Zuständen (States) S des Agenten, seinen möglichen Aktionen A, der stochastischen Matrix P_{ij} der Übergänge von einem State i in den State j (auch Transition Probability Matrix (TPM) genannt) und schließlich der Belohnungs- bzw. Reward Matrix R.

States S: Die Zustände des Systems sind alle möglichen Konfigurationen des Lagers, sowie eine von 6 Möglichkeiten für den nächsten Befehl (ein-/auslagern rot/blau/weiß). Damit ergibt sich für das 2x2 Lager 4^4 * 6 = 1536 Zustände und für das 2x3 Lager 4^6 * 6 = 24576 Zustände.

Actions A: Eine Aktion beschreibt den Platz den der Agent im Lager auswählt.

Transition Probability Matrix P_{ij}: Für jede Aktion gibt es eine Zustandsübergangsmatrix P_{ij}. Sie beschreibt die Wahrscheinlichkeit von einem Zustand i in den Zustand j überzugehen. Nach einer Aktion gibt es genau 6 Zustände in die der Agent wechseln kann, alle anderen Werte sind 0. Es muss beachtet werden, dass der Agent im gleichen State bleibt, falls ein auslagern nicht möglich war oder ein Platz zum lagern schon besetzt ist. Die Wahrscheinlichkeiten wurden aus dem zur Verfügung gestellten Trainingsset berechnet.

Reward Matrix R: Für die Reward Matrix zu einer Aktion wurde die negative Manhatten Distanz benutzt. Ist ein einlagern nicht möglich wurde ein Wert von -10 gesetzt, wodurch der Agent manchmal Produkte ablehnt, da er langfristig mehr Nutzen sieht das Lager divers zu halten. Soll ein im Lager nicht existentes Produkt herausgenommen werden, wurde das mit -100 bestraft.

Für die Berechnung der Strategie (Policy) des MDP wurde die MDPToolbox in Python verwendet. Ein sehr wichtiger Parameter war hier der Discount Faktor. Desto näher dieser Wert gegen 1 läuft, desto weitsichtiger plant der Agent seine Entscheidungen. Ein Wert von 0.999 lieferte trotz der langen Laufzeit die besten Ergebnisse.

Evaluation

Policy vs. Value Iteration: Policy Iteration konnte nur für ein Lager der Größe 2x2 berechnet werden, da der Rechen- und Speicheraufwand der Policy Iteration zu hoch war (alle Berechnungen in Google Colab). Felder mit der gleichen Manhatten Distanz wurden sporadisch verschieden belegt. Hierdurch variiert der Reward für den Agenten jedoch nicht. Alle folgenden Ergebnisse wurden mithilfe der Value Iteration berechnet, da diese eine deutlich schnellere Laufzeit zeigte.

Performance Analyse: Um die Performance des MDP gesteuerten Robotors in Relation zu setzen, wurde ein zweiter Roboter implementiert dessen Storing und Restoring Verfahren als greedy beschrieben werden kann. Er lagert ein Produkt immer an der ersten freien Stelle und entnimmt entsprechend auf Befehl das erste Produkt in der richtigen Farbe. Die Manhatten Distanz, welche die Roboter zurücklegen müssen, um ein Produkt im Lager zu platzieren oder herauszunehmen wird in Schritten als Performance Metrik herangezogen.

	Anzahl Produkte	Greedy Robot (in Schritten)	Smart Robot (in Schritten)	Abgelehnte Produkte / Nicht im Lager	Relative Ver-bessserung Smart/Greedy
Warehousetraining2x2	8177	14401	14127	318	- 1.9 %
Warehouseorder2x2	65	114	121	1	+ 6.1 %
Warehousetraining2x3	12108	25686	24564	490	- 4.6 %
Warehouseorder2x3	60	124	114	4	- 8.8 %

Sowohl der Roboter des 2x2 Lagers, als auch der des 2x3 Lagers können auf einem großen Datenset (Warehousetraining) den greedy Algorithmus überholen. Jedoch lehnen die KI gesteuerte Roboter gelegentlich Produkte ab und konnten diese später nicht mehr zurückgeben. Für diesen Fall wurden keine Bestrafungen verteilt, wodurch das bessere Ergebnis wieder relativiert wird. Weiterhin muss berücksichtigt werden, dass hier die Trainingsdaten zur Evaluation hinzugezogen wurden.

Verhalten des Agenten und Einfluss des Trainingssets: Bei einem Blick auf 3 Verhaltensmuster der KI, kann man weitere Schlüsse zur Leistungsfähigkeit ziehen (im Screenshot auf der nächsten Seite visualisiert).

- Bei einer Lagerung zweier gleichen Produkte in ein leeres Lager, wir das zweite Produkt weiter hinten gelagert, um vorne Platz zu lassen für alternative Produkte.
- Bei der Lagerung des ersten Produkts in ein leeres Lager wird das blaue Produkt, welches im Trainingsset die geringste Auftrittswahrscheinlichkeit hat, von der KI im leeren Lagerhaus direkt an eine hintere Stelle gelegt.
- Die KI lehnt Produkte ab, wenn das Lager nicht divers genug ist. An dieser Stelle kann auch sehr gut der Einfluss des Trainingssets beobachtet werden. Während rote Produkte mit höchster Auftrittswahrscheinlichkeit, erst bei mindestens 3 vorhandenen roten Produkten abgelehnt werden, werden blaue/ weiße Produkte gelegentlich schon bei einem vorhandenen blauen/ weißen Produkt abgelehnt, trotz noch freier Plätze.

Bewertung der Ergebnisse: Das Verhalten des KI gesteuerten Roboters hebt sich merkbar von einem greedy arbeitenden Roboter ab. Wie eben dargelegt, zeigt die aus dem MDP berechnete Policy intuitive und weitsichtige Entscheidungen, die definitv das Potential haben herkömmliche Produktionsabläufe zu verbessern.

Ein negativer Aspekt sind die enormen Rechen- und Speicherkapazitäten die schon bei einem 2x3 Lager nötig waren. Für eine Berechnung für ein riesiges Amazon Warenlager mit mehrern tausend Stellplätzen müssten andere Lösungsalgorithmen verwendet werden.

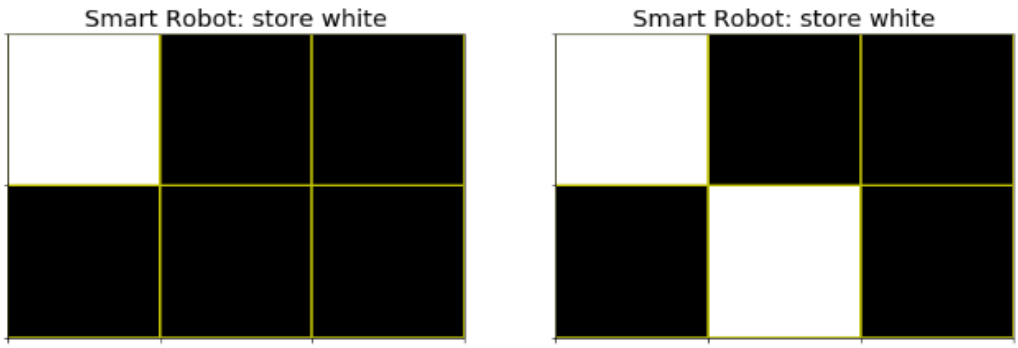
Screenshot

Test Performance

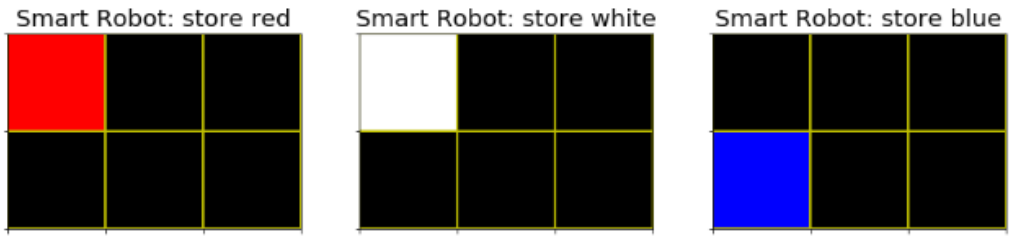
```
In [19]: test = UnitTest(6, value_policy, states, 'Exercise 3 - Reinforcement Learning - warehousetrain
         test.plot(24216, plot=False)
         test.print_reward()
         print('Performance difference in steps: ' + str(test.greedy.reward - test.smart.reward))
         print('Relative performance difference: ' + str(round((test.greedy.reward/ float(test.smart.re
```

```
Could not restore red in [0, 'b', 0, 0, 0, 0]
refused to store blue in ['w', 'b', 'r', 'b', 0, 'b']
Could not restore blue in [0, 0, 0, 0, 0, 0]
refused to store blue in ['w', 'b', 'r', 'w', 'b', 0]
Could not restore blue in [0, 0, 0, 0, 0, 0]
refused to store red in ['r', 'b', 'r', 'b', 'r', 0]
Could not restore red in [0, 0, 0, 'w', 0, 0]
refused to store blue in ['w', 'b', 0, 'w', 'b', 'w']
Could not restore blue in [0, 0, 0, 'w', 0, 'w']
refused to store red in ['r', 0, 'r', 'w', 'r', 'r']
Could not restore red in [0, 0, 0, 0, 0, 0]
refused to store blue in ['r', 'r', 'b', 'b', 0, 'b']
Could not restore blue in [0, 0, 0, 0, 0, 0]
refused to store white in [0, 'b', 'w', 'b', 'r', 'w']
Could not restore white in [0, 0, 0, 0, 0, 0]
Greedy Robot: 25686
Smart Robot: 24564
Refused: 490
Performance difference in steps: 1122
Relative performance difference: -4.57
```

Storing two Products of the same Color



Check Empty Warehouse First Storing



Check Policy for Refusal of Storing