SAKI SS19 Homework 3

Author: Joachim Wagner

Program code: https://github.com/audiofrequenz/oss-saki-ss19-exercise-3.git

Summary

Aufgabe 3 umfasst das Erstellen einer automatisierten Lagereinheit mittels Reinforcement Learning. Hierbei wurde sowohl mit einem 2x2 Lager wie auch mit einem 2x3 Lager gearbeitet. Das Lager soll autonom mithilfe eines Roboters mit roten, blauen und weißen Containern verwaltet werden.

Zu Beginn der Aufgabe war es notwendig sich einen Überblick über die benötigten Zustände zu verschaffen. Diese werden nachfolgend zur Erarbeitung der Übergangswahrscheinlichkeitsmatrix benötigt. Diese Matrix wird anhand der Wahrscheinlichkeit einer Änderung von einem Zustand in den nächsten Zustand erstellt. Es resultieren x (n*n) Matrizen. „n“ ist hierbei die Anzahl der Zustände, x ist die Anzahl der Aktionen, die durchgeführt werden können. In diesem Fall umfassen die Aktionen die Lagerposition in welchem ein Ablegen oder Aufnehmen eines Containers geschehen soll.

Um die Übergangswahrscheinlichkeit eines Zustandes in den nächsten zu definieren gibt es, nachdem definiert wurde, welche Übergänge möglich sind, mehrere Möglichkeiten. Zum einen kann eine Gleichverteilung angenommen werden, zum anderen kann mittels eines Trainingssets an Daten eine Wahrscheinlichkeitsbestimmung vorgenommen werden. In unserem Fall war die Wahrscheinlichkeit, dass ein roter Container abgelegt oder geholt werden soll, höher als die der beiden anderen Container-Arten. Diese so ermittelte Wahrscheinlichkeit wird wie folgt in die Übergangsmatrix eingetragen. Eine Reihe einer solchen n*n-Matrix darf in der Summe nur 1 ergeben. Um dies zu gewährleisten wurden die Summen überprüft und je nach über- oder unterschreiten entsprechend angepasst.

Nach der Ermittlung der Übergangswahrscheinlichkeits-Matrizen wurden die Belohnungen definiert, welche sowohl positiv als auch negativ ausfallen können. Reinforcement Learning benötigt diese Belohnungs-Information, um die optimale Strategie zu errechnen und damit einhergehend eine möglichst hohe Belohnung zu erreichen.

Die Auswahl der Belohnungen hat Auswirkung auf das Verhalten des Reinforcement-Algorithmus. Zunächst wurde die Auswahl so gewählt, dass sowohl der nächste freie Platz, im Falle des Ablegens eines Containers, als auch der erste Container mit der richtigen Farbe im Falle des Aufnehmens eines Containers die höchste Belohnung bekommt.

Nachfolgend wurde diese Art der Belohnung überarbeitet, um die Leistung des Algorithmus zu testen.

Hierfür wurde beim Ablegen eines Containers die prozentuale Häufigkeit des Vorkommens auf die Platzbewertung addiert. Somit bekommt der Algorithmus eine höhere Bewertung, wenn er einen roten Container auf den ersten Platz legt gegenüber anders farbigen Containern.

Zum Trainieren des Algorithmus werden neben den Übergangswahrscheinlichkeiten und den Belohnungen noch die maximale Anzahl der Durchgänge sowie der Diskontierungsfaktor eingestellt. Dies dient dazu, dass während des Lernprozesses verhindert werden soll in eine unendliche Schleife zu gelangen. Die Wahl des Diskontierungsfaktors wurde im ersten Fall des Belohnungssystem mit 0,2 gewählt, spielte aber keine Rolle für das Ergebnis.

Als Algorithmen wurden zur Auswertung ValueIteration wie auch PolicyIteration gewählt.

Ein Test der angelernten Algorithmen wurde mittels eines Testdatensatzes gewährleistet. Hierfür wurde die errechnete Strategie der beiden Reinforcement-Algorithmen angewendet, um von einem Zustand in den nächsten zu gelangen.

Ein Überblick über die Performance wurde sowohl über die Zustände, die im Lagerhaus erreicht werden, ermöglicht, als auch mittels der Anzahl der Schritte, die ein Roboter benötigt, welche anhand der Container-Position ermittelt werden. Da allein hierdurch keine Aussage möglich ist, ob ein Algorithmus gute Ergebnisse liefert, wurde ein Greedy-Algorithmus implementiert, der dieselben Aufgaben tätigen musste.

Die Ergebnisse der Algorithmen wurden schlussendlich miteinander verglichen.

Evaluation

Die Evaluation der Algorithmen ergab im Falle der „nächst möglichen“ Verteilung der Belohnung dasselbe Ergebnis wie der Greedy-Algorithmus (Abb. 1).

Die Ursache ist hierbei, dass sowohl Greedy als auch Value/Policy Iteration dem nächsten freien Platz, oder dem nächsten verfügbaren richtig gefärbtem Container, die meiste Bedeutung beimessen.

Zu sehen ist dies an der Abfolge der Zustände, die alle drei Algorithmen einnehmen (Abb. 2).

Aufgrund dessen wurde das Belohnungssystem wie in Summary beschrieben angepasst. Im Falle der Testdaten hatte dies keine Auswirkungen auf die Algorithmen (Abb. 3).

Anhand der Zustände, die während des Verfahrens durchlaufen werden, zu sehen in Abb. 4, wird dieses Vorgehen der Algorithmen ebenfalls deutlich. Eine Unterscheidung zwischen zwei Zuständen ist nur zu finden, wenn beide Felder die gleiche Anzahl an Wegpunkten für den Roboter haben. Dies ist bei einer 3x2 Matrix, wie in Abbildung 5 zu sehen, zum Beispiel Feld 2 und Feld 3.

Aufgrund der langen Laufzeit der Algorithmen sowie der hohen RAM-Auslastung ist ein weiteres Verbessern des Belohnungssystem nicht möglich.

Eine mögliche Verbesserung hierbei könnte sein die Belohnung so zu überarbeiten, dass ein prozentualer Anteil an Feldern, gemessen an den Trainingsdaten, für bestimmte Farb-Container reserviert werden. Sollte nur noch ein solcher reservierter Lagerplatz frei sein, könnte dieser auch für andere Farben freigegeben werden.

Zudem könnten weitere Algorithmen getestet werden. In diesem konkreten Fall steht die Performance des zur Verfügung stehenden Rechners der Auswertung im Weg, siehe Abbildung 6.

Auch das Ändern des Diskontierungsfaktors hatte keine Auswirkung auf die Zustandsfolge und die Schrittlänge des Roboters.

Screenshot

```
#test algorithms with dataset
traveled_fields_mdpResultValue = test_algorithm(mdpResultValue.policy)
traveled_fields_mdpResultPolicy = test_algorithm(mdpResultPolicy.policy)
traveled_fields_greedy = greedy_distance(travel_costs)
print("ValueIteration Robot traveled: ", traveled_fields_mdpResultValue)
print("PolicyIteration Robot traveled: ", traveled_fields_mdpResultPolicy)
print("Greedy Robot traveled: ", traveled_fields_greedy)

ValueIteration Robot traveled:  248
PolicyIteration Robot traveled:  248
Greedy Robot traveled:  248
```

Abbildung 1 Robot Steps Reward Solution #1

Abbildung 2 Robot States Reward Solution #1

```
ValueIteration Robot traveled:  248
PolicyIteration Robot traveled:  248
Greedy Robot traveled:  248
```

Abbildung 3 Robot Steps Reward Solution #2

Abbildung 4 Robot States Reward Solution #2

Feld 1	Feld 3	Feld 5
Feld 2	Feld 4	Feld 6

Abbildung 5 Warehouse

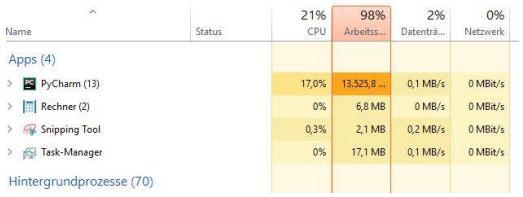

Abbildung 6 Ram-Usage