SAKI SS 2021 Homework {4}

Author: Volodymyr Marych (idm: uv26ukas)

Program code: https://github.com/west9906/SAKI_SS2021/tree/master/homework-4

Summary

Task 3 involves creating an automated repository using reinforcement learning. First, I determined the probabilities of actions(store_red, restore_white, …) based on the warehousetraining files. These probabilities will be further used as probabilities of transition from one state to another. The number of all states is determined by the formula, Ns = A * N ^ S, where A is the number of actions, N is the number of colors including the 'empty', and S is the size of the warehouse. Total number of states:

2x2: Ns = 6 * 4 ^ 4 = 1536

mdptoolbox were used as a library with reinforcement-learning-based algorithm. Two algorithms were chosen: Value Iteration, and Policy iteration. Both need two, the transition probability matrix (T), and the reward matrix (R). T has the shape of S x Ns x Ns, and R = Ns x S. T was row wise normalized, if there are no possible transitions for a state, in this case it is considered a terminal, and then the probability of transition to itself is 1. R was defined base on distance that robot have to travel to change the state, 0 - if it is impossible to store/restore on certain place in warehouse. To train the algorithm, in addition to the transition probabilities and rewards, the maximum number of runs and the discount factor are set. This is to prevent the learning process from looping. The choice of the discount factor was chosen in the first case of the reward system with 0.9, but did not play a role in the result. To evaluate algorithms, a greedy algorithm was implemented that had to perform the same tasks. The results of the algorithms were finally compared with one another.

Evaluation

The evaluation of the algorithms yielded the same result as the greedy algorithm (Fig. 1).

The reason is that both greedy and value / policy iteration attach the most importance to the next free space, or the next available properly colored container. Since the position 1 and 2 has the same reward, state [red red blue empty store_blue] and [red blue red empty store_blue] consider be the same.

This can be seen in the sequence of states (Fig. 2).

```
# -> [0, 1]
#    [2, 3]
self.positions = [0, 1, 2, 3]

# -> [3, 2]
#    [2, 1]
self.reward = np.asarray([3, 2, 2, 1])
```

Due to the long running time of the algorithms and the high RAM utilization, further improvements to the reward system are not possible.

A possible improvement here could be to revise the reward so that a percentage of fields, measured against the training data, are reserved for certain color containers. If there is only one such reserved storage space available, it could also be released for other colors.

In addition, other algorithms could be tested. In this specific case, the performance of the available computer stands in the way of the evaluation.

Screenshot

```
ValueIteration Robot traveled:  228
PolicyIteration Robot traveled:  228
Greedy Robot traveled:  228
```

Fig.1

	0	1	2	3	transaction
0	red	empty	empty	empty	store_red
1	red	white	empty	empty	store_white
2	red	empty	empty	empty	restore_white
3	red	empty	red	empty	store_red
4	red	blue	red	empty	store_blue
5	red	blue	red	red	store_red
6	empty	blue	red	red	restore_red
7	empty	blue	empty	red	restore_red
8	empty	blue	empty	empty	restore_red
9	red	blue	empty	empty	store_red
10	red	empty	empty	empty	restore_blue
11	empty	empty	empty	empty	restore_red
12	blue	empty	empty	empty	store_blue
13	empty	empty	empty	empty	restore_blue
14	red	empty	empty	empty	store_red
15	red	empty	red	empty	store_red
16	red	blue	red	empty	store_blue
17	red	empty	red	empty	restore_blue
18	empty	empty	red	empty	restore_red
19	red	empty	red	empty	store_red
20	red	blue	red	empty	store_blue
21	red	empty	red	empty	restore_blue
22	empty	empty	red	empty	restore_red
23	blue	empty	red	empty	store_blue
24	blue	white	red	empty	store_white
25	blue	white	red	red	store_red
26	blue	white	empty	red	restore_red
27	blue	white	empty	empty	restore_red
28	empty	white	empty	empty	restore_blue
29	red	white	empty	empty	store_red
30	empty	white	empty	empty	restore_red

	0	1	2	3	transaction
0	red	empty	empty	empty	store_red
1	red	white	empty	empty	store_white
2	red	empty	empty	empty	restore_white
3	red	red	empty	empty	store_red
4	red	red	blue	empty	store_blue
5	red	red	blue	red	store_red
6	empty	red	blue	red	restore_red
7	empty	empty	blue	red	restore_red
8	empty	empty	blue	empty	restore_red
9	red	empty	blue	empty	store_red
10	red	empty	empty	empty	restore_blue
11	empty	empty	empty	empty	restore_red
12	blue	empty	empty	empty	store_blue
13	empty	empty	empty	empty	restore_blue
14	red	empty	empty	empty	store_red
15	red	red	empty	empty	store_red
16	red	red	blue	empty	store_blue
17	red	red	empty	empty	restore_blue
18	empty	red	empty	empty	restore_red
19	red	red	empty	empty	store_red
20	red	red	blue	empty	store_blue
21	red	red	empty	empty	restore_blue
22	empty	red	empty	empty	restore_red
23	blue	red	empty	empty	store_blue
24	blue	red	white	empty	store_white
25	blue	red	white	red	store_red
26	blue	empty	white	red	restore_red
27	blue	empty	white	empty	restore_red
28	empty	empty	white	empty	restore_blue
29	red	empty	white	empty	store_red
30	empty	empty	white	empty	restore_red

Fig. 2 Value/Policy iteration (left), Greedy (right)