SAKI SS 2021 Homework 4

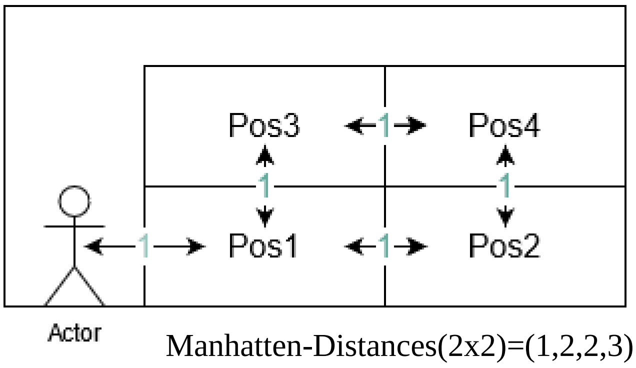
Author: Stefan Fischer

Program code: https://github.com/StefanFischer/SAKI-Project4/releases/tag/homework4

Summary

With the globalization of the world, there is also an increase in globalized trading. Therefore, the importance of logistics are dramatically rising. Besides that, there is a boom in E-Commerce, where the amount of orders is even greater due to the COVID-19 pandemic, costumers which seek for shorter delivery times and higher demand of the costumers regarding product diversity. Those needs do all put more stress on the logistics and the underlying warehouses. The option of smart warehouses is also emerging with current technology, using robots to store and restore items. Artificial Intelligence can be used to optimize the storing strategy in such a smart warehouse, reducing time and costs for the warehouse holder.

In the following the use of reinforcement learning on optimizing a storage strategy is analyzed. A small warehouse with 4 shelves (2x2), 6 shelves (2x3) or 9 shelves (3x3) with 3 different items (red, blue and white) is simulated. A robot interacts with the warehouse by storing and restoring items into the different shelf locations.



Manhatten-Distances(2x2)=(1,2,2,3)

In the Figure there is a scheme of the 2x2 warehouse. To store items the robot (actor) can only enter the shelves from the position 1 (Pos1). Each move has a cost of 1, therefore the distances can be defined as the Manhatten distance, as the robot can only move to adjacent fields and not diagonally. With a given set of orders ("store red", "restore red", "store blue",…) the goal is to optimize the storage strategy by minimizing the distance the actor has covered.

The problem can be modeled as a Markov Decision Process (MDP), where the actor interacts with the surrounding environment and takes action which change the environment in each timestep. In MDPs the goal is to maximize the reward over the episode, which is the whole set of orders. The covered distance is introduced as negative reward. A an MDP is defined with a set of possible environment states **S**, a set of possible actions **A** the actor or robot can take, a transition model given by the transition probability matrix **P**, which defines the probability from going from one state to another state given an action and as last building block the reward matrix **R**, which contains the rewards for all pairs of action and state.

States **S**: The set contains all possible configurations of the storage and additional the order given (store/restore specific item). For the 2x2 warehouse, there are 1536 possible states and for the 2x3 warehouse even 24576, as the number of states is exponentially in the order of the shelf-size.

Actions **A**: The set of actions is the set of positions which are in the warehouse, so there are 4 actions in a 2x2 grid (Pos1, Pos2, Pos3, Pos4). Those actions have a direct impact on the environment and lead to the following state by the transition probability matrix **P**.

Transition Probability Matrix **P**: This matrix defines the transition model for each action **a** between pairs of each states (s_i, s_j). For impossible transitions the value is $P(a)_{i,j} = 0$. For this problem there are six different following states for each state given an action **a**, as only the requirements change. The used transitions probabilities were computed on the given trainings-set, where the frequency of each item was calculated.

Reward Matrix **R**: for a given state s_i and action **a** the reward matrix returns the reward of this specific return. For each movement (store/restore) of the robot a negative reward according to the covered Manhatten Distance was given. If the item could not be stored or restored, a penalty of -50 was given. As **P** has to be stochastic, there have to be self-transitions, but rejecting an item while there was free space was forbidden by an enormous penalty.

Policy $\boldsymbol{\pi}$: A policy $\boldsymbol{\pi}$ describes a strategy, which action should be taken given the current state, so it is a mapping from each state to a specific action to maximize the overall return.

The MDPToolbox offers different algorithms to solve the MDP problem: Policy-Iteration, Value-Iteration and Q-Learning, where Q-Learning is a deep-learning-based method and both other methods are dynamic programming approaches.

Evaluation

As the dynamic programming approaches are both optimal they are also returning the exact optimal results. Q-Learning is not guaranteed to converge to the optimal solution. Therefore, the Value-Iteration was picked as it is faster and needs less RAM and time.

Another hyperparameter of the Value-Iteration algorithm is the discount value. It regulates the impact of future return on the decision which current action to take. If the discount value is zero, the optimized policy and the greedy policy do not differ. In practice the discount value is set to 0.999 to maximize the overall return.

Furthermore, another penalty was introduced. If rejecting of items is not penalized strongly, the optimized agent would reject multiple objects to keep the warehouse divers, but the reward is much worse on all given scenarios.

To evaluate the optimized policy, a greedy policy was implemented, which stores every item on the nearest free position and for restoring also takes the first available item.

	Orders	Rejected Stores/Restores	Greedy Policy	Optimized Policy
Warehouse Order 2x2	60	12	-684	-686
Warehouse Training 2x2	12108	2200	- 128186	- 128228
Warehouse Order 2x3	60	0	-124	-130
Warehouse Training 2x3	12108	0	-2139	-2183

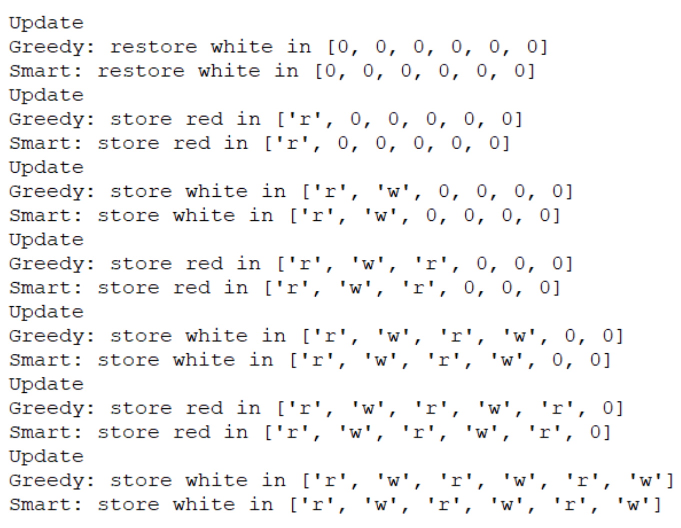
The greedy policy is for the both examples for the 2x3 and 2x2 grid superior to the optimized policy. As there are only two different scenarios the computational evaluation is not representative, but it can be seen that the return difference is quite small regarding the number of orders.

By qualitative evaluation of the optimized policy with a 1x6 grid to strengthen the effects, different behaviors are occurring. The optimized policy stores less common items (white, blue) at farther free space to keep space for the more common red item Furthermore, if a second item of same kind is stored, it is kept in a place further at the end to leave space for an item which is not contained yet and keeping the storage diverse (see Screenshots).

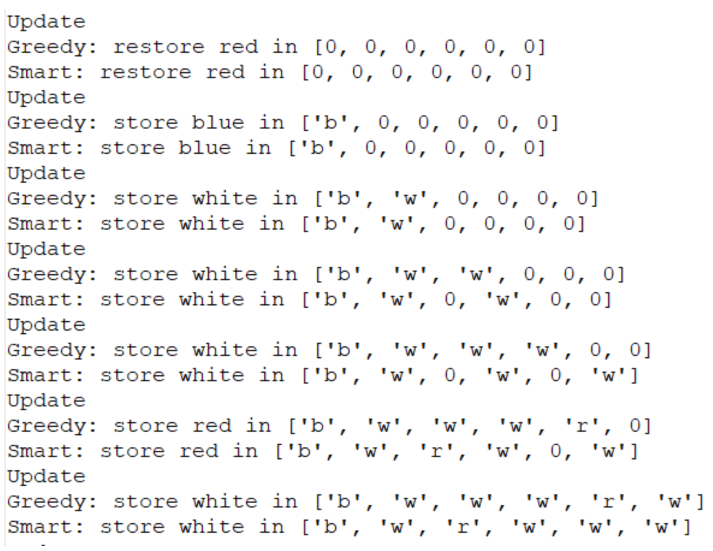
The optimized policy has advantages in edge-cases like storing a item on the first place over a long time, but this was not the case for the two data sets. Furthermore, the size of the storage grid will also improve the advantage of the

The use of reinforcement learning for automatic warehousing does have great potential to reduce costs and to improve greedy processes. In contrast, the warehouses in real industry are much bigger than this toy example, but already for a 3x3 grid the problem can not be solved by normal hardware, even with the implemented sparse matrix approach. Therefore, there is also model-free reinforcement learning which does not need a transition model and solve those problems.

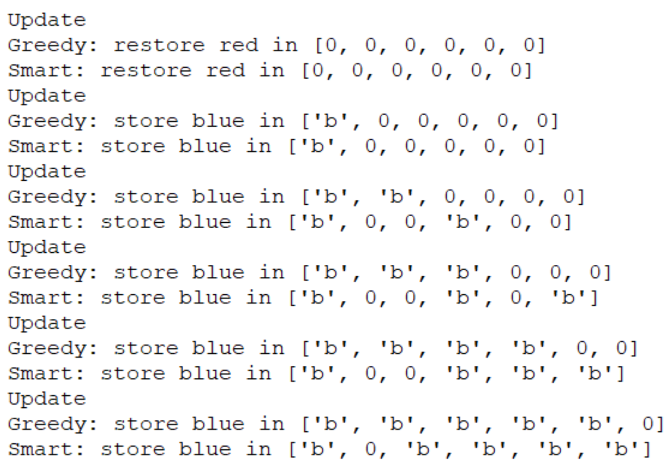
Screenshot



```
Update
Greedy: restore white in [0, 0, 0, 0, 0, 0]
Smart: restore white in [0, 0, 0, 0, 0, 0]
Update
Greedy: store red in ['r', 0, 0, 0, 0, 0]
Smart: store red in ['r', 0, 0, 0, 0, 0]
Update
Greedy: store white in ['r', 'w', 0, 0, 0, 0]
Smart: store white in ['r', 'w', 0, 0, 0, 0]
Update
Greedy: store red in ['r', 'w', 'r', 0, 0, 0]
Smart: store red in ['r', 'w', 'r', 0, 0, 0]
Update
Greedy: store white in ['r', 'w', 'r', 'w', 0, 0]
Smart: store white in ['r', 'w', 'r', 'w', 0, 0]
Update
Greedy: store red in ['r', 'w', 'r', 'w', 'r', 0]
Smart: store red in ['r', 'w', 'r', 'w', 'r', 0]
Update
Greedy: store white in ['r', 'w', 'r', 'w', 'r', 'w']
Smart: store white in ['r', 'w', 'r', 'w', 'r', 'w']
```

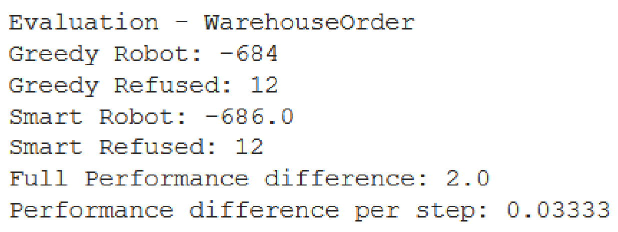


```
Update
Greedy: restore red in [0, 0, 0, 0, 0, 0]
Smart: restore red in [0, 0, 0, 0, 0, 0]
Update
Greedy: store blue in ['b', 0, 0, 0, 0, 0]
Smart: store blue in ['b', 0, 0, 0, 0, 0]
Update
Greedy: store white in ['b', 'w', 0, 0, 0, 0]
Smart: store white in ['b', 'w', 0, 0, 0, 0]
Update
Greedy: store white in ['b', 'w', 'w', 0, 0, 0]
Smart: store white in ['b', 'w', 0, 'w', 0, 0]
Update
Greedy: store white in ['b', 'w', 'w', 'w', 0, 0]
Smart: store white in ['b', 'w', 0, 'w', 0, 'w']
Update
Greedy: store red in ['b', 'w', 'w', 'w', 'r', 0]
Smart: store red in ['b', 'w', 'r', 'w', 0, 'w']
Update
Greedy: store white in ['b', 'w', 'w', 'w', 'r', 'w']
Smart: store white in ['b', 'w', 'r', 'w', 'w', 'w']
```



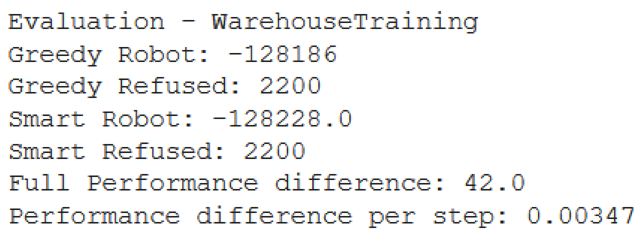
```
Update
Greedy: restore red in [0, 0, 0, 0, 0, 0]
Smart: restore red in [0, 0, 0, 0, 0, 0]
Update
Greedy: store blue in ['b', 0, 0, 0, 0, 0]
Smart: store blue in ['b', 0, 0, 0, 0, 0]
Update
Greedy: store blue in ['b', 'b', 0, 0, 0, 0]
Smart: store blue in ['b', 0, 0, 'b', 0, 0]
Update
Greedy: store blue in ['b', 'b', 'b', 0, 0, 0]
Smart: store blue in ['b', 0, 0, 'b', 0, 'b']
Update
Greedy: store blue in ['b', 'b', 'b', 'b', 0, 0]
Smart: store blue in ['b', 0, 0, 'b', 'b', 'b']
Update
Greedy: store blue in ['b', 'b', 'b', 'b', 'b', 0]
Smart: store blue in ['b', 0, 'b', 'b', 'b', 'b']
```

Differences in behaviour of smart and greedy agent (1x6 grid) with [-1,-2,-3,-4,-5,-6] costs



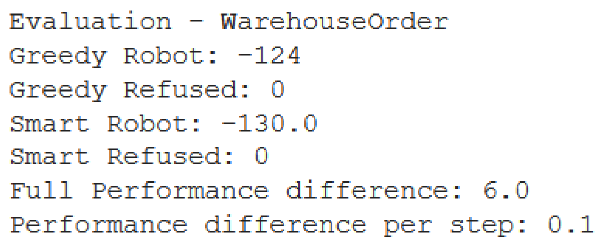
```
Evaluation - WarehouseOrder
Greedy Robot: -684
Greedy Refused: 12
Smart Robot: -686.0
Smart Refused: 12
Full Performance difference: 2.0
Performance difference per step: 0.03333
```

Evaluation Warehouse-Order (2x2)



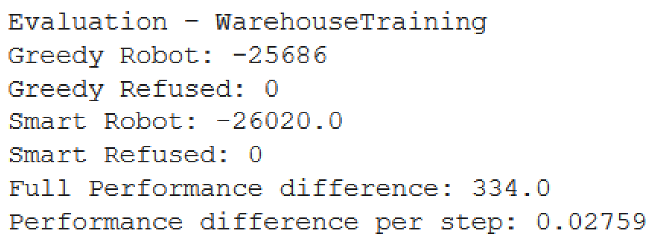
```
Evaluation - WarehouseTraining
Greedy Robot: -128186
Greedy Refused: 2200
Smart Robot: -128228.0
Smart Refused: 2200
Full Performance difference: 42.0
Performance difference per step: 0.00347
```

Evaluation Warehouse -Training (2x2)



```
Evaluation - WarehouseOrder
Greedy Robot: -124
Greedy Refused: 0
Smart Robot: -130.0
Smart Refused: 0
Full Performance difference: 6.0
Performance difference per step: 0.1
```

Evaluation Warehouse-Order (2x3)



```
Evaluation - WarehouseTraining
Greedy Robot: -25686
Greedy Refused: 0
Smart Robot: -26020.0
Smart Refused: 0
Full Performance difference: 334.0
Performance difference per step: 0.02759
```

Evaluation Warehouse-Training (2x3)