Exercise 3
Implementing a deliberative Agent

Group №27: Carolin Beer, Francesco Pase

23rd October 2018

1 Model Description

The state consists of tasks, the current city and the remaining capacity of the vehicle. This is implemented as an array of integers, `stateList` and the variable `capacityLeft` which is of type double.
The stateList is a list of all tasks which represents it's current status for the agent as well as the current city. Assume we have `n` tasks available, then `stateList[0]` to `stateList[n-1]` represent the status of the Tasks with id 0 to `n-1`. The integer `0` denotes that a task is available for pickup, `1` indicates that a task is currently carried by this agent and `2` is the encoding for already delivered tasks.
`stateList[n]` indicates the id of the current city and `capacityLeft` is the capacity of the vehicle minus the weights of the currently carried tasks.

1.1 Intermediate States

Whenever the agent moves the current city is updated. With every pickup or delivery of a task, the capacity and status of the task in the `stateList` change. To handle multi-agent operation, missing tasks that were already picked up by others, are set to the status `2`, so that the agent does not consider them during the optimization.

1.2 Goal State

A goal state can be any state where `stateList[0]` to `stateList[n-1]` are set to `2` (thus, indicating that all tasks are delivered). `stateList[n]` can be the index of any city.

1.3 Actions

Move: change `stateList[n]`, denoting the current city
Pickup: decrease `capacityLeft` by task weight, set `stateList[task id]=1`.

Deliver: increase `capacityLeft` by task weight, set `stateList[task id]=2`.

2 Implementation

2.1 BFS

The BFS algorithm is implemented, as indicated in the exercise description. However, the algorithm will not stop after finding one `goalState`, but until the list `Q`, which contains all nodes, is empty. Core to the implementation is the method returning the successor nodes. These are found by iteratively searching through all the tasks in stateList which have the status `0` (available) or `1` (currently carried) and appending the succeeding state where this task then has a status of `1` or `2` (delivered), respectively.
Afterwards, a method called `computePlan` is called, which allows to transform the `goalState` with the lowest costs into a plan of actions.

2.2 A*

We implemented A* as an extension of the BFS algorithm. It overrides the search algorithm to reflect the A* search as given in the exercise, but used the same method to find a successor and to compute a plan.

2.3 Heuristic Function

The heuristic function uses the cost that were required to get to the current state plus the cost to deliver the first available task from pickup to delivery city. This function turned out to perform best – even though it is only a very rough estimate – as it is very easy to compute. It clearly underestimates the actual cost as it only contains a fraction of it. This guarantees optimality, as discussed in during the lecture.

3 Results

3.1 Experiment 1: BFS and A* Comparison

3.1.1 Setting

We tested the setup for a variable number of tasks and a variable number of capacity per vehicle.

3.1.2 Observations

Both BFS and A* compute the same (optimal) plan, therefore Reward/km and Costs match. With 10kg/30 kg capacity, the maximum number of tasks that can be computed within 60 seconds is 12/10 for BFS and 13/12 for A*. We can clearly see that the additional steps by sorting the list of states according to the heuristic in A* pay off, and increase the efficiency of the algorithm compared to BFS.
Furthermore, a higher capacity per vehicle drives computation times as well since this increases the space of allowed states that have to be explored.

#Tasks	Capacity (kg)	Cost	Reward/km	Time BFS (s)	Time A* (s)
10	10	10100	26	3.34	-
12	10	10650	31	49.9	8.18
13	10	10800	33	»60s	24.14
14	10	11650	33	»60s	87.39
10	30	9100	29	14.66	2.54

Table 1: Comparison of different setups for BFS and A* with a single agent.

3.2 Experiment 2: Multi-agent Experiments

3.2.1 Setting

In the following experiment we compared the average reward per agent for different numbers of agents with a fixed capacity and number of tasks.

3.2.2 Observations

As we can see, the independent optimization of the agents doesn't lead to a mutually optimal outcome. The opposite is the case: the average reward decreases with every additional agent in the simulation. To resolve this, either global planning or collaboration between agents could be possible.

# Agents	#Tasks	Capacity (kg)	Avg. reward/km
1	10	30	29
2	10	30	17
3	10	30	16
4	10	30	15
5	10	30	13

Table 2: Comparison of single- and multi-agent setups using A*.