Exercise 2: A Reactive Agent for the Pickup and Delivery Problem

Group №46: Yann Vonlanthen, Loïc Vandenberghe

October 8, 2019

1 Problem Representation

1.1 Representation Description

Each state $s \in S$ is described by the City of its geographical location l and the desination of the available task t. A state where no task is available is characterized by $t = none$. More formally the state space S can be characterized as follows:

$$S = \{\ [l,t]\ \mid\ l \in Cities, t \in (Cities \setminus \{l\})\ \cup \{none\}\}$$

Note that $|S| = numberOfCities^2$.
For each state there is one action per neighbor, and if $t \neq none$ one additional action to pickup the task. More formally for state $s = [l,t]$, the possibles actions are a_i where $i \in (neighbors(l) \cup \{t\}) \setminus \{none\}$ and the action is defined by:

$$a_i = \begin{cases} \textit{"go to neighbor i"} & \forall i \in neighbors(l) \setminus \{t\} \\ \textit{"pickup task and go to taskDestination t"} & \text{if } i = t \end{cases}$$

Note that when t is a neighbor this description does not include the action of moving to t without accepting the task. However we don't loose enything, as in our setting it will always be better to accept the task while moving to t.
The reward table is defined as follows:

$$R(s = [l,t], a_i) = \begin{cases} -costPerKm * distance(l,i) & \text{if } i \neq t \\ r(l,t) - costPerKm * distance(l,t) & \text{if } i = t \end{cases}$$

Finally the transition table can be defined $\forall s' \in S$ as:

$$T(s = [l,t], a_i, s' = [l',t']) = \begin{cases} p(l',t') & i = l' \\ 0 & otherwise \end{cases}$$

Note that $p(l, null)$ returns the probability that no task is available in city l, which corresponds to the logist implementation.

1.2 Implementation Details

First of all each state is represented by a private class called *State* with two attributes: l for the location city and t for the task destination. The implementation of the $hashCode()$ and $equals()$ methods allow us to use the Java HashMap datastructure efficiently.

Next we initialize said hashmaps, namely $V(s)$, $Best(s)$, as well as an array containing all states.

Agent discount factor γ	0.60	0.85	0.95	0.999
Profit after 700 actions	26_434_018	26_689_699	27_615_746	27_693_112
Average profit per action (To optimize)	37_762	38_128	39_451	39_561
Average profit per km (Graph)	61.451	65.655	67.882	69.348

Table 1: Results of experiment 1

We then perform the RLA algorithm during the setup phase, i.e. we perform value iteration to solve the MDP problem, filling $V(s)$ and $Best(s)$, until the change between one iteration and the other is below the constant ϵ. Note that as seen in class, we will get a guarantee that not only depends on ϵ, but also on the discont factor γ that was chosen.

The implementation is quite straighforward and follows the definitions from above. However, contrary to the above actions are represented by integers, where a negative integer represents the action of picking up a task, and a non-negative integer represents the index in the neighborlist of the city the vehicle is moving to.

Finally, in the act function this has to be taken into account, in order to return the correct PickUp or Move object.

2 Results

2.1 Experiment 1: Discount factor

2.1.1 Setting

We performed this experiment on the *france.xml* topology. Additionally we changed the starting cities to be the same, even though eventually this doesn't really matter. We compare 4 instances of our agent that have respective discount factors of 0.999, 0.95, 0.85 and 0.6. Note that due to a constant ϵ in our setting, the number of iterations, as well as the guarantee on the values in $V(S)$ will not be the same. The experiment can be run using the following program arguments:

```
config/reactiveAllSameStart.xml reactive-rla999 reactive-rla95 reactive-rla85 reactive-rla60
```

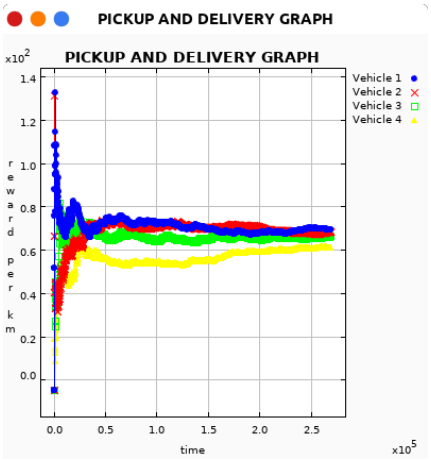
2.1.2 Observations

We observe that after an initial period, from which no conclusions can be drawn due to the high variance of this probabilistic setting, we can observe that the closer the discount factor is to 1, the better the agent performs. We note though that the difference between a discount factor of 0.999 and 0.95 is very small and probably not signicative. Fig.1a shows how eventually all agents are converging to similar average profit per km, but in increasing order of discount factor size. Table 1 further shows how the average profit per action, for which we actually optimize, also increases with a higher discount factor γ.

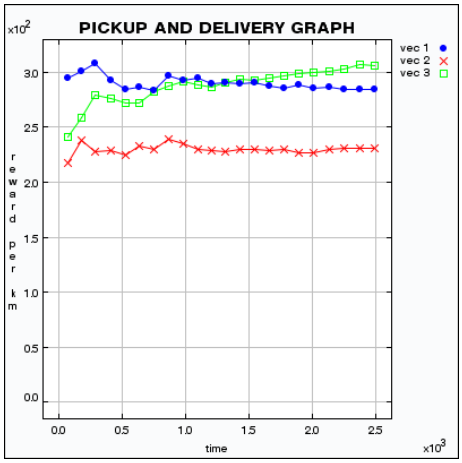
2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

For this experiment we used the Switzerland topology, all vehicles start in Lausanne and the discount factor is set to 0.99 for the trained agent. To launch the experiment the followings arguments are needed:
`config/reactive3.xml reactive-rla reactive-random reactive-dummy`



(a) Experiment 1: Discount factors 0.999 for vehicle 1, 0.95 for vehicle 2, 0.85 for vehicle 3 and 0.6 for vehicle 4



(b) Experiment 2: Comparisons with dummy agents Green is the dummy agent, red is the random agent and blue is the agent using the training algorithm.

Figure 1: Results of experiments 1 and 2

Topology	Switzerland		
Agent	**Random Agent**	**Dummy Agent**	**Trained Agent**
Profit after 1000 actions	37_973_973	39_351_892	45_286_865
Average profit per action	37_973	39_351	45_286
Average profit per km	231	300	284

Table 2: Results of experiment 2

The dummy agent takes an input (acceptance ratio). Then, every time the agent gets an available task, it computes its rewards per kilometer. If the reward/km is bigger than the acceptance ratio, the agent accepts the task. Otherwise it refuses the task and moves to a random neighbor. For this experiment the acceptance ratio is set to 200. We tried many ratios, the dummy agent is performing best with a ratio of around 200 on this topology.

2.2.2 Observations

If we look at the graph, the first impression would be that the dummy agent is performing better than the trained agent. In a sense that's true since the dummy agent is trying to optimize (as good as it can) the average reward per kilometer while our agent is optimizing the reward per action. If we look at the average profit per action however, the trained agent is performing 13% better than the dummy agent. The random agent on the other side performs worse in every regard. This experiment shows that our RLA algorithm is not perfect, and does not give an optimal strategy if we look at the rewards per kilometers. Furthermore it shows that even a simple strategy is able to approach the result of an optimal strategy for a reactive agent in this setting.