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

Group 47: Elia Anzuoni (312827), Francesco Ballestrazzi (312494)

October 6, 2020

1 Problem Representation

1.1 Representation Description

State The set of possible states was chosen to be $S = \{(v, i, j) : \ v \in [V],\ i \in [N],\ j \in [N] \cup \{\perp\}\}$, where V is the number of vehicles and N is the number of cities. A state (v, i, j) represents the situation where vehicle v is currently in city i and is proposed a task with destination city j (if $j \in [N]$), or no task is available (if $j = \perp$).

Action The set of possible actions was chosen to be $A = [N] \cup \{\perp\}$. Action $a \in [N]$ means moving to city a (ignoring the possibly available task); action $\perp$ means to accept the available task. Clearly, not all actions are compatible with all states: for example, action $\perp$ is incompatible with state $(v, i, \perp)$ for every $v \in [V]$ and $i \in [N]$.

Reward Provided that action a is compatible with state s, the next city the vehicle will be in is deterministically determined by the pair (s, a) - even though the next state is a-priori unknown -, and is denoted by $n(s, a)$. Thus, the reward table was chosen as

$$R((v,i,j),a) = -c(v) \cdot d(i,\ n(s,a)) \ + \ r(i,j) \cdot \mathbb{1}\{j \neq \perp\}$$

where $s = (v, i, j)$, $c(v)$ is the cost per kilometer incurred by vehicle v, $d(i, k)$ is the distance between cities i and k, and $r(i, j)$ is the gross reward for a task with pickup city i and delivery city j.

Transition The transition probability $T(s, a, s') = T((v, i, j),\ a,\ (v', k, l))$ is non-zero only if a is compatible with s, $v = v'$, and $k = n(s, a)$. For such feasible triplets, the transition probability is simply $p(k, \ell)$, the probability of there being a task with delivery city ℓ when arriving in city k.

1.2 Implementation Details

A class `State` was implemented, exposing utility methods to the RLA. The situation $j = \perp$ is represented internally as the field `destCity` being `null`.

An action is represented as a `City`, with the correspondence $\perp \rightarrow$ `null`.

The `MyReactive` class maintains a 3D array of all possible states, constructed once and for all before the RLA starts: the RLA updates the fields `value` and `bestAction` in the states at each iteration. To optimise the space, the triplet (v, i, i) was made to index the state $(v, i, \perp)$, since no state can exist where a task is available with pickup city equal to the delivery city.

To enforce the compatibility between states and actions, the iteration over possible actions in the RLA was implemented as an iteration over the list returned by the method `State.possibleActions`;

the methods `State.getNextCity` and `State.reward`, implementing the functions $n(s,a)$ and $R(s,a)$, respectively, assume compatibility between s and a.

The parameter ϵ of the RLA, governing the stopping criterion (iterations are done until the maximum variation over all states since the last iteration is less than ϵ) is determined by a user-defined parameter δ, representing the maximum desired distance between $V(s)$ and $V^*(s)$, over all states s. The inverse formula to determine ϵ is $\epsilon = \frac{\delta}{2} \cdot \frac{1-\gamma}{\gamma}$.

2 Results

In this section we present the results obtained with different parameters configurations (discount factor, delta, type of policy). The performance metrics used are the average reward per kilometer and the average reward per action taken. For each configuration we also show the number of iterations required by the value iteration to converge to the required precision and the number of iterations for which the policy remains unchanged.

2.1 Experiment 1: Discount factor

The discount factor was found to have very little influence on the performance; this can be explained by noting that, with $\gamma = 0$, the resulting policy is greedy, i.e. the optimal one with a horizon of one time-step: in this particular instance, this turned out to be a fairly good policy.

The number of vehicles was also found to be inconsequential to the performance: this can only be explained by assuming that the Logist platform treats every agent independently.

2.1.1 Setting

$\delta = 1$; #Agents $= 3$; $\gamma = \{0, 0.99\}$

2.1.2 Observations

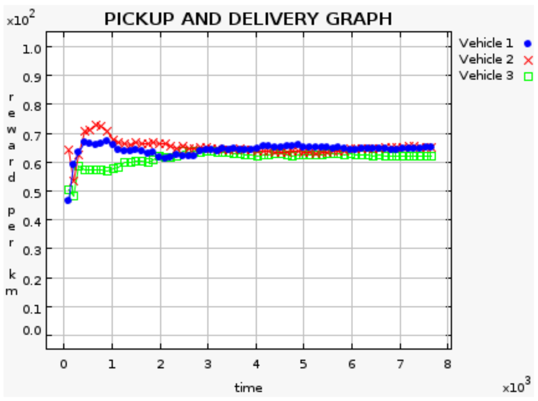

Figure 1: $\gamma = 0,\ R/act = 37000$,
$IterToLearn = 0,\ IterPolicyUnchanged = 0$

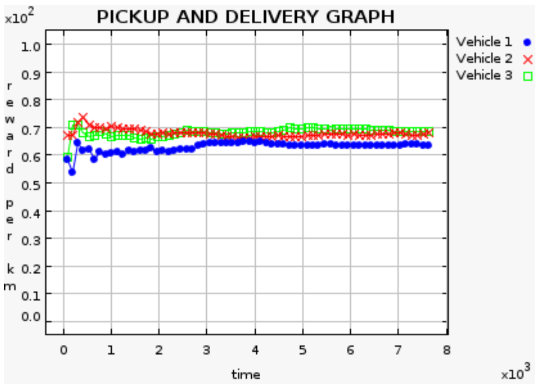

Figure 2: $\gamma = 0.99,\ R/act = 39000$
$IterToLearn = 807,\ IterPolicyUnchanged = 590$

All the policies present a first period of adjustment that then stabilises toward the same reward/km and reward/action for all vehicles. The difference between the results is quite small, with only a slightly worse performance of the policy with discount factor 0. Instead, the number of iterations it takes the RLA to learn the policy is clearly much higher with $\gamma = 0.99$, although the last 590 of them are useless, as the policy does not change.

2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

Our dummy agent has a hardcoded `maxDistance` parameter, set to 500: it always accepts the task, if available, if the delivery city is less than `maxDistance` away from the current city, otherwise it moves to a random neighbour.

We ran an experiment in France and one in the Netherlands with the same agent parameters.

$\delta = 1$; #Agents = 3 (1 RLA, 2 dummy, 3 random); $\gamma = 0.5$ (both for RLA and random).

2.2.2 Observations

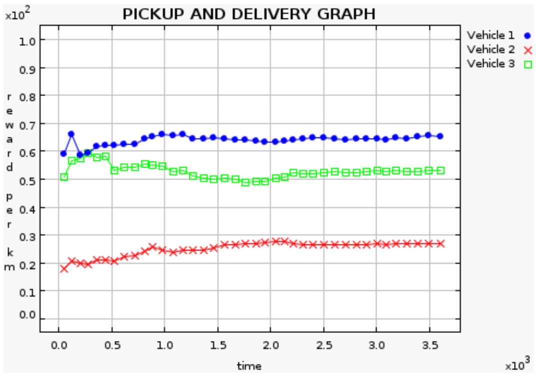

Figure 3: Topology=France, R/act RLA=39000, R/act dummy=11000, R/act random=36000

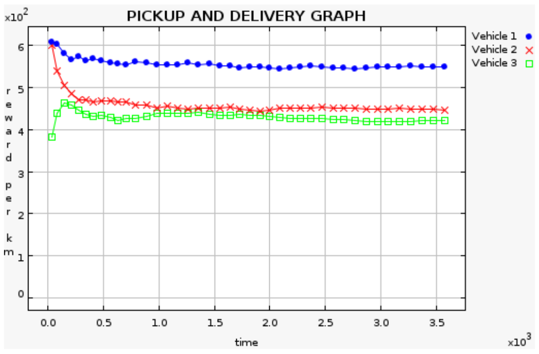

Figure 4: Topology=Netherlands, R/act RLA=46000, R/act dummy=45000, R/act random=43000

As expected, the RLA agent clearly outperforms the other ones, with both metrics (reward/action and reward/km). As a sidenote, our dummy agent has very different performances in the two topologies.

2.3 Experiment 3: Parameter delta

Here, we tweak the desired distance from optimum. Again, even with large delta and thus only few iterations, the final policy is at least as good as the greedy one achieved after just one iteration.

2.3.1 Setting

$\delta = \{1000, 100000\}$; $\gamma = 0.5$; #Agents = 1;

2.3.2 Observations

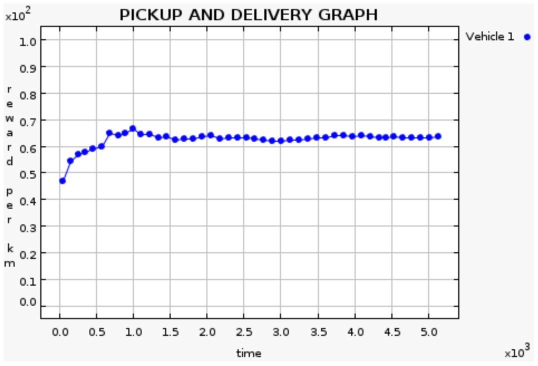

Figure 5: $\delta = 1000$, $R/act = 38500$, $IterToLearn = 6$, $IterPolicyUnchanged = 3$

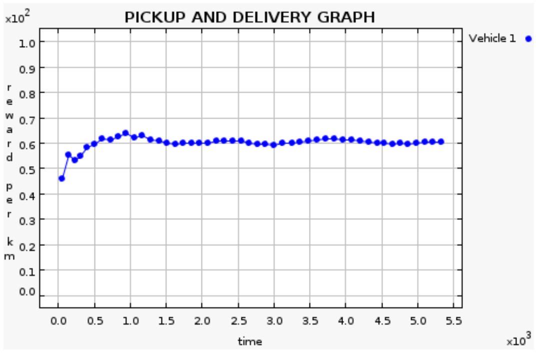

Figure 6: $\delta = 100000$, $R/act = 37600$, $IterToLearn = 1$, $IterPolicyUnchanged = 0$