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

Group №:3 Yannick Grimault, Vincent Petri

October 11, 2016

## 1 Problem Representation

### 1.1 Representation Description

We chose for our state representation that $s$ would be a pair $(City, Task)$. Here $City$ represents the current city in which the agent is and $Task$ is the destination of the proposed task. If no task is provided, $Task = null$.

The different action that our agent can take in a state is either accept the proposed task if one is provided or move to a neighbour city. The reward table is defined as follow:

- $R(s, a) = 0$ for move action
- $R(s, a) = r(a)/D$ for pickup actions where $r(a)$ is the given reward for the action and $D$ is the distance between the two cities

With this representation $T(s, a, s') = p(NewCity, NewTask)$ with $s' = (NewCity, NewTask)$

### 1.2 Implementation Details

In order to implement our learning algorithm we created a list called *allState* of all possible state over which we iterate to compute our optimal strategy.

We decided to implement our action as integers:

- if the agent decide to pickup the task $a = -1$
- else $a = i$ where $i$ is the index of the neighbour city the agent decide to go to

We created a class actionReward which has two attributes, an *action* represented as an integer and a $reward$ represented as a double.

In order to be able to use the Map we created between our state list and corresponding actionReward we had to override the hashcode method on the state object. This allowed us to get the actionReward we wanted by using a new state object instead of the one in allState list.

To compute the n-th iteration of the strategy we start by computing the reward for the expected pickup action if there is one, else it is set to a negative integer. We then compute over all neighbour cities the expected reward as stated in the given formula. We finally choose the highest one and store it in our $newStrategy$.

In our implementation we choose to repeat the learning process for a maximum of 100000 iteration or when the difference in V(S) between two iteration is smaller than 0.01. To test this difference we compute $\sqrt{\sum(strategy.s.reward - newStrategy.s.reward)^2}$

## 2 Results

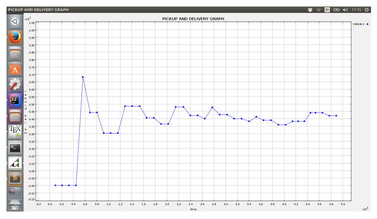
### 2.1 Experiment 1: Discount factor

#### 2.1.1 Setting

For this experiment we chose different value for the discount factor: 0.5; 0.75; 0.90; 0.999. We then analyze the computed strategy table and the reward graphs.

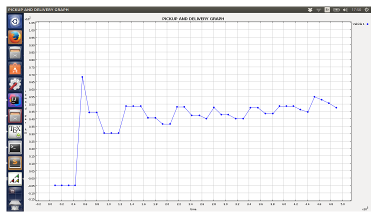
#### 2.1.2 Observations

- $\gamma = 0.5$



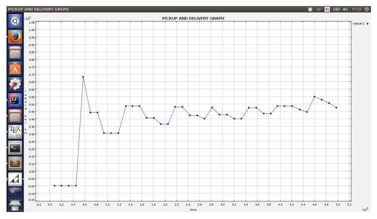
With this configuration we reach a total profit of 346838 in 10 actions. The strategy converges in 8 iterations.

- $\gamma = 0.75$



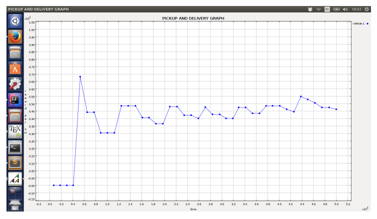
With this configuration we reach a total profit of 325865 in 10 actions. The strategy converges in 17 iterations.

- $\gamma = 0.9$



With this configuration we reach a total profit of 325865 in 10 actions. The strategy converges in 44 iterations.

- $\gamma = 0.999$



With this configuration we reach a total profit of 325865 in 10 actions. The strategy converges in 4489 iterations.

We can see that all these results are really close. This is easy to explain. Indeed with the representation we choosed, the strategy is often the same. It accepts most of the given tasks. It only refuse the one which are really far away (for example from Marseille to Brest). The strategy being really close between the different setups we end up with really close results.

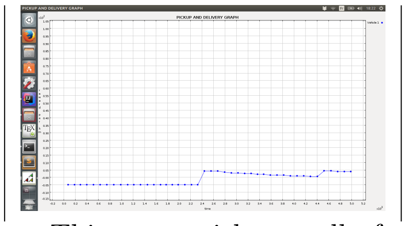
## 2.2 Experiment 2: Comparisons with dummy agents

### 2.2.1 Setting

We chose to compare our agent (with the results we found in the previous experiment) to two dummy agents. The first one is provided in the template, it is a random agent moving randomly and accepting tasks with a probability of 0.5. The second agent is moving randomly when no task is provided and accept all the tasks.

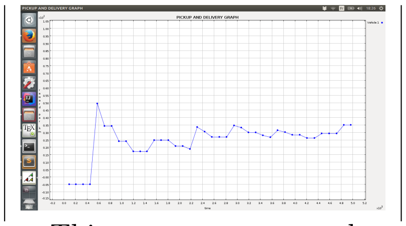
### 2.2.2 Observations

- Random Agent



This agent pickup really few tasks. It generates almost no profit. After 10 action it reached 20020. That is more than 10 times less than our reactive agent.

- Deterministic Agent



This agent generates a better profit than the random one with a total of 292953 after 10 tasks. Still our reactive agent managed to do better thanks to the learning algorithm we implemented.