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

Group 23: Alessandro Bianchi, Davide Nanni

October 5, 2020

1 Problem Representation

1.1 Representation Description

- State ::= ⟨ `City` *current city*, `City` *task destination*⟩ where *task destination* can be `null` .
- Action ::= ⟨ `City` *destination city*, `boolean` *deliver task*⟩. Possible actions:
  - Actions with *deliver task* = `true` only if *destination city* == State's *task destination*
  - Actions with *destination city* neighbor of State's *current city* and *deliver task* = `false` if State's *task destination* == `null`
- State transition $(s, a) \rightarrow s'$ where *current city* of s' = *destination city* of a
- Transition probabilities
  $P =$ `TaskDistribution.probability` $(s\ destination\ city, s'\ destination\ city)$
- Reward
  $R(s, a) = task\ reward - cost\ per\ KM \cdot distance(current\ city,\ destination\ city)$.
  If no task is picked, $task\ reward = 0$

1.2 Implementation Details

- The `State` and `ActionReactive` classes represent the States and Actions, respectively
- A `Pair` class has been created in order to represent the `<State, ActionReactive>` entry
- The `setup()` function performs the following actions:
  1. Fill a map `cityStates` with a list of all possible States in each City
  2. Fill a map `stateProbabilities` with the transition probability of each State
  3. Fill a map `stateActionSpace` with a list of all possible Actions from each State
  4. Fill a map `stateActionRewards` with the reward for each `<State, ActionReactive>`
  5. Initialize all `vValues` to `- (double) (int) Double.MAX_VALUE`
- The `train()` function implements the *Q-lerning* algorithm. The function iterates in order to compile the `stateActionBest` map, which represents the best Action for each State. Convergence is reached when no changes were made to `stateActionBest` throughout the latest iteration.
- The `act()` function simply checks the State and performs the best Action based on `stateActionBest`

2 Results

In this section, we will present the results of a series of experiments we conducted to verify the impact of the discount factor, the performance of our agent compared to those of dummy and random ones and how different agents operate in competition. Because of space constraints, only experiments in the 'France' topology were reported. A small diversion concerning the results in other topologies are reported below.

2.1 Experiment 1: Discount factor

2.1.1 Setting

In order to test the impact of the discount factor, two different simulations were made:

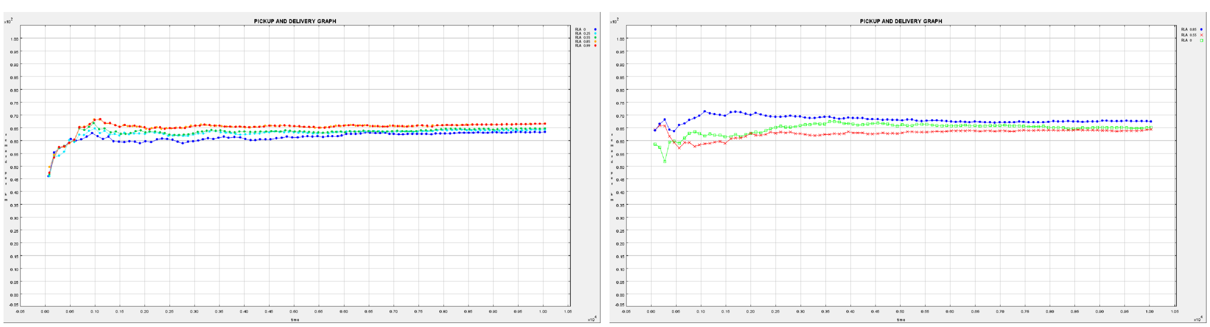
1. The first one aims at verifying the role of the `discountFactor` in a context with a single agent running. The following values were tested, and plotted when possible: $0, 0.25, 0.55, 0.85, 0.99$ and 1.

2. The second test uses the results of the first one, and aims at showing how different agents perform when competing against each other using various `discountFactor` s. In accord to the results obtained in the first simulation, only the most meaningful values $(0, 0.55$ and $0.85)$ were kept.

2.1.2 Observations

1. Figure 1a compares the performance of a single agent, running solo with the aforementioned values of `discountFactor` . As we can see, a higher value implies a better performance of the agent. Obviously, the number of iterations needed in order to reach convergence grows with `discountFactor` as well. This relationship, however, is not linear. On the one hand, the improvement in performance given by increasing `discountFactor` from 0.55 to 0.85 possibly justifies the 131 iterations respect to 41. On the other hand, the improvement is barely noticeable when increasing the `discountFactor` to 0.99, but a grand total of 1837 iterations is necessary in order to reach convergence. It is worth pointing out that, as expected, convergence cannot be reached with `discountFactor` $= 1$, with the training phase resulting in a timeout error (`logist.LogistException: agent reactive-rla timed out`).

2. Figure 1b shows how agents with a significantly different `discountFactor` behave in competition. As we can observe, the agent with the highest `discountFactor` still has the highest reward per km. The other two agents, however, eventually do not differ by much.



(a) Reactive solo, various discount factors (b) Competitive Reactive, various discount factors

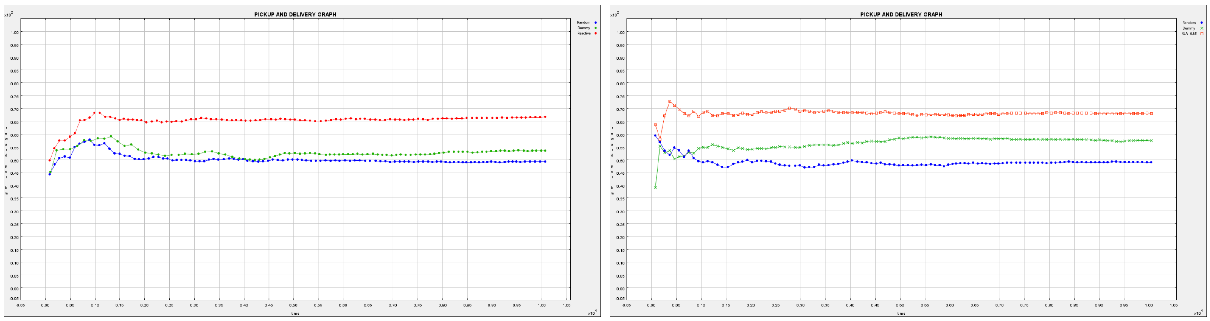
2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

In order to perform this experiment, a new agent `DummyAgent` was defined, in addition to the existing `RandomAgent` and `ReactiveAgent` . The policy defined for `DummyAgent` is to always accept a task, when one is proposed. In this experiment, we will compare the results of the three agents, seeing how they behave both when running solo and in competition.

2.2.2 Observations

As we can see both in Figure 2a and Figure 2b, there is a big performance gap between the three agents. In particular, `ReactiveAgent` makes good use of its training and always performs much better than the other two. The results of `RandomAgents` and `DummyAgents` look comparable in the solo case, while the latter performs clearly better in the scenario of a competitive execution.



(a) Random, Dummy, Reactive solo (b) Competitive Random, Dummy, Reactive

The three agents were run concurrently in all the other topologies as well. In general, the results are similar, although with higher absolute values. In 'The Netherlands', however, the percentage gain of `ReactiveAgent` was better compared to 'France': 45% instead of 30% against the `RandomAgent` , 32% instead of 18% against the `DummyAgent` .

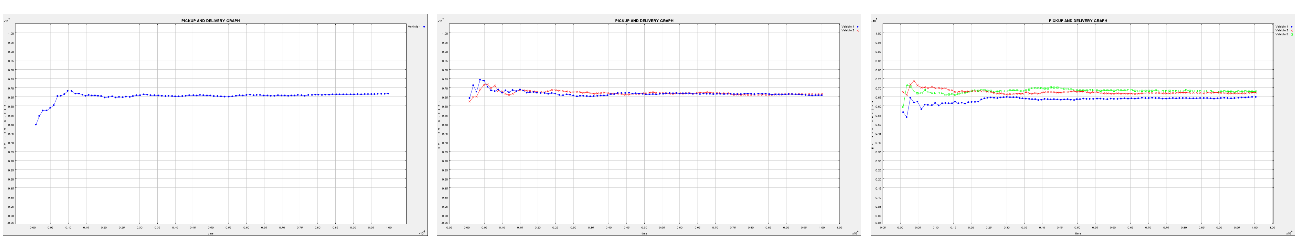
2.3 Experiment 3: Behavior in competition

2.3.1 Setting

In this experiment, we try to analyse the performance of identical agents (`ReactiveAgent` with `discountFactor` $= 0.85$) when executed in competition and, therefore, in competition. In particular, we will run simulations with one, two and three agents executing at the same time.

2.3.2 Observations

As we can see from Figure 3a, Figure 3b, and Figure 3c, the overall performance of the agents varies slightly as the number of competing agents changes.
In Figure 3b it is interesting to observe how, despite the presence of one more agent compared to Figure 3a, there seems to be enough resources for the two to co-exist and obtain a similar reward per km as they would when acting alone. However, we can see how the increase in one's reward is usually met with a decrease in the other's. This, however, balances out in the long run, as both agents have the same policy.
In Figure 3c we can see that agent 1, despite having the same policy as the other two, started out slow, and did not manage to catch up with the other two.



(a) Single Reactive (b) Two concurrent reactive (c) Three concurrent reactive