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

Group №: Student 1, Student 2

October 10, 2017

1 Problem Representation

1.1 Representation Description

States correspond to the current city of the agent and the available task. We decided to represent them as a pair `(cityFrom, cityTo)` where `cityTo` is the city of delivery of the task and can be `null` if there is no task available. From a state `(cityFrom, cityTo)`, possible actions are either to pickup – if `cityTo` is not `null` – the task (going to `cityTo`) or – in any case – to move to a neighbor of `cityFrom`. The reward $R(s, a)$ for a pair [state s; action a] is the value of the delivery (if the action is a pickup otherwise 0) minus the `costPerKm` times the distance traveled.

1.2 Implementation Details

1.2.1 Representations

The set of possible states is implemented as a HashMap of HashMaps:

```
states = new HashMap<City, HashMap<City, State>>();
```

To each city `cityFrom` is associated an hash map `hm` that associates each possible task `cityTo` available in `cityFrom` and the `null` task with an object `State`.

An object `State` contains 6 parameters:

- `City cityFrom`: *city where is the car*
- `City cityTo`: *city where is the car*
- `HashMap<City,Double> values`: *associating possible actions (neighboring cities + cityTo) with Q-values*
- `HashMap<City,Double> rewards`: *associating possible actions (neighboring cities + cityTo) with rewards*
- `double bestValue`: *value* $V(s) = \max_a Q(s, a)$
- `City bestAction`: *policy* $\pi(s) = \text{argmax}_a\, Q(s, a)$

While setting up the reactive agent's class `ReactiveMDP`, we fill the hash map of states `States` with created object states for each possible pair `(cityFrom, cityTo)`, and we also initialize Q-values and rewards for each state by looping over possible actions (neighboring cities + `cityTo` if not `null`).

1.2.2 Value Iterations

Then, the value of each `State` s contained in the hashmap `states` is updated with a value iteration step:

```
Algorithm 1: Value iterations for state $s$.
  oldValue = $V(s)$
  for action $a$ in State $s$ do
      newCityFrom ← $a$
      $Q(s,a) \leftarrow R(s,a)$
      for each city newCityTo do
          $s' \leftarrow$ [newCityFrom ; newCityTo]
          $Q(s,a) \leftarrow Q(s,a) + \gamma \mathcal{P}$ [task from newCityFrom to newCityTo] $V(s')$
      end
      $Q(s,a) \leftarrow Q(s,a) + \gamma \mathcal{P}$ [no task from newCityFrom] $V(s')$
  end
  $V(s) = \max_a Q(s,a)$
  $\pi(s) = \text{argmax}_a Q(s,a)$
  $\delta(s) = (\text{oldValue} - V(s))^2$
```

When value iterations converge to fixed point V^* (when $\sum_s \delta(s) < 1e^{-6}$), we stop value iterations and we launch the MDP with computed policies $\pi(s) \forall s$.

2 Results

2.1 Experiment 1: Discount factor

2.1.1 Setting

We ran our optimisation algorithm for discount values of 0.01, 0.25, 0.50, 0.75, and 0.99

Command line:

```
java -jar ../logist/logist.jar config/reactive.xml reactive-.01 reactive-.25 ...
... reactive-.50 reactive-.75 reactive-.99
```

2.1.2 Observations

After letting the simulation run for a long time (1.510^4 ticks), we observe that discounts factors closer to 1 have a higher average profit.

```
reactive-.01
AVERAGE PROFIT: 39580.19921052631

reactive-.25
AVERAGE PROFIT: 41657.42321755028

reactive-.50
AVERAGE PROFIT: 42159.52963477821

reactive-.75
AVERAGE PROFIT: 42300.31010737628

reactive-.99
AVERAGE PROFIT: 42788.204704532414
```

2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

We compare the reactive agent with discount factors of 0.99 and 0.01 to two dummy agents. The first one that always do pickUp actions if available and the second one who takes the pickups only with a probability of 50%. Both dummy agent move randomly when not doing pickups.

Command line:

```
java -jar ../logist/logist.jar config/reactive2.xml reactive-.01 reactive-.99 ...
... reactive-random reactive-alwaysPickUp
```

2.2.2 Observations

We observe that the random agent performs very badly, but that always picking up is not so bad compared to the reactive with discount factor of 0.01 we can understand that as a small discount factor means that the agent will care more about immediate rewards than future value, which means always picking up available tasks.

```
reactive-.99
AVERAGE PROFIT: 2607.9454202880934

reactive-.01
AVERAGE PROFIT: 2297.9035733919736

reactive-alwaysPickUp
AVERAGE PROFIT: 2246.7879190175904

reactive-random
AVERAGE PROFIT: 968.0287038588261
```

2.3 Experiment 3

2.3.1 Setting

Command line:

```
java -jar ../logist/logist.jar config/reactive3.xml reactive-.01 reactive-.25 ...
... reactive-.50 reactive-.75 reactive-.99
```

and:

```
java -jar ../logist/logist.jar config/reactive4.xml reactive-.01 reactive-.25 ...
... reactive-.50 reactive-.75 reactive-.99
```

2.3.2 Observations

These two experiments tend confirm the results of experiment 1 even though the differences between different discount factors are sometime very small.