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

Group №:9 Fengyu Cai, Wanhao Zhou

October 8, 2019

## 1 Problem Representation

In this section, we briefly explain our design of the *state representation*, *action space*, *reward table* and *probability transition table* along with the implementation details.

### 1.1 Representation Description

- State representation: $\mathcal{S}$. We uniquely define each state of the agent as a tuple consisting of two cities, namely `(currentCity, packageDest)`. The latter can be `null` if there is no task available when the agent is in the `currentCity`.

- Action space: $\mathcal{A}$. The possible actions of a given state is defined by the potential moving directions of a vehicle. To be specific, the moving directions include: all neighbouring cities of the current city plus the package destination field of the state if it is not `null` or not contained in the neighbouring cities already. For discriminating between actions `Move` and `Pickup`, see details below.

- Reward table: $\mathcal{R}$. Entries of the reward table consists of two parts: the cost of travelling and the reward for accomplishing the task. The cost is computed by the total length of two cities times the cost per kilometres of the vehicle. The reward is determined similarly by source city and the destination city. This reward is counted if and only if the `packageDest` field of the current city is not `null` and it equals to our current action, meaning that we are actually carrying out the task.

- Probability transition table: $\mathcal{P}$. Denote the old state as $s$, and the action executed is $a$. This means that for the next state $s'$, we immediately have that $s'.\texttt{currentCity} = a$. The new state is thus defined by a new package destination (can be `null`) and the related transition probability is calculated via the probability of there exists a task from $s'.\texttt{currentCity}$ to $s'.\texttt{packageDest}$. We enumerate all possibilities and the rest is considered as probability of no task available.

### 1.2 Implementation Details

The file organisation is as follows. `ReactiveTemplate` is the original dummy random agent. `ReactiveTrained` is the agent of our reinforcement learning algorithm. `ReactiveState` is the state definition class. `ReactiveControl` is the main class that runs the learning algorithm and yields the optimal policy.

- State representation $\mathcal{S}$ and action space $\mathcal{A}$. We define a class called `ReactiveState` which consists of three members: `currentCity`, `packageDest` and `actions`. The action space for each state is enclosed within the class `ReactiveState` member `actions`. It is defined as an `ArrayList` of all possible actions (to the next city). All possible states and actions are initialised in `setup` method. To determine the actual action, `Pickup` is assigned if and only the destination of the `availableTask` is the same as our chosen action of the state. Otherwise, we choose `Move`.

- Reward table $\mathcal{R}$ and probability transition table $\mathcal{P}$. There is no explicit definition as data structures in our implementation. They are called via the `TaskDistribution` object in the training process.

- Key ingredients of the reinforcement learning algorithm.

  (a) $V$ Table. Defined as $V : \mathcal{S} \to \mathbb{R}$, is implemented as a `HashMap<ReactiveState, Double>`.

  (b) $Q$ Table. Defined as $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, is implemented as a `HashMap<ReactiveState, HashMap<City, Double>>`.

  (c) Policy $\pi$. Defined as $\pi : \mathcal{S} \to \mathcal{A}$, is implemented as a `HashMap<ReactiveState, City>`.

- Reinforcement learning algorithm. Given below.

**Algorithm 1** Reinforcement learning

**Require:**
State value function $V$, Q-value function $Q$, policy $\pi$.
Sates $\mathcal{S}$, Actions $\mathcal{A}$, Reward function $\mathcal{R}$, Transition probabilities $\mathcal{P}$, Discounting factor $\gamma$, epsilon $\epsilon$.
**procedure** ReinforcementLearning($V, Q, \pi$, $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$, $\mathcal{P}$, $\gamma, \epsilon$)
  Initialise $V, Q, \pi$.
  **while** `true` **do**
    $V' \leftarrow V$, `diff` $\leftarrow 0$
    **for** $s \in \mathcal{S}$ **do**
      **for** $a \in \mathcal{A}(s)$ **do**
        $Q(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \cdot \sum_{s'} \mathcal{P}(s,a,s') \cdot V'(s')$
      $V(s) \leftarrow \max_a\{Q(s,a)\}$, $\pi(s) \leftarrow \text{argmax}_a\{Q(s,a)\}$
      `diff` $\leftarrow$ `diff` + `abs`$(V(s) - V(s'))$
    **if** `diff` $< \epsilon$ **then** break
  **return** $V, Q, \pi$

# 2 Results

## 2.1 Experiment 1: Discounting factor

The discounting factor will influence the extent how the agent weighs the future reward. The smaller discounting factor indicates that the agent more focus on the present and its following steps, while the larger discount indicates that the agent consider more on the long-term future.

### 2.1.1 Setting

We set the discounting factor $\gamma$ as evenly spaced numbers over the interval from 0 to 1, in order to find the relationship between the discounting factor and time consumption to train the agent. For the clear visualization, we select $\gamma = 0.1, 0.5, 0.9$ to observe the relationship between the unit cost and the discounting factor.

### 2.1.2 Observations

From Table 1, we could find out that with the discounting factor increasing, we can figure out that the time will be taken longer, which quite makes sense, as mentioned that more attention has been placed for the longer future.

In the Figure 1, the discounting factor are $\gamma = 0.1$ (blue), $0.5$ (red), $0.9$ (green). We could find out the fact that when the discounting factor is larger, the reward per km will be higher. The difference between 0.1 and 0.5 is larger than the difference between 0.5 and 0.9.

| $\gamma$ | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time(ms) | 13 | 19 | 18 | 18 | 25 | 27 | 37 | 57 | 72 | inf |

Table 1: The relationship between time and discounting factor


Figure 1: The cost per km with discount factor $\gamma = 0.1$ (blue), $0.5$ (red), $0.9$ (green)

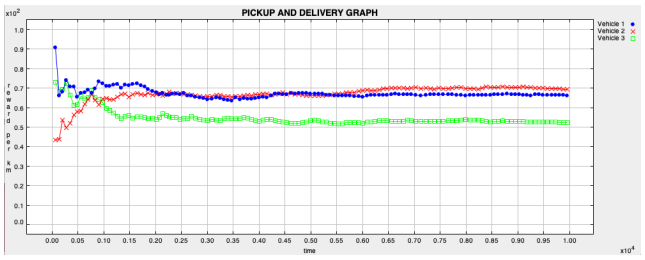
## 2.2 Experiment 2: Comparisons with dummy agents

### 2.2.1 Setting

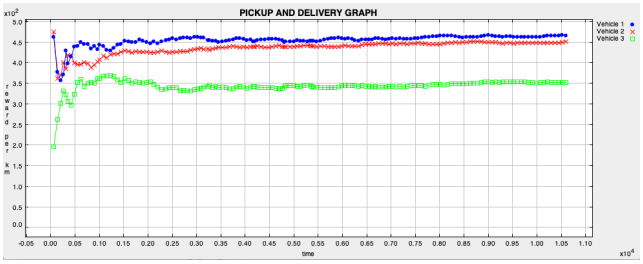
In this experiment, we compare our well-trained agent (blue) (discount factor $\gamma = 0.9$) with two dummy agents, which one of them is only the simple trained agent (red) with the discount factor $\gamma = 0.1$ and the other is random agent (green). In the Figure 2(a) and Figure 2(b), we plot the graph in France and the Netherlands.

### 2.2.2 Observations

We could find out that the trained models ($\gamma = 0.1, 0.9$) perform quite similarly, and they are much better than the dummy agent in both topologies.




(a) Compared with dummy agent France




(b) Compared with dummy agent Netherlands

Figure 2: $\gamma = 0.1$ (blue), $0.9$ (red) and dummy agent (green)