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

Group №23: Jean-Thomas FURRER, Emily HENTGEN

October 8, 2017

## 1 Problem Representation

### 1.1 Representation Description

In our implementation, a state is described by three different attributes: the current city of the agent, whether there is an available task is this city or not, and the destination city of the task.

An action is described by three different attributes as well: whether it consists of a pick up or just a move between two cities, the departure city and the destination city.

The reward table consists of a table mapping a *State* to a numerical reward. Similarly, the probability transition table maps a *State* to the probability of arriving at this *State*. For a *State* with an available task, this is simply the probability of having an available task in the current city to the destination city (given by the task distribution probability). For a *State* with no available task, this is 1 minus the sum of all probabilities of having a task from the current city to another city in the network.

### 1.2 Implementation Details

The agent behaviour is implemented in the *ReactiveTemplate* class. For comparison purposes, there is also a dummy agent who randomly travels between cities, and randomly picks the available task or not; its behaviour implemented in the *ReactiveDummyTemplate* class. The two classes *State* and *Action* implement a state respectively an action representation, both defined by their respective above-mentioned attributes.

The main data structures are implemented as *HashMap*s, so that accessing elements is fast. The data structure representing the set of all states is a *List*. The additional data structure *statesForCity* is not strictly necessary, but speeds up the retrieval of all states associated to a given city (instead of looping through the entire *allStates* list).

**rewards** *HashMap<template.Action, Double>*

**probabilities** *HashMap<State, Double>*

**bestActions** *HashMap<State, Action>*

**bestValues** *HashMap<State, Double>*

**allStates** *List<State>*

**statesForCity** *HashMap<City, List<State>>*

The redundant attributes between a *State* and an *Action* (in particular the current city/departure city) make the use of the HashMaps more practical: the keys, which conceptually consists of a *(State, Action)* pair are here either a *State* or an *Action*, which simplifies the overall implementation.

**Algorithm 1** Learning Strategy (Value iteration)

INPUT

*rewards* ▷ the table mapping an action to its expected reward

*probabilities* ▷ the table mapping a state to the probability of being in this state

*allStates* ▷ the set of all possible states

*statesForCity* ▷ the table mapping a city to its set of possible states

*discountFactor* ▷ the probability an agent picks up a task

OUTPUT

*bestMoves* ▷ the table mapping a state to the next best action and the associated value

$hasConverged \leftarrow false$
initialize $bestMoves$ to an empty table

**while** not $hasConverged$ **do**
  **for** $state$ in $allStates$ **do**
    $maxQ \leftarrow -\infty$
    **for** $action$ in $state.actions$ **do**
      $acc \leftarrow 0$
      **for** $nextState$ in $statesForCity[action.cityTo]$ **do**
        $acc = acc + probabilities[nextState] * bestMoves[nextState].value$
      **end for**
      $Q = rewards[action] + discountFactor * acc$
      **if** $Q > maxQ$ **then**
        $maxQ \leftarrow Q$
        $bestMoves[state] = (action, Q)$
        $hasConverged \leftarrow false$
      **end if**
    **end for**
  **end for**
**end while**
**return** bestMoves

# 2 Results

## 2.1 Experiment 1: Discount factor

### 2.1.1 Setting

### 2.1.2 Observations

## 2.2 Experiment 2: Comparisons with dummy agents

### 2.2.1 Setting

### 2.2.2 Observations

⋮

## 2.3 Experiment n

### 2.3.1 Setting

### 2.3.2 Observations