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

Group 24: Eloi Jean Benvenuti, Timothée-Florian Sébastien Bronner

October 10, 2017

1 Problem Representation

1.1 Representation Description

State representation A state is defined by the city in which the vehicle is, whether there is an available task in the city and, if there is, to which city the task requires the vehicle to move to. Since the agent will be presented only one task even if there are multiple available tasks in the city, there is no need to add extra states for the cases where they are some combinations of available tasks in the city.

This means that, for each city, there is (n-1), where n is the number of cities, state where a task is available (one for each other city) and 1 state for when no task is available. Since there are n cities, this gives us a total of n^2 states.

Possible actions When he is in a city the vehicle can either pick up the available task and go to the corresponding city, if a task is available, or not pick up anything and move to a neighboring city.

Reward table In order to decide on the best action we need to know, for each state, the best expected rewards. We store those in a dictionary with an entry for each state.

Probability transition table By picking our actions we can control in which city we end up. The random part is the state of the city (whether there will be a task and to which city the task require us to go). We didn't need to construct this table because we could query all the probabilities by an API call.

1.2 Implementation Details

State class The state class has 3 variables:

- startCity: The city in which the vehicle currently is.
- endCity: The city in which the task requires us to go. This is equal to null if there is no available task.
- task: A boolean who is equal to true if there is a task in the city, and false otherwise.

AgentMove class The AgentMove class encodes the different actions the vehicle can take. It has 3 variables:

- startCity: The city in which the vehicle currently is.
- endCity: The city in which the vehicle will move.
- task: A boolean who is equal to true if the vehicle do this movement in the context of picking up and delivering a task and false otherwise.

Reward HashTable We encoded the best reward for each state into a HashMap<State,Double> who takes states as key and returns the expected rewards.

Best action HashTable We encoded the best action for each state into a HashMap<State,AgentMove> who takes states as key and returns the best move encoded in the AgentMove class.

Iterative computation of the reward table We computed the best actions and optimal rewards following this procedure:

Data: States $s \in S$, actions $a \in A$, forgetting factor γ, probability transition table T
Result: Best expected reward for each state V, best action for each state

Initialize V=0 and V_=0 for all states;
do
  V_ = V ;
  for $s \in S$ **do**
    for $a \in A$ **do**
      $R(s,a)$ = expected rewards of the task − cost of movement ;
      $Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') * V\_(s')$;
    end
    a_{max} = find_max($Q(s,a \quad \forall a \in A)$) ;
    $V(s) = Q(s, a_{max})$;
    bestAction(s) = a_{max} ;
  end
while $|V(s) - V\_(s)| < \epsilon \quad \forall s \in S$;

Algorithm 1: Computation of the best action and the best expected reward.

2 Results

2.1 Experiment 1: Discount factor

This experiment aims at providing an understanding of the influence that the discount factor has over the efficiency of the agent.

2.1.1 Setting

To analyse the role of discount factor 5 measurements of the average reward (reward per step) for each discount factor have been averaged. The measurements have been made at 6000 steps. This average is shown in Figure 1, note that most measurements are made with discount factors close to 1.

2.1.2 Observations

The result shown in Figure 1 indicates that at low discount factors will be prejudicial to the efficiency of the agent. However, after 0.75 a plateau can be observed and increasing the discount factor does not improve the result. It should also be noted that changing the discount factor could only improve the efficiency by a couple of percentage (3.1% between the two maxima). This seems to indicate that the algorithm implemented is only marginally improving compared to the greedy algorithm (discount factor of 0).

2.2 Experiment 2: Comparisons with dummy agents

The reactive agent will now be compared to two dummy agents.

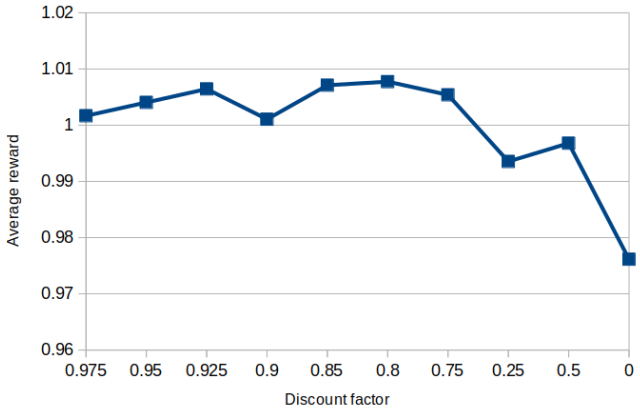

Figure 1: Normalized average reward per step

2.2.1 Setting

The comparison will be made between the worst case of the reactive agent in function of the discount factor (0) and two dummy classifiers. To see if the worst case is still more efficient. The two dummies are implemented as follows:

1. If a task is available randomly chose to pick it up. If no task taken randomly move to a connected city.
2. If there is no task randomly move to a connected city, otherwise pick the task.

2.2.2 Observations

The results are again made by averaging 5 average reward (reward per step). The result is the following.

- Worst reactive: 37844
- Dummy 1 (pure random): 31212
- Dummy 2 (always pick up an available task): 36135

This shows that the reactive agents perform, even in the worst case, better that the dummy agents. Even though the simple fact that an agent who will always take an available task can make it relatively close (4.5 % of difference) to our greedy agent.