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

Group №24: Lucas Burget, Lucas Waelti

October 9, 2018

1 Problem Representation

1.1 Representation Description

The state space S is composed of two sub-states s_0 (no task available) or s_1 (task available) per city id, which yields to $2 \times N_{cities}$ different states. Since Logist takes care of the routing and the delivery of the task once it was decided to pick it up, we don't need to include a carrying state in our model. Indeed, we can only influence the agent once it is free of a task.

$$S = \{(id = 0, s_0), (id = 0, s_1), (id = 1, s_0), (id = 1, s_1), ...\}$$

We define actions that an agent can undertake in each city:

- a_0^n: Move to neighbour city n.
- a_1^d: Pickup the presented task in town i and deliver it to its destination d.

We have 3 valuable pieces of information about the environment:

- $p(i)$: The probability that a task exists in city i.
- $r(i, j)$: The reward perceived for transporting the task from city i to j.
- $p(i, j)$: The probability of having a task from city i to j.

State transition probabilities Knowing that we are in city i, in sub-state s_0^i or s_1^i, the next states can only be s_0^j and s_1^j in city j. So the state transition probability only depends on the next city j and the presence of the task $p(j)$ in this city. We express the state transition probabilities as follow:

	$p(s_0^j)$	$p(s_1^j)$
(s_0^i, a_0^n)	$1 - p(j)$	$p(j)$
(s_1^i, a_0^j)	$1 - p(j)$	$p(j)$
(s_1^i, a_1^j)	$1 - p(j)$	$p(j)$

Rewards of state action pairs The reward is computed in two possible ways, depending on the action that is taken. Given the fact that the agent is in city i and considering moving to neighbour n or destination d (carrying a task):

- a_0^n: reward = $cost(i, n)$
- a_1^d: reward = $r(i, d) - cost(i, d)$

with $cost(i, j)$ the cost of travel defined by the cost per kilometer of the vehicle and the distance between cities i and j.

1.2 Implementation Details

Value Function Iteration Therefore the structure of the *Value Iteration Algorithm* takes the following form, with stopping criterion (note: $V(s)$ encapsulates the city id and the sub-states):

- do
- $error_{max} \leftarrow 0$, $error \leftarrow 0$
- for $c \in Cities$ do
- for $s \in [s_0, s_1]$ do
- $Q_{max} \leftarrow 0$
- for $a \in A$ do
- if $a = a_0$ do
- for $n \in Neighbours$ do
- $Q(s, a_0) \longleftarrow R(s, a_0) + \gamma \sum_{s' \in [s_0, s_1]} T(s, a_0, s') \cdot V(s')$
- if $Q(s, a_0) > Q_{max}$ do
- $Q_{max} \leftarrow Q(s, a_0)$
- else if $a = a_1$ and $s = s_1$ do
- for $d \in DestinationCities$ do
- $Q(s, a_1) \longleftarrow R(s, a_1) + \gamma \sum_{s' \in [s_0, s_1]} T(s, a_1, s') \cdot V(s')$
- if $Q(s, a_1) > Q_{max}$ do
- $Q_{max} \leftarrow Q(s, a_1)$
- $error.insert(V(s) - Q_{max})$
- $V(s) \leftarrow Q_{max}$
- $error_{max} \leftarrow max(|error|)$
- while $|error_{max}| > tolerance$

Selecting the best action During the simulation, we choose the best possible action by maximising the following expression:

$$BestAction \longleftarrow \underset{a}{\operatorname{argmax}} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V(s') \right)$$

where a is picked among all possible actions at the current time. $R(s, a)$ is the actual reward and not the expected one, as used for the value iteration.

2 Results

2.1 Experiment 1: Discount factor

2.1.1 Setting

The Figure 1 shows the influence of the discount factor γ. To test the effect on a given topology, we let the agent accumulate reward with $\gamma = [0.1, 0.25, 0.5, 0.75, 0.85, 0.99]$ over 500 actions. The graph displays the normalised accumulated reward (within $[0, 1]$) for each value of γ, given a topology. For the Netherlands, we made a second test with a new seed to see how the agent would perform with a new task distribution.

We used the default settings provided in the skeleton of the project otherwise.

2.1.2 Observations

From the 4 tests we lead, we could extract an average and interpolate it to show the general influence of the discount factor. It seems there is a sweet spot around 0.85, where the reward tends to be maximal and is otherwise lower for bigger or smaller values of γ. For bigger values of γ, the agent gives too much

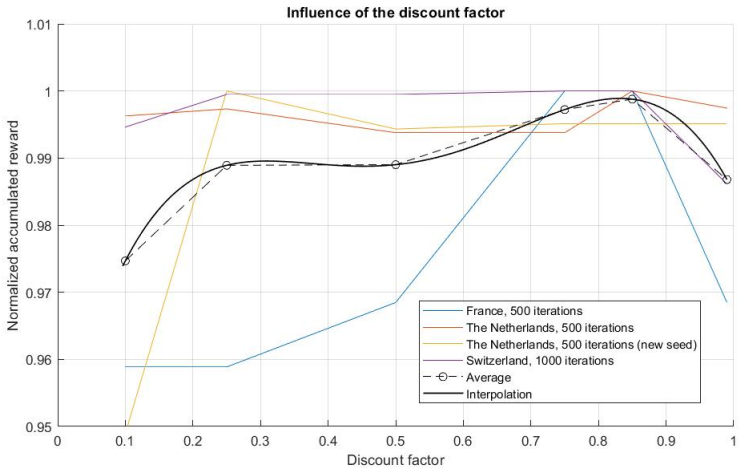

Figure 1: Performance of the agent depending on its discount factor

credits to its future predictions, while with lower values of γ, the agent might neglect some valuable information from the Value function.

2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

To test the efficiency of our *reactive* agent which plans rather in the long run, we created a *short-term* agent to compare them. Our dummy agent only takes into account its immediate and next rewards to decide what to do. If a task is available where it is, it will take it. Otherwise, it will move to the closest city to minimise its cost of travel. We tested the three different agents (*Reactive, Short-term and Random*) one after the other on different topologies and probability distributions.

2.2.2 Observations

We noticed that the *random* agent behaved poorly in all the cases, as expected. However, the *short-term* agent performed almost as well as a *reactive* one with a low *discount factor*. It can be explained because the latter doesn't really take $V(s)$ (the policy learned offline) into account to choose which action to do, so as the *short-term*.

2.3 Experiment 3

2.3.1 Setting

Our environment parameters (*probability distribution of tasks*) are static between the learning phase and the acting phase. To test the limit of our offline learning strategy, we decided to let play more than one agent at a time and let their respective vehicles carry tasks through the topology.

2.3.2 Observations

We observed that the ranking of the agents, relative to their respective accumulated reward, may vary to a point where the agent that selects random actions will outperform an agent with a learned policy! This is due to the fact that other agents will remove tasks from cities and therefore modify the task distribution probabilities, hence influence the environment, which is not supported by a *reactive* agent.