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

Group №:44 Valentin Kindschi, Cyril van Schreven

December 2, 2019

1 Problem Representation

1.1 Representation Description

We define N as the total number of cities.

A *state* s is defined by the city the agent is at, and the task that it is offered. There is one distinct task per destination city, plus there is the possibility that there are no tasks available. There are thus $N + 1$ states per city, and $N * (N + 1)$ states in total.

An *action* a is a movement to another city. It is defined by a destination, and the carrying of a package or not.

The reward table $R(s, a)$ has a reward value for each combination of state and action. It accounts for the cost of travelling, and for the (optional) reward of a task.

Formally, the transition probability table $T(s, a, s')$ has a value for each combination of state, action, and next state. In our problem the transition is a movement to a new city. The transition is only non-deterministic because different tasks (or none) can be present at that city. The original state is not relevant. Also, the probability transition table has a distinct value for each destination city (action) and each possible state at that city, thus $N * (N + 1)$ values.

1.2 Implementation Details

First, note that even if there can be tasks of varying rewards going from origin city to another, this is not taken into account. These tasks are considered the same, and the average reward is used.

A state is implemented as a class *ReactiveState* with three objects:

– the current city, a *City* object as defined in the logist.topology package.

– the presence of a task, a boolean.

– the destination city, a *City* object as defined in the logist.topology package.

If there is no task available, the destination city is per definition null.

An action is implemented as a class *ReactiveAction* with three objects:

– the destination city, a *City* object as defined in the logist.topology package.

– the use of a task, a boolean.

– the reward of the action, a double.

Note that a different *ReactiveAction* instance is created for each state the action originates from. $N * (N+1)$ instances are created. It is therefore possible for it to hold the reward value because it will depend on the departing and the destination cities.

Finally, some map objects are created to link the different states, actions and cities: *statesInCity* links each city to its own list of states, *possibleActions* links each city to its possible actions and *strategy* links each state to the optimal action determined by the reinforcement learning algorithm.

The action reward is equal to $taskReward - cost$. The reward for a task is defined in the *TaskDistribution* class in the logist.task package. The cost of a travel can be computed as the $costPerKm * distanceTravelled$. If an action is performed without task, the value of that action in the reward table will always be negative. Also, the reward table is not explicitly created. As mentioned above, the reward is saved within each instance of the action class.

The transition probability table is technically already implemented in the *TaskDistribution* class in the logist.task package. Indeed, the action defines the transition to a new city. From that city, the task distribution to other cities defines the probability of each state to occur.

2 Results

2.1 Default setup

The setup used for the experiments is the following:

– The default settings: *settings-default.xml*

– The topology of France: *france.xml*

– A uniformly drawn probability distribution of task availability between 0 and 1.

– A constant reward distribution with a policy giving higher values to short distances, between 10000 and 99999.

– A uniformly drawn probability of there being no task between 0 and 1.

2.2 Experiment 1: Discount factor

2.2.1 Setting

All the parameters are fixed except the discount factor.

2.2.2 Observations

As one can see on Figure (1) below, there is almost no noticeable difference between the curves, excpet for the beginning, where the smallest discount factor takes more time to reach its steady state. This is surprising as the theory would predict that the agents with a higher discount factor look further in the potential future states and achieves better results on the long term. A possible explanation for this observation would be that the optimal strategy is rather greedy.

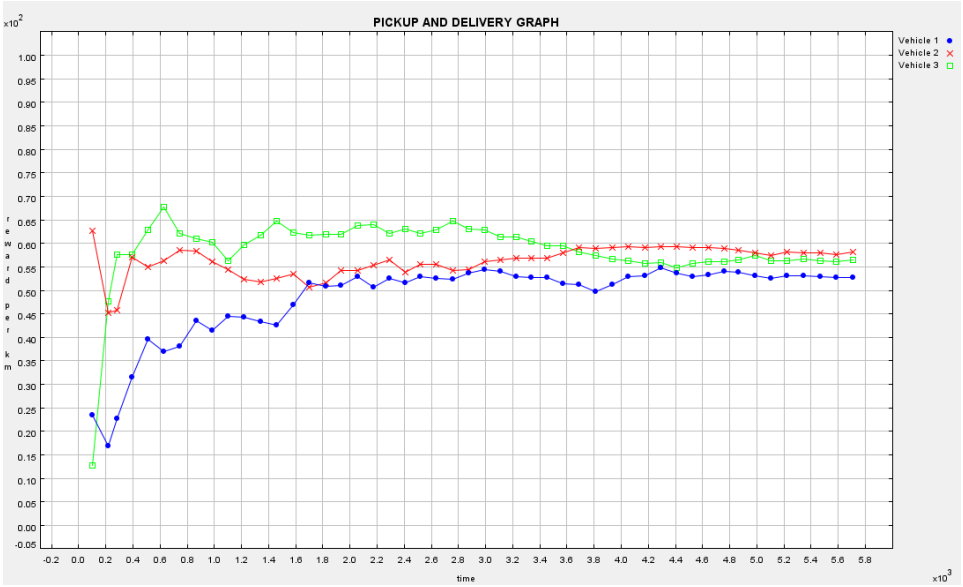

Figure 1: Results with different discount rates (blue: 0.5, red: 0.95, green: 0.999)

2.3 Experiment 2: Comparisons with dummy agents

2.3.1 Setting

The reactive agent is compared with two dummy agents. The random agent and the neighbor agent. The neighbor agent accepts tasks if and only if they are to be deliver in a neighboring city.

2.3.2 Observations

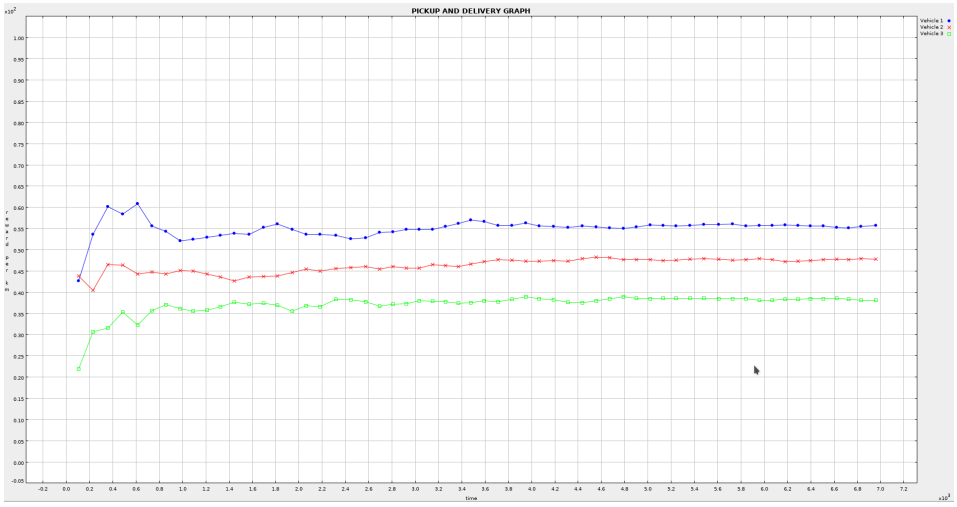

Figure 2: Results with different agents (blue: reactive (rla), red: random, green: neighbor)

The results shown on Figure (2) are clear. The reactive agent based on the reinforcement learning algorithm is the most cost efficient. The random agent comes second. A purely random agent would come lower but this agent takes the task 95/100 of the time. The neighbor agent comes last. This makes sense as it is an arbitrary rule and the agent refuses many potential rewards.