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

Group №: 14 : Launay Clement, Student 2

October 10, 2017

1 Problem Representation

1.1 Representation Description

Given the design of the environment our agent will evolve in, it is possible to classify the environmental features into two groups. First are those which are fixed and given during the *setup* method: the topography with a number $numCity$ of cities and the distance between them, the distribution of tasks and their reward. Second and more interesting are the parameters given when the agent have to make a decision due to *act*. These parameters are the current city the agent is in and the destination and reward of a task if available where the reward can already be deduce as a function $taskReward(currentCity, depositCity)$. Based on this observation, the state representation chosen for our agent is $S = \{currentCity, deliveryCity\}$ where $currentCity$ is any reachable city and $deliveryCity$ is also the set of cities in the topology but to which is added $NOTASK$, which represents the unavailability of tasks when arriving in a city (so $\{deliveryCity\} = \{city\} \cup \{NOTASK\}$). This means that we account for $numCity * (numCity + 1)$ states in our representation. With that state representation, we can define the set of possible actions: in any given state $s = \{currentCity, deliveryCity\}$ we can either move to a neighboring city of $currentCity$ or $PICKUP$ the task if $deliveryCity \neq NOTASK$.
So the for a given $s = \{currentCity, deliveryCity\}$, $A(s) = \{currentCity.neighbors()\}$ if $deliveryCity = NOTASK$, else $A(s) = \{currentCity.neighbors()\} \cup \{PICKUP\}$.

We will apply the algorithm seen in class which consists in iterating on the state values until convergence. We loop other these values and updates them at each step using the formula $V(s) = max_a(R(s,a) + \gamma \sum'_s T(s,a,s')V(s')) = max_a Q(s,a)$ where $Q(s,a)$ is the intermediate function which keep track of the expected overall profit for doing action a in state s. Once convergence is met we extract our strategy: $\forall s, strategy(s) = max_a Q(s,a)$
In order to apply this formula, we need to define:

- $R(s,a)$ the reward function for doing action a in state s. In our case we have, we either pickup the task or move to another city, hence:
  $R(\{currentCity, deliveryCity\}, PICKUP) = expectedReward(currentCity, deliveryCity)$
  $- cost(currentCity, deliveryCity)$
  $R(\{currentCity, deliveryCity\}, neighborCity) = -cost(currentCity, neighborCity)$
  where $expectedReward(city1, city2)$ and $cost(city1, city2)$ comes directly from our knowledge of the environment.

- $T(s,a,s')$, the transition probability function from state s to state s' if we do action a. In our case, given the chosen state representation, we notice that from a state $s = \{currentCity, deliveryCity\}$, the only reachable states are $\{neighborCity, deliveryCity'\}$ (where $neighborCity$ is a neighbor of $currentCity$ if we choose the action to move ($a = neighborCity$), or $\{deliveryCity, deliveryCity'\}$ if we picked up the task ($a = neighborCity$), since the agent will then find itself if the $deliveryCity$ of the task.

From these elements, we have simply:
$T(\{currentCity, deliveryCity\}, action, \{currentCity', deliveryCity'\})$
$= T(\{currentCity, deliveryCity\}, action, \{action, deliveryCity'\})$
$= probabilityForTask(action, deliveryCity')$ with $action \in \{currentCity.neighbors()\} \cup \{PICKUP\}$
and in that later case
$probabilityForTask(PICKUP, deliveryCity') = probabilityForTask(deliveryCity, deliveryCity')$

To sum up, the agent always knows at each step in which city it will find itself after it chooses an action, the uncertain part is the delivery city for the task, which is given by knowing the environment probabilistic distribution for tasks between cities.
With these points made clear, we can run the algorithm until the state values have converged, giving us for each state the expected overall profit of the available actions ($Q(s, a)$). We use this to find out what is the best action for each state.

1.2 Implementation Details

The implementation relies heavily on the fact that the topology contains $numCity$ whose id range from 0 to $numCity - 1$. This allows to use arrays for most of the data to store, using $city.id$ as the index and permits to assign $NOTASK$ or $PICKUP$ the value $numCity$.

- Typically, the probabilities for tasks are stored in a $numCity * (numCity + 1)$ array of doubles called $probabilityForTask$, and $probabilityForTask[0][NOTASK] = probabilityForTask[0][numCity]$ is the probability there are no task when arriving in the city 0, while $probabilityForTask[0][1]$ is the probability for a task from the city 0 to the city 1.

- The value of state function $V(s)$ that indicates the expected profit that can be made from state s is also stored and updated in a similar array $numcity * (numCity + 1)$ because it matches our state representation.

- Finally, the table giving the expected overall profit by doing action a in state s $Q(s, a)$ is stored in an array (representing) the states) of Hashmaps(Integer,Double) where the integer represents the action code (which is simply 0:move to city with id 0, ... numCity=PICKUP:pickup the task), and the double is the profit. So $q[0][1].get(2)$ is the expected profit that would be made if the agent is in city 0, is proposed a task to deposit in city 1 and choose to move to city 2.

The *setup* method is used to apply the algorithm of value iteration described in slide 26 of the course. The iterations are stopped when after a cycle that transformed $V(s)$ in $V'(s)$, $\forall s, |V(s) - V'(s)| < \epsilon$, with epsilon a small value indexed on the cost to travel (which seems an appropriate measure). We then extract $strategy(s)$ as described in the analysis.
The *act* method then only consists in reading the current state s and doing action $strategy(s)$.

2 Results

2.1 Experiment 1: Discount factor

2.1.1 Setting

This experiment features 5 ReactiveRLA agent with different discount (.95,.75,.50,.25,.10).

topology:the_netherlands; all agents' home is Amsterdam; rngSeed:35904357424152424; probability:uniform, long-distances, min=0.0, max=0.9; reward:uniform, long-distances, min=1000, max=5000; no-task:uniform, min=0.1, max=0.2

2.1.2 Observations

The settings were chosen such that the agents' strategy varies (because their value state does not converge at the same speed, hence different results). A general observation is that strategies still looks

a lot alike, the main difference being the city they choose to go when no task is available. (e.g.: in city 3 agent 0.1 goes to city 5, .25 goes to 0 while the other goes to city 8). Running the simulation confirms the expected intuition that agents with higher discount perform better in the long run because they "planned for it". Note that this may not be that clearly visible with other settings, which is blamed on the randomness of tasks' availability. Below are the profit (in thousands of CHF) made by the agents every 10000 actions.

discount / number of tasks	10000	20000	30000	40000	50000	#iterations
0.1	17403	34904	52510	70112	87407	6
0.25	17656	35345	52925	70663	88339	8
0.5	17759	35408	53303	71031	88771	13
0.75	17707	35336	53091	70917	88798	27
0.95	17614	35515	53459	71350	89194	137

In the experiments ran, the outputted strategy was never modified by further increasing the *discount* after 0.9, while the umber of iterations needed grew fast ($\simeq$ 600 iterations at 0.99). Hence the default 0.95 seems like a safe bet.

2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

The 3 agents deployed in this experiment are reactive-random, our reactive-rla with discount=0.95 and a custom dummy agent reactive-ins. ReactiveInstant is a greedy agent that use a table of the average reward proposed in each city to make the best action at each step. It is not truly reactive as it only pickups a task if it is higher than the average for that city (or at least better than what he could expect in the neighboring cities).

topology:switzerland; agents start in different cities; rngSeed=7770420242192152424; probability:constant, short-distances,min=0.1, max=0.9; reward:constant, long-distances, min=1000, max=99999; no-task:constant, value=0.3

2.2.2 Observations

Without surprise, the reactive agent is the best performer of the three. It is (fortunately) clearly better than the random agent, typically making 20% more profit in a standard case like this one. Our dummy agent ReactiveInstant also performs better than the random one (
simeq 10% more profit) and can sometimes be close to the RLA in some edge cases (notably if all properties are constant).

2.3 Experiment 3: Edge case

2.3.1 Setting

This experiment is similar to a test case: the no-task distribution is very high (0.9)

topology:england; agents start in different cities; rngSeed=35904202421921524 72; probability:constant, value=1; reward:uniform, min=0, max=50000; no-task:constant, value=0.9

2.3.2 Observations

As planned, the "aware" agents will go to a specific edge (the most interesting one) and keep on doing two-ways trips until they get a task, deliver it and return to start again. ReactiveInstant performs quite well in these settings, even better than a RLA with low discount ($\preceq$ 30) that can get caught in another edge instead of always returning to the most profitable one.