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

Group 39: Hugo Bonnome, Pedro Amorim

October 11, 2016

1 Problem Representation

1.1 Representation Description

The topology T is graph defined by $T = \{C, P\}$ where C is the set of cities in the topology and P the set of paths connecting these cities.

1.1.1 State representation

The state s of a given agent is defined by $s = \{c, t_d, N_c\}$ where $c \in C$ is the city where the agent currently is, $t_d \in C \cup \{None\}$ indicates whether there is a task to city d in c (being equal to $None$ when no task is available) and $N_c \subseteq C$ is the set of cities that can be reached from c, in other words the neighbours of c.

1.1.2 Actions

The agent can:

- Move towards a neighbour n, this will be denoted $M(n)$
- Pickup a task in the current city and deliver it to the destination city, this will be denoted $D(t_d)$. We assume that the agent never attempts the pickup action if there is no task available in its current city.

1.1.3 Reward

For the action of moving to a neighbour:

$$R(\{c, t_d, N_c\}, M(n)) = -dist(c, n)$$

where $n \in N_c$ and $dist(c, n)$ is the shortest path distance between c and n. This value can be justified by the fact every km that we travel without a profit implies a loss.

For the action of picking up a task and delivering it:

$$R(\{c, t_d, N_c\}, D(t_d)) = AR(c, d)\frac{1}{dist(c, d)}$$

with $AR(c, d)$ being the average reward from delivering a task from city c to city d which is weighted by the distance between both cities.

1.1.4 Probability transition table

The uncertainty in the world state only comes from the presence of a task in a given city or not. It does not depend on the type of action taken by the agent.

$$p(\{c, t, N_c\}, (M(n)|P(n)), \{n, t_d, N_n\}) = P(n, d)$$
$$p(\{c, t, N_c\}, (M(n)|P(n)), \{n, None, N_n\}) = probNoTask(n)$$

where $P(n, d)$ is the probability of there being a task in city n whose destination is d and $probNoTask(n)$ is the probability of city n having no task which can be computed by $1 - \Sigma_{c \in C} P(n, c)$.

1.2 Implementation Details

1.2.1 ActionType

an ENUM with two Plans : MOVE or PICKUP

1.2.2 State

Define a city where the state belongs and a taskDestination that represents a package to deliver (it can be null if there is no task in this city). So the state is represented the both attributes : CITY + TASK.

It implements :

- getAvailableAction() : returns a list of ActionState (an edge to another city with an ActionType), i.e all decisions the agent could make in a given state.
- probability() : returns the probability to be in the state knowing that we are in the city of the state i.e : probability(this.taskDestination — this.city)

1.2.3 ActionEdge

Define an action that can be taken by an agent in a state so it's instantiated with fromState + ActionType(MOVE/PICKUP) + CityDestination.

It implements :

- getImmediateReward() : return the immediate reward associated with the action, calculated as follow :
- MOVE : -this.fromState.getCity().distanceTo(this.getDestination())
- PICKUP : this.taskDistribution.reward(fromState.getCity(), getDestination())/ this.fromState.getCity().distanc

1.2.4 Graph

It generates all possibles states and provides the following functions :

- getStateIt() : iterate over all states
- getStateIt(city) : iterate over all states in a given city

1.2.5 State & Graph

STATE contains two more attributes :

- bestAction : the most rewarded ActionState (initially null)
- evReward : the average expected reward of the best action (initially 0)

GRAPH contains a method :

- expectedRewardIn(city) : returns the sum of each state's evReward multiplied by the state probability in the given city i.e : what will be the average evReward if I go to this city and if I act perfectly in this city.

STATE contains a method :

- update(discount, graph) : update evReward and bestAction by reevaluate all their available actions. for each availableAction take max : reward = availableAction.getImmediateReward() + discount*graph.expectedRewardIn(availableAction.getDestination()); i.e : immediate reward of the action + discount*weighted sum of state.reward

GRAPH contains a method :

- update(discount) : call for each state in the graph state.update(discount)

To make each evReward/bestAction converge to their respective values we just have to call graph.update(discount) until the differences between each update are lower than a preset threshold.

2 Results

2.1 Experiment 1: Discount factor

Discount factor allow MDP to weight the future reward, the closer the discount factor is to one, the more efficient MDP will be in gaining long-term expected value. A priori, we think that the discount factor in this setting will not have a strong influence on the average reward, because the rules of the world are simplistic enough to not require long-term consideration of future states.

2.1.1 Setting

We ran two experiments, the first one with a discount factor equal to 0.85 and the second one at 0.15.

2.1.2 Observations

- REACTIVE df : 0.85 : The total profit after 16386 actions is 613281328 (average profit: 37427.152935432685)
- REACTIVE df : 0.15 : The total profit after 16744 actions is 625253670 (average profit: 37341.95353559484)

As expected the discount factor does not produce substantial variation on the average reward, so we have 37341 for a discount factor at 0.15 vs 37427 for a discount factor at 0.85 : it only makes about 0.2% improvement. However it should be a more relevant parameter in more complex games like chess when one action could provide the best short-term reward but lead to a dead-end further down the road.

2.2 Experiment 2: Comparisons with the given dummy agent

2.2.1 Setting

Let's test our agent against the dummy one that follows a basic strategy :

- Move to a random neighbor if there is no task available or with 15% probability if a task is provided.
- Deliver the package otherwise.

Reactive agent : MDP based action. Dummy agent : Random agent given in the skeleton.

2.2.2 Observations

After about 19'000 actions we have the following result:

- DUMMY : The total profit after 18843 actions is 593389062 (average profit: 31491.220187868174)
- REACTIVE : The total profit after 19769 actions is 738573948 (average profit: 37360.207800091055)

The reactive agent has an average reward that is about 18% higher in comparison to the dummy agent.

2.3 Experiment 3: Comparisons with modified dummy agent

2.3.1 Setting

An intuitive strategy would be to always deliver a package when the state allows it. This is the reason we modified the dummy agent to always accept a task when available and compared the result with the MDP reactive agent.

Reactive agent : MDP based action. Dummy agent : pPickup was increased to 1.0 (random movement when no task is available)

2.3.2 Observations

After about 17'000 actions we have the following results:

- DUMMY : The total profit after 16522 actions is 612041360 (average profit: 37044.023725941166)
- REACTIVE : The total profit after 18550 actions is 696576186 (average profit: 37551.276873315364)

The reactive agent has an average reward that is about 1% higher in comparison to the dummy agent

This small difference comes from the fact that all cities have the same uniform distribution of tasks and as such delivering a package is almost always the best action to take.

The slight difference can be explained by the fact that our reactive agent improved his score by choosing a better action when no work is offered in a city. In this case it will choose the best trade of between taking the closest city (low immediate cost) or the most central city. (best reward in the future).