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

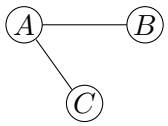
Group №66: Amaury Combes, Tho Nikles

October 9, 2018

1 Problem Representation

1.1 Representation Description

In our model, a state is either a city, or a tuple of city. A tuple is of the form (A, B) where A and B are cities and means that we are in state A, that proposes a task for city B. The possible actions are to move to a neighbour city, or to deliver a task from the left to the right member in a tuple. Note that it is not possible to move in the same city, and that there is no tuple of the form (A, A). Suppose we have a simple model with 3 cities, namely A, B and C, with the following connections:



The transition graph will be the following

Transition	A	B	C	(A,B)	(A,C)	(B,A)	(B,C)	(C,A)	(C,B)
Move A	*None*	$\{A,(A,B),(A,C)\}$	$\{A,(A,B),(A,C)\}$	*None*	*None*	$\{A,(A,B),(A,C)\}$	$\{A,(A,B),(A,C)\}$	$\{A,(A,B),(A,C)\}$	$\{A,(A,B),(A,C)\}$
Move B	$\{B,(B,A),(B,C)\}$	*None*	*None*	$\{B,(B,A),(B,C)\}$	$\{B,(B,A),(B,C)\}$	*None*	*None*	*None*	*None*
Move C	$\{C,(C,A),(C,B)\}$	*None*	*None*	$\{C,(C,A),(C,B)\}$	$\{C,(C,A),(C,B)\}$	*None*	*None*	*None*	*None*
Deliver	*None*	*None*	*None*	$\{B,(B,A),(B,C)\}$	$\{C,(C,A),(C,B)\}$	$\{A,(A,B),(A,C)\}$	$\{C,(C,A),(C,B)\}$	$\{A,(A,B),(A,C)\}$	$\{B,(B,A),(B,C)\}$

The *None* in the table are the action that are illegal in our model. For the rest of this document, we will ignore them, since the program takes care to make these transitions unfeasible. We see that the table can be (and is) separated in four regions. These will be useful to define the reward table. We will use the cardinal points to refer to each quarter (N, S, E, W). Let's begin with the SW one, since it is the easiest. We cannot deliver something in a city from a city that offers no task, so these rewards are not defined. Next, let's see the NW and NE ones. Since we simply move, without delivering anything to the next town. The reward is negative and simply corresponds to the cost (in km) of the trip. Let's take a look at the last region, the SE one. Here we are in a town which offers a task for the one in which we move. So the task is completed and we gain the reward (known by looking in the table we receive at the beginning of the program). But moving to the town still has a cost, so we subtract the distance (in kilometres) to the town where the task need to be delivered.

As for the transition probability matrix (T(s, a, s'), we first set all illegal triplet to have probability 0 (illegal triplets are the ones that cannot be found in the transition graph). We then notice that the probability of all legal triplets only depends on the value of s'. Indeed, regardless of the previous state and the action taken, the probability that a task spawns at the incoming city is the same.
This means that there are only n*n values to find for T. We distinguish the case (a) where s' is a city state (i.e. a state that represent a city without tasks) and (b) where s' is a task state (i.e. a state that represent a city with a task to deliver to a specific location).

In the following the function p corresponds to the function given in the assignment description. emptyS is the set of states that represent a city without tasks. taskS is the set of states that represent a city with a particular task; those states have to parameters (fromCity, toCity) corresponding respectively to the actual location and the delivery location
(a) T(s, a, s') = p(s', null); for all s in S, a in A, s' in emptyS and (s, a, s') in the transition graph
(b) T(s, a, s') = p(s'.fromCity, s'.toCity); for all s in S, a in A, s' in taskS and (s, a, s') in the transition graph

1.2 Implementation Details

We opted for a highly OO approach to program this assignement. States are modeled accordingly by the superclass State and subclasses EmptyState and TaskState. The same holds for the classes StateActions, MoveAction and DeliverAction that models the actions an agent can take. In order to find our strategy, we used the value iteration algorithm which is implemented in the ValueIteration class. Then, we link our strategy with the logist interface through the act method in the ReactiveTemplate class.

2 Results

2.1 Experiment 1: Discount factor

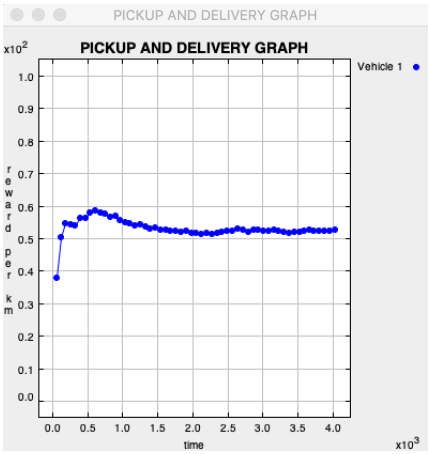
We will use this experience to see how the discount factor affect the simulation for different values of this variable.

2.1.1 Setting

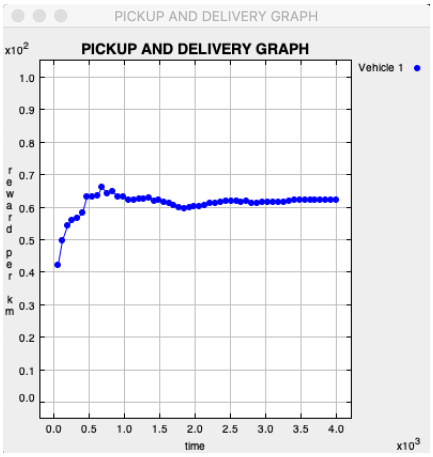
For this experiment, we make the discount factor vary and take the values 1e-30, 1e-10 and 0.99999. We will comment on how these different values affect the results.

2.1.2 Observations

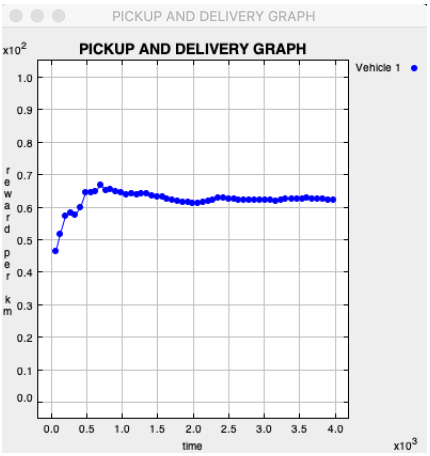
From our experiments we notice two main phenomena. First, as we increase the value of the discount factor, the number of iterations required for the value iteration algorithm converges is higher. Secondly, the higher the discount factor, the higher the reward per km will be after letting the simulation converge.



value of 0.001



value of 0.2



value of 0.999

2.2 Experiment 2: Comparisons with dummy agents

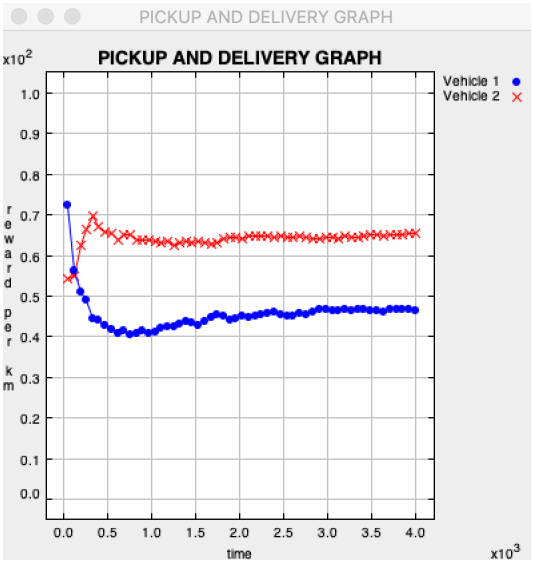
In this experiment, we will see how our agent compares to the random agents already implemented in the program version that was provided. In expectation, ours should do better since they had a learning phase and make thoughtful decision. we will now check whether those expectations are met or not, and try to explain why it is this way.

2.2.1 Setting

We use the following settings, 0.95 as discount factor, the default topology and two agents, one random, one clever.

2.2.2 Observations

As expected, our agent (in red), has globally a higher profit per km than the other agent (in blue). The differences can be explained by the fact that our agent uses some kind of reasoning, knowing what move to do in order to maximize its profit (based on probabilities), while the random agent moves without any reasoning. We also see that their profit tend to stabilize at a certain point, after which there are only small deviation, due to what they get as rewards.



Battle between a random (blue) and a clever (red) agent