Intelligent agents

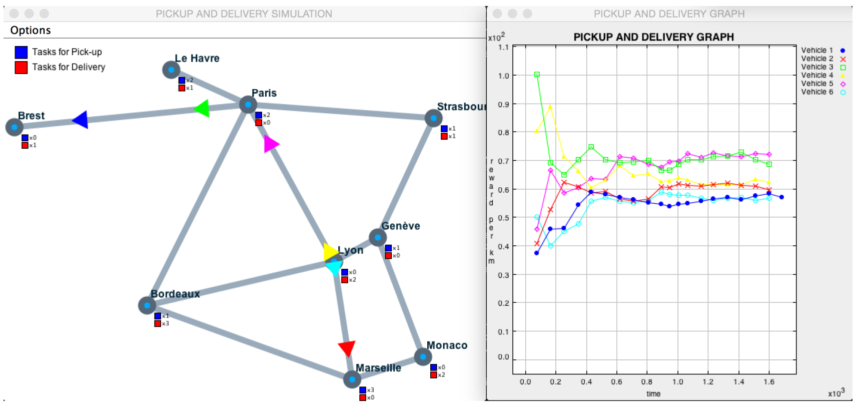
CS-430

Lab2: Reactive Agents

Submitted By Group 36:

Christopher Salotti & Marco Antognini

October 2015



Introduction

In this lab, we use reactive agent to solve the *Pickup and Delivery problem*. We implemented the Reinforcement Learning Algorithm (RLA) with a known task probability distribution. The goal is to find the optimal task to take and deliver by taking in account the task probability distribution, the rewards that we can get, the topology, the state in which we currently are and the next state we will be in after performing an action. By using the state action transition and reward tables, we compute the optimal action in the given topology.

Simulation Parameters

A clever and very important step is to define the possible states of our agent and the actions that it can execute.

State and Actions

We define a state with two variables:

City c The current city we are in. c ranges over the cities of the topology. We denote by C this range.

Task t The destination of the currently available task. Like c, t ranges over the cities of the topology but it can also be *null* when no task is available. We denote by T this range.

We define our set of **states** as $S = C \times T$.

In order to have the simplest agent model, we defined only one kind of action:

Move to city c The move action simplify the decision of *taking or not* the task. If the action is to move to the task's destination, then we choose to take the task with us. Otherwise, we ignore it and move to a neighboring city.

However, due to the probabilistic distribution of tasks, performing an action on a given state doesn't result in a deterministic new state.

Given a topology of N cities, we have $N + 1$ possible states per city, $|S| = O(N^2)$ and each of the N actions can result in any of the state in S. As we can see, it become quite large with respect to the number of cites. However, as we will highlight below, many states are invalid and in practice the number of action available to an agent in city c is proportional to the number of neighboring cities. Therefore, using a sparse representation of the reward table can significantly reduce the memory cost of this algorithm.

From the assignment statement the following states and actions are illegal:

- $c_i \times t_i$: A task can only be pickup from a city i when $i \neq j$.
- Move from c_i to c_i: The agent cannot stay at the same position.
- Move from c_i to c_j without a task if c_i and c_j are not directly connected: The agent can only move to neighboring cities if it is not transporting a task.
- Move from c_i to c_j while carrying a task with destination $c_k \neq c_j$: When transporting a task, the agent must go to the task destination without stopping in another city on the way.

We chose to represent our state action matrix as a graph representing the probabilistic transition from a state to the next one when performing an action.

Implementation

We can split our implementation in two principal sections: the **setup** and the **act**.

Setup

The setup is the first step of our simulation that computes and stores the best decisions for the agent when it is in a state s_i. We will only explain some key parts that can differ from the lab description.

State Our `State` class keeps track of the city-task pair and can be used in Java associative containers thanks to the standard `hashCode` and `equals` methods.

Potentials Inside our `ReactiveTemplate`, a.k.a. the agent, we store the potentials of each state using a `HashMap<State, Double>`. We found it more elegant and convenient to use with sparse `State`s than an array of `Double` since we don't have to explicitly define a one-to-one mapping from state to indexes.

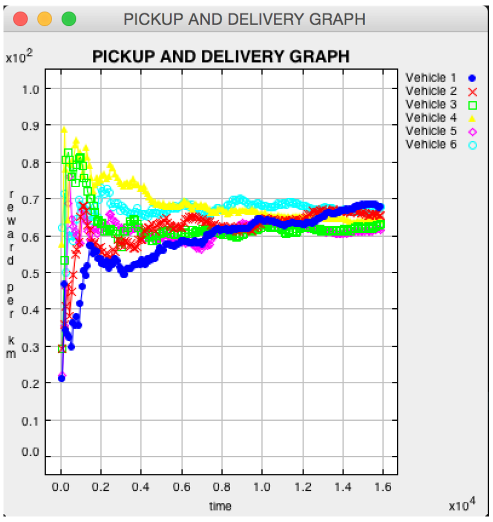
Actions When learning offline about the potential of each state, if we update the potential of a given state we also update its associated best action in a dedicated `HashMap<State, City>`.

Act

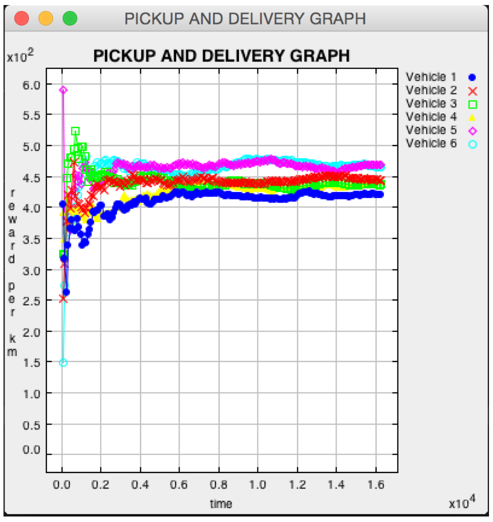
The act is quite simple and short compared to the setup. We can divide into very few steps.

1. We select the best action for our current state.
2. If there is a task available:
    - If our action is the task destination, then we take the task as well.
    - Otherwise we simply ignore it.

Results



(a) France Topology



(b) Netherlands Topology

Figure 1: Multiple agent run for $\gamma = 0.2, 0.4, 0.6, 0.8, 0.9, 0.95$. The vehicle order is the same as the γ's.

As we can see on the graphs, the agents seems to converge to nearly the same value (between 60-70 for the France and 410-470 for the Netherlands). We can observe that the gamma coefficient has an influence on the converged value.