Reactive agents

Jeremy Gotteland & Quentin Praz

October 14, 2015

World representation

State representation

Recall that the agent here refers to a single vehicle. We chose to represent the state of an agent as the following:

- Current city: The city in which the agent is currently located.
- Destination city: At any time the agent is a state where it could pick up a task to bring to a city. This variable is this potential city. If there is no task to be pick up, this variable is set to $NULL$.
- Possible moves: A list of cities which the agent can travel to from current state i.e. its neighbor cities and its potential destination city.

Actions

In our state representation, an action is just a move to another city.

Reward

The reward for moving from a state to another city c is defined as:

$$R(s, c) = \mathbb{1}_{\{c\ =\ s.destinationCity\}} \cdot AR(s.currentCity, c) - Km(s.currentCity, c) \cdot CostPerKm$$

Probability of transition

The probability of transition to the state s' from the state s by moving to city c is defined as:

$$T(s, c, s') = \begin{cases} P(c, s'.destinationCity) \text{ if } destinationCity \text{ is not } NULL \\ 1 - \sum_{c' \in Cities} P(c, c') \text{ otherwise} \end{cases}$$

Algorithm

Here we describe the algorithm we used

Implementation details

Most of the code was implemented in the class

`ReinforcementLearningModel.java`

which provides static access to compute the best possible actions offline, before the simulation starts.

This class contains the following variables

```
private static Map<Integer, State> states = new HashMap<>();

private static Map<Integer, Map<City, Double>> rewards = new HashMap<>();

private static Map<Integer, Map<City, Integer>> stateTransitions = new HashMap<>()

private static Map<Integer, Map<City, Double>> Q = new HashMap<>();

private static Map<Integer, Double> V = new HashMap<>();

private static Map<Integer, City> best = new HashMap<>();
```

Here we used Integers as identifiers for the 'State' objects, as they are keys in many other maps and having objects as key in maps is a very bad programming practice. The main method, which is called from the 'setup' method from ReactiveTemplate.java is

```
public static void offlineProcessing(Topology t, TaskDistribution td, Vehicle v, do
```

This methods starts by generating all states thanks to the 'generateAllStates(Topology t)', and then generates all the possible actions from each state with the private 'generateAllActions' method. The states and possible actions are described in Section **State representation**

Then the method computes the optimal moves for each states following the algorithm described in class.

The tricky part in the code is the *getIdForState(State s)* method which returns the integer identifier of an event. It iterates through the entire map of states and finds the one for which the currentCity and destinationCity are the same (possible moves are always the same if the other two are matching).

Move selection happens in the **public Action act(Vehicle vehicle, Task availableTask)** method implemented by the *ReactiveTemplate.java* class.

```
@Override
public Action act(Vehicle vehicle, Task
    availableTask) {

        Action action;

        City currentCity = vehicle.getCurrentCity()
            ;
        City potentialDeliveryCity = (availableTask
            == null) ? null : availableTask.
            deliveryCity;

        State currentState = new State(-1,
            currentCity, potentialDeliveryCity);


        City newDestinationCity =
            ReinforcementLearningModel.
            getNextMoveForState(currentState);

        if(availableTask != null &&
            newDestinationCity.equals(availableTask.
            deliveryCity)) {
                System.err.println("Picking up task
                    from [" + currentCity + "] to
                    [" + newDestinationCity + "]") ;
                action = new Pickup(availableTask);
        } else {
                System.err.println("Simply move
                    from [" + currentCity + "] to ["
                    + newDestinationCity + "]");
                action = new Move(
                    newDestinationCity);
        }
```

```
            return action;
        }
```

We reconstruct the state of the agent, and get the next move. If there is a task available, and that the next move corresponds to the destination city for this task, we return a Pickup action to pick it up. Otherwise, we leave the task and move to the city that was computed offline, and that should eventually give a better reward.

Observations

The only parameter we could play on was the discount factor. We tried some very small values and some values close to 1 but we didn't notice major changes. The only thing that was modified was the time of the pre-computation: the convergence was faster with a small discount factor.
One reason could be that networks are quite small and then the futur events don't count that much in the equation.