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

Group №76: Simon Honigmann, Arthur Gassner

October 9, 2018

1 Problem Representation

1.1 Representation Description

Defining the states required considerable forethought, which took the form of a small scale notebook simulation. We first used only 3 cities, arranged in a triangle, to determine what information the agent would require to make an optimal decision. Using this method, we decided to structure the states and actions as follows. The agent's state is defined by it's current city and the destination city of any task (plus one case where there is no available task). To simplify the structure of our state transition table, we included entries for impossible states (e.g. having a task destination be the same city as the current city) and simply assigned a zero probability to these states. For N cities each with N+1 possible destinations (including no destination), this yields (N)x(N+1) possible states. Similarly, each state has a possible action. Again to simplify the logic, we defined one action for each city, that would represent ignoring any available tasks and proceeding to a given city, and one additional action for taking the available task.

Once the structure of the states and actions were defined, we created rules that could be used to generate the state transition probabilities. Based on our small scale test, we defined the following rules:

1. The task destination cannot equal the origin city. This applies to both present state (s) and future state (s').
2. The present origin city cannot equal the future state's origin city (i.e. the agent must change cities every step of the simulation).
3. Regarding tasks:
   (a) If the task is accepted by the agent, the current destination city must equal the future state's origin city.
   (b) In all other cases:
      i. the action cannot equal the current origin city and,
      ii. the future origin city must equal the action city.
4. All matrix entries that satisfy all above conditions are to be populated as follows:
   (a) If the future state has a task available, then the probability is $T(s, a, s') = p(s'_{origin}, s'_{destination})$ where p is the task probability distribution defined in the xml document.
   (b) If the future state has no task available, the probability is $T(s, a, s') = 1 - \sum p(s'_{origin}, s'_i)$ where i indexes all cities.

For all cases that were defined as impossible by the rule set, a zero probability was assigned.

Defining the expected rewards, denoted "Profit" in our code was easy enough. For each of (N)(N+1) states and (N+1) actions, the profit was defined as follows:

- If the agent takes an available task, then Profit equals
  $P(s, a) = r(s_{origin}, s_{destination}) - c(s_{origin}, s_{destination})$

- Else if the agent decides instead to go to an adjacent city, then Profit simply equals: $P(s, a) = -c(s_{origin}, a_{destination})$

Where $P(s, a)$ denotes the profit for a given state, s, and action, a, and $r(origin, destination)$ and $c(origin, destination)$ each represent the respective delivery reward and cost of travel between two cities. Cost of travel was defined as the travel distance multiplied by a constant cost-per-distance. We defined the cost-per-distance to be one for all experiments.

To test our created tables, we manually reviewed the generated matrix, and tested the central and boundary cases for correctness. Checks were repeated for all provided city .xml files to ensure the robustness of the algorithms. Once satisfied with our results, we proceeded to developing an off-line reinforcement learning algorithm for the agents.

1.2 Implementation Details

Despite vehicles only being able to travel to directly adjacent cities, agents are allowed to decide to move to any city if it the decision algorithm indicates it is the best decision. However, when the agent reaches an intermediary city, the agent is still able to re-evaluate the decision based on the actual state reached. For instance, if the intermediary city has an improbable yet valuable task available, the agent could change course and accept the task.

To compute the best action, $Best(s)$ for each possible state, an iterative optimization algorithm was is computed at the start of each simulation. For each starting state, every action was considered and the reward for that action was determined using the previously determined Profit matrix. Then, to consider future possibilities leading from the resulting state, for each possible future state, the state transition probability was multiplied by the max value of the future state, $V(s')$, and discounted by a factor γ. The max possible value, $V(s)$, for all actions value is stored, and then serves as the baseline for the next iteration of calculations. Once the max value converges (due to the discount factor), the best action is set to be the action leading to the max value.

All mentioned tables and matrices were structured as nested ArrayLists so that memory could be dynamically allocated when different .xml files are used. By looping through all possible index combinations and applying the conditions above, the tables could be initialized with values. Again, all possible states and actions were considered and the best action and maximum values were saved in nested ArrayLists for future use.

2 Results

The simulation was tested thoroughly with different conditions to better understand value of using offline reinforcement learning algorithms, and the importance of different discount factors. It is important to note that our collected data displays each agent's profit, divided by their distance travelled, instead of simply profit which was the value optimized in the reinforcement learning algorithm. That being said, the data presented is still expected to be generally representative of the simulation behaviours.

2.1 Experiment 1: Discount factor

2.1.1 Setting

The first experiment investigates the long-term value of different discount factors. This was achieved by running 5 agents with discount factors of 0.05, 0.1, 0.5, 0.85 and 0.99, using the configuration specified by the `E1React.xml` file. Reward per distance travelled *(RPD)* is plotted for each agent in Figure 1.

2.1.2 Observations

As shown in Figure 1, the *RPD* of each agent converges after enough time has elapsed. Another important observation is how the variation of the discount factor influences the *RPD*. The higher the discount factor, the higher the *RPD*. That is because a high discount factor implies taking into account the possible "future" rewards an agent can receive. Lower discount factors instead favour immediate rewards. Short term performance is predominantly based on chance, allowing lower discount factor agents the possibility of performing well. However, as the randomness is averaged out over thousands of actions, the agents with higher discount factors and more foresight prevail.

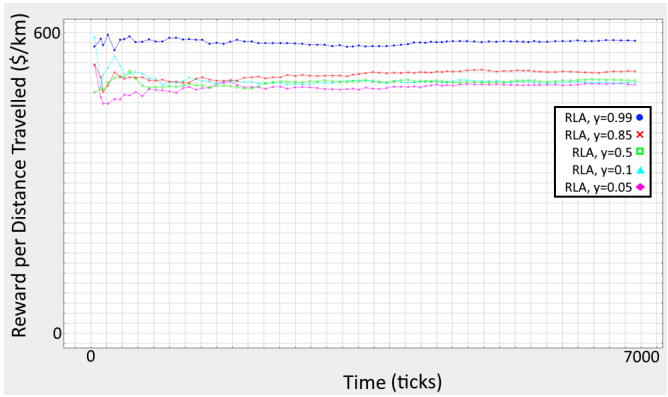

Figure 1: *RPD* for 5 RLA trained agents with different discount factors travelling in the Netherlands

2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

The second experiment investigates the performance difference between RLA trained agents and dummy agents. Dummy agents were simulated using the template code provided. We used the two dummy agents specified in `agents.xml`: `reactive-random`, and `reactive-random-50`. The dummy agents had a 85% and 50% probability of accepting an available task respectively. We compared these dummy agents to an RLA trained agent using a discount factor of 0.99, specified by `reactive-rla-99`.

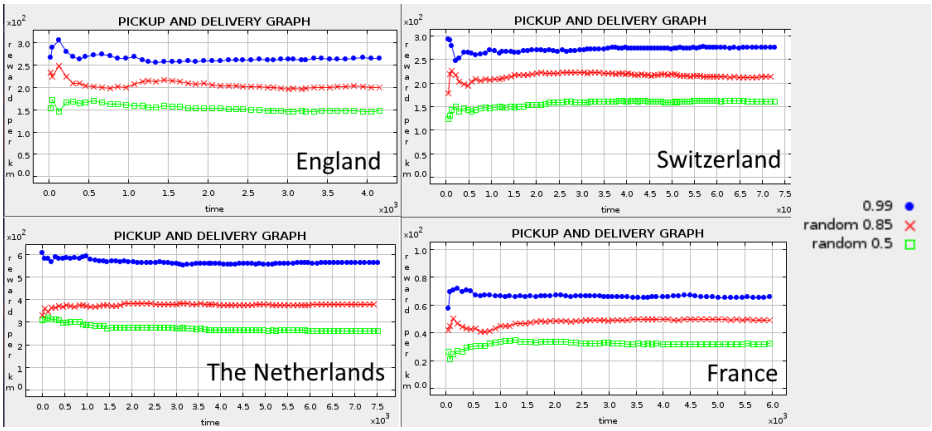

Figure 2: *RPD* of an RLA trained agent (gamma = 0.99) and two dummy agents, across four topologies

We ran each simulation for the four different topologies provided (the Netherlands, England, France and Switzerland).

2.2.2 Observations

We can see that the RLA trained agent performs consistently better than both dummy agents across all four topologies. Comparisons across different topologies help highlight the reliance of early performance on random task generation. As all topologies show a notable performance margin between the RLA and Dummy agents, we can conclude that RLA is an effective strategy to improve decision making when programming intelligent agents.