A Reactive Agent for the Pickup and Delivery Problem

In this exercise, you will learn to use a reactive agent to solve the Pickup and Delivery Problem. For that, you will implement a reinforcement learning algorithm (RLA) to compute an optimal strategy off-line. This strategy is then used by the agent to travel through the network.

We make the following assumption about the reactive agent:

1. The vehicle starts from its home city and can move freely through the network.
2. When the vehicle arrives in a city, it finds out whether or not a task is available in that city.
3. If a task is available, the agent can decide to pick up the task and deliver it, or it can refuse the task and move to another city (without load).
4. If no task is available or if the task was refused, the vehicle moves along a route to a neighboring city.
5. The vehicle can transport only one task at a time.
6. The agent receives a reward for each task that it delivers and pays a cost for each kilometer that it travels. Since the agent tries to maximize its profit, it will always deliver a task on the shortest path.
7. A task which was refused disappears immediately. The next time the agent moves to that city a new task is generated according to the probability distribution.

Applying reinforcement learning (Markov Decision Processes) to learn an optimal strategy

An intelligent reactive agent can improve its performance by learning to optimally respond to the percepts received. Using reinforcement learning the agent can learn to react optimally on the basis of a probability distribution of the tasks in the network. This approach assumes that the probability distribution is known. The learning phase is done offline before the agent travels through the network.

The following steps should therefore be taken:

1. On the basis of a probability distribution of the tasks, learn offline the actions to take in order to handle tasks optimally.

2. Use the created table (the strategy) to travel through the network and to quickly fetch the best response to a given state of the world.

3. As soon as a task has been picked up, deliver it on the shortest path to the destination city. (This is already implemented by the platform.)

At the beginning there are two tables:

$p(i, j)$: The probability that in city i there is a task to be transported to city j.

$r(i, j)$: The reward for a task that is transported from city i to city j.

These tables are created from the task set settings in the configuration file and can be accessed through the `TaskDistribution` object that is given during setup.

At runtime, a task from the current city i to some other city j will be created with probability $p(i, j)$ and will have a reward of $r(i, j)$.

Implementing the Reinforcement Learning Algorithm

In this exercise, you must implement the offline reinforcement learning algorithm. The first thing that you need to do (and this is the most difficult part of this exercise) is to define (on paper!):

- An adequate state representation of the world
- Which actions you can take in each state, and
  - the corresponding state transition,
  - the probability of the transition and
  - the reward of the transition.

Remember that a reactive agent chooses an action depending *only* on its perceived state of the world. So, be very careful and test your representation on paper before you start programming !

Furthermore, you need to define $Best(S)$ and $V(S)$ as the vectors indicating the best action from a state and the corresponding accumulated value. You must learn $V(S)$ by value iteration:

1: **repeat**
2: **for** $s \in S$ **do**
3: **for** $a \in A$ **do**
4: $Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V(s')$
5: **end for**

6: $V(S) \leftarrow max_a Q(s,a)$
7: **end for**
8: **until** good enough

Following this algorithm, you must first define a table for $R(s,a)$ that indicates the rewards for taking action a being in state s. You also must define a table for $T(s,a,s')$ which defines the probability to arrive in state s' given that you are in state s and that you take action a, or in other words: $T(s,a,s') = Pr\{s'|s,a\}$.

You should define the discount factor γ in order to ensure that the algorithm converges. This must be a value between 0 and 1, preferably close to 1.

At each iteration, update $Q(s,a)$ and $V(S)$. The algorithm stops whenever there is no more a change in $V(S)$. When you have learned $V(S)$, the agent can start to move through the topology following the actions indicated in $V(S)$.

Your task

1. Define your state representation s, the possible actions a, the reward table $R(s,a)$ and the probability transition table $T(s,a,s')$ - this is the hardest and most crucial part of the exercise.

2. Implement the offline reinforcement learning algorithm for determining the actions to take in order to search and deliver tasks optimally. This algorithm should be executed before the vehicles start moving.

3. Run simulations of one, two and three agents using your optimally learned strategy $V(S)$. Look at the performance graph of the agents. How does it change for different discount factors ? Explain your results.

Implementation hints

- There is a seperate document that elaborates the relevant parts of the *LogistPlatform* for this exercise.
- Do note that there is a continuous supply of tasks and that the simulation will run forever. Use the exit button from the *Repast* UI to shutdown the platform cleanly.
- The prepared package comes with various configuration files that allow you to test your solution. Feel free to change them and see how it affects your program. When you hand-in your solution please make sure that you correctly set your discount factor and any other user-defined values (if you use them).
- The task distribution can be given a seed in order to make the task generation deterministic. This is invaluable when doing code debugging, but please test your program with other seeds, too. If your program has just crashed look at the *history.xml* file - it contains a detailed history of events. If you reuse the seed values

provided there (by changing the configuration file) you should be able to repeat the crash.