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

Group №: 90 Kyle Gerard, Yann Bolliger

October 8, 2018

1 Problem Representation

1.1 Representation Description

We wanted our state to contain two types of information: the current city of the agent as well as the fact whether there is a task available in this city. For the tasks, we also need to distinguish between the different destinations of the tasks. Therefore our state is modelled by a tuple from the set:

$$\mathcal{S} = \{(c_{curr}, c_{dest}) \,|\, c_{curr} \in \mathcal{C}, c_{dest} \in (\mathcal{C} - \{c_{curr}\}) \cup \{\texttt{null}\}\}$$

where $\mathcal{C}$ is the set of all cities. Note that `null` denotes the case where there is no task available in the current city. The exclusion of c_{curr} in the second place encodes that there are no tasks with delivery city equal to the pickup city.

Our actions are either a `Pickup` for the given task or a `Move` to a neighboring city given by the set:

$$\mathcal{A} = \{\texttt{Pickup}\} \cup \{\texttt{Move}(c) \,|\, c \in \mathcal{C}\}$$

Given this representation of the problem the reward table $R(s, a)$ is defined in straight-forward way. It is the expected reward the agent gets for a delivery $r(c_i, c_j)$ minus the cost for the travelled distance:

$$R(s = (c_i, c_j), a) = \begin{cases} r(c_i, c_j) - k \cdot d(c_i, c_j), & \text{if } a = \texttt{Pickup} \\ -k \cdot d(c_i, c_{neighbor}), & \text{if } a = \texttt{Move}(c_{neighbor}) \end{cases}$$

where $d()$ is the function that calculates the optimal distances and k is the cost per kilometer of the vehicle.

In order to explain the transition table $T(s, a, s')$ we have to think about what it means to go from a state (c_i, c_j) to a state (c_n, c_m). It is important to note that $T(s, a, s') = \mathbb{P}(s'|s, a)$; the action that the agent takes is given.

1. If the agent does `Move`, he ends up in state (c_n, c_m) if c_n is a neighbor of c_i and if there is a task to city c_m (or `null`) in c_n. This happens with the probability given by the provided distribution $p(c_n, c_m)$.

2. If the agent does `Pickup`, he ends up in state (c_n, c_m) if c_n is the destination of the task and if there is a task to city c_m (or `null`) in c_n. This happens also with the probability $p(c_n, c_m)$.

3. In all other cases the transition is impossible, therefore $T(s, a, s') = 0$. This can happen if the action is `Pickup` but there is no task or if the end and start cities are not the same or not neighbors.

1.2 Implementation Details

For our states and actions, we created two specific classes to model them exactly as described above: `State, ActionSpaceElem`. Additionally, there are two methods that return the state space $\mathcal{S}$ and the set of actions $\mathcal{A}$. This allows us to loop naturally over the entire spaces like in the theoretical pseudo-code for value iteration (e.g. $\forall s \in \mathcal{S}$ `do ...`).

However, in order to avoid bugs in the extreme case where $\gamma = 0$ and in order to be faster, we filter the actions that are used in the loop based on the current state of the outer loop. This is done by the method `getPossibleActions`. The method filters for example all `Move` actions that lead to non-neighboring cities. Or when $\gamma = 0$, it prevents the agent from wanting to move to the city where it currently is.

The value iteration algorithm stops when the approximation of $V(s)$ is “good enough”. We chose to calculate the mean square error between the $V(s)$ of two consecutive iterations and to stop the algorithm when the error was $< \epsilon = 10^{-16}$. This corresponds to approximately 200 iterations for the french map and we found that an $\epsilon < 10^{-16}$ didn’t provide any advantage because the strategy $Best(s)$ usually converges much faster.

Finally, our $V(s)$ and $Best(s)$ are implemented using `HashMap`s for optimal performance. We provided the `State` class with an appropriate hash-function. In the `act` function it is therefore sufficient to instantiate the state based on the arguments of the function and to retrieve the best action from the latest $Best(s)$. This happens in $\mathcal{O}(1)$ time thanks to the `HashMap`.

2 Results

2.1 Experiment 1: Discount factor

2.1.1 Setting

2.1.2 Observations

2.2 Experiment 2: Comparisons with dummy agents

2.2.1 Setting

2.2.2 Observations

⋮

2.3 Experiment n

2.3.1 Setting

2.3.2 Observations