Practice Work I — Object Transportation Problem

Rafael Schubert Campos
Regional University of Blumenau — FURB
Reinforcement Learning, Specialization in Data Science

Experiments Report

1) Modeling of the Markov Decision Problem (MDP)

This section describes how this particular solution was modeled, and is divided into three topics:

1. Modeling of the problem's states;
2. Modeling of the agent's actions;
3. and Definition of the reward function.

1.1) Modeling of the problem's states

Each problem state is modeled as P, such that P is a pair of non-negative integer numbers representing the coordinates of a particular tile in the scenario's map (e.g. $(0,1)$, $(3,2)$, $(4,0)$). The first number is the index of the tile's *row*, while the second one is the index of the tile's *column*. Both those coordinates are measured from the top left of the map, starting at $(0,0)$.

Given that the tested scenario consists of a 6 (six) rows by 7 (seven) columns map, there will be a total of 42 (forty-two) states for this particular scenario.

1.2) Modeling of the agent's actions

Since the agent's actions are limited to movements across the map, each one of them is modeled as an enumeration item (the `GridWorldAction` enumeration in the source-code). Each is associated with a specific displacement vector row–column that is added to the agent's current position.

- `MOVE_AGENT_NORTH`: the agent walks one tile upwards ($< 0, -1 >$);
- `MOVE_AGENT_SOUTH`: the agent walks one tile downwards ($< 0, +1 >$);
- `MOVE_AGENT_EAST`: the agent walks one tile to the right ($< +1, 0 >$);
- `MOVE_AGENT_WEST`: the agent walks one tile to the left ($< -1, 0 >$).

Mind that since the tiles are mapped from the top left, an upwards move means a negative displacement in the rows axis, hence the $< 0, -1 >$ and $< 0, +1 >$ vectors for the upwards and downwards moves, respectively.

There's a caveat to those actions, however. Whenever the agent attempts an action that would result in it occupying an unreachable tile — i.e. a wall, a tile outside the map or the tile occupied by the package —, it remains in place instead.

The exception to that caveat is the case where the agent attempts to move while holding the package. In that case, the agent may freely move into the tile occupied by the package, as long as the previously described rules are satisfied for both the agent and the package.

1.3) Definition of the reward function

The reward function $r(S, S')$ takes the agent's current state S, and the resulting state S' as arguments, and rewards the agent's behavior accordingly to the rules below, in the specified order:

1. If $S = S'$, then the agent is rewarded with the value assigned to `punishmentForInvalidMovement`;
2. Else, if S' is an extraction zone, and the agent is currently holding the package, then the agent is rewarded with the value assigned `rewardForPackageExtraction`;

3. Else, if S' is a tile immediately horizontally adjacent to the package's position, and the package has not been captured yet, then the agent is rewarded with the value assigned to `rewardForPackageCapture`;
4. Else, the agent is rewarded with the value assigned to `punishmentForMovement`.

The values of the rewards will be those specified in the file `problem-config.csv`, and for these experiments, I chose the following values:

Reward	Value
`punishmentForMovement`	-0.1
`punishmentForInvalidMovement`	-1.0
`rewardForPackageCapture`	$+1.0$
`rewardForPackageExtraction`	$+1.0$

2) Setup of the Experiments

This section describes how the experiments were set up, and is divided into two topics:

1. Hyperparameters;
2. and Learning horizon.

2.1) Hyperparameters

For the experiments with this particular solution, I decided on experimenting with the following set of values for the hyperparameters:

Hyperparameter	Value
α (`learningRate`)	0.01
γ (`decayRate`)	0.9

2.2) Learning horizon

For the experiments with this particular solution, I decided on imposing the limits below:

Limit	Value
# of episodes	1,000
Maximum # of steps[1]	64

1: The maximum number of steps was arbitrarily calculated as the number of non-wall tiles, minus two tiles (one for the agent, and one for the package), then two times the result. $2 \times (6 \times 7 - 8 - 2) = 64$

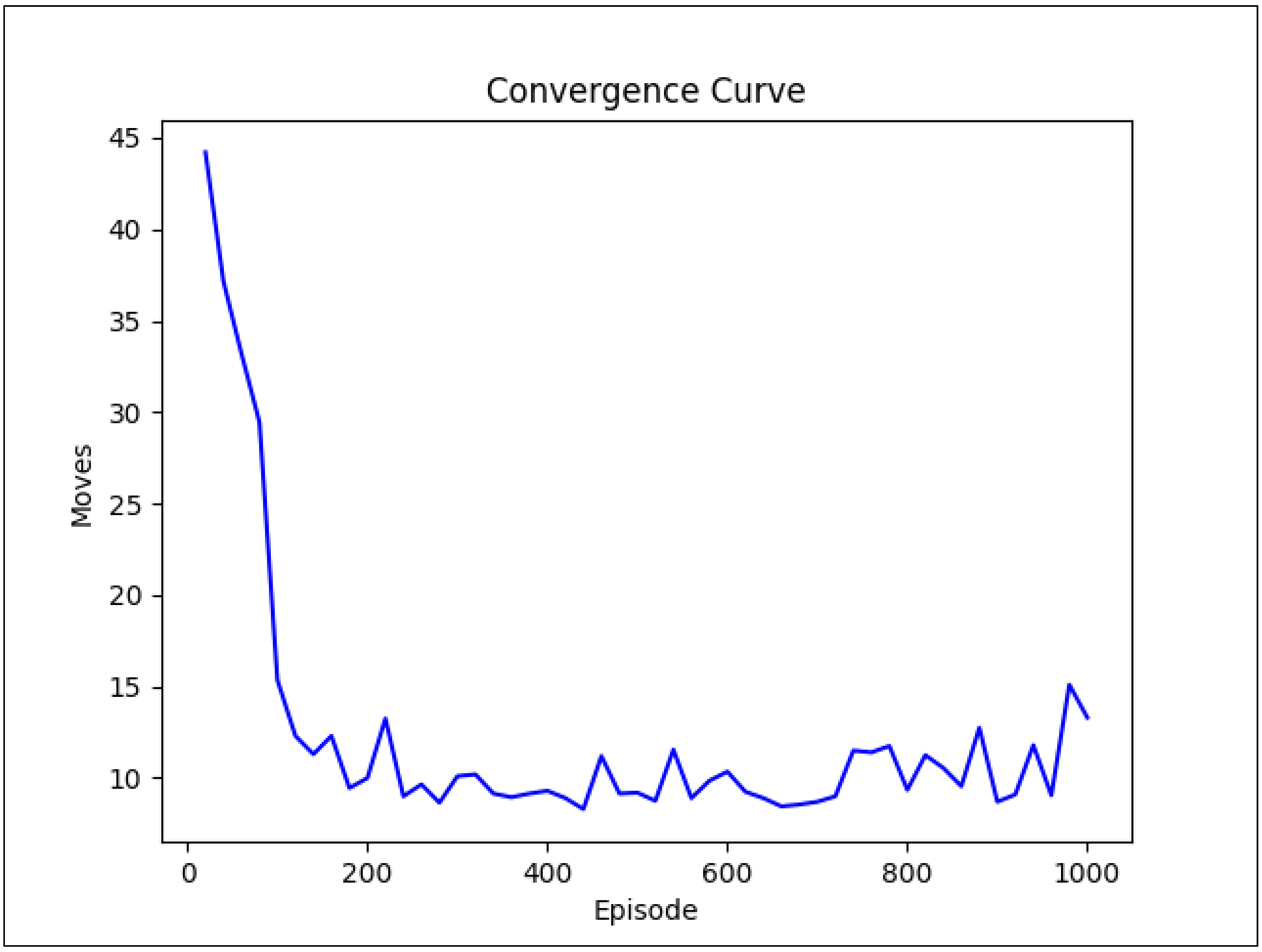
3) Results of the Experiments

This section describes the results of the experiments, and is divided into two topics:

1. Convergence curve;
2. and Time of execution.

3.1) Convergence curve

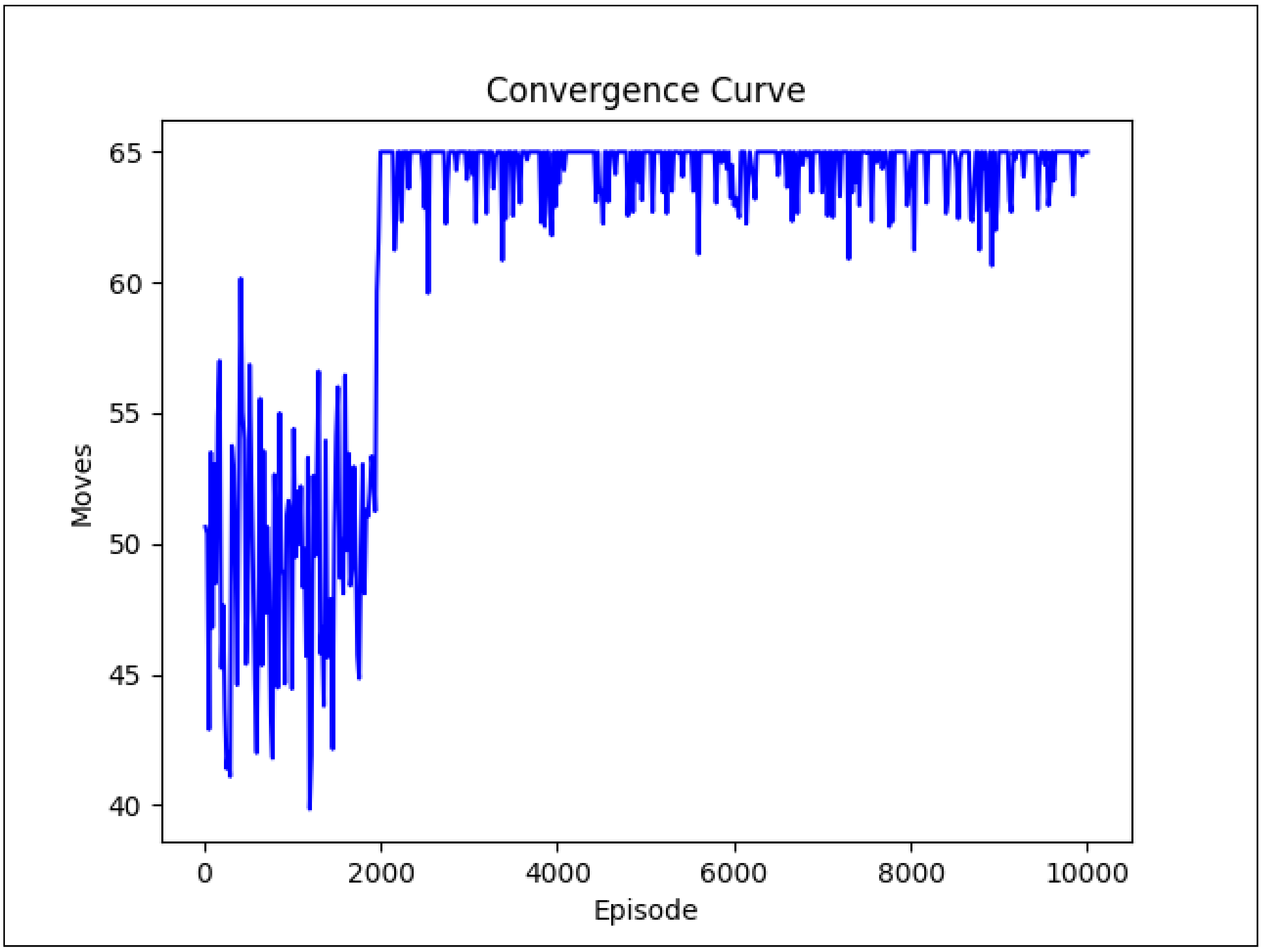
Running this solution with the setup described along the previous topics, it can be noticed that the agent quickly learns a short path to achieve its goal. Each point in the plot presents the average number of steps needed to end a set of twenty episodes.



It can also be noticed that there is some oscillation in the number of steps beyond the 200th episode. I believe that is due to the negative reward awarded to the agent on each valid, non-eventful action. Since the agent tries to stick to the optimal path, the more it travels that same path, the more it is negatively scored.

Perhaps a neutral reward — or rather, no reward at all — on each non-eventful action may reduce that oscillation. It would probably require more episodes, but it may be worth it.

Just for the sake of curiosity, I experimented with 10,000 episodes and set `punishmentForMovement` to 0, but alas, the result was not as promising as it seemed initially. Here is the convergence curve.



It seems that, without the negative reward for uneventful moves, all actions per tile are equally (approximately) scored, because any viable path is therefore acceptable, regardless of length. “All roads lead to Rome,” quite literally.

3.2) Time of execution

A single run of the experiment would be unmeaningful in regards to execution time, since that would depend on the resources currently available. So I fixed Python’s random generator’s seed to a constant value and ran the experiment five times. Here are the results.

Run	Time (s)
1	0.88
2	0.88
3	0.88
4	0.87
5	0.88
Average	**0.88**

Considering how this solution was implemented, I believe that most of the execution time is due to hard disk IO. Had I better implemented the episode reset process, the execution could become faster.