MAST881L – Project

Jacob Chenette (40223429), Gabriel Morin (40223667), and Julien St-Pierre (40224462)

December 7^{th}, 2021

Introduction

The objective of this project was to conduct reinforcement learning in order to find the solution of a small size maze which has multiple optimal deterministic solutions. Indeed, in our maze, multiple sequences of actions can be candidate to be the optimal policy. The maze is defined as a Markov decision process. In order to find one of the optimal paths to the exit of the maze, we applied different off-policy algorithms and one dynamic programming method. Off-policy learning means learning the value function for one policy while following another policy. Therefore, we simulated episodes using a simple fixed policy and compare how the following methods performed for this particular problem: Generalized Policy Iteration (GPI), Q-learning, n-step off-policy SARSA and Tree Backup.

Problem description

We created a maze game of dimension 10 by 10 as presented in Figure 1. The player starts in the left upper corner of the board ($s = 1$) and try to reach the right bottom cell ($s = 100$) by transitioning from one square to another, with diagonal transitions forbidden. The player is rewarded with -1 point for each transition and 0 point if he transitions to the terminal state. Furthermore, the maze has walls, represented by unnumbered black squares, and boundaries that the player cannot cross but can hit, in which case he stays at the same state and is rewarded with -1 point. Thus, from the player's point of view, the objective of the game is to go from initial state (s=1) to terminal state ($s^* = 100$) as quickly as possible. Each simulation of an episode generates a sequence of states, actions and rewards as follows:

$$S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T,$$

where the state-action pair S_{T-1} and A_{T-1} leads to the terminal state with reward $R_T = 0$.
All sets are defined by :

$$\begin{aligned} \mathcal{A} &= \{a : a \in \{1, 2, 3, 4\} \equiv \{\text{left, up, right, down}\}\}, \\ \mathcal{S} &= \{s : s \in \{1, ..., 100\} \setminus \{12, 13, 23, 32, 35, 36, \\ &\qquad 45, 46, 56, 57, 60, 61, 63, 64, 67, \\ &\qquad 68, 69, 75, 76, 84, 85, 89, 97, 98, 99\}\}. \end{aligned} \tag{1}$$

We decided to use a simple exploratory policy to generate episodes, that is,

$$\pi(a \mid s) = 1/4, \ \forall a \in \mathcal{A}, \forall s \in \mathcal{S}.$$

The transition probabilities used in our simulations are all equal for every state-action pair, meaning that

$$p(s', r \mid s, a) = 1 \quad \forall s, a.$$

For a visual view of all possible states, see Figure 1.

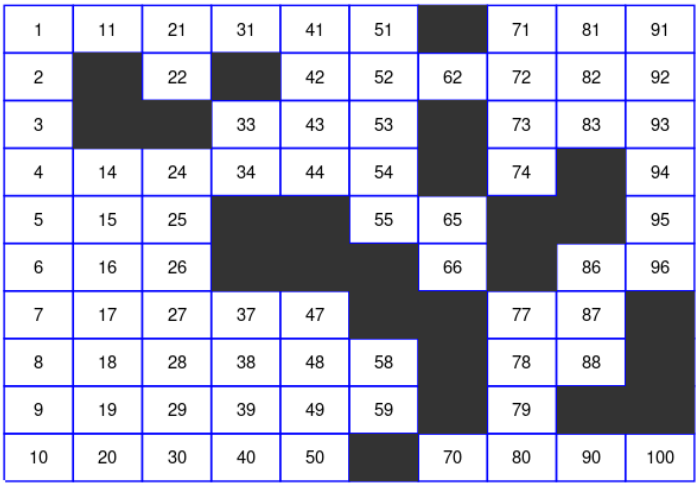

Figure 1: The board game with all states, defined within Equation (1), labelled by its number. Any unnumbered black square represents a wall. State 100 is terminal.

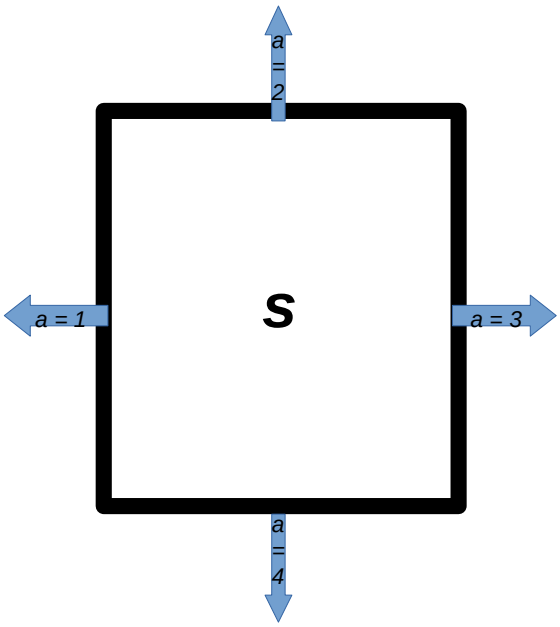

Figure 2: Possible action to all states s

Analyses

Algorithms, approximations, and efficient measures function

We have compared the following off-policy algorithms to solve the maze:

- Generalized Policy Iteration (GPI);
- Q-learning;
- n-step SARSA;
- TreeBackup algorithm.

The previously listed algorithms were compared using the root sum of squared errors (RSSE) defined by

$$RSSE_t = \sqrt{\sum_{s,a} (Q_t(s,a) - q_\pi(s,a))^2}.$$

In order to to calculate the RSSE, the optimal action value function $q_{\pi^*}(s,a)$ must be calculated. To do so, the optimal actions are found using the final estimates of the GPI, and the optimal policy is then obtained for each state ($\pi(s|a) = \pi^*$). The optimal state value function is solved as

$$\mathbf{v}_\pi = (I - \mathbf{M})^{-1}\mathbf{b},$$

using the optimal policy. $\mathbf{b}$ is a vector, where the s^{th} element is

$$\mathbf{b}_s = \sum_{s',r,a} rp(s', r|s, a)\pi(a|s),$$

I is the identity matrix and M is as matrix whose element (s,s') is

$$M_{s,s'} = \gamma \sum_{r,a} p(s', r|s, a)\pi(a|s).$$

When $\mathbf{v}_\pi$ is known, $q_\pi(s,a)$ is computed with the following formula :

$$q_\pi(s,a) = \sum_{s'} \sum_{r} p(s'r|s'a)(r + \gamma v_\pi(s')).$$

The GPI (dynamic programming approach) is known to converge to an optimal policy as long as the number of iteration is infinite. Though, the number of iterations can be truncated and we still obtain convergence. Hence, the value of $q_{\pi^*}(s,a)$ $\forall s, a$ can be found once π^* is obtained with the GPI, since v_{π^*} is unique.

Finally, since there are more than a single optimal policy, to assess convergence of the estimated optimal policy for all methods, we considered the estimated action value function $Q_t(s,a)$ after each iteration (or episode) and compared the number of states which optimal actions matched with the optimal policy that considers equally optimal actions as ties.

GPI

The generalized policy improvement (dynamic programming) is an algorithm that requires a knowledge of the entire environment (i.e. of all state-action pairs, transition probabilities and rewards). Since the resolution of the maze does not involve a large number of states and actions, this technique is appropriate. The principle behind the GPI is to make the two processes of policy improvement and policy evaluation interact. This algorithm is proven to converge to v_{π_*} as long as the number of iterations is infinite. However, in the case of this problem, convergence is achieved in a small number of iterations.

The optimal path obtained from the GPI estimation is one among a set of optimal sequence of actions. For instance, at the 5^{th} step, it is possible to go down and right instead of going right and down (like in Figure 3). In fact, that would lead to the same number of steps before the terminal step is reached.

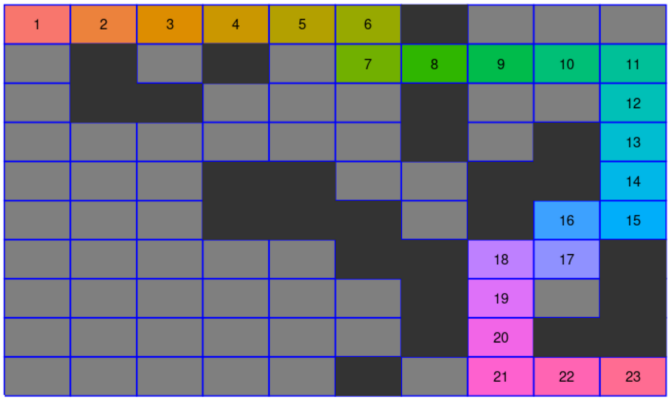

Figure 3: One of the optimal set of states at each iteration for the GPI algorithm. Each number gives the corresponding iteration.

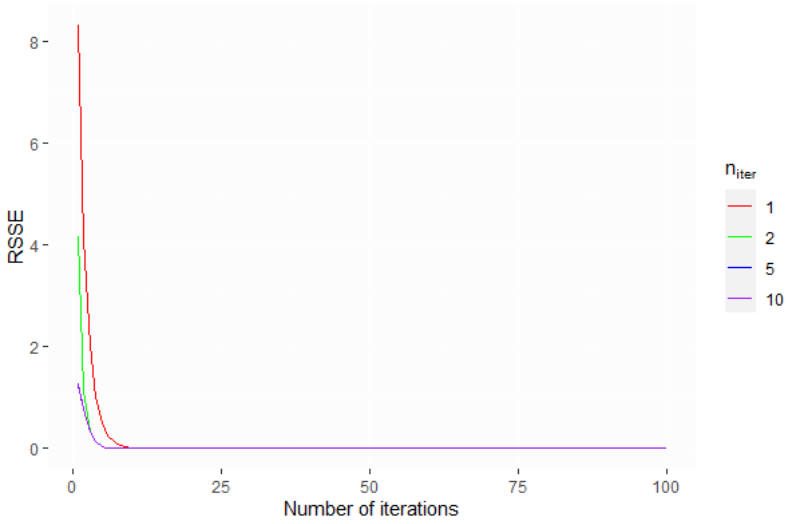

Figure 4: RSSE calculated at each iteration for the generalized policy iteration.

As we can see from Figure 4, the total number of the times the policy is improved and evaluated is equal to 100. The parameter n_{iter} corresponds to the number of sweeps within the policy evaluation. When n_{iter} is 1, the value iteration is performed. According to the plot, as n_{iter} increases, the RSSE decreases much faster. At the 27^{th} iteration, the green, blue, and purple curves are exactly equal to 0. The value iteration algorithm converges at the 28^{th} with a RSSE that equals 0; the GPI learns pretty fast.

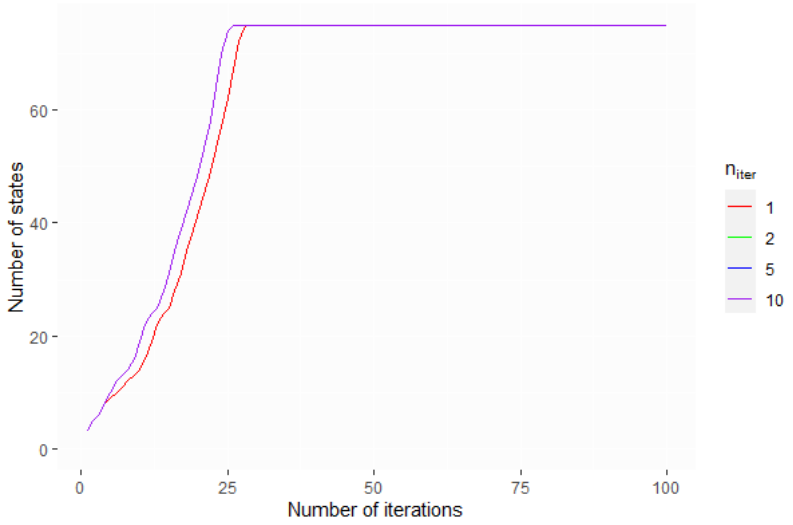

Figure 5: Convergence in policy versus the number of episodes.

In Figure 5, the algorithms performed with n_{iter}=1, 2, and 10 appear as the exact same curve. Hence, they converge equally in policy in exactly 26 iterations. The value iteration algorithm learns a little bit slower than the others (28 iterations). Since the optimal policy is not unique (more than one action might be optimal in a given state), the estimate of $Q(s, a)$ is used to find the optimal actions in each state. The number of states that corresponds to the optimal policy (that considers ties if there are more than one optimal action) are represented in the y-axis. Hence, the number of states in which the optimal actions are found is bounded between 0 and 74. The algorithm using $n_{iter} = 2$ will be used to compare its performance with the other reinforcement learning techniques, because it requires less iterations in the policy evaluation than n_{iter}=5 and 10.

Q-learning

Q-learning suffers from overestimated bias since its algorithm needed to compute the maximum value of the Q-function for each iteration. In practice, since the number of iterations is finite, this procedure will conduct bias in the estimates explain by the Jensen inequality :

$$\mathbb{E}[\max_a Q_t(s, a)] \geq \max_a \mathbb{E}[Q_t(s, a)]. \tag{2}$$

Nevertheless, it is known to be a consistent estimate of q_* meaning that for $\forall \epsilon > 0$:

$$\lim_{t\to\infty} \mathbb{P}[|Q_t(s, a) - q_*(s, a)| > \epsilon] = 0, \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A},$$

where

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q_t(A_{t+1}, a) - Q_t(S_t, A_t)].$$

Thus, for this application, we will use a number of iterations sufficiently large to approximate the Q-function and stopping when the Q-function is stable enough to procure a solution of the puzzle. We have not considered double Q-learning method because of the simplicity of the problem and because the base Q-learning gives us enough performance.

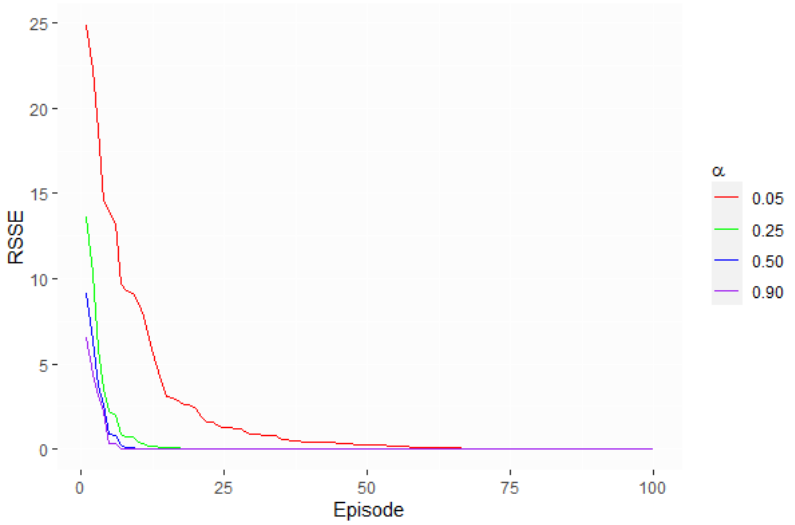

Figure 6: RSSE calculated at each episode for the Q-learning algorithm.

In Figure 6, the RSSE is calculated for each algorithm performed with a learning rate $\alpha = [0.05, 0.25, 0.50, 0.90]$. The figure shows that, as the parameter α increases, the algorithm learns much faster. In fact, the better suited parameter is $\alpha = 0.9$ which allows the Q-learning method to learn the action-value function in only 22 simulated episodes.

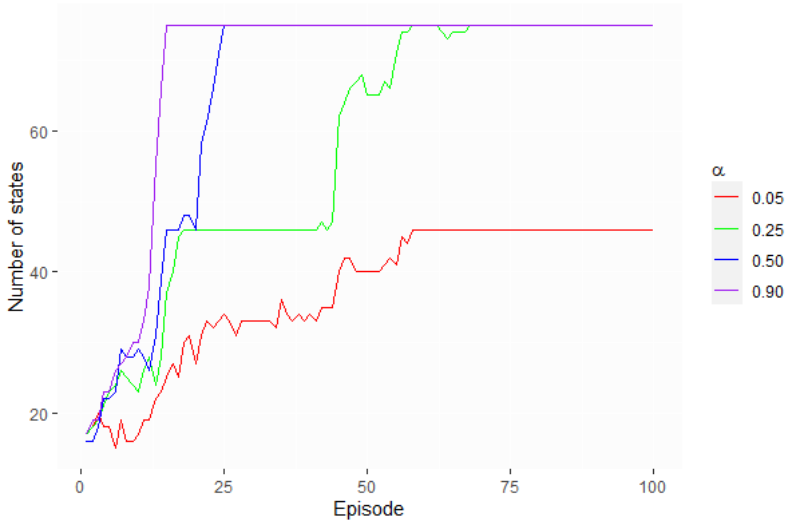

Figure 7: Convergence in policy versus the number of episodes for the Q-learning algorithm.

In Figure 7, we can see which value of α allowed Q-learning to converge in policy in less than 100 episodes. The Q-learning approximations performed with $\alpha = [0.25, 0.5, 0.9]$ plateau at 74 which is the total number of states with correct optimal actions learned by the algorithm ($|\mathcal{S}| = 74$). When α is set to 0.90, it only takes 15 episodes to find the optimal policy (that considers ties). Hence, this value of α will be used to compare the performance of the Q-learning method with other reinforcement techniques.

Off-policy n-step SARSA

In off-policy n-step SARSA, we must take into account the difference between the policy that was used to generate episodes and the policy that we want to learn the value function from, often the greedy policy,

using their relative probability of taking the n actions that were taken. This entails using an *importance sampling ratio* ρ defined by

$$\rho_{t+1:t+n} \equiv \prod_{k=t+1}^{\min(t+n,\tilde{T}_{I(t)}-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}. \tag{3}$$

The off-policy n-step Sarsa update is given by:

$$\begin{aligned} Q_{t+n}(S_t, A_t) &= Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \\ Q_{t+n}(s, a) &= Q_{t+n-1}(s, a) \quad \forall (s, a) \neq (S_t, A_t), \end{aligned}$$

where

$$G_{t:t+n} \equiv \sum_{j=1}^{\min(n,\tilde{T}_{I(t)}-t)} \gamma^{j-1} R_{t+j} + \mathbb{1}_{\{t+n\leq\tilde{T}_{I(t))}\}} \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}).$$

Because the importance sampling ratio ρ is highly variable, especially as step size increases, this algorithm often results in large estimation variance [Asis et al., 2017]. Thus, the resulting high variance must be compensated for by using small step sizes, which can considerably slow learning and requires more care when tuning parameters. Therefore, we allow π in (3) to be ϵ-greedy by varying the probability of exploring actions rather than always selecting the greedy actions because we want to study the performance of this method as a function of ϵ.

We present in Figure 8 and Figure 9 the RSSE after each episode for the n-step SARSA algorithm in function of the step length $n = [1, 5]$ and exploration probability $\epsilon = [0, 0.05, 0.1, 0.15]$, respectively for the first and last 100 of 400 simulated episodes. For the 1-step and 5-steps SARSA, we fixed the learning rate to $\alpha = 0.05$ and $\alpha = 0.005$ respectively, since these values have shown to make the algorithm behaves well. Increasing the probability of exploring from 0 to 0.05 and above decreases the RSSE at a given episode for $n = 5$ and has little effect for $n = 1$ for the first 100 episodes. For $n = 1$, we observe that for $\epsilon = 0$ the method converges to a RSSE different from zero. For other combinations of n and ϵ, the RSSE slowly converges towards zero because of the small values for the learning rate. Of note, slow convergence is the trade off to reduce the variance of the estimator and ensures convergence of this method. Compared to other algorithms in this report, this method has been by far the one that requires the most attention when tuning learning rate α and step-size n in order to avoid RSSE diverging to infinity.

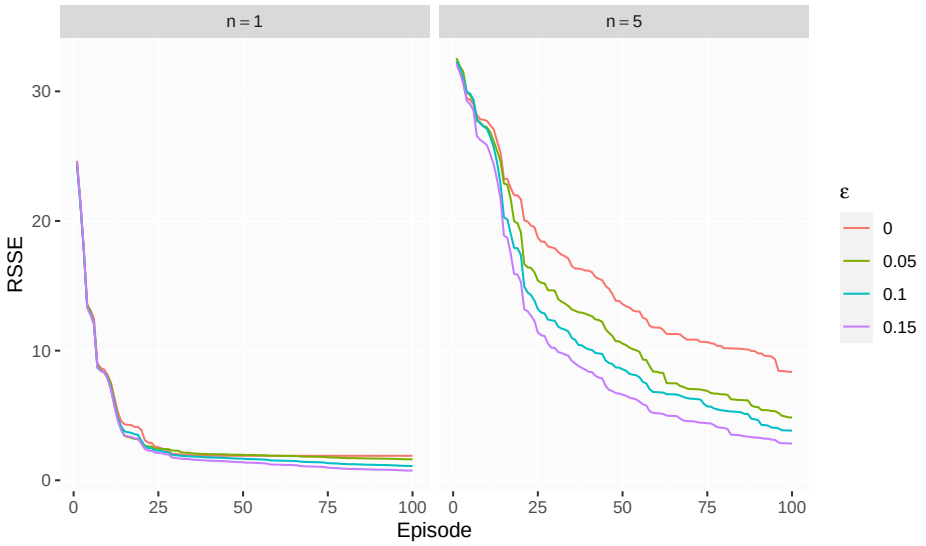

Figure 8: RSSE after the first 100 episodes for the n-step SARSA algorithm.

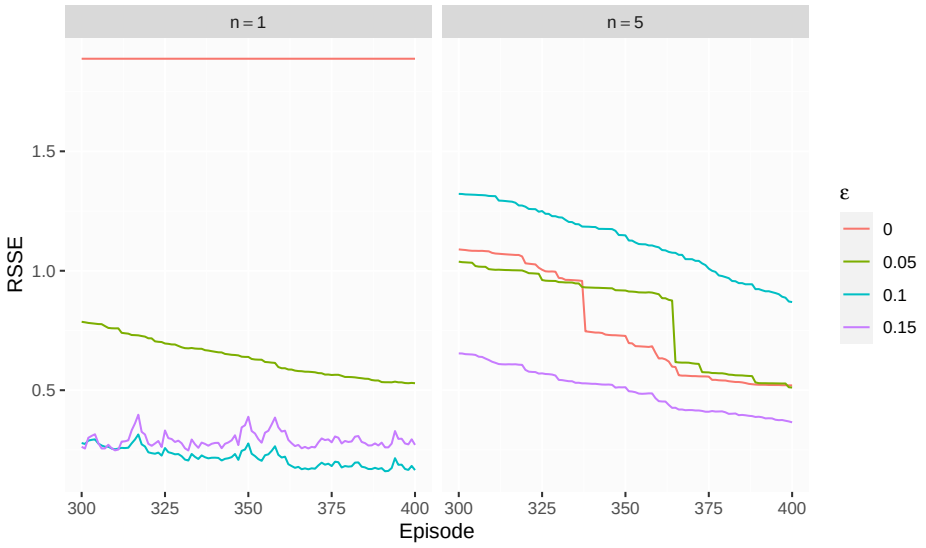

Figure 9: RSSE after the last 100 episodes in Figure 8 for the n-step SARSA algorithm.

We present in Figure 10 the number of state-action pairs that concord with the optimal policy after each episode for the n-step SARSA algorithm in function of the step length and the exploration probability ϵ, for a total of 400 simulated episodes. For $n = 1$ we observe that π has converged to a policy that is only different than the optimal policy for one action-state pair (73 on a total of 74), after 99 episodes. For other values of ϵ, it has not converged to π^*. For $n = 5$, the policy π still has not converged towards the optimal policy after 400 episodes.

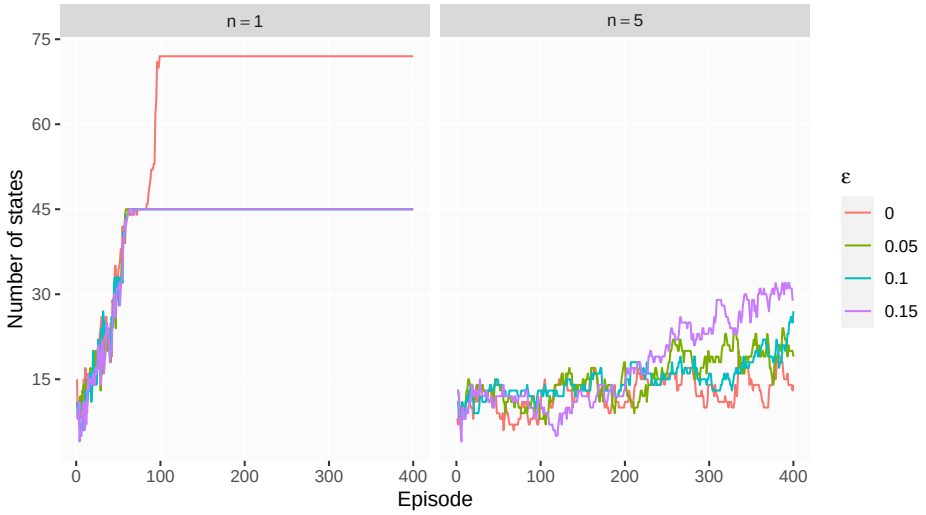

Figure 10: Number of states that match the optimal policy after each episode for the n-step algorithm.

TreeBackup algorithm

In order to avoid importance sampling in n-step off-policy methods, we need expectations over actions in each of the n-step updates. To do so, one can estimate their values by using the current action-value function $Q_t(s, a)$, such that the n-step reward is given by

$$G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+n-1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}.$$

In the Tree Backup algorithm, the previous target is used with the usual action-value update rule from n-step Sarsa:

$$\begin{aligned} Q_{t+n}(S_t, A_t) &= Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)] \\ Q_{t+n}(s, a) &= Q_{t+n-1}(s, a) \quad \forall (s, a) \neq (S_t, A_t). \end{aligned}$$

We present in Figure 11 and Figure 12 the RSSE after each episode for the Tree Backup algorithm in function of the step length $n = [1, 2, 5, 10]$ and learning rate $\alpha = [0.05, 0.25, 0.5, 0.9]$, respectively for the first and last 50 of 200 simulated episodes. Increasing the learning rate from 0.05 to 0.25 and above significantly accelerates convergence. For $n = 1$, we observe that for all values of α the RSSE does not converge to 0 after 200 episodes. For $n = 2, 5, 10$, the RSSE is equal to zero after 200 episodes for all learning rates except for $\alpha = 0.05$ which slowly converges towards zero.

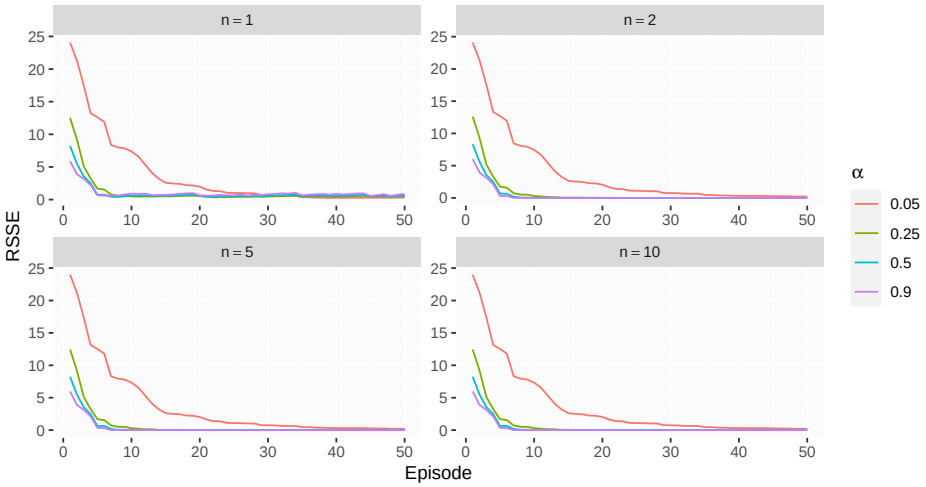

Figure 11: RSSE after the first 50 episodes for the Tree Backup algorithm.

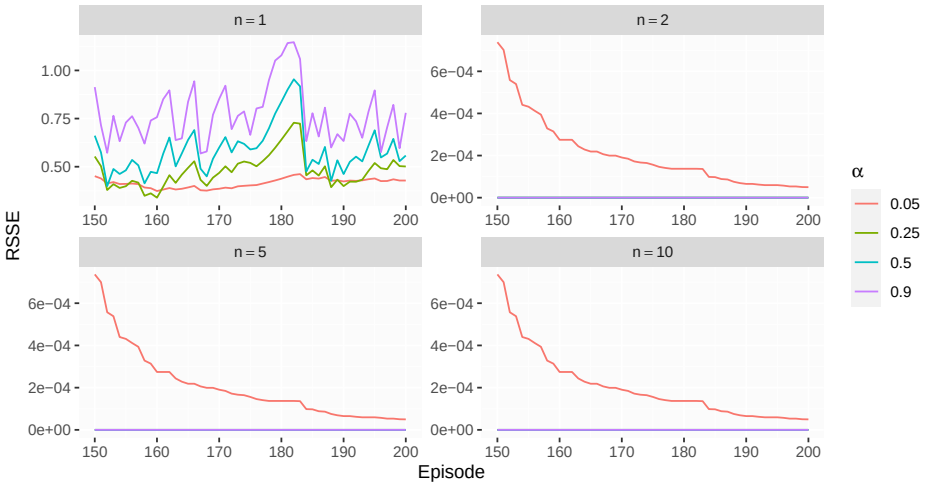

Figure 12: RSSE after the last 50 episodes in Figure 11 for the Tree Backup algorithm.

We present in Figure 13 the number of state-action pairs that concords with the optimal policy after each episode for the Tree Backup algorithm in function of the step length $n = [1, 2, 5, 10]$ and learning rate $\alpha = [0.05, 0.25, 0.5, 0.9]$, for a total of 200 simulated episodes. Again, for $n = 1$ we observe that for all values of α we dot not converge to the optimal policy after 200 episodes. For $n = 2, 5, 10$, the optimal policy is achieved for all learning rates except $\alpha = 0.05$. Moreover, each increase in the learning rate accelerates convergence to the optimal policy; for $\alpha = 0.90$, convergence is attained in 15 episodes.

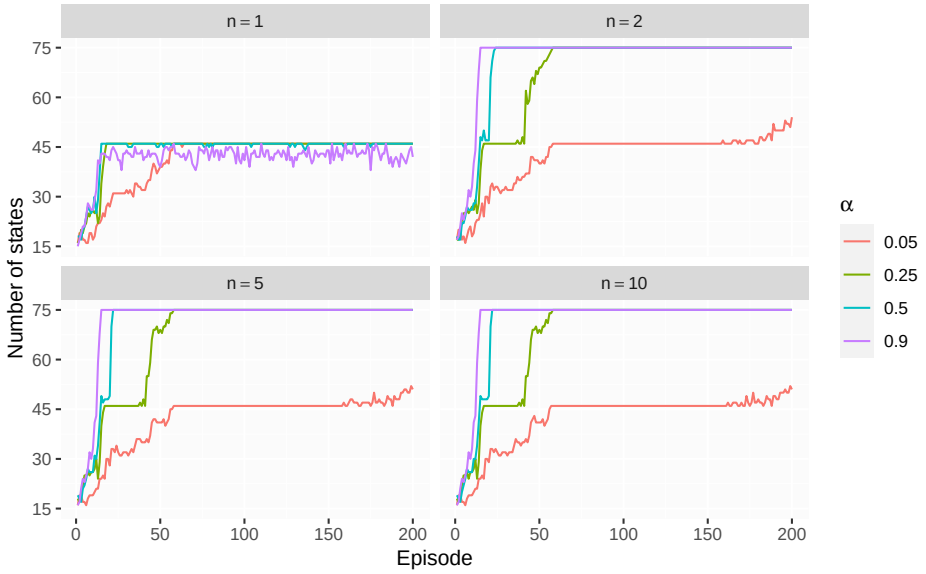

Figure 13: Number of states that match the optimal policy after each episode for the Tree Backup algorithm.

Conclusion

In this report, we wanted to solve a simple given 10×10 maze using multiple reinforcement learning algorithms seen in class. We compared the generalized policy iteration (GPI), the Q-learning, the off-policy n-step SARSA and the Tree Backup algorithm. Using the RSSE as a function of the estimated true action value function, we saw that the GPI showed the fastest convergence rate. Among off-policy methods, the Tree Backup and Q-learning algorithms performed quite comparably, though implementation of the Q-learning algorithm was easier. Off-policy n-step SARSA required the most attention when tuning parameters in order to avoid RSSE diverging to infinity, and exhibited slow convergence compared to other methods. The use of weighted importance sampling ratio could help alleviate these issues in future work as it has a much smaller variance in practice than the ordinary importance sampling. Even though it introduces bias, the weighted importance sampling estimator is consistent, meaning the bias falls asymptotically to zero as the number of samples increases.

Since the puzzle was pretty small and manageable in terms of computation power requirement, the GPI method behaved extremely well. This method was easy to implement since it did not require any simulation and all the information of the environment is stored in a relatively small matrix. Obviously, this general class of problems can be complexified by adding multiple additional dimensions to the puzzle, like a time dependent third dimension. A practical application is the algorithm required by a robot to move into an environment, for instance Boston Dynamics robot-dog Spot, which is a much more complex problem than the one we solved for this project.

References

[Asis et al., 2017] Asis, K. D., Hernandez-Garcia, J. F., Holland, G. Z., and Sutton, R. S. (2017). Multi-step reinforcement learning: A unifying algorithm.

Appendix - Code structure

```
ProjetRL
 ¦--Code_structure.R
 ¦   °---> function* : checkScriptVars
 ¦--Library.R
 ¦   °--<!>LIST OF PACKAGES NEEDED TO BE INSTALLED<!>
 °--Scripts
    ¦--Main.R
    ¦   ¦---<|> Dependency with : Scripts/Simulations/SimulateEpisodeMaze.R
    ¦   °---<|> Dependency with : Scripts/RL_algorithm/miscfunc.R
    ¦--RL_algorithm
    ¦   ¦--GPI_Jacob.R
    ¦   ¦   ¦---> function* : PolicyEvaluation
    ¦   ¦   ¦---> function* : PolicyImprovement
    ¦   ¦   ¦---> function* : GPI
    ¦   ¦   ¦---<|> Dependency with : /Scripts/Main.R
    ¦   ¦   ¦---<|> Dependency with : Scripts/RL_algorithm/miscfunc.R
    ¦   ¦   °---<|> Dependency with : Scripts/RL_algorithm/PolicyAVF.R
    ¦   ¦--Graph_path_to_flag.R
    ¦   ¦   ¦---<|> Dependency with : /Scripts/Main.R
    ¦   ¦   ¦---<|> Dependency with : /Scripts/RL_algorithm/Q_learning_Jacob.R
    ¦   ¦   °---<|> Dependency with : /Scripts/RL_algorithm/GPI_Jacob.R
    ¦   ¦--miscfunc.R
    ¦   ¦   ¦---> function* : arrows
    ¦   ¦   °---> function* : QtoMaze
    ¦   ¦--PolicyAVF.R
    ¦   ¦   °---<|> Dependency with : /Scripts/Main.R
    ¦   ¦--Q_learning_Jacob.R
    ¦   ¦   ¦---> function* : ApplyQLearning
    ¦   ¦   ¦---<|> Dependency with : /Scripts/Main.R
    ¦   ¦   ¦---<|> Dependency with : Scripts/RL_algorithm/miscfunc.R
    ¦   ¦   °---<|> Dependency with : Scripts/RL_algorithm/PolicyAVF.R
    ¦   ¦--SARSA_Julien.R
    ¦   ¦   ¦---> function* : ApplySARSA
    ¦   ¦   ¦---<|> Dependency with : /Scripts/Main.R
    ¦   ¦   °---<|> Dependency with : Scripts/RL_algorithm/PolicyAVF.R
    ¦   °--TreeBackup_Julien.R
    ¦       ¦---> function* : ApplyTreeBackup
    ¦       ¦---<|> Dependency with : /Scripts/Main.R
    ¦       °---<|> Dependency with : Scripts/RL_algorithm/PolicyAVF.R
    °--Simulations
        °--SimulateEpisodeMaze.R
            °---> function* : SimulateEpisodeMaze
```

Figure 14: All functions and script dependency. It is required to open the Library.R script and install all packages within the script. All algorithm naming SARSA, TreeBackup and GPI are the script that will run the reinforcement learning algorithms. Other script will create plots (e.g. graph_to_flag) or the data structure seen in this figure. Main script defines the directory for all dependent file, plus it is a connection to miscfunc and SimulateEpisodeMaze function. Data folder stored the maze puzzle with optimal policies.