Assignment 3 - ECS7002P - Frozen Lake

Alexander Sworski (190004273), Amir Sepehr Aminian (210453289), Dimitrios Mylonas (180326363)

School of Electronic Engineering and Computer Science Queen Mary University of London, UK

I. Question 1

The code in this assignment was organised in separate files for each task. Firstly, the environment for frozen lake was implemented according to the specification. The next 3 files were functions for implementations of the 6 algorithms, grouped in tabular model based, tabular model free and non-tabular model free methods respectively.

The main deviation from the suggestions was in our fifth file, where our *main* function, which calls all algorithms we have implemented, is changed to accept parameters. With these parameters we can decide if we want to run all algorithms together or each task individually. Furthermore, we can also decide if the algorithms perform learning on the big or small lake. These changes facilitate easier experimentation for finding optimal policies and iterations required for each reinforcement learning (RL) algorithm.

A change we added in our Sarsa and Q-Learning algorithms (including the linear approximation versions) was that we forced the first 4 moves to be 1 in each possible direction. This was done to increase exploration initially and help build the state space.

A further addition to our code was a file that allows the policy and values for each of the states to be displayed on a visualisation of the frozen lake. An example of that can be seen in Fig.1.

II. Question 2

Performing policy iteration on the big frozen lake resulted in the policy and values seen in Fig.1 and took 6 iterations to converge to the optimal policy. In the case of value iteration, the algorithm took more iterations to converge to the same optimal policy: 19 iterations. This behaviour was expected, however, policy iteration was taking 50% longer time to find the optimal policy than value iteration, despite performing less iterations. This can probably be attributed to the fact that policy iteration requires calling the functions for policy

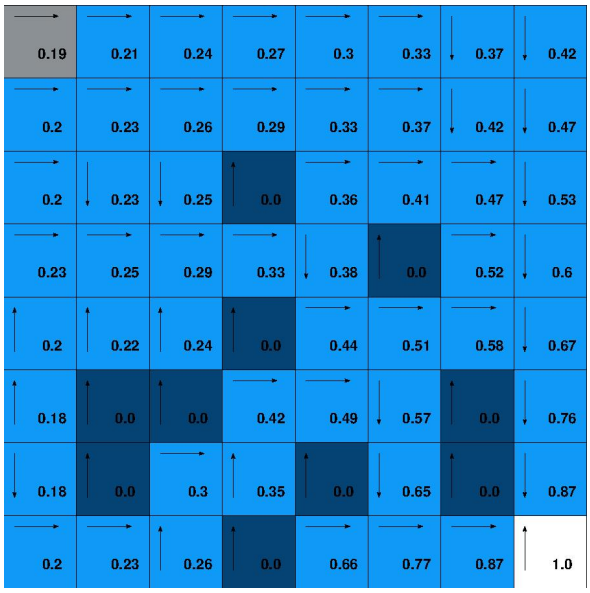

Figure 1: Resulting policy and values using policy iteration and value iteration. Both converge to the same policy.

evaluation and policy improvement. This meant that additional computation was occurring during each iteration, leading to a slower policy iteration.

III. Question 3

For the tabular model free algorithms performed on the small lake, policies were found and compared to the optimal policies found by policy evaluation. Q-learning, converged to the optimal policy after 3000 episodes. However, Sarsa failed to converge after 10000 episodes (which was the maximum episode count decided by us), but arrived at a policy that was quite close to the one found by our model based algorithms. The resulting policies and their state values can be seen in Fig.2.

IV. Question 4

In linear action-value approximation, the state value is approximated by a parametric function. A state is often represented by a feature vector ϕ(s). These features represent different aspects of the game that affect the value of a state. Given a parameter θ we can represent the value function as

$$V(s;\theta) = \theta^T \phi(s)_i = \sum_i \theta_i \phi(s)_i \quad (1)$$

This means that for a feature vector where all elements except one are 0 (which is a one hot encoding), each vector represents one possible state-action value. This results in our value function to be

$$V(s;\theta) = \sum_i \theta_i \phi(s)_i = \theta_n * 1 = \theta_n \quad (2)$$

assuming there are n states. This means that the parameters θ can simply be interpreted as the weights given to each state.

The tabular model-free reinforcement learning algorithms we implemented are a special case of the non-tabular model-free reinforcement learning algorithms because they are the case where the features $\phi(s)$ represent only the value referring to the quality of the state. This is possible for finite and small environments and is called

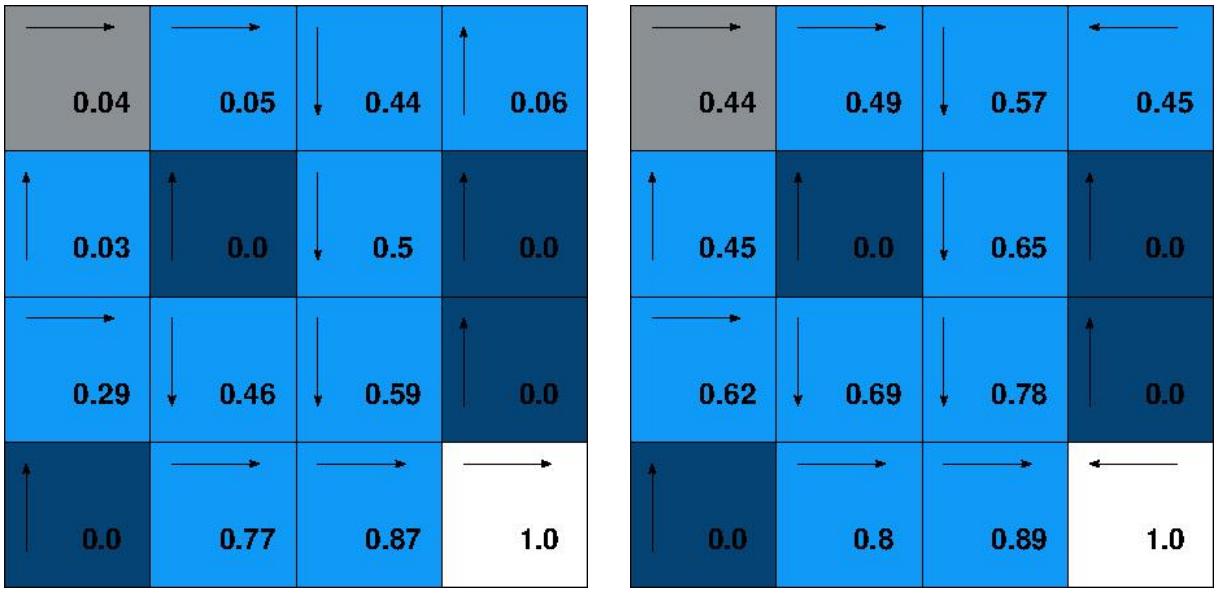

Figure 2: Policies and state values for small lake using Sarsa (left) and Q-Learning (right).

a tabular model. However, when the features include values representing different aspects of the game, this is a non-tabular model, and is simply an extension of the tabular model with features that are not one hot encoding.

V. Question 5

Next, policies were found using Q-learning and Sarsa for the big frozen lake. This was done by tweaking the parameters and comparing the resulting policy using these tabular model-free algorithms to the optimal policy achieved by policy evaluation.

Using the given parameters from the assignment description (2000 episodes, 0.001 learning rate and 0.5 exploration factor) resulted in non-converged policies. The algorithms clearly did not have enough episodes to converge and many of the state values were 0. This led to the number of episodes being increased incrementally to find a number that leads towards the wanted optimal policy.

A further change was increasing the value of the exploration constant from 0.5 to 0.9. This is to prioritise exploration, because our algorithms seem to have a lot of 0's, suggesting these states were not visited during run time. This change paired with the increase in episode number lead to the Q-Learning algorithm converging very close to the optimal policy at around 50 thousand episodes. This is not however the case for Sarsa. Even for very large number of episodes of more than 100 thousand, the algorithm could not find the optimal policy. Changing the learning rate did not affect the convergence of Sarsa. The policies found can be seen in Fig.3.

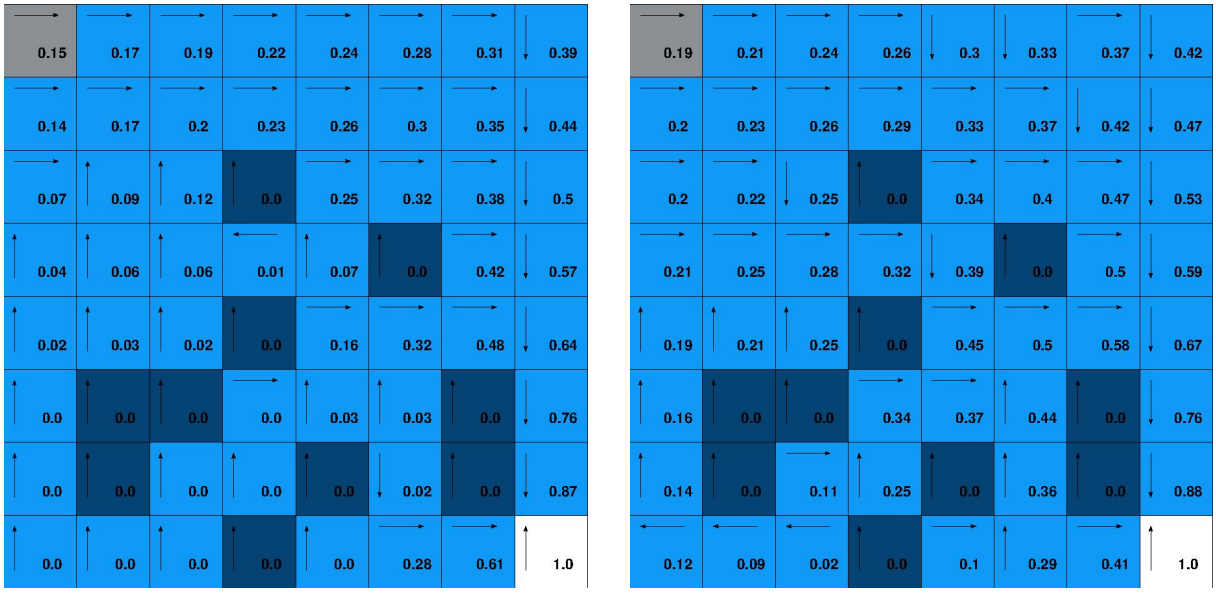

Figure 3: Policies and state values for big lake using Sarsa (left) and Q-Learning (right).