ECS7002P – Artificial Intelligence in Games

Assignment 2 Report: Frozen Lake

Vilma Sinani (210843664)

Stella Ivy Enuaye Annor (210717675)

Maha Alanazi (210856066)

Q1: Explain how your code for this assignment is organised. Did you make implementation decisions that deviate significantly from what we suggested? [5/100]

The code for this assignment was written in python using the suggested implementation and instructions defined in the assignment specification. The code folder consists of the following:

1. *Environment.py*: defines a generic frozen lake environment with the following classes: *EnvironmentModel* (represents a model in the environment); *Environment* (inherits from *EnvironmentModel* and represents an interactive environment); *FrozenLake* (represent the frozen lake environment and implements the two methods, *p*, and *r*).
2. *tabular_model_based.py*: defines the tabular model-based reinforcement learning. This consist of the policy evaluation, policy improvement, policy iteration and value iteration functions.
3. *tabular_model_free.py*: defines ε-greedy, *Sarsa* control and Q-learning control.
4. *Non-tabular_model-free_algorithm.py*: define the *Sarsa* control and Q-learning control using linear function approximation. This uses *LinearWrapper* class (emulates the environment), *linear_SARSA*, and *linear_q_learning* methods.
5. *main.py*: uses the implemented algorithms listed above and different parameters to output estimated values.
6. *REAME.txt:* specifies how to run the *main.py* code.
7. *Output.txt:* provides the output of *main.py* code when using 100 *max_iterations*, 0.9 for *gamma*, and 0.001 for *theta*.

Q2. How many iterations did policy iteration require to find an optimal policy for the big frozen lake? How many iterations did value iteration require? Which algorithm was faster? [5/100]

The number of iterations required to find the optimal policy for the big frozen lake is 6 for policy iteration and 19 for value iteration, which are significantly lower than 100, the maximum number of iterations defined in *main.py*. Both iterative algorithms are guaranteed to find an optimal policy, however, the policy iteration algorithm converges faster (6 iterations required) in comparison to the value iteration algorithm as shown by the results obtained. This is expected due to the fundamental difference in the algorithm: the value iteration iterates over the value function and the policy iteration updates the policy by first evaluating the policy and then improving the policy. The value iteration finds the maximum action value by looping through all possible actions which requires many iterations to converge and is therefore more computationally expensive.

Q3. How many episodes did Sarsa control require to find an optimal policy for the small frozen lake? How many episodes did Q-learning control require? Hint: you may use policy evaluation to compare the value of each policy obtained by these algorithms to the value of an optimal policy [5/100]

Executing the SARSA algorithm using the small frozen lake environment led to the policy being returned after 2000 episodes, which is the maximum number of episodes set for the problem. As for the case where Q-learning was implemented, the optimal policy was returned after 399 episodes, hence the algorithm did not have to execute all the iteration steps in order to reach its optimal solution. This implies Q-learning converges faster. After

altering the maximum number of episodes and setting it to 20000, the tabular model free algorithms were tested again. Doing so demonstrated that SARSA's rate of convergence is slower, as the condition indicating optimality was reached after 14942 episodes and its optimal policy was returned.

Q4. In linear action-value function approximation, how can each element of the parameter vector θ be interpreted when each possible pair of state s and action a is represented by a different feature vector φ(s, a) where all elements except one are zero? Explain why the tabular model-free reinforcement learning algorithms that you implemented are a special case of the non-tabular model-free reinforcement learning algorithms that you implemented. [5/100]

The parameter vector θ corresponds to the weights assigned with each feature, where the features are encoded as columns in a features matrix. Each column of the matrix corresponds to a feature, which has a number of elements computed by multiplying the number of actions by the number of states. Hence, each feature has elements encoding state-action pairs. As the feature vectors are represented as one-hot encodings with zero values across all entries except for 1, all irrelevant state-action pairs are discarded and only the relevant pair is preserved. Hence, updating the parameters vector θ during training will not affect the irrelevant state-action pairs.

The tabular model-free approach is a special case of the non-tabular model-free in which a table is constructed to store values of each state-action pair, these values being stored in the memory. In the case of non-tabular model-free algorithms, a table is not constructed, hence values are not stored in the memory. Instead, they are computed at each step using the encoding provided by the feature vectors.

Q5. Try to find an optimal policy for the big frozen lake by tweaking the parameters for Sarsa control and Q-learning control (maximum number of episodes, learning rate, and exploration factor). You must use policy evaluation to confirm that the resulting policy is optimal. Even if you fail, describe your experience. [5/100]

Executing the tabular model free algorithms using the big frozen lake environment led to policies containing only 0 values being returned for both SARSA and Q-learning. Experimenting with different values of the parameters, such as increasing the maximum number of episodes up to 100,000 episodes and altering the learning rate and exploration factor did not result in convergence either, the policies returned still having 0 entries only. This suggests both algorithms are not able to learn as the policies are not being optimised and remain the same as when they were initialised. This is due to the increased size of the search space, which in turn increases the complexity of the problem.