COMP 560: Assignment 2

Reinforcement Learning

Group Members: Arturo Schmidt (1 contributor)

**Environment Setup**

The environment is parsed from stdin as is represented as a dictionary mapping a state and action to a state with certain probability. The agents interact with the environment by taking an action in a state, which returns a reward and a new state based on the probability model described by the dictionary. The environment takes as parameters the reward of the goal state and intermediary states. The agents have no knowledge of the probability table maintained by the environment.

**Model-Based Learning**

Passed 🡪 environment, minExplorations, maxIterations, discountValue, learnRate

The approach used in model-based learning was that of two steps. In the first step, the probability model is calculated. This is done by playing the game randomly and noting the P(s’|s,a). This is done until the values converge with a diff < 0.001 . The second step is the use value iteration with a overestimated utility value. This is done by having the exploration function set the utility of a (state,action) pair according to the number of times that state action combination has occurred. If it is less than minExplorations then it is set to a parameter which corresponds to a utility overestimate.

The way the iteration works is for maxIterations, the agent starts at ‘Fairway’, we enter a loop in which we check if we are in state ‘In’. If we are not, we get the action resulting from the exploration function, which is either the highest utility valued action or a random action. Once we have our action, we take that action in the previous state and note the new state and the reward seen. We use this to update our utility function

U(s) += learnRate\*(reward + discountValue\*explorationFunction(s,a,n[s,a]))

We iterate until we hit a terminal state and repeat this for n epochs. From this created utility table we can simply choose from every state the action that maximizes V = P(s’|s,a)\*U(s’) to create our policy.

**Model-Free Learning**

Passed 🡪 environment, maxShots, learnRate,discountValue,epochs

The technique used for model-free learning was using q-learning. The agent receives a starting state by the environment and get the next action to execute. This is done by either returning the highest q-valued action or retuning a random state dependent on an exponential exploration function. Once we have our action, we execute the action. The environment returns a new state and its reward and the q-table[previous state, action] gets updated according to the q-learning function:

Q(s,a) = learnRate(Q(s,a) + discountValue Q(s’,a’) – Q(s,a))

This is repeated until the agent reaches a terminal state or the number of actions taken in the current epoch is greater than accepted. The game is repeated epoch times, updating the same q-table. Once we have our q-table, the optimal policy quickly follows by choosing the action that leads to the highest q-value at every state.

**Correlations**

When analyzing discount and epsilon (learnRate) values of both algorithms, the approach taken was to graph the outcomes of calculating the average number of shots per game as we increased the discount and epsilon value (separately) by 0.02 from 0.1 to 1. Results showed no conclusion correlation to imply that the values have a strong influence in the outcome of the utilities (at least in the sample example of the assignment. With regards to the exploration function, a similar process was done. For model-based it was accomplished by setting the number of times a state had to be visited before being trusted incrementally through trials. For model-free is was done by adjusting the exponential function. As previously, no strong correlation was found to indicate an optimal value to balancing exploration vs utility. This might be due to the sample game being very small and the values oscillating around the average. Graphing of the results is available in source code.

The stopping of learning processes was done by predetermining a number of epochs. 10000 epochs were used in the testing mentioned previously.