COMP 767 (Reinforcement Learning) Assignment 1

Arna Ghosh (260748358), Arnab Kumar Mondal (260906419)

February 2020

1 Theory Part

Answer 1.

We have $\mu_i = \mathbb{E}[R_i]$ and $\mu^* = \max_{i=1}^{K} \mu_i$ where K is the number of arms. Let us denote $\bar{\mu}_i$ be the estimate of i-th arm such that $\bar{\mu}_i = \sum_{t=1}^{T/K} \frac{R_{it}}{T/K} = \sum_{t=1}^{T/K} \frac{K}{T} R_{it}$ where R_{it} is the random variable denoting the reward sample obtained from t-th trial of the i-th arm. If we take $n = T/K$ then we can define random variable $\bar{\mu}_i = \frac{\sum_{t=1}^{n} R_{it}}{n}$. By Hoeffding's inequality we have:

$$\mathbb{P}[|\bar{\mu}_i - \mathbb{E}[\bar{\mu}_i]| \geq \lambda] \leq 2e^{-2n\lambda^2}$$

where $\lambda \geq 0$. We also have:

$$\mathbb{E}[\bar{\mu}_i] = \mathbb{E}[\frac{\sum_{t=1}^{n} R_{it}}{n}] = \frac{\sum_{t=1}^{n} \mathbb{E}[R_{it}]}{n} = \frac{\sum_{t=1}^{n} \mu_i}{n} = \mu_i$$

$$\implies \mathbb{P}[|\bar{\mu}_i - \mu_i]| \geq \lambda] \leq 2e^{-2n\lambda^2}$$

Now let E_i be the event of $|\bar{\mu}_i - \mu_i| < \lambda$, then $\mathbb{P}[\Omega \setminus E_k] = \mathbb{P}[|\bar{\mu}_i - \mu_i]| \geq \lambda] \leq 2e^{-2n\lambda^2}$ where Ω is the universal set.

$$\mathbb{P}[\bigcap_{i=1}^{K} E_i] = \mathbb{P}[\Omega \setminus (\bigcup_{i=1}^{K} \Omega \setminus E_i)] = 1 - \mathbb{P}[\bigcup_{i=1}^{K} \Omega \setminus E_i]$$

By Union bound we have:

$$\mathbb{P}[\bigcup_{i=1}^{K} \Omega \setminus E_i] \leq \sum_{i=1}^{K} \mathbb{P}[\Omega \setminus E_i]$$

$$\implies \mathbb{P}[\bigcap_{i=1}^{K} E_i] \geq 1 - \sum_{i=1}^{K} \mathbb{P}[\Omega \setminus E_i] \geq 1 - \sum_{i=1}^{K} 2e^{-2n\lambda^2}$$

$$\implies \mathbb{P}[\bigcap_{i=1}^{K} E_i] \geq 1 - 2Ke^{-2n\lambda^2}$$

$\bigcap_{i=1}^{K} E_i$ implies that $|\bar{\mu}_i - \mu_i| < \lambda \quad \forall i \in \{1, 2, .., K\}$. Let $\hat{i} = \arg\max \bar{\mu}_i$
and if $\hat{i}$ is the optimal arm chosen then $|\mu^* - \mu_{\hat{i}}| = 0$ else we have two inequalities $|\mu^* - \bar{\mu}^*| < \lambda$ and $|\mu_{\hat{i}} - \bar{\mu}_{\hat{i}}| < \lambda$. Solving the last two inequalities gives us $\mu^* - \mu_{\hat{i}} \leq 2\lambda$. Choosing $\lambda = \epsilon/2$ we get $\mu^* - \mu_{\hat{i}} \leq \epsilon$ with probability $1 - \delta$ for $T = nK$ trails where:

$$\delta = 2Ke^{-2n\lambda^2} = 2Ke^{-\frac{T}{2K}\epsilon^2}$$

$$\implies T = \frac{2K}{\epsilon^2} \ln \frac{2K}{\delta} = \mathcal{O}(\frac{1}{\epsilon^2} \ln \frac{1}{\delta})$$

Answer 2.

i) Let us write down the definition of the value functions according to their definition:

$$V_M^\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}|S_t = s]$$

$$V_{\bar{M}}^\pi(s) = \mathbb{E}[\bar{G}_t|S_t = s] = \mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1} \bar{R}_{t+i}|S_t = s]$$

For any policy $\pi(a|s)$ we have

$$\bar{R}(s) = \sum_a \pi(a|s)\bar{R}(s,a) = \sum_a \pi(a|s)(R(s,a) + \mathcal{N}(\mu, \sigma^2))$$

$$\implies \bar{R}(s) = R(s) + \mathcal{N}(\mu, \sigma^2)$$

This gives us:

$$V_{\bar{M}}^\pi(s) = \mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1}(R_{t+i} + \mathcal{N}(\mu, \sigma^2))|S_t = s]$$

$$= \mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}|S_t = s] + \mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1}\mathcal{N}(\mu, \sigma^2)|S_t = s]$$

$$= V_M^\pi(s) + \sum_{i=1}^{\infty} \gamma^{i-1}\mathbb{E}[\mathcal{N}(\mu, \sigma^2)] = V_M^\pi(s) + \sum_{i=1}^{\infty} \gamma^{i-1}\mu$$

$$\implies V_{\bar{M}}^\pi(s) = V_M^\pi(s) + \frac{\mu}{1-\gamma}$$

i) We are given that $\bar{P} = (\alpha * P + \beta * Q)$ where $\alpha + \beta = 1$. let the state transition matrix following policy π be P_π and $\bar{P}_\pi$ which implies $\bar{P}_\pi = (\alpha * P_\pi + \beta * Q_\pi)$. Using Bellman equation in matrix form we have:

$$V_M^\pi = R_\pi + \gamma P_\pi V_M^\pi \quad \& \quad V_{\bar{M}}^\pi = R_\pi + \gamma \bar{P}_\pi V_{\bar{M}}^\pi$$

Subtracting both we get that :

$$V_{\bar{M}}^\pi - V_M^\pi = \gamma \bar{P}_\pi V_{\bar{M}}^\pi - \gamma P_\pi V_M^\pi$$

$$\implies V_{\bar{M}}^\pi - \gamma \bar{P}_\pi V_{\bar{M}}^\pi = V_M^\pi - \gamma P_\pi V_M^\pi$$

$$\implies (I - \gamma \bar{P}_\pi)V_{\bar{M}}^\pi = (I - \gamma P_\pi)V_M^\pi$$

$$\implies (\alpha(I - \gamma P_\pi) + \beta(I - \gamma Q_\pi))V_{\bar{M}}^\pi = (I - \gamma P_\pi)V_M^\pi$$

Assuming $(I - \gamma P_\pi)$ is not singular we get:

$$(\alpha I + \beta(I - \gamma P_\pi)^{-1}(I - \gamma Q_\pi))V_{\bar{M}}^\pi = V_M^\pi$$

$$V_{\bar{M}}^\pi = (\alpha I + \beta(I - \gamma P_\pi)^{-1}(I - \gamma Q_\pi))^{-1} V_M^\pi$$

Answer 3.

.We have in this question $|V^*(s) - \hat{V}(s)| \leq \epsilon \quad \forall s \in S$ and $L_{\hat{V}}(s) = V^*(s) - V_{\hat{V}}(s)$ where $V_{\hat{V}}$ is the value function obtained after evaluating greedy policy with respect to $V_{\hat{V}}$. Let us write the greedy policy:

$$\hat{a} = \pi_{\hat{V}}(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r(s,a) + \gamma \hat{V}(s')]$$

and the optimal policy:

$$a^* = \pi_{V^*}(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V^*(s')]$$

$$V_{\hat{V}}(s) = \sum_{s'} p(s'|s,\hat{a})[r(s,\hat{a}) + \gamma V_{\hat{V}}(s')]$$

$$= \sum_{s'} p(s'|s,\hat{a})[(r(s,\hat{a}) + \gamma \hat{V}(s')) + (\gamma V_{\hat{V}}(s') - \gamma \hat{V}(s'))]$$

$$= \sum_{s'} p(s'|s,\hat{a})[r(s,\hat{a}) + \gamma \hat{V}(s')] + \gamma \sum_{s'} p(s'|s,\hat{a})[V_{\hat{V}}(s') - \hat{V}(s')]$$

$$\geq \sum_{s'} p(s'|s,a^*)[r(s,a^*) + \gamma \hat{V}(s')] + \gamma \sum_{s'} p(s'|s,\hat{a})[V_{\hat{V}}(s') - \hat{V}(s')]$$

The last statement is true as $\hat{a}$ is the greedy action with respect to value function $\hat{V}(s)$ and so any other action a^* will result in a sub-optimal value. Now $|V^*(s) - \hat{V}(s)| \leq \epsilon \implies V^*(s) - \epsilon \leq \hat{V}(s) \leq V^*(s) + \epsilon \quad \forall s \in S$. Substituting $\hat{V}(s)$ in the above inequality we get:

$$\geq \sum_{s'} p(s'|s,a^*)[r(s,a^*) + \gamma V^*(s') - \gamma\epsilon] + \gamma \sum_{s'} p(s'|s,\hat{a})[V_{\hat{V}}(s') - V^*(s') - \epsilon]$$

$$\geq V^*(s) - 2\gamma\epsilon + \gamma \sum_{s'} p(s'|s,\hat{a})[V_{\hat{V}}(s') - V^*(s')]$$

Rearranging the terms in the above inequality we get:

$$V^*(s) - V_{\hat{V}}(s) - \gamma \sum_{s'} p(s'|s,\hat{a})[V^*(s') - V_{\hat{V}}(s')] \leq 2\gamma\epsilon$$

$$\implies L_{\hat{V}}(s) - \gamma \sum_{s'} p(s'|s,\hat{a}) L_{\hat{V}}(s') \leq 2\gamma\epsilon$$

The above is true for all s and its corresponding $\hat{a}$. To prove the rest, it is sufficient to show that the inequality holds for the peak value of $L_{\hat{V}}(s)$. Therefore, let us assume that $\bar{s}$ has the highest $L_{\hat{V}}(s)$ among all $s \in S$ so $L_{\hat{V}}(\bar{s}) \geq L_{\hat{V}}(s') \forall s' \in S$:

$$\implies \sum_{s'} p(s'|\bar{s},\hat{a}) L_{\hat{V}}(\bar{s}) \geq \sum_{s'} p(s'|\bar{s},\hat{a}) L_{\hat{V}}(s')$$

$$\implies L_{\hat{V}}(\bar{s}) \geq \sum_{s'} p(s'|\bar{s},\hat{a}) L_{\hat{V}}(s')$$

$$\implies L_{\hat{V}}(\bar{s}) - \gamma L_{\hat{V}}(\bar{s}) \leq L_{\hat{V}}(\bar{s}) - \gamma \sum_{s'} p(s'|\bar{s},\hat{a}) L_{\hat{V}}(s') \leq 2\gamma\epsilon$$

$$\implies L_{\hat{V}}(\bar{s}) \leq \frac{2\gamma\epsilon}{1-\gamma}$$

Now as the peak value of $L_{\hat{V}}(s)$ is less than $\frac{2\gamma\epsilon}{1-\gamma}$ so for all s we can write:

$$L_{\hat{V}}(s) \leq \frac{2\gamma\epsilon}{1-\gamma}$$

2 Coding Part

2.1 Question 1. Part A.

For this part of the question we initialize a toy bandit with the arm's reward distribution as shown in Fig 1. To evaluate the performance of each algorithm, we ran each sampling method on the same bandit problem for 1000 steps and repeated it for 50 different seeds to obtain mean and standard deviation of each performance metric. For each sampling method, we performed a hyperparameter search to choose the best value of the hyperparameter and then subsequently compared the performance of different algorithms with the best hyperparameter value.

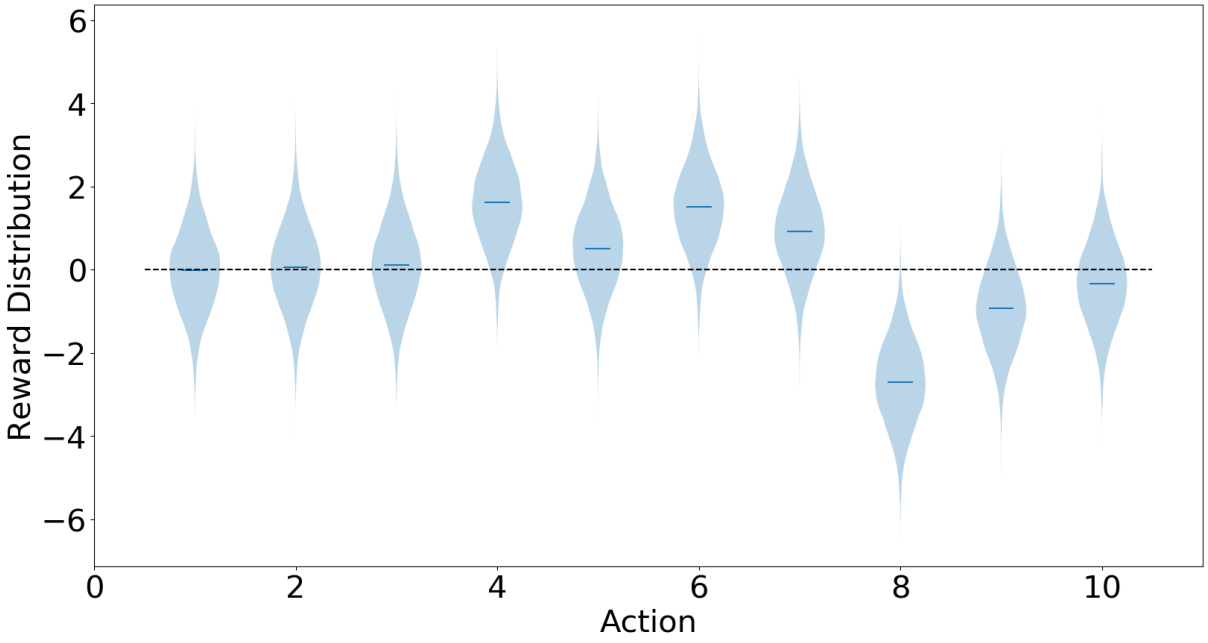

Figure 1: Bandit arm distribution

Regret was defined to be a theoretical quantity, that was calculated using the true mean reward of the bandit's arms. We defined regret for a particular step to be the difference between the true means of the optimal arm and the arm chosen by the algorithm at that step. We ran the algorithm for 10 training steps wherein, the estimates of each arm's reward was updated. Subsequently, the performance of the algorithm was evaluated on 5 testing steps, wherein the arm with the best estimate of mean reward was chosen. Each experiment was run for 1000 steps and repeated for 50 different seeds. All experiments and associated plots can be found in the public Colab notebook.

2.1.1 Boltzmann Sampling

In Boltzmann sampling, we maintain an estimate of the mean reward (Q_i) for each arm and the number of times the arm was tried ($trial_i$). Initially, all estimates are set to 0. The probability of selecting an arm, i, is proportional to $exp(\frac{Q_i}{T})$, where T is the temperature of the system. With each observation, the estimates are updated in a running mean fashion, i.e. $Q_i = \frac{(trial_i - 1) * Q_i}{trial_i} + \frac{reward}{trial_i}$. T controls the extent of exploration of the Boltzmann algorithm. For very large T, all arms are chosen with roughly equal probability and for very small T, the arm with the highest Q estimate is chosen (greedy choice). In our experiments, we used a simulated annealing schedule to decay the T from an initial value. Specifically, temperate at step n is given by: $T_n = \frac{T}{1+n}$

We perform a hyperparameter search to choose the optimal value of T. We compared the performance of the algorithm using 4 metrics, namely training return, testing return, regret and percentage of times

optimal action was chosen. These metrics were averaged over 50 seeds to yield average performance. The experiments and associated plots for 5 different values of T, namely 10, 50, 100, 500 and 1000, can be found in the Colab notebook. $\mathbf{T = 100}$ yielded the best performance among all values that we tried. For lower values of T, the algorithm converged to the greedy policy sooner but often settled for a suboptimal solution (lower training/testing returns or higher regret values or lower percentage of optimal action choice). Higher values of T explored for a very long time and often has very high variability in their step-by-step regret. This is because the algorithm explored for far too long and therefore seldom chose the greedy action. All plots in the associated Colab notebook cell were smoothed using 'gaussian_filter1d' function (with $\sigma = 2$) from the scipy.ndimage.filters library.

2.1.2 Upper Confidence Bound (UCB) Sampling

In UCB sampling, we maintain an estimate of the mean reward (Q_i) for each arm and the number of times the arm was tried ($trial_i$). Initially, all estimates are set to 0. The arm with the highest upper uncertainty bound is selected at each step. Specifically, the arm with the highest H value at step n is selected, where $H_i = Q_i + c * \sqrt{\frac{\ln n}{trial_i}}$. c determines the extent of optimism in choosing an uncertain arm. With each observation, the estimates are updated in a running mean fashion, i.e. $Q_i = \frac{(trial_i - 1) * Q_i}{trial_i} + \frac{reward}{trial_i}$. For very large c, arms with high uncertainty are preferred even if their estimated Q is low and for very small c, the arm with the highest Q estimate is chosen (greedy choice).

We perform a hyperparameter search to choose the optimal value of c. As before, we compared the performance of UCB with different values of c using 4 metrics, namely training return, testing return, regret and percentage of times optimal action was chosen. These metrics were averaged over 50 seeds to yield average performance. The experiments and associated plots for 3 different values of c, namely 0.2, 1 and 5, can be found in the Colab notebook. $\mathbf{c = 1}$ yielded the best performance among all values that we tried. For lower values of c, the algorithm did not explore enough, eventually converging to incorrect estimates of Q and thereby choosing a suboptimal arm while exploiting (lower testing returns, higher regret values and lower percentage of optimal action choice). Higher values of c explored for a long time and eventually had higher regret and lower optimal action choice percentage. The performance during training was far worse for UCB with higher values of c because the algorithm would explore suboptimal arms more often than UCB with $c = 1$. We expected the UCB with higher c values to eventually have similar test performance as UCB with $c = 1$. But the plots showed a high step-by-step variance in the test metrics, which can be attributed to the high exploratory behavior of the algorithm. During the training steps, the algorithm would choose suboptimal arms more often and obtain a biased estimate of Q. All plots in the associated Colab notebook cell were smoothed using 'gaussian_filter1d' function (with $\sigma = 2$) from the scipy.ndimage.filters library.

2.1.3 Thompson Sampling

In Thompson sampling, we maintain a distribution of expected reward (Q_i) for each arm. We choose an arm using the maximum value of the samples obtained from the expected reward distribution of each arm and then update the distribution of the selected arm based on the reward observed. An underlying assumption in the algorithm we used is that we consider a fixed variance($\sigma_{likelihood}$) of the likelihood Gaussian distribution to simplify our computation. Since a Gaussian distribution is the conjugate prior for the Gaussian likelihood, our update equation for the mean and variance of the posterior Gaussian distribution as a function of observed reward ($r_{observed}$) is given by:

$$\sigma^2_{posterior} \leftarrow \frac{1}{1 + 1/\sigma^2_{prior}} \qquad \mu_{posterior} = \frac{1}{1 + 1/\sigma^2_{prior}}(r_{observed} + \frac{\mu_{prior}}{\sigma^2_{prior}})$$

Note that, only the distribution of the arm for which we got maximum expected reward from the sample is updated. Also, during test time we choose the arm with the maximum mean of expected reward distribution. In this part of the problem we have chosen $\sigma_{likelihood}$ as a hyperparameter and the plots of average reward, average regret and percentage of optimal actions can be found in the associated Colab notebook cell. Though this problem doesn't explicitly have a hyperparameter but we tried varying the variance of the likelihood function to see it's effect on the performance. It is interesting to note that the performance of both $\sigma_{likelihood} = 0.1$ and $\sigma_{likelihood} = 1$ are almost similar whereas the performance of the $\sigma_{likelihood} = 10$

is quite poor. We believe that putting more confidence on the observed reward value with small standard deviation of the likelihood makes the convergence of posterior distribution faster. We can see that while the average reward for training increases first and quickly saturates for $\sigma_{likelihood} = 0.1$ and $\sigma_{likelihood} = 1$, it continuously increases and later convergence to the same value for $\sigma_{likelihood} = 10$. We conclude that using a smaller $\sigma_{likelihood}$ is preferable for faster convergence and gives better results.

2.1.4 Comparison of sampling algorithms

As required in the question, we compared the performance of all algorithms on the same bandit problem and plotted 3 metrics, namely training return, testing return, training regret and testing regret. The plots can be found in the associated Colab notebook. All algorithms yielded similar performance for testing returns. However, Boltzmann exploration yielded worse performance in terms of training return and train/test regret. This is because Boltzmann exploration required more exploration steps to decide the best arm to exploit. UCB performed the best among all algorithms, and only slightly better than Thompson sampling in the initial steps. However, as we observed before Thompson sampling is more robust to the choice of hyperparameter σ_{prior} as compared to UCB. These design choices would impact the algorithm to be deployed depending on the bandit problem to be tackled.

2.2 Question 2. Part B.

For both the parts of this question we have used asynchronous updates. Let us consider the number of states to be $|S|$ and the number of actions to be $|A|$. Our implementation of tabular policy iteration and tabular value iteration can be found in the respective Colab notebook cells. The working of the algorithm for each iteration can be visualized using the functions *policy_iteration*() and *value_iteration*(). It shows both the policy and visualises the value functions for each state. For both the algorithms, we have used asynchronous updates by using in-place dynamic programming which helped in faster convergence. To evaluate the performance of each environment, we used 90 iterations and 5 different seeds to obtain the mean and standard deviation of the metrics. The time and space complexity of the algorithm are given below:

Algorithms	Time Complexity	Space Complexity
Value Iteration	$\mathcal{O}(\|S\|^2 \times \|A\|)$	$\mathcal{O}(\|S\| \times \|A\|)$
Policy Iteration	$\mathcal{O}(\|S\|^2 \times \|A\|)$	$\mathcal{O}(\|S\| \times \|A\|)$

Table 1: Time and Space Complexity of the algorithms

To evaluate the training and testing behavior of the algorithm, we used a slightly different policy than mentioned in the assignment. Since the environments used in our experiments were relatively simple and tractable, we trained or updated the policy for 1 iteration and then tested the performance of that policy in the environment for 5 iterations. The testing metrics were averaged over these 5 iterations to observe the improvement in policy with time.

2.2.1 Frozen Lake

To test our value iteration and policy iteration algorithms we used two environments. The first environment is the Frozen lake 8×8. Note that we used a deterministic environment by setting the slippery flag to be 0 for our experiments. We have used a discount factor of 0.9. The plots of cumulative reward and total steps per episode are obtained in this problem setting and they can be found in this Colab notebook cell. Also, we verified the empirical optimal policy and the expected reward with the theoretically obtained values. As illustrated in the plots, policy iteration converged faster than value iteration.

2.2.2 Grid World

The second environment is the Grid world where we set our own toy problem as shown in the Fig 2. The outcomes of actions are deterministic in this problem and cell A transports the agent to cell B with probability

1 irrespective of the action chosen but will only get a reward of 2 if it chooses to go down. We have used a discount factor of 0.9. We have designed a wrapper class of the emdp library by Zafarali Ahmed with a custom function where we can design the grid problem. Also when the episodes are run, the initial state to start is chosen randomly. The plots of cumulative reward and total steps per episode are obtained in this problem setting and they can be found here. We also verified the empirical optimal policy and the expected reward with the theoretically obtained values for this part. Once again, policy iteration converged faster than value iteration. Value iteration plots indicate that the algorithm momentarily prefers to exploit the +2 reward multiple times as that is a local optima in the environment.

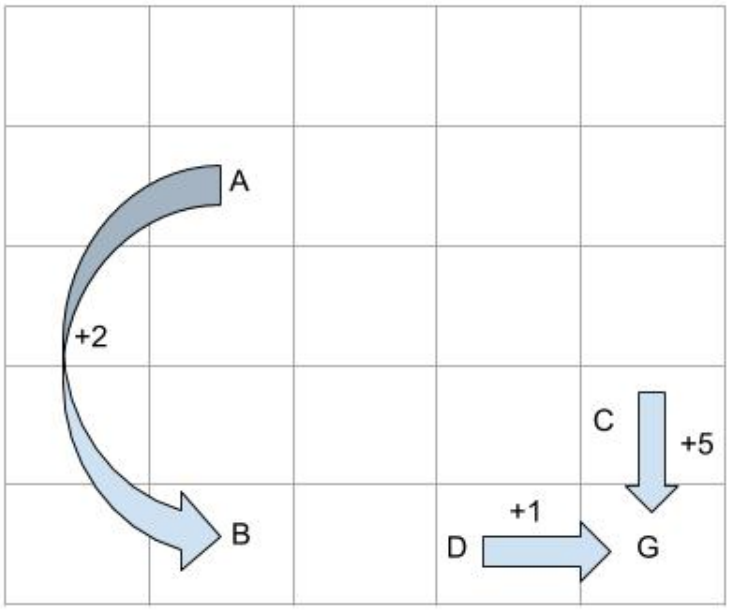

Figure 2: Gridworld toy problem design

Contributions

AKM and AG worked on the theory questions together. AG implemented the Boltzmann and UCB sampling algorithms. AKM implemented the Thompson sampling algorithm. AG implemented the policy and value iteration for FrozenLake environment and AKM implemented the custom GridWorld environment, which was thereafter used for evaluating policy and value iteration algorithms. Both AG and AKM contributed equally to writing the report.

Reproducibility Checklist

- ☒ Description of algorithms and environments
- ☐ Time and space complexity for bandit algorithms – **not applicable**
- ☒ Time and space complexity for MDP
- ☒ Link to code – added link to Colab notebook where applicable
- ☐ Description of data collection process - **not applicable**
- ☒ Link to download dataset or environment
- ☐ Explanation of excluded data – **not applicable**

☒ Explanation of train/test splitting

☒ Range of hyperparameters and method to select best hyperparameter

☒ Number of evaluation runs

☒ Description of experiments

☒ Definition of metrics used

☒ Error bars in plots – plots indicate standard deviation bars

☒ Result description with mean and standard deviation – shown in plots

☒ Description of computing infrastructure used – Colab notebook provided to enhance reproducibility of results