[COMP6247] Lab Three Report

Feng Xie — fx1n18@soton.ac.uk — Student ID:30502322

1 Question One

Each site can only contain one flag or no flag and the hider should choose two sites to hide her flags at each round. Therefore, after 100 rounds, the total times of the flags hidden at each site should equal to 200. And for each site, the total times of flags hidden at this site should be no more than 100.

From the analysis above, we can know that the totals of (47,53,31,36,34) for each site are incorrect because the total number is 47 + 53 + 31 + 36 + 34 = 201 which is bigger than 200. And the totals of (102, 26, 24, 40, 8) are also incorrect because 102 is bigger than 100.

2 Question Two

When the number of times the hider left the flags at each site is given by the distribution (36, 24, 40, 63, 37), the best fixed strategy of the seeker in hindsight is choosing site 3 and site 4 which correspond to 40 and 63 in the distribution. Because in hindsight, the two sites with the highest numbers are site 3 and site 4 so it is more likely that according to this distribution, the hider will choose these two sites more often in the future.

3 Question Three

3.1 Approach

3.1.1 Uniform random opponent

If the opponent follows a uniform-random policy, the 2 flags will be uniformly randomly placed at the start of each round. The code was written as follows.

```
def uniform_random_opponent():
    num_of_sites = 5
    sites = np.zeros(num_of_sites)
    while True:
        flags = np.random.choice(num_of_sites, size=2)
        if flags[0] != flags[1]:
            break
    sites[flags] = 1
    return sites
```

3.1.2 The opponent with an epsilon-greedy policy

When the opponent follows an epsilon-greedy policy, it will try to minimize the average reward of the seeker. The epsilon was set to 0.2. At the start of each round, the opponent observes what the reward the seeker has got for each site, and chooses two sites with the least rewards with the probability of 0.8 to hide the flags and chooses two random sites with the probability epsilon(i.e. 0.2). The code was written as follows.

```
def epsilon_greedy_opponent(R):
    # R: the rewards that the seeker has got for each site
    num_of_sites = 5
    sites = np.zeros(num_of_sites)
    epsilon = 0.2
    if np.random.uniform() < epsilon:
        while True:
```

```
            flags = np.random.choice(num_of_sites, size=2)
            if flags[0] != flags[1]:
                break
    else:
        flags = np.argsort(R)[0:2]
    sites[flags] = 1
    return sites
```

3.1.3 The opponent with a FPL policy

When the opponent follows a FPL policy, at the start of each round, it observes what the reward the seeker has got for each site and generate a random noise from an exponential distribution for each site. And it computes the sum of the rewards observed and the noises and chooses two sites with the lowest values to hide the flags. The code was written as follows. The scale of the exponential distribution was set to 1.

```
def FPL_opponent(R):
    # R: the rewards that the seeker has got for each site
    num_of_sites = 5
    sites = np.zeros(num_of_sites)
    Z = np.random.exponential(scale=1, size=num_of_sites)
    flags = np.argsort(R + Z)[0:2]
    sites[flags] = 1
    return sites
```

3.1.4 The guessing algorithm for the seeker

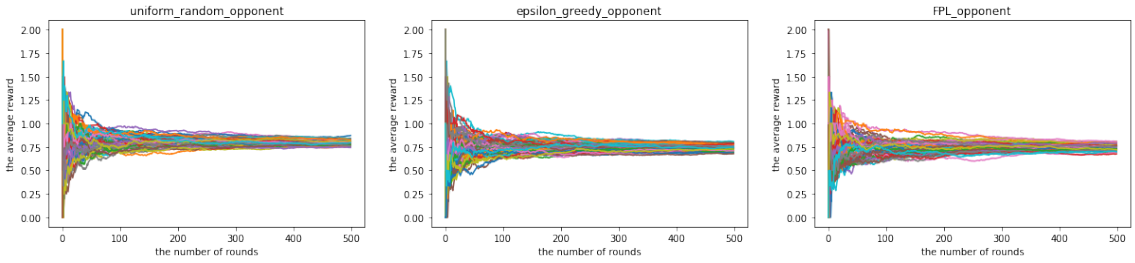
The Exp3 was chosen as the guessing algorithm for the seeker because the regret of Exp3 is at most $O(\sqrt{KTlogK})$ which is proportional to $O(\sqrt{T})$.

The algorithm was played with an opponent for 500 rounds and the average reward of the seeker was plotted over the number of plays. This was repeated for 100 times to make sure we can get statistical significance. For each time, the weight ω for each site was reset to 1.0. And weights were updated during each of the 500 rounds.

The implementation of the Exp3 was added in the appendix. The parameter γ in the Exp3 was set to 0.3.

3.2 Results

From the figure 1, we can see that after running the process for 100 times, the average reward of the seeker seems to converge to a value close to 0.75 over the number of plays. It didn't matter if we play the guessing algorithm against an opponent with a uniform-random policy, an epsilon-greedy policy or a FPL policy. It looks that the converged value of the algorithm with a uniform-random opponent is a bit higher than the results of playing against the other two opponents.



(a) (b) (c)

Figure 1: The average reward of the seeker with different opponents

4 Conclusion

When the opponent follows a uniform-random policy, it doesn't use the information of rewards from the seeker. When the opponent applies an epsilon-greedy policy or FPL policy, it always tries to minimise

the average reward the seeker will get. But we can see from the results that using Exp3 as the guessing algorithm for the seeker helps achieve an average reward of about 0.75 when playing against the three different opponents.

5 Appendix

```
def play_exp3(opponent_policy):
    gamma = 0.3
    num_of_sites = 5
    num_of_rounds = 500
    num_of_times = 100
    for _ in range(num_of_times):
        average_reward = []
        w = np.ones(num_of_sites)
        R = np.zeros(num_of_sites)
        total_reward = 0
        for i in range(num_of_rounds):
            x = np.zeros(num_of_sites)
            if opponent_policy.__name__ in ['epsilon_greedy_opponent', 'FPL_opponent']:
                sites_reward = opponent_policy(R)
            else:
                sites_reward = opponent_policy()
            p = (1 - gamma) * (w / sum(w)) + np.ones(num_of_sites) * gamma / num_of_sites
            while True:
                choices_index = np.random.choice(num_of_sites, size=2, p=p)
                if choices_index[0] != choices_index[1]:
                    break
            x[choices_index] = sites_reward[choices_index] / p[choices_index]
            w = w * np.exp(gamma * x / num_of_sites)
            R[choices_index] += sites_reward[choices_index]
            average_reward.append(sum(R) / (i + 1))
        plt.plot(average_reward)
        plt.xlabel('the number of rounds')
        plt.ylabel('the average reward')
        plt.title(opponent_policy.__name__)
```

Listing 1: The implementation of Exp3