# [COMP6247] Lab Four Report

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

## 1 Question One

Since the game setting is asymmetric, the hider is able to make use of how many rewards the seeker has got for each site and choose where to hide the flags in order to minimise the total reward of the seeker.

### 1.1 Approach

The standard FPL and Exp3 were chosen as the learning algorithm for both players.

In the first experiment, FPL was used for both players. At each round, the hider will receive the information from the seeker which tells her how many rewards the seeker has got for each site. The hider will apply FPL to make decisions based on this information and choose the two sites with the lowest values to hide flags, trying to minimise the reward of the seeker. The code showing how the hider hides flags is shown as below.

```
def FPL_hider(R, eta):
    # 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=eta, size=num_of_sites)
    flags = np.argsort(R + Z)[0:2]
    sites[flags] = 1
    return sites
```

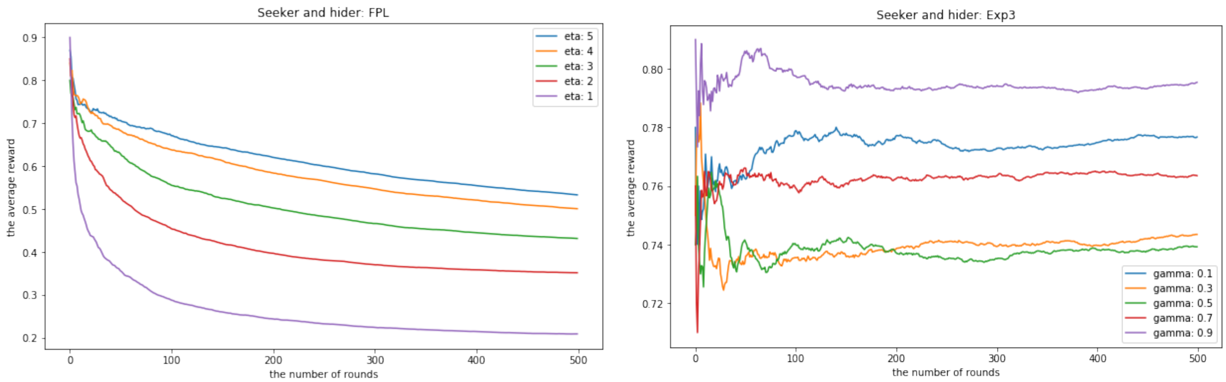
In the second experiment, Exp3 was used for both players. At each round, the hider will receive the weights information from the seeker and apply Exp3 to choose two sites to hide her flags. The hider has the probability of 1 - gamma to choose the two sites with the lowest weights to hide flags. The code showing how the hider hides flags is shown as below.

```
def exp3_hider(gamma, weights, num_of_sites):
    # weights: The weights that the seeker has gained for each site.
    sites = np.zeros(num_of_sites)
    p = weights / sum(weights)
    if np.random.uniform() < gamma:
        flags = np.random.choice(num_of_sites, size=2, replace=False)
    else:
        flags = np.argsort(p)[0:2]
    sites[flags] = 1
    return sites
```

### 1.2 Results

Each game consists of 500 rounds and the game was repeated for 100 times. For the FPL and Exp3 algorithms, different values of the parameters(i.e. eta and gamma) were tried.

From the figure 1, we can see that the average reward converged over the number of rounds for both algorithms. As the eta value increased, the average reward converged slower and it converged to a higher value. The probable reason for this is that as the noise becomes larger and larger, FPL becomes an algorithm which is more likely to make random decisions. When the gamma value increased, the converged value of the average reward first decreased and then increased. The gamma is a tradeoff between exploration and exploitation. When the gamma is too large, exp3 becomes an algorithm that makes uniform random decisions.




(a) Seeker and hider: FPL.

(b) Seeker and hider: Exp3

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

## 2 Question Two

### 2.1 Approach

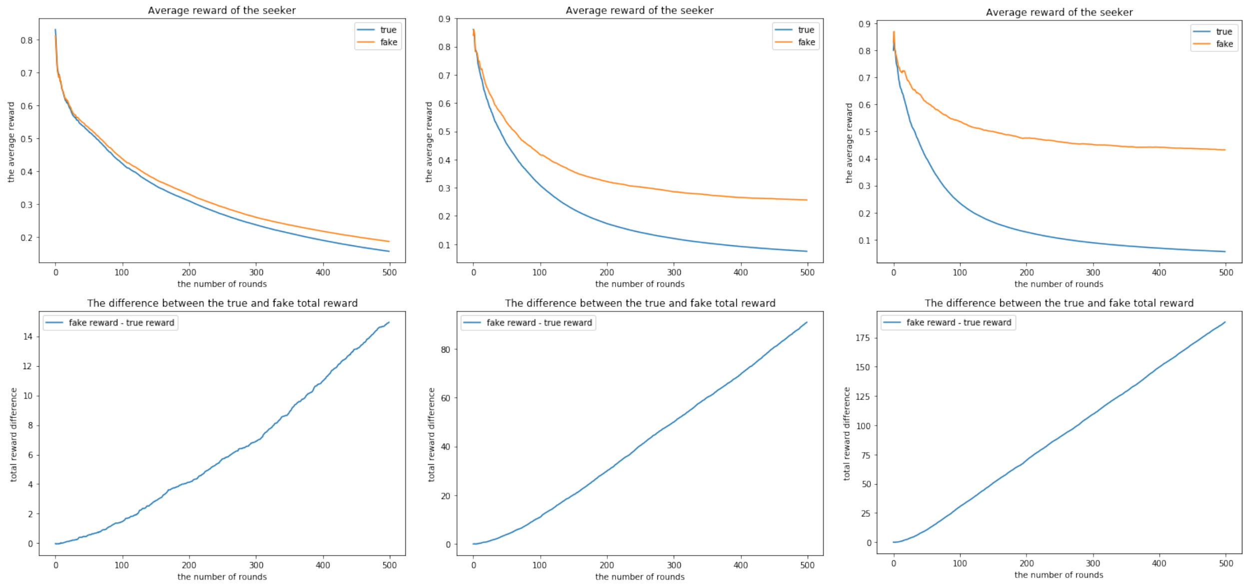
The hider can lie about the position of the true flags during the game. She can lie up to B times during a game. A random faking strategy was used to decide when to lie. At the start of each game, the faking timings were randomly decided as the code shown below.

```
# assign the fake timings
# MAX_N_FAKE: the number of fake times in a game.
fake_timings = np.random.choice(500, size=MAX_N_FAKE, replace=False)
```

The standard FPL and Exp3 were chosen as the learning algorithms. Two experiments were done. In the first experiment, both the seeker and hider chose FPL as the learning algorithm. For the hider, in order to make different decisions when she is faking, she needs to make use of the true rewards and the fake rewards which the seeker has got. When she is faking, she will say the flags are hidden in the two sites which can help the seeker maximise rewards. And the true sites for hiding the flags are the ones that can minimise the rewards of the seeker. The code written for the hider is shown as follows.

```
def FPL_hider(R_t, R_f, fake, eta):
    '''
    :param R_t: the true rewards that the seeker has got for each site.
    :param R_f: the fake rewards that the seeker has got for each site.
    :param fake: a boolean value which signifies whether to fake or not.
    '''
    num_of_sites = 5
    true_sites = np.zeros(num_of_sites)
    Z = np.random.exponential(scale=eta, size=num_of_sites)
    # assign true flags
    t_flags = np.argsort(R_t + Z)[0:2]
    true_sites[t_flags] = 1
    # assign fake flags
    if fake:
        fake_sites = np.zeros(num_of_sites)
        # Give the seeker what she wants so that she tries to maximise the fake rewards.
        f_flags = np.argsort(R_f + Z)[-2:]
        fake_sites[f_flags] = 1
    else:
        fake_sites = true_sites
    return fake_sites, true_sites
```

In the second experiment, both the seeker and hider chose Exp3 as the learning algorithm. For the hider, as what was discussed above, she needs to make use of the fake information and the true information. Here the information is the weights in the Exp3 algorithm. When the hider is faking, she will claim that the two sites are the ones with the higher fake weights and in fact the flags are hidden in the two sites with the lower true weights. Both the true and fake weights were updated at each round.




(a) Number of B(fake time): 10 (b) Number of B(fake time): 50 (c) Number of B(fake time): 100

Figure 2: Hider: FPL; Seeker: FPL

## 2.2 Results

For the FPL algorithm, the eta in the exponential distribution was set to 2. For the Exp3 algorithm, the gamma value was set to 0.3. The average reward of the seeker was plotted over the number of rounds. And the difference between the true total reward and the fake total reward of the seeker over the number of rounds was also plotted. Here the difference refers to the value of (fake total reward - true total reward).

It can be seen from the figure 2 that as the number of B increased, the gap between the true average reward and the fake average reward of the seeker became larger and the converged point seems to get earlier. As for the difference of the total reward, it is shown that the difference value continued to increase over the number of rounds and the total reward difference at the end of the game increased dramatically as the B value(fake times) increased.

For the results of Exp3, similar trends can be found in the figure 3(the figure is in the next page) compared with the figure 2. The difference is that the gap between the true average reward and the fake average reward is smaller than the gap when FPL was used. Besides, the difference value between the true and fake total reward is not as large as the one when we applied FPL to both players.

It can be concluded that the hider can manipulate the seeker to have a low performance at the end of the game by using the faking strategy.

# 3 Appendix

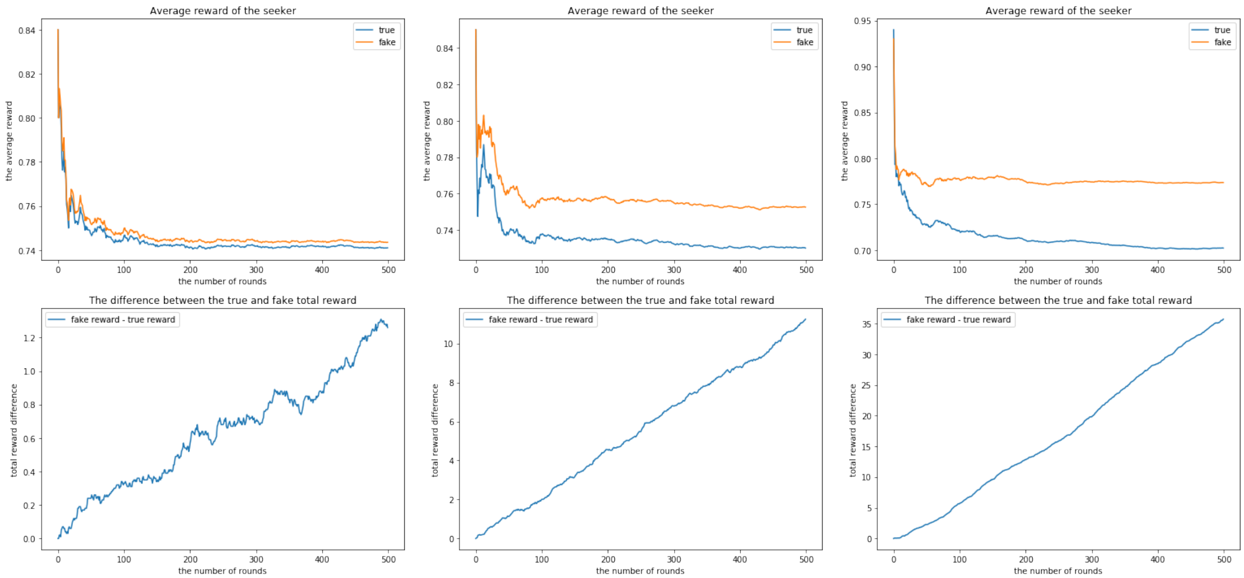
The code for the two questions.

## 3.1 Question 1

```python
import numpy as np
import matplotlib.pyplot as plt

def FPL_hider(R, eta):
    # 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=eta, size=num_of_sites)
    flags = np.argsort(R + Z)[0:2]
    sites[flags] = 1
    return sites

```




(a) Number of B(fake time): 10 (b) Number of B(fake time): 50 (c) Number of B(fake time): 100

Figure 3: Hider: Exp3; Seeker: Exp3

```python
def play_fpl(opponent_policy, eta=1):
    all_rewards = []   # reward: 100 * 500
    for _ in range(100):
        rewards_list = []
        R = np.zeros(5)  # reset rewards of the seeker
        for i in range(500):
            sites_reward = opponent_policy(R, eta)
            Z = np.random.exponential(scale=eta, size=5)
            choices_index = np.argsort(R + Z)[-2:]
            rewards = sites_reward[choices_index]
            R[choices_index] += rewards
            rewards_list.append(sum(R) / (i + 1))
        all_rewards.append(rewards_list)
    all_rewards = np.asarray(all_rewards)
    average_rewards = np.average(all_rewards, axis=0)  # compute the average reward for every round
    plt.plot(average_rewards, label=('eta: '+str(eta)))
    plt.xlabel('the number of rounds')
    plt.ylabel('the average reward')
    plt.title('Seeker and hider: FPL')
    plt.legend()

def exp3_hider(gamma, weights, num_of_sites):
    # weights: The weights that the seeker has gained for each site.
    sites = np.zeros(num_of_sites)
    p = weights / sum(weights)
    if np.random.uniform() < gamma:
        flags = np.random.choice(num_of_sites, size=2, replace=False)
    else:
        flags = np.argsort(p)[0:2]
    sites[flags] = 1
    return sites


def play_exp3(opponent_policy, gamma=0.3):
    num_of_sites = 5
    num_of_rounds = 500
    num_of_times = 100
    all_rewards = []   # reward: 100 * 500
    for _ in range(num_of_times):
        rewards_list = []
        w = np.ones(num_of_sites)  # initialize the weights
        R = np.zeros(num_of_sites)
```

```python
        for i in range(num_of_rounds):
            x = np.zeros(num_of_sites)
            sites_reward = opponent_policy(gamma, w, num_of_sites)
            p = (1 - gamma) * (w / sum(w)) + np.ones(num_of_sites) * gamma / num_of_sites
            # draw a uniform sample of size 2 without replacement
            choices_index = np.random.choice(num_of_sites, size=2, p=p, replace=False)
            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]
            rewards_list.append(sum(R) / (i + 1))
        all_rewards.append(rewards_list)  # add the average reward for the current episode
    all_rewards = np.asarray(all_rewards)
    # print(all_rewards)
    average_rewards = np.average(all_rewards, axis=0)  # compute the average reward for every round
    plt.plot(average_rewards, label=('gamma: '+str(gamma)))
    plt.xlabel('the number of rounds')
    plt.ylabel('the average reward')
    plt.title('Seeker and hider: Exp3')
    plt.legend()
```

## 3.2 Question 2

```python

def play(game, *args):
    all_rewards_t = []   # reward: 100 * 500
    all_rewards_f = []
    total_diff = []  # difference between the true reward and fake reward
    for _ in range(100):
        rewards_list_t, rewards_list_f, diff = game(*args)
        total_diff.append(diff)
        all_rewards_t.append(rewards_list_t)
        all_rewards_f.append(rewards_list_f)
    all_rewards_t = np.asarray(all_rewards_t)
    all_rewards_f = np.asarray(all_rewards_f)
    total_diff = np.asarray(total_diff)
    total_diff = np.average(total_diff, axis=0)
    average_rewards_t = np.average(all_rewards_t, axis=0)  # compute the average reward for every round
    average_rewards_f = np.average(all_rewards_f, axis=0)
    plt.figure(figsize=[8, 12])
    plt.subplot(211)
    plt.plot(average_rewards_t, label='true')
    plt.plot(average_rewards_f, label='fake')
    plt.title('Average reward of the seeker')
    plt.xlabel('the number of rounds')
    plt.ylabel('the average reward')
    plt.legend()
    plt.subplot(212)
    plt.plot(total_diff, label='fake reward - true reward')
    plt.title('The difference between the true and fake total reward')
    plt.xlabel('the number of rounds')
    plt.ylabel('total reward difference')
    plt.legend()

def fpl_vs_fpl(MAX_N_FAKE, eta):
    rewards_list_t = []
    rewards_list_f = []
    diff = []  # total fake reward - total true reward
    # reset rewards of the seeker
    R_t = np.zeros(5)  # true rewards
    R_f = np.zeros(5)  # fake rewards
    fake_timings = np.random.choice(500, size=MAX_N_FAKE, replace=False)  # assign the fake timings
    fake_timings = sorted(fake_timings, reverse=True)  # descending order
    for i in range(500):
        fake = False
        if len(fake_timings) > 0 and fake_timings[-1] == i:
            fake = True
            fake_timings.pop()
        fake_sites, true_sites = FPL_hider(R_t, R_f, fake, eta)
        Z = np.random.exponential(scale=eta, size=5)
```

```
        # the seeker chooses 2 sites based on the fake rewards
        choices_index = np.argsort(R_f + Z)[-2:]
        f_rewards = fake_sites[choices_index]
        t_rewards = true_sites[choices_index]
        R_t[choices_index] += t_rewards
        R_f[choices_index] += f_rewards
        rewards_list_t.append(sum(R_t) / (i + 1))
        rewards_list_f.append(sum(R_f) / (i + 1))
        diff.append(sum(R_f) - sum(R_t))
    return rewards_list_t, rewards_list_f, diff

def exp3_vs_exp3(MAX_N_FAKE, gamma):
    num_of_sites = 5
    rewards_list_t = []
    rewards_list_f = []
    diff = []  # total fake reward - total true reward
    # reset rewards of the seeker
    R_t = np.zeros(5)  # true rewards
    R_f = np.zeros(5)  # fake rewards
    w_f = np.ones(5)  # initialize the fake weights
    w_t = np.ones(5)  # initialize the true weights
    fake_timings = np.random.choice(500, size=MAX_N_FAKE, replace=False)  # assign the fake timings
    fake_timings = sorted(fake_timings, reverse=True)  # descending order
    for i in range(500):
        fake = False
        if len(fake_timings) > 0 and fake_timings[-1] == i:
            fake = True
            fake_timings.pop()
        fake_sites, true_sites = Exp3_hider(w_t, w_f, fake, gamma)
        p_f = (1 - gamma) * (w_f / sum(w_f)) + np.ones(num_of_sites) * gamma / num_of_sites
        p_t = (1 - gamma) * (w_t / sum(w_t)) + np.ones(num_of_sites) * gamma / num_of_sites
        # draw a uniform sample of size 2 without replacement
        choices_index = np.random.choice(num_of_sites, size=2, p=p_f, replace=False)
        # update the fake weights
        x_f = np.zeros(num_of_sites)
        x_f[choices_index] = fake_sites[choices_index] / p_f[choices_index]
        w_f = w_f * np.exp(gamma * x_f / num_of_sites)
        # update the true weights
        x_t = np.zeros(num_of_sites)
        x_t[choices_index] = true_sites[choices_index] / p_t[choices_index]
        w_t = w_t * np.exp(gamma * x_t / num_of_sites)
        # accumulate rewards
        R_t[choices_index] += true_sites[choices_index]
        R_f[choices_index] += fake_sites[choices_index]
        # compute the average reward for the current step
        rewards_list_t.append(sum(R_t) / (i + 1))
        rewards_list_f.append(sum(R_f) / (i + 1))
        diff.append(sum(R_f) - sum(R_t))
    return rewards_list_t, rewards_list_f, diff

def FPL_hider(R_t, R_f, fake, eta):
    '''
    :param R_t: the true rewards that the seeker has got for each site
    :param R_f: the fake rewards that the seeker has got for each site
    :param fake: a boolean value which signifies whether to fake or not
    '''
    num_of_sites = 5
    true_sites = np.zeros(num_of_sites)
    Z = np.random.exponential(scale=eta, size=num_of_sites)
    # assign true flags
    t_flags = np.argsort(R_t + Z)[0:2]
    true_sites[t_flags] = 1
    # assign fake flags
    if fake:
        fake_sites = np.zeros(num_of_sites)
        # Give the seeker what she wants so that she tries to maximise the fake rewards.
        f_flags = np.argsort(R_f + Z)[-2:]
        fake_sites[f_flags] = 1
    else:
        fake_sites = true_sites
    return fake_sites, true_sites
```

```python

def Exp3_hider(w_t, w_f, fake, gamma):
    '''
    :param w_t: True weights.
    :param w_f: Fake weigths.
    :param fake: a boolean value which signifies whether to fake or not.
    '''
    num_of_sites = 5
    # assign true sites, minimise the reward of the seeker
    p_t = w_t / sum(w_t)
    true_sites = np.zeros(num_of_sites)
    if np.random.uniform() < gamma:
        t_flags = np.random.choice(num_of_sites, size=2, replace=False)
    else:
        t_flags = np.argsort(p_t)[0:2]
    true_sites[t_flags] = 1
    # assign fake sites, maximise the reward of the seeker
    p_f = w_f / sum(w_f)
    if fake:
        fake_sites = np.zeros(num_of_sites)
        f_flags = np.argsort(p_f)[-2:]
        fake_sites[f_flags] = 1
    else:
        fake_sites = true_sites
    return fake_sites, true_sites
```