Homework 10
Colorado CSCI 5454

Sai Siddhi Akhilesh Appala

November 30, 2022

People I studied with for this homework: Pradyumnna

Problem 1

We know that from class notes and lectures if a series of losses occurs where the losses are marginally greater than the best approach. The average daily regret of the algorithm is close to the ideal value, which is close to zero. lets solve it my taking an example

- where there are 2 events and the say the first one has loss as 0 and for the second one loss in t^{th} round is $1/2^t$.
- if we say, the algorithm always chooses the second event, it won't be choosing the optimal at any round.
- Then the total loss is $L_A = \sum_{t=1}^{T} 1/2^t$ and $OPT = 0$ when we select the first one in all rounds.
- Then the average per day represented by L_{avg} given below

$$\begin{aligned} L_{avg} &= (L_A - OPT)/T \\ &= 1/T * \sum_{t=1}^{T} 1/2^t \\ &\leq 1/T * \sum_{t=1}^{\infty} 1/2^t \\ &= 1/T * 1 \ \ (By \ \ infinity \ \ GP \ \ formula) \\ &= 1/T \end{aligned} \tag{1}$$

When $L_{avg} = 0$ for $T \to \infty$. From above we can say that it's possible to have an algorithm that never picks the solution and has an average regret of 0.

Problem 2

Part a

- Consider two actions x and y on t^{th} day.
- $Pr_x^t = \frac{w_y^t}{W_t}$ and $Pr_y^t = \frac{w_y^t}{W_t}$
- $w_x^t = e^{-\epsilon \sum_{s=1}^{t-1} l_x^s}$ and $w_y^t = e^{-\epsilon \sum_{s=1}^{t-1} l_y^s}$
- where l_x^s, l_y^s represent loss in choosing action x and y on day s respectively,

$$
\begin{aligned}
r &= Pr(x)/Pr(y) \\
&= w_x^t / w_y^t \\
&= e^{-\epsilon \sum_{s=1}^{t-1} l_x^s} / e^{-\epsilon \sum_{s=1}^{t-1} l_y^s} \\
&\leq e^{-\epsilon \sum_{s=1}^{t-1} 0} / e^{-\epsilon \sum_{s=1}^{t-1} l_j^s} \\
&= e^0 / e^{-\epsilon \sum_{s=1}^{t-1} l_j^s} \\
&= 1 / e^{-\epsilon \sum_{s=1}^{t-1} l_j^s} \\
&= 1 / e^{-\epsilon(t-1)} \\
&= 1 / e^{-(t-1)/T} \\
&\leq 1 / e^{-1} \quad (\because t - 1 \leq T) \\
&= e
\end{aligned}
$$

Therefore by the above equation ratio of any action probabilities is less than e.

Part b

We know that regret R is given by $E[L_T] - OPT$.
In our case $OPT = 0$, by choosing action 1 all the time.

$$
\begin{aligned}
R &= E[L_T] - OPT \\
&= E[L_T]
\end{aligned}
$$

- lets say we have 2 cases here those are p_1^i and p_2^i which represents probability of 1 or 2 at round i .
- As we all know that multiplicative weights algorithm has more preference for lower loss actions.
- At any round 1 we will be having a lower loss of 0.

- We know that $p_1^i + p_2^i = 1$. as we only chose 2, we need to find the 2nd one lower bound.
- From part we know the relationship between p_1 and p_2 is e. we will substitute that in the below equations

$$\begin{aligned} p_1^i + p_2^i &= 1 \\ \Rightarrow e * p_2^i + p_2^i &\geq 1 \\ \Rightarrow p_2^i &\geq 1/(e+1) \end{aligned}$$

- Now we will find $E[L^i]$. It's a Bernoulli random variable with value 1 with p_2 and 0 with p_0 and i based on the rounds. so out $E[L^i]$ so we need to find thep_2 from below.

$$\begin{aligned} E[L^i] &= 1 * p_2^i + 0 * p_1^i \\ &= p_2^i \end{aligned}$$

- Now we will represent the expectation as below

$$\begin{aligned} E[L_T] &= E[\sum_{i=1}^{T} L_T] \\ E[L_T] &= E[\sum_{i=1}^{T} L_i] \\ E[L_T] &= \sum_{i=1}^{T} E[L_i] \\ E[L_T] &= \sum_{i=1}^{T} p_2^i \\ E[L_T] &\geq \sum_{i=1}^{T} 1/(e+1) \\ E[L_T] &= T/(e+1) \end{aligned}$$

- if we substitute the above values in our starting equation, we get below

$$\begin{aligned} R &= E[L_T] \\ &\geq T/(e+1) \end{aligned}$$

As from above, we have seen a constant i.e $C = 1/(e+1)$ so the regret is $\Omega(T)$.

part c

In the same way if we take 2 actions i.e x and y, We know that regret R is given by $E[L_T] - OPT$.
In our case $OPT = T/2$, by choosing action any one of the 2 actions so it will be t/2.

$$R = E[L_T] - OPT$$
$$R = E[L_T] - T/2$$

- Lets start with the loss for action x and action y for i^{th} round are respectively l_x^i and l_y^i.
- lets consider the example of having the even and odd one for x as below

$$l_x^i = \begin{cases} 0 \ \ (if \ \ i \ \ is \ \ odd) \\ 1 \ \ (for \ \ other \ \ case) \end{cases}$$

- lets consider the example of having the even and odd one for y as below

$$l_y^i = \begin{cases} 0 \ \ (if \ \ i \ \ is \ \ even) \\ 1 \ \ (for \ \ other \ \ case) \end{cases}$$

- Let's calculate the probabilities based on the conditions for x and y and respectively for p_x^i and p_y^i for odd and even cases.
- when i is odd case for p_x and p_y.

$$\begin{aligned} p_x^i / p_y^i &= e^{-\epsilon \sum_{s=1}^{i-1} l_x^s} / e^{-\epsilon \sum_{s=1}^{i-1} l_y^s} \\ &= e^{-\epsilon * (\sum_{s=1}^{i-1} l_1^s - \sum_{s=1}^{i-1} l_2^s)} \\ &= e^{-\epsilon * \sum_{s=1}^{i-1} (l_1^s - l_2^s)} \\ &= e^{-\epsilon * 0} \\ &= 1 \end{aligned}$$

- when i is other cases for p_x and p_y.

$$\begin{aligned} p_x^i / p_y^i &= e^{-\epsilon \sum_{s=1}^{i-1} l_1^s} / e^{-\epsilon \sum_{s=1}^{i-1} l_2^s} \\ &= e^{-\epsilon * (\sum_{s=1}^{i-1} l_1^s - \sum_{s=1}^{i-1} l_2^s)} \\ &= e^{-\epsilon * \sum_{s=1}^{i-1} (l_1^s - l_2^s)} \\ &= e^{-\epsilon * -1)} \ (By \ \ equation \ \ 11 \\ &= e^{\epsilon} \end{aligned}$$

- When i is even, L_i is a Bernoulli random variable, then we define the expectation as

$$E[L_i] = \begin{cases} e^\epsilon/(1+e^\epsilon) \ \ (if \ \ i \ \ is \ \ even) \\ 1/2 \ \ (for \ \ other \ \ case) \end{cases}$$

- We know that L_T be the total loss as a summation of independent ones like below

$$L_T = \sum_{i=1}^{T} L_i \tag{2}$$

From above all we can define the $E[L_{total}]$ as

$$\begin{aligned} E[L_{total}] &= E[\sum_{i=1}^{T} L_i] \\ &= \sum_{i=1}^{T} E[L_i] \\ &= \sum_{i=odd}^{<=T} E[L_i] + \sum_{i=even}^{<=T} E[L_i] \\ &= \sum_{i=odd}^{<=T} 1/2 + \sum_{i=even}^{<=T} e^\epsilon/(1+e^\epsilon) \\ &= T/4 + T/2 * (e^\epsilon/(1+e^\epsilon)) \end{aligned} \tag{3}$$

if we substitute the above values in our 1st point to get the regret

$$\begin{aligned} R &= E[L_T] - T/2 \\ R &= T/4 + T/2 * (e^\epsilon/(1+e^\epsilon)) - T/2 \\ R &= T/2 * (e^\epsilon/(1+e^\epsilon)) - T/4 \end{aligned} \tag{4}$$

From above, there exist positive constants which make $R >= C * T$, $\forall \ T >= N$, so the regret, in this case, is $\Omega(T)$.

part d

we chose ϵ as 1 as we want to be as high as possible

Problem 3

Part a

Execution PDF is attached

```
def probs_from_weights( wts ) :
    total = sum ( wts )
    probArr = []
    for i in wts:
        probArr.append(i/total)
    return probArr
```

0.1 Part b

Execution PDF is attached

```
def best_response ( probArr ) :
    values = []
    for p in range (len(probArr)) :
        tmp =0.0
        for q in range(len( probArr ) ) :
            tmp = tmp + u_matrix[p][q]* probArr[q]
        values.append(tmp)
    max_value = max(values)
    opt_action = values.index(max_value)
    return opt_action
```

Part c

Execution PDF is attached

```
def util_of_each_action(response) :
    util =[]
    for i in range(len(u_matrix)):
        util.append(u_matrix[i][response])
    return util
```

Part d

Execution PDF is attached

```
def convert_utils_to_losses(utilities):
    losses =[]
```

```
    for i in range(len(utilities)):
        losses.append(-1 * (utilities[i]/2) +0.5)
    return losses
```

Part e

Execution PDF is attached

```
def update_weights(wts, lose, epsilon):
    w = []
    for i in range(len(wts)):
        w.append(wts[i]*(1 - (epsilon*lose[i])))
    return w
```

Part f

```
def Compute_Equilibrium(T, multiply_epsilon=1) :
    numberOfActions = 4
    epsilon = math.sqrt(math.log(numberOfActions)/ T )
    epsilon = epsilon* multiply_epsilon
    # epsilon =0.093
    print ("Value of T:", T )
    print("Value of Epsilon: ",epsilon)
    weights = [1 ,1 ,1 ,1]
    list_of_probs = []
    for r in range ( T ) :
        probs = weights_prob(weights)
        list_of_probs.append( probs )
        response = optimised_value(probs)
        utilities = u_per_action(response)
        losses = convert_utils_to_losses( utilities)
        weights = update_weights (weights,losses,epsilon)
    average_probability =[]
    for x in zip(*list_of_probs):
        average_probability.append(sum(x)/T)
    print ( " Probability distribution : " )
    print ( average_probability )


    return list_of_probs , average_probability
```

```
u_matrix =[
                [0 , -1 ,1 , -0.2],
                [1 ,0 , -1 ,0.2],
                [ -1 ,1 ,0 ,0.6],
                [0.2 , -0.2 , -0.6 ,0]
               ]
```

```
============50================
Value of T: 50
Value of Epsilon:  0.16651092223153954
 Probability distribution :
[0.2978417384563051, 0.23118339400652899, 0.29644986791885913, 0.17452499961830675]
Opponent utility:  0.0380566553212079
============500================
Value of T: 500
Value of Epsilon:  0.05265537695468319
 Probability distribution :
[0.31364648858568706, 0.2761940012324579, 0.3135593281757281, 0.09660018200612748]
Opponent utility:  0.02050762185044734
============5000================
Value of T: 5000
Value of Epsilon:  0.016651092223153956
 Probability distribution :
[0.32701146006074466, 0.3128720513040118, 0.3269036239775043, 0.033212864657738914]
Opponent utility:  0.007388999419446934
============50000================
Value of T: 50000
Value of Epsilon:  0.005265537695468318
 Probability distribution :
[0.33144506578310373, 0.3266668731492541, 0.33140461193313014, 0.01048344913449275]
Opponent utility:  0.0026410489569775163
============500000================
...
Value of Epsilon:  0.0016651092223153954
 Probability distribution :
[0.3326358341390643, 0.3313416847938585, 0.33270953787154056, 0.0033129431977548624]
Opponent utility:  0.000705264438131085
```

Clearly, as T increases, probability distribution is converging at [1/3, 1/3, 1/3, 0] which is equilibrium distribution.

Part g

Results are attached as a iamge

```
for T in [50,500,5000,50000,500000]:
    print("=============" + str(T) + "=================")
    list_of_probs , average_probab = Compute_Equilibrium( T )
    best = best_response ( average_probab )
    opponent_utility = 0.0
##part g
    for i in range ( len ( average_probab ) ) :
        opponent_utility = opponent_utility + u_matrix[best][i]*average_probab [
            i]
    print("Opponent utility: ",opponent_utility)
```

```
============50================
Value of T: 50
Value of Epsilon:  0.16651092223153954
 Probability distribution :
[0.2978417384563051, 0.23118339400652899, 0.29644986791885913, 0.17452499961830675]
Opponent utility:  0.0380566553212079
============500================
Value of T: 500
Value of Epsilon:  0.05265537695468319
 Probability distribution :
[0.31364648858568706, 0.2761940012324579, 0.3135593281757281, 0.09660018200612748]
Opponent utility:  0.02050762185044734
============5000================
Value of T: 5000
Value of Epsilon:  0.016651092223153956
 Probability distribution :
[0.32701146006074466, 0.3128720513040118, 0.3269036239775043, 0.033212864657738914]
Opponent utility:  0.007388999741944693 4
============50000================
Value of T: 50000
Value of Epsilon:  0.005265537695468318
 Probability distribution :
[0.33144506578310373, 0.3266668731492541, 0.33140461193313014, 0.01048344913449275]
Opponent utility:  0.0026410489569775163
============500000================
...
Value of Epsilon:  0.0016651092223153954
 Probability distribution :
[0.3326358341390643, 0.3313416847938585, 0.33270953787154056, 0.0033129431977548624]
Opponent utility:  0.000705264438131085
```

As, T increases, utility value is tending to zero which is the equilibrium value.

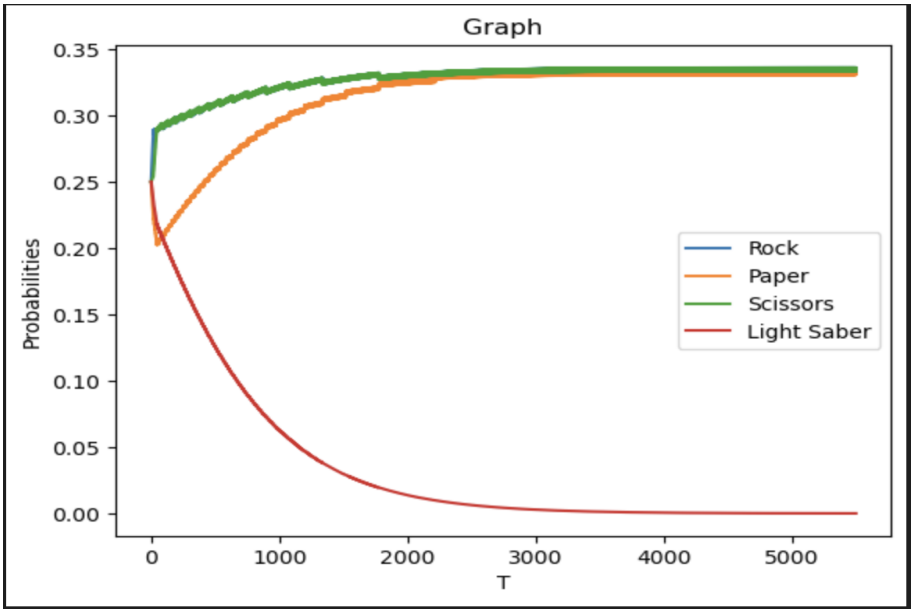
Part h

Code snipped to plot the probabilities on every iteration is in the code part at the end of the file as function **plot_graph**

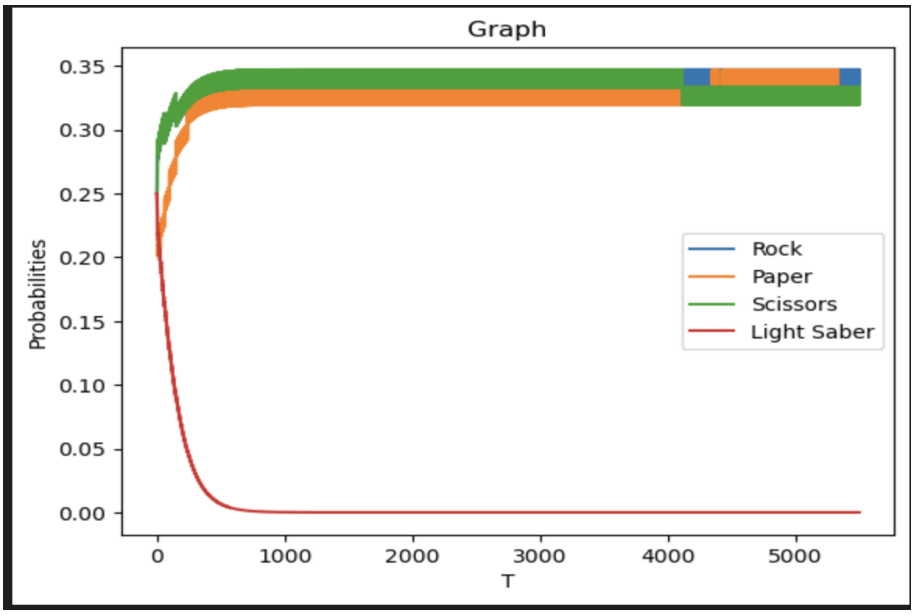
```
def plot_graph(list_of_probs):
    rounds = range ( T )
    rock_probs = [ list_of_probs [ i ][0] for i in range ( T ) ]
    paper_probs = [ list_of_probs [ i ][1] for i in range ( T ) ]
    scissor_probs = [ list_of_probs [ i ][2] for i in range ( T ) ]
    saber_probs = [ list_of_probs [ i ][3] for i in range ( T ) ]
    plt.plot ( rounds , rock_probs , label = 'Rock')
    plt.plot ( rounds , paper_probs , label = 'Paper')
    plt.plot ( rounds , scissor_probs , label = 'Scissors')
    plt.plot ( rounds , saber_probs , label = 'Light Saber')
    plt.xlabel ( 'T')
    plt.ylabel ( 'Probabilities')
    plt.title ( " Graph " )
    plt.legend ()
    # function to show the plot
    plt . show ()
```

T: value 5500
Epsilon value: 0.0166494273636564. by applying $\epsilon = \sqrt{ln(N)/T)}$

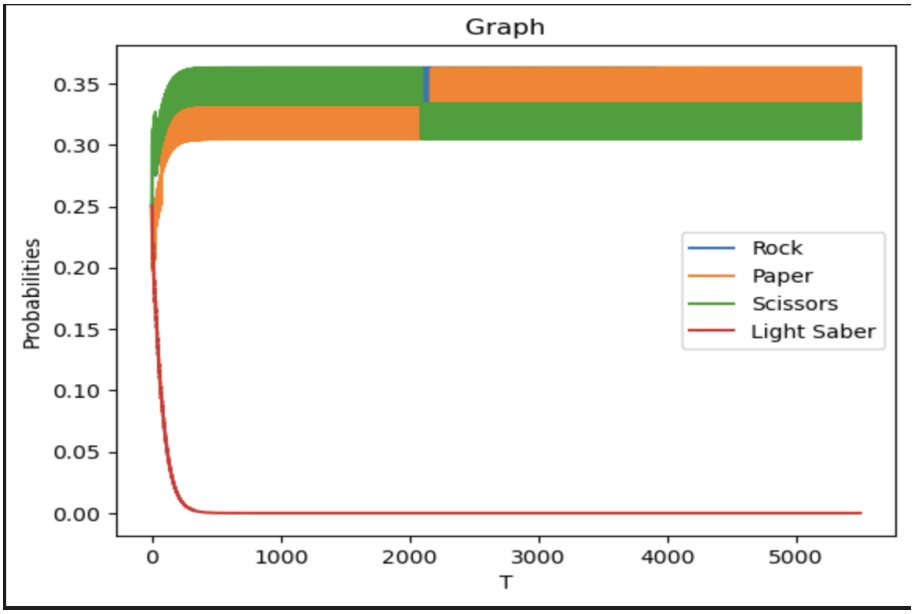
From graph we conclude that, algo is very close to equilibrium and when T is increasing distribution as 1/3 for rock, paper, scissor and 0 for light saber.



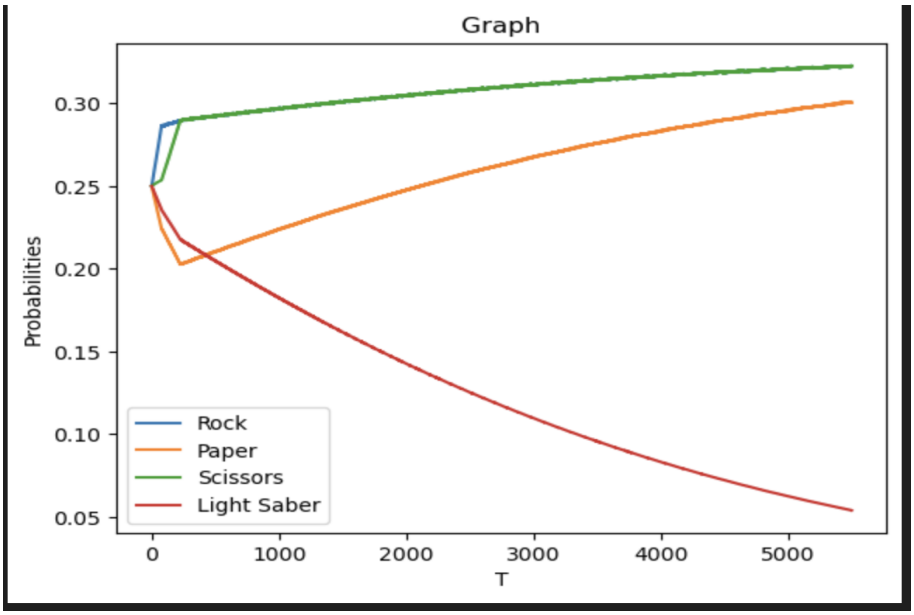
If we take five times bigger ideal epsilon. From graph, even for the same number of T, values are from the equilibrium compared to ideal epsilon.



Now say we chose a different than the ideal one.when we increase epsilon 10 times. For a higher say which is 10 times larger than the one mentioned above, for = 0.0166, we get a graph as below



Now take the epsilon five times smaller than the ideal epsilon. In this graph, even for the same value of T, distribution is far from equilibrium state.



From the above, we can figure out that, if we choose any epsilon other than the ideal one it wont be converging to the equilibrium state. because we know tat multi weight algo wont guarentee no regret only to the epsilon

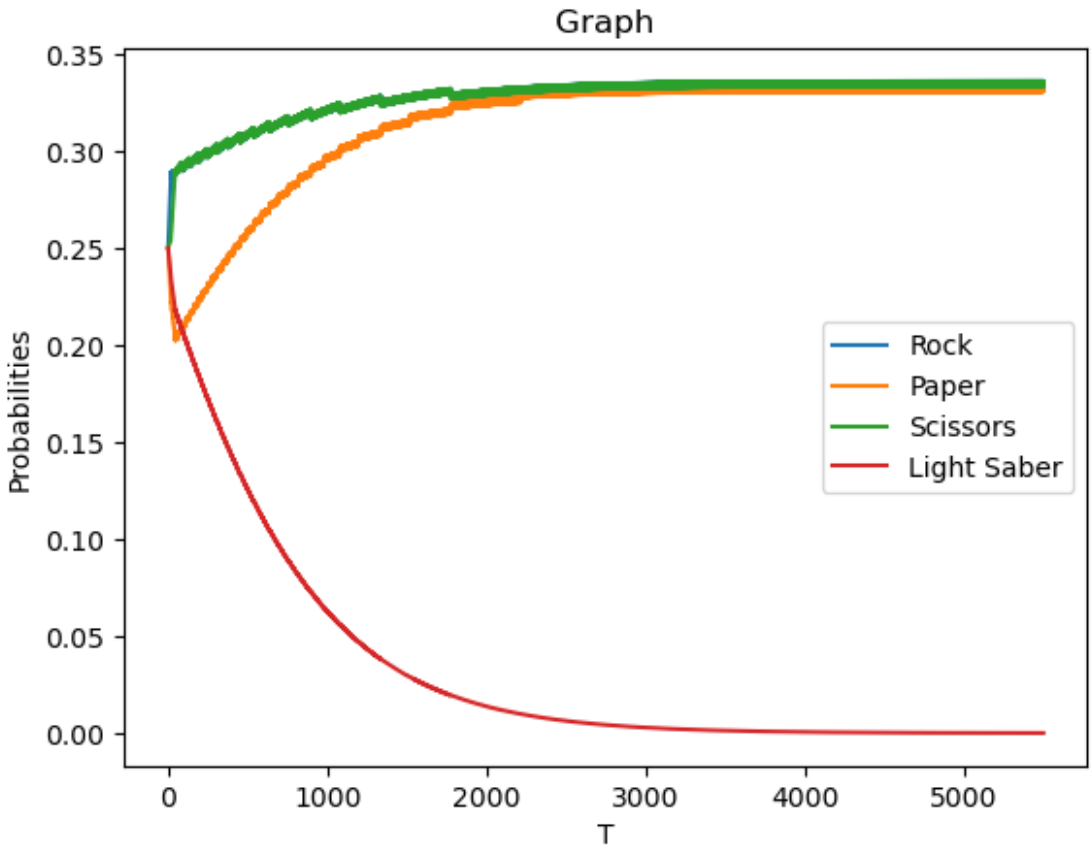
```
import matplotlib.pyplot as plt
import math

## Part a
def probs_from_weights( wts ) :
    total = sum ( wts )
    probArr = []
    for i in wts:
        probArr.append(i/total)
    return probArr

## part b
def best_response ( probArr ) :
    values = []
    for p in range (len(probArr)) :
        tmp =0.0
        for q in range(len( probArr ) ) :
            tmp = tmp + u_matrix[p][q]* probArr[q]
        values.append(tmp)
    max_value = max(values)
    opt_action = values.index(max_value)
    return opt_action

## part c
def util_of_each_action(response) :
    util =[]
    for i in range(len(u_matrix)):
        util.append(u_matrix[i][response])
    return util

## Part d
def convert_utils_to_losses(utilities):
    losses =[]
    for i in range(len(utilities)):
        losses.append(-1 * (utilities[i]/2) +0.5)
    return losses

## part e
def update_weights(wts, lose, epsilon):
    w = []
    for i in range(len(wts)):
        w.append(wts[i]*(1 - (epsilon*lose[i])))
    return w

## part f
def Compute_Equilibrium(T, multiply_epsilon=1) :
    numberOfActions = 4
    epsilon = math.sqrt(math.log(numberOfActions)/ T )
    epsilon = epsilon* multiply_epsilon
    # epsilon =0.093
    print ("Value of T:", T )
```

```
    print("Value of Epsilon: ",epsilon)
    weights = [1 ,1 ,1 ,1]
    list_of_probs = []
    for r in range ( T ) :
        probs = probs_from_weights(weights)
        list_of_probs.append( probs )
        response = best_response(probs)
        utilities = util_of_each_action(response)
        losses = convert_utils_to_losses( utilities)
        weights = update_weights (weights,losses,epsilon)
    average_probability =[]
    for x in zip(*list_of_probs):
        average_probability.append(sum(x)/T)
    print ( " Probability distribution : " )
    print ( average_probability )


    return list_of_probs , average_probability


u_matrix =[
                [0 , -1 ,1 , -0.2],
                [1 ,0 , -1 ,0.2],
                [ -1 ,1 ,0 ,0.6],
                [0.2 , -0.2 , -0.6 ,0]
               ]

for T in [50,500,5000,50000,500000]:
    print("=============" + str(T) + "=================")
    list_of_probs , average_probab = Compute_Equilibrium( T )
    best = best_response ( average_probab )
    opponent_utility = 0.0


    for i in range ( len ( average_probab ) ) :
        opponent_utility = opponent_utility + u_matrix[best]
[i]*average_probab [ i]
    print("Opponent utility: ",opponent_utility)


=============50=================
Value of T: 50
Value of Epsilon:  0.16651092223153954
 Probability distribution :
[0.2978417384563051, 0.23118339400652899, 0.29644986791885913,
0.17452499961830675]
Opponent utility:  0.0380566553212079
=============500=================
Value of T: 500
Value of Epsilon:  0.05265537695468319
```

```
 Probability distribution :
[0.31364648858568706, 0.2761940012324579, 0.3135593281757281,
0.09660018200612748]
Opponent utility:  0.02050762185044734
=============5000================
Value of T: 5000
Value of Epsilon:  0.016651092223153956
 Probability distribution :
[0.32701146006074466, 0.3128720513040118, 0.3269036239775043,
0.033212864657738914]
Opponent utility:  0.0073889997419446934
=============50000================
Value of T: 50000
Value of Epsilon:  0.005265537695468318
 Probability distribution :
[0.33144506578310373, 0.3266668731492541, 0.33140461193313014,
0.01048344913449275]
Opponent utility:  0.0026410489569775163
=============500000================
Value of T: 500000
Value of Epsilon:  0.0016651092223153954
 Probability distribution :
[0.3326358341390643, 0.3313416847938585, 0.33270953787154056,
0.0033129431977548624]
Opponent utility:  0.000705264438131085
```

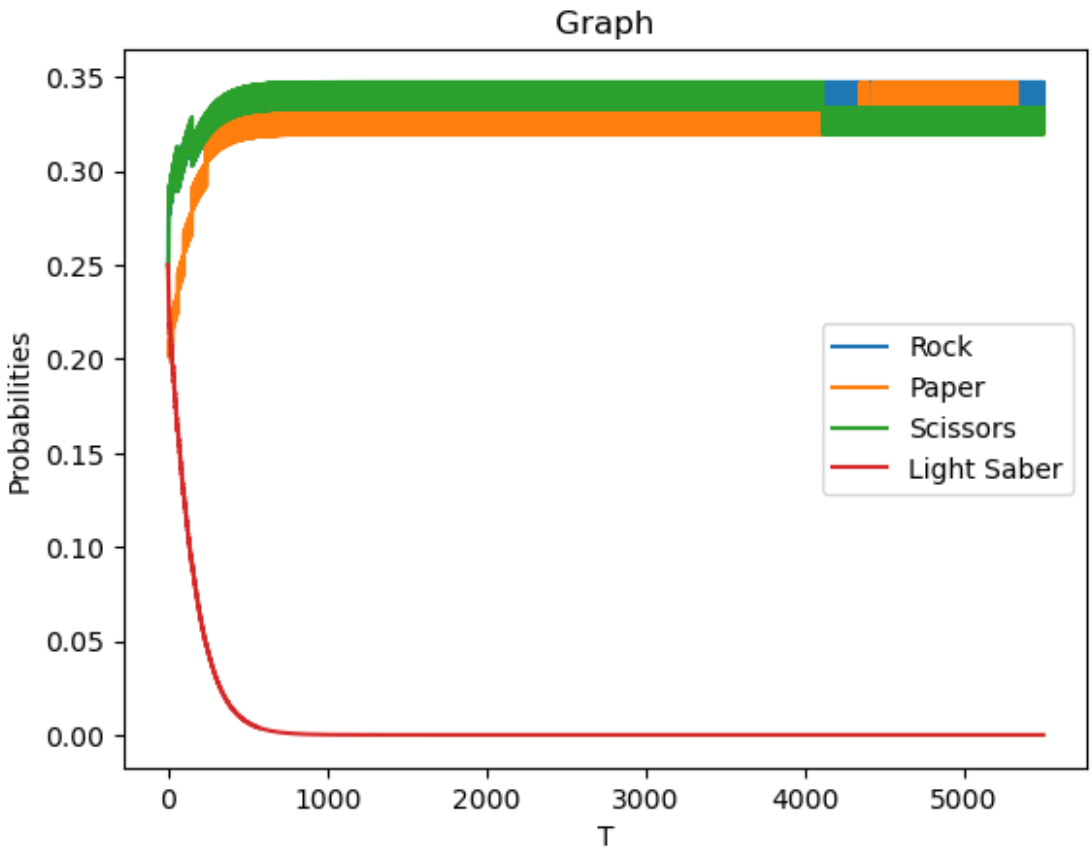
```
## Part h
def plot_graph(list_of_probs):
    rounds = range ( T )
    rock_probs = [ list_of_probs [ i ][0] for i in range ( T ) ]
    paper_probs = [ list_of_probs [ i ][1] for i in range ( T ) ]
    scissor_probs = [ list_of_probs [ i ][2] for i in range ( T ) ]
    saber_probs = [ list_of_probs [ i ][3] for i in range ( T ) ]
    plt.plot ( rounds , rock_probs , label = 'Rock')
    plt.plot ( rounds , paper_probs , label = 'Paper')
    plt.plot ( rounds , scissor_probs , label = 'Scissors')
    plt.plot ( rounds , saber_probs , label = 'Light Saber')
    plt.xlabel ( 'T')
    plt.ylabel ( 'Probabilities')
    plt.title ( " Graph " )
    plt.legend ()
    # function to show the plot
    plt . show ()

T=5500
list_of_probs , average_probab = Compute_Equilibrium( T )
plot_graph(list_of_probs)
```

```
Value of T: 5500
Value of Epsilon:  0.015876193504855515
 Probability distribution :
[0.3273181443432907, 0.31379285933774326, 0.3272214451262361,
0.03166755119272934]
```



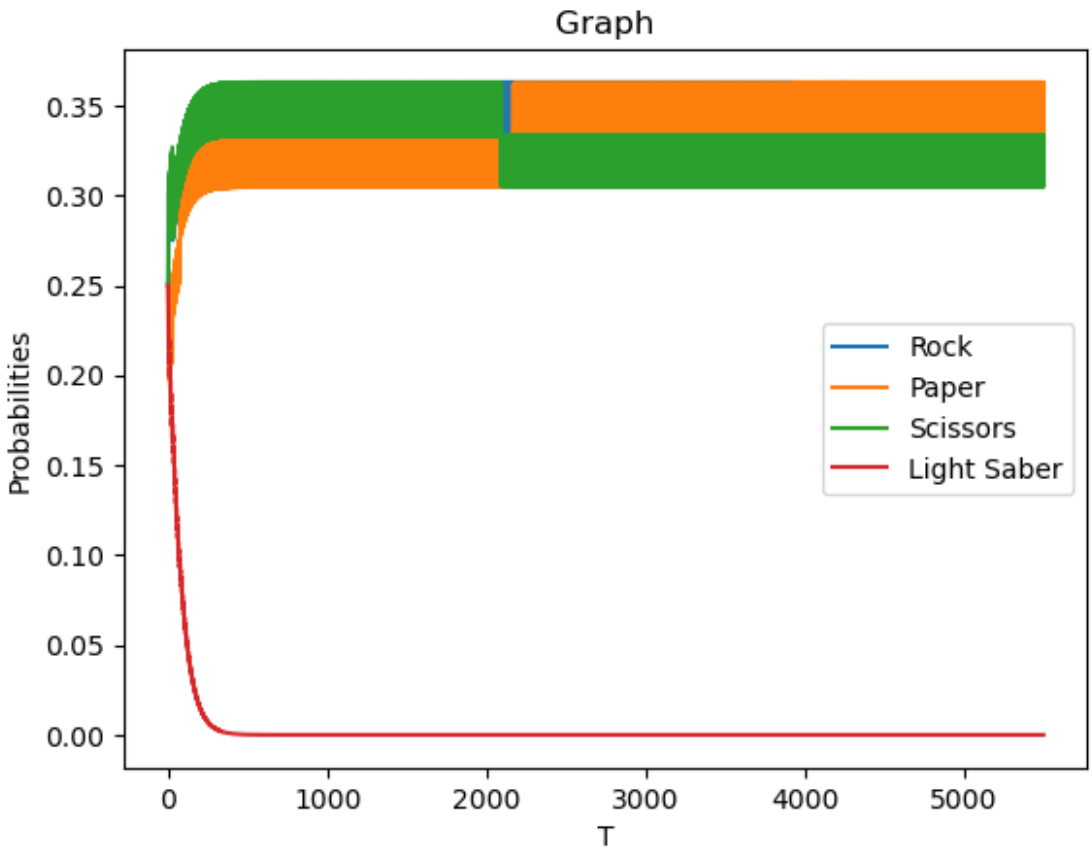

```
#Making epsilon 5 times larger
T=5500
list_of_probs , average_probab = Compute_Equilibrium( T,5)
plot_graph(list_of_probs)
```

```
Value of T: 5500
Value of Epsilon:  0.07938096752427758
 Probability distribution :
[0.3365890228109997, 0.3237306014391129, 0.33325786509590216,
0.006422510653961033]
```



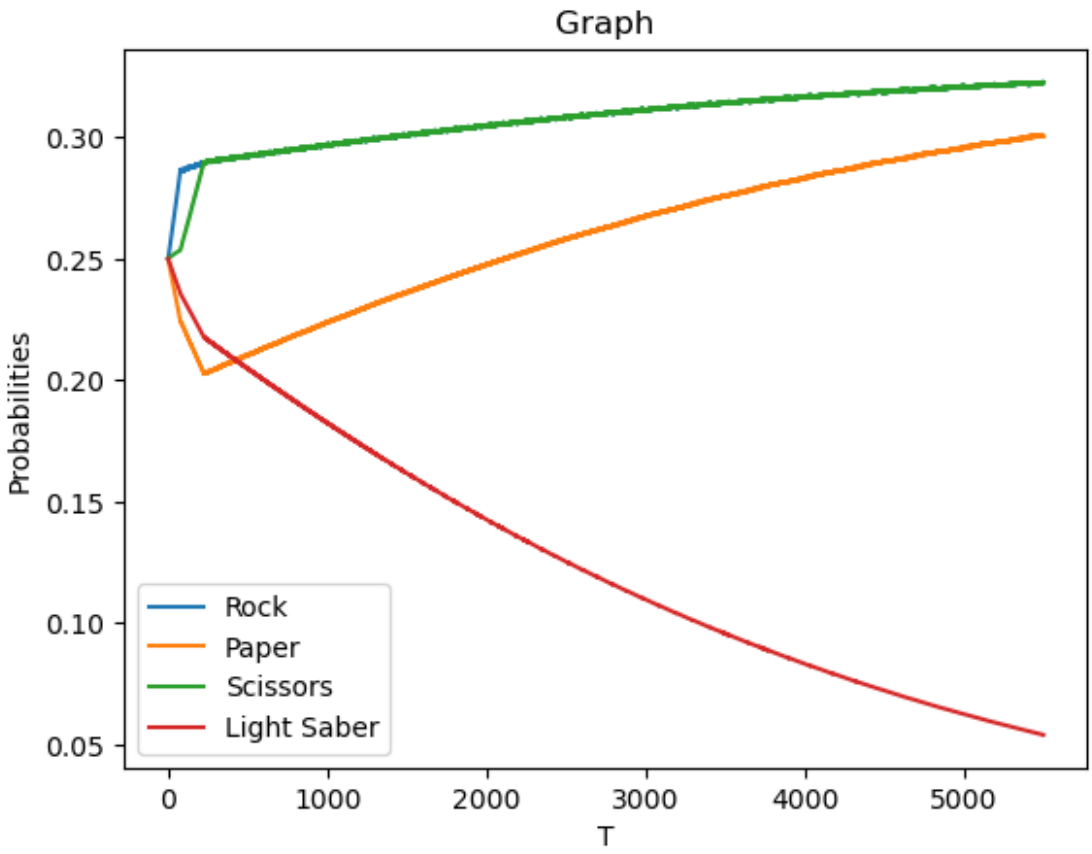

```
#Making epsilon 10times larger
T=5500
list_of_probs , average_probab = Compute_Equilibrium( T,10)
plot_graph(list_of_probs)
```

```
Value of T: 5500
Value of Epsilon:  0.15876193504855515
 Probability distribution :
[0.33683588851147306, 0.3329451325428245, 0.3269395371620793,
0.003279441783565892]
```




```
#Making epsilon 5 times smaller
T = 5500
list_of_probs , average_probab = Compute_Equilibrium( T,1/5 )
plot_graph(list_of_probs)
```

```
Value of T: 5500
Value of Epsilon:  0.003175238700971103
 Probability distribution :
[0.3079150004681154, 0.2589385683849046, 0.3072766764769623,
0.12586975467001665]
```


Graph
Probabilities
0.30
0.25
0.20
0.15
0.10
0.05
0
1000
2000
3000
4000
5000
T
Rock
Paper
Scissors
Light Saber