PH415 Assignment 2
Monte Carlo Spiral Random Walk

S Venkat Bharadwaj

September 15, 2023

Problem

Consider a random walk of a particle on a 2D lattice in the presence of a rotational bias B directed into the plane. In presence of such a bias, the probability of jump depends on the direction from which the walker arrived. With respect to the direction of arrival, the probability of clockwise movement (p_a) and anti clockwise movement (p_b) are given by

$$p_a = \frac{1}{4}(1 - B) \qquad p_c = \frac{1}{4}(1 + B)$$

Generate a random walk of steps randomly chosen between N = 40,000 and 50,000 and average the samples over randomly chosen N_s between 10,000 and 20,000. Evaluate the following by taking B as 0.5, 0.75, 1:

i Plot the configuration of single random walk of 10,000 steps,

ii Calculate the end-to-end distance and plot their distribution.

iii Check how the average distance goes with time t.

iv Evaluate local exponent ν_t and plot it against 1/t.

v Determine the number of sites(N_{cov}) visited by the walker with time t.

General Algorithm

- Step 1: Generate random N and N_s values.
- Step 2: Set Bias B = 0.5, 0.75, 1 in each iteration
- Step 3: Generate a random number between 0 and 1. Select the 1st step based on a uniform distribution i.e. p for each direction = 0.25. Set prevstep = 1,2,3,4 for left, down, right, up respectively. Change x,y accordingly.
- Step 4: Generate another random number. Depending on prevstep, set probabilities of clockwise and and anti-clockwise as p_a and p_c respectively. Update prevstep and coordinates. If Coordinates are unique, increase N_{cov} by 1.
- Step 5: Reduce N by 1. Calculate ν_t and r_t. Store these values and N_{cov} Go back to step 4 if N!=0.
- Step 7: Reset N value and Repeat from Step 3 N_s times. Find the Average values of N_cov, ν_t and r_t over N_s walks.

Code

```
import numpy as np
import matplotlib.pyplot as plt
import random
import math as m
import seaborn as sns

B = [0.5,0.75,1]
Ns = int(10000 + 10000*random.random())
N = int(40000 + 10000*random.random())
re = np.zeros((Ns,len(B)))

for k in range(len(B)):
    p_a  = 1/4*(1-B[k])
    p_c = 1/4*(1+B[k])
    avg_unique = np.zeros(N, dtype=float)
    rt = np.zeros(N, dtype=float)
    nu = np.zeros(N-2, dtype=float)
    x = np.zeros(N)
    y = np.zeros(N)
    unique = set()
    for j in range(Ns):
        x[0] = 0
        y[0] = 0
        unique.add((x[0],y[0]))
        avg_unique[0] = avg_unique[0]+len(unique)/Ns
        prev_step = 0
        step = random.random()
        if(step<0.25):
            x[1] = -1
            y[1] = 0
            prev_step = 1
        if (0.25<=step<0.5):
            x[1] = 0
            y[1] = -1
            prev_step = 2
        if(0.5<=step<0.75):
            x[1] = 1
            y[1] = 0
            prev_step = 3
        if (0.75<=step):
            x[1] = 0
            y[1] = 1
            prev_step = 4
        rt[1] = rt[1]+(x[1]*x[1]+y[1]*y[1])/Ns
        unique.add((x[1],y[1]))
        avg_unique[1] = avg_unique[1]+len(unique)/Ns
        for i in range(2,N):
            step = random.random()
            if prev_step==1:
                if(step<p_c):
                    x[i] = x[i-1]-1
                    y[i] = y[i-1]
                    prev_step = 1
                if (p_c<=step<p_c+p_a):
                    y[i] = y[i-1]-1
                    x[i] = x[i-1]
                    prev_step = 2
                if(p_c+p_a<=step<p_c+2*p_a):
                    x[i] = x[i-1]+1
                    y[i] = y[i-1]
                    prev_step = 3
                if (p_c+2*p_a<=step):
                    y[i] = y[i-1]+1
                    x[i] = x[i-1]
                    prev_step = 4
            if prev_step==2:
                if(step<p_c):
                    x[i] = x[i-1]-1
                    y[i] = y[i-1]
```

```
                    prev_step = 1
                if (p_c<=step<2*p_c):
                    y[i] = y[i-1]-1
                    x[i] = x[i-1]
                    prev_step = 2
                if(2*p_c<=step<2*p_c+p_a):
                    x[i] = x[i-1]+1
                    y[i] = y[i-1]
                    prev_step = 3
                if (2*p_c+p_a<=step):
                    y[i] = y[i-1]+1
                    x[i] = x[i-1]
                    prev_step = 4
            if prev_step==3:
                if(step<p_a):
                    x[i] = x[i-1]-1
                    y[i] = y[i-1]
                    prev_step = 1
                if (p_a<=step<p_a+p_c):
                    y[i] = y[i-1]-1
                    x[i] = x[i-1]
                    prev_step = 2
                if(p_a+p_c<=step<2*p_c+p_a):
                    x[i] = x[i-1]+1
                    y[i] = y[i-1]
                    prev_step = 3
                if (2*p_c+p_a<=step):
                    y[i] = y[i-1]+1
                    x[i] = x[i-1]
                    prev_step = 4
            if prev_step==4:
                if(step<p_a):
                    x[i] = x[i-1]-1
                    y[i] = y[i-1]
                    prev_step = 1
                if (p_a<=step<2*p_a):
                    y[i] = y[i-1]-1
                    x[i] = x[i-1]
                    prev_step = 2
                if(2*p_a<=step<2*p_a+p_c):
                    x[i] = x[i-1]+1
                    y[i] = y[i-1]
                    prev_step = 3
                if (2*p_a+p_c<=step):
                    y[i] = y[i-1]+1
                    x[i] = x[i-1]
                    prev_step = 4

            rt[i] = rt[i] + (x[i]*x[i] + y[i]*y[i])/Ns
            unique.add((x[i],y[i]))
            avg_unique[i] = avg_unique[i] + len(unique)/Ns
        unique.clear()
        re[j,k] = m.sqrt(x[-1]*x[-1]+y[-1]*y[-1])
        #print(j)
    for i in range(2,N):
        nu[i-2] = m.log(rt[i])/(2*m.log(i))



    sns.distplot(re[:,k], bins = 100)
    plt.title('Distribution of Average End to End disatnce for B = ' + str(B[k]))
    plt.xlabel('Average End to End distance')
    plt.ylabel('Frequency')
    plt.show()
    tmp = np.arange(N, dtype=float)
    plt.scatter(x[1:10000],y[1:10000],s=0.5)
    plt.plot(x[1:10000],y[1:10000])
    plt.title('Spiral Random Walk of 10000 Steps for B = ' + str(B[k]))
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```

```
    plt.plot(tmp,avg_unique)
    plt.title('Number of Sites Visited for B = ' + str(B[k]))
    plt.xlabel('Number of Steps (Time)')
    plt.ylabel('Number of Unique Sites Visited (N_cov)')
    plt.show()
    sum_x = 0
    sum_x2 = 0
    sum_y = 0
    sum_xy = 0
    for i in range(1,N):
        tmp[i] = m.log(tmp[i])
        rt[i] = m.log(rt[i])
        sum_x = sum_x+tmp[i]
        sum_x2 = sum_x2 + tmp[i]*tmp[i]
        sum_y = sum_y+rt[i]
        sum_xy = sum_xy+tmp[i]*rt[i]
    plt.plot(tmp,rt)
    plt.title('Square of Distance Travelled vs Time for B = ' + str(B[k]))
    plt.xlabel('Number of Steps (Time)')
    plt.ylabel('Square of Distance Travelled')
    plt.show()
    nu_fin = ((N*sum_xy-(sum_x*sum_y))/(N*sum_x2-(sum_x*sum_x)))/2
    print('The diffusion constant of the Random walk comes as', nu_fin)
    tmp = np.arange(2,N)
    plt.plot(1/tmp,nu)
    plt.title('Diffusion Coefficient vs 1/Time for B = ' + str(B[k]))
    plt.ylabel('Diffusion Coefficient')
    plt.xlabel('1/Number of Steps (1/Time)')
    plt.show()
```

Discussion

- From the image of the Random Walk, it is clear to see that a Bias of such sort leads to a tendency of rotational motion. However, it is not easy to differentiate between 2 different values of Bias as they do not direct them in a particular direction.
- The distribution of average end-to-end distance for each walk is plotted. From the plot it can be observed that the distribution is a Rayleigh distribution. Rayleigh distribution is a continuous probability distribution for for non-negative random numbers which corresponds to a chi-distribution with 2 DOF.
- From the graph of log R_t^2 vs log t, it can be seen that the plot follows a linear trend. By least square fitting, the slope can be found as 0.538, 0.540, 0.491 for B=0.5, 0.75, 1 respectively.
- The ν vs t graph seems to be decreasing and converging to a value close to 0.5 for each random walk. For bias B = 0.5,0,75,1 the converging nu comes out to be 0.516, 0.514, 0.499 respectively.
- From the values of nu, we can conclude that for B = 0.5 and 0.74 it is super diffusion and for B=1, it is sub diffusion. However, this value varies randomly and the value of nu can alternate to super and sub diffusion for each B.
- The plot of N_{cov} vs t shows that it is continuously increasing. The trend it follows is much higher than t/lnt(which is expected of a unbiased 2D random walk).

Plots

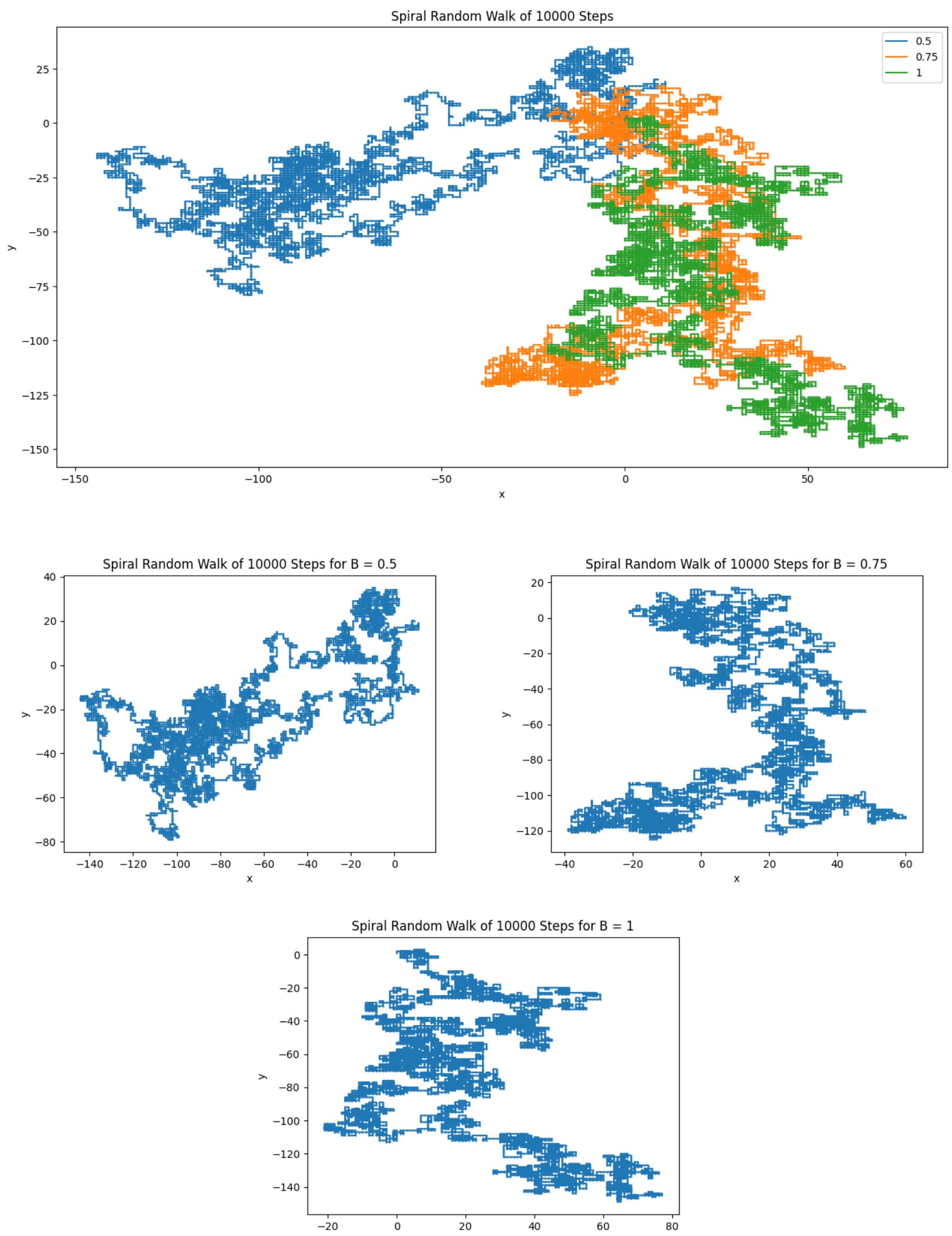

Figure 1: Random Walk of 10000 Steps

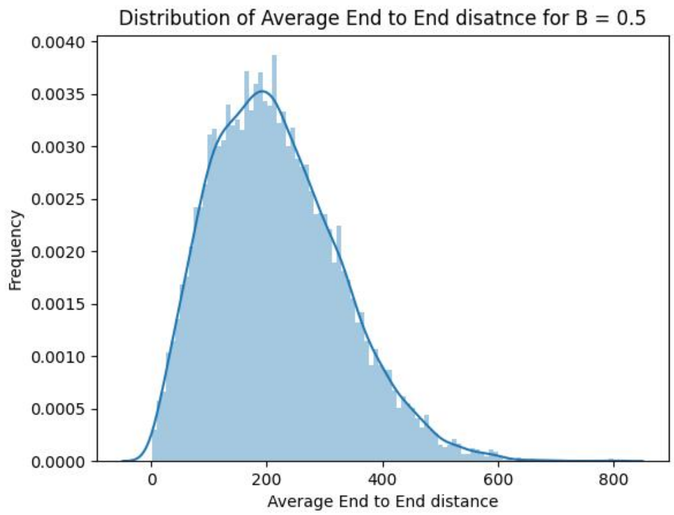

Figure 2: Distribution of Average End to End distance for B = 0.5 and bins=100

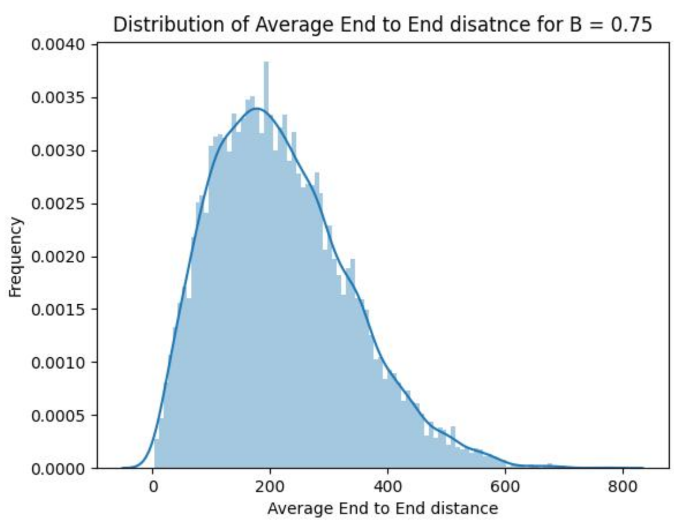

Figure 3: Distribution of Average End to End distance for B = 0.75 and bins=100

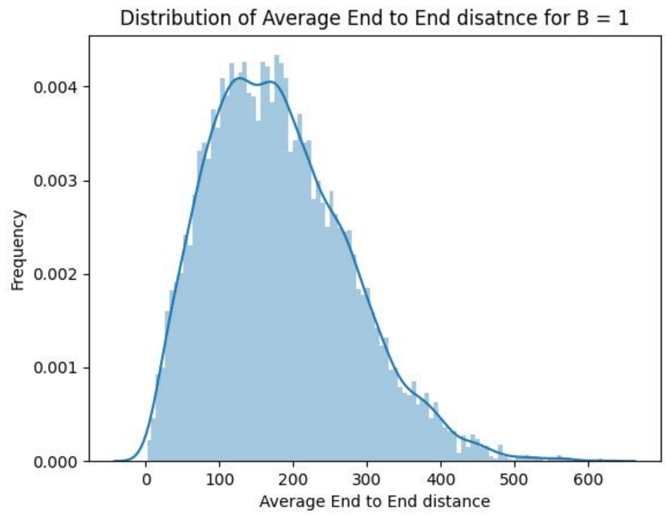

Figure 4: Distribution of Average End to End distance for B = 1 and bins=100

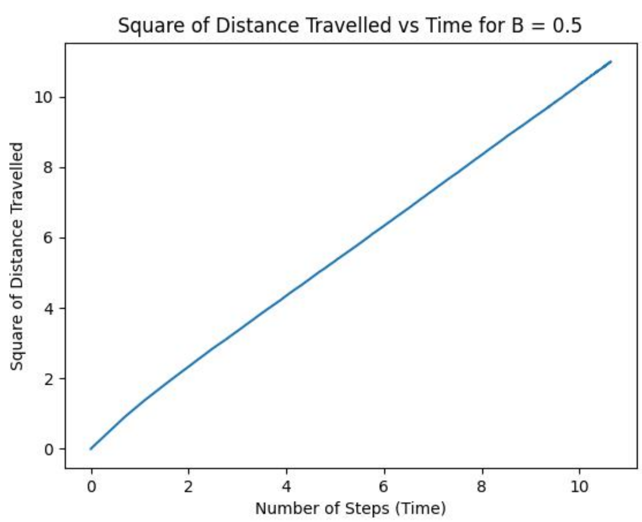

Figure 5: Square of Distance Travelled vs Time for B = 0.5 in log scale

Figure 6: Square of Distance Travelled vs Time for B = 0.75 in log scale

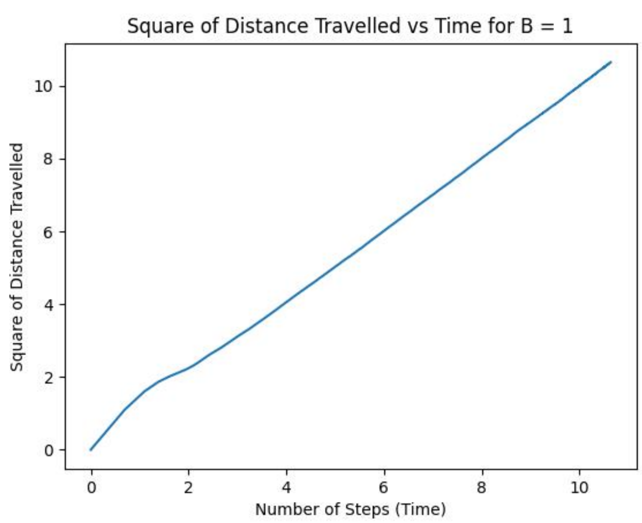

Figure 7: Square of Distance Travelled vs Time for B = 1 in log scale

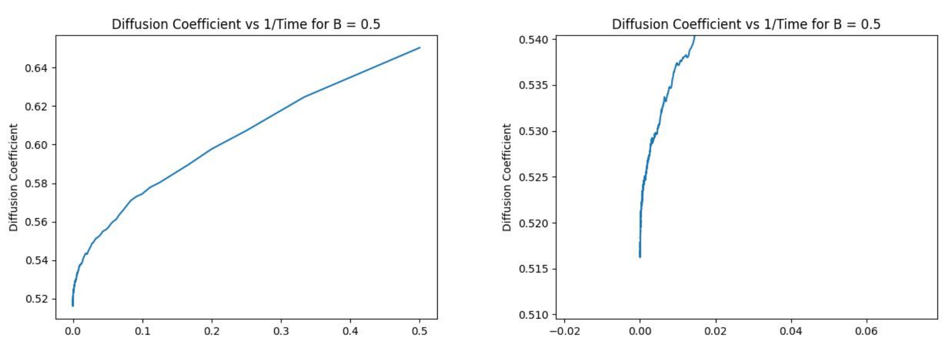

Figure 8: Diffusion Coefficient vs 1/Time for B = 0.5

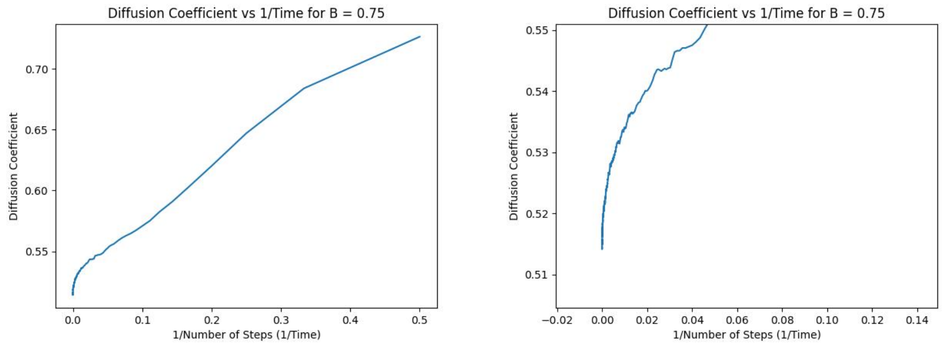

Figure 9: Diffusion Coefficient vs 1/Time for B = 0.75

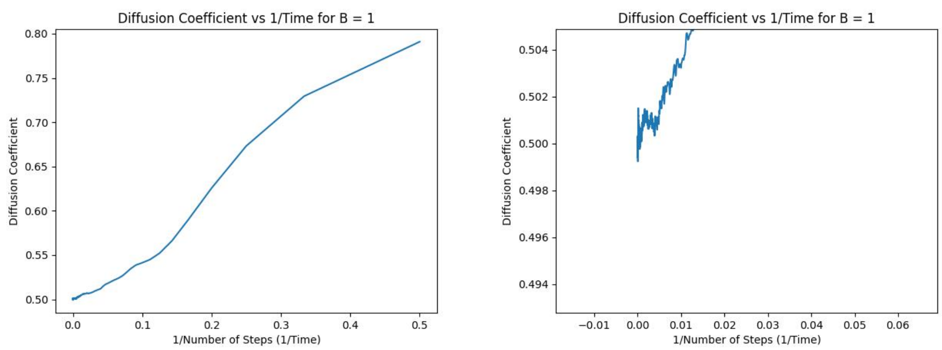

Figure 10: Diffusion Coefficient vs 1/Time for B = 1

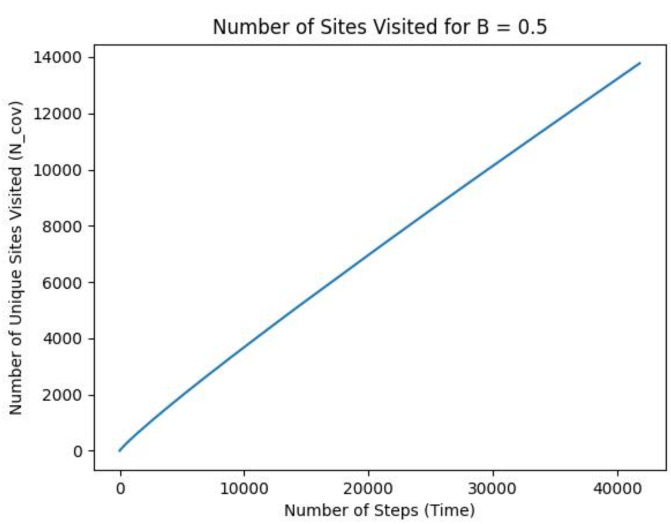

Figure 11: Number of Unique Sites Visited (N_{cov}) for B = 0.5

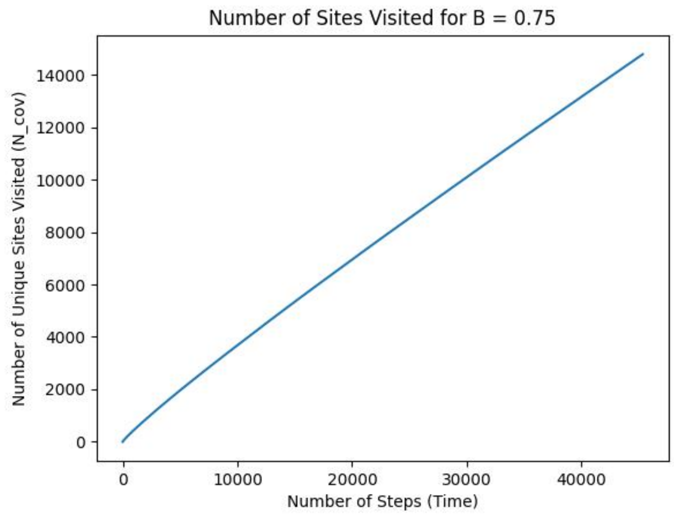

Figure 12: Number of Unique Sites Visited (N_{cov}) for B = 0.75

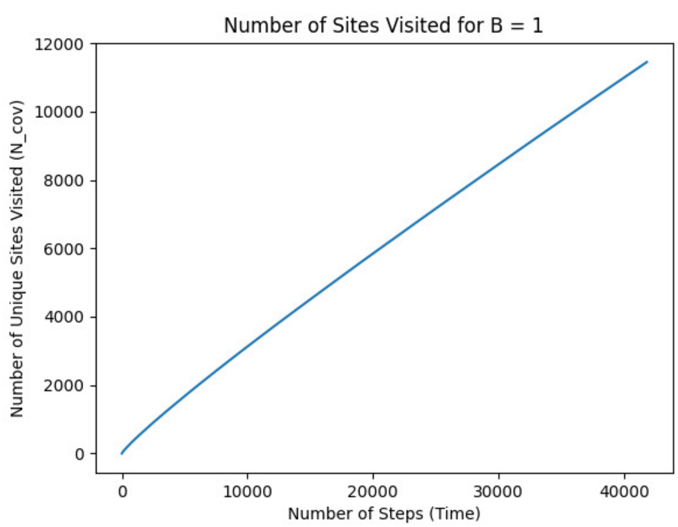

Figure 13: Number of Unique Sites Visited (N_{cov}) for B = 1