PHY407 (Lab 10: Monte Carlo Simulations)

Utkarsh Mali[1] and Aslesha Pokhrel[1,2]

[1]Department of Physics, University of Toronto
[2]Department of Computer Science, University of Toronto

December 5, 2020

Contents

Work Allocation:
Aslesha: Question 1, Question 3
Utkarsh: Question 2, Question 3

Question 1: Simulated Annealing Optimization

(a) In this section, we test the sensitivity of the simulated annealing optimization method in the travelling salesman problem to the cooling schedule time constant τ. First, we fix a set of points by using $seed = 10$. Then, we vary only the second seed before the while loop each time. The resulting distance D and the plot showing the different paths taken is given below:

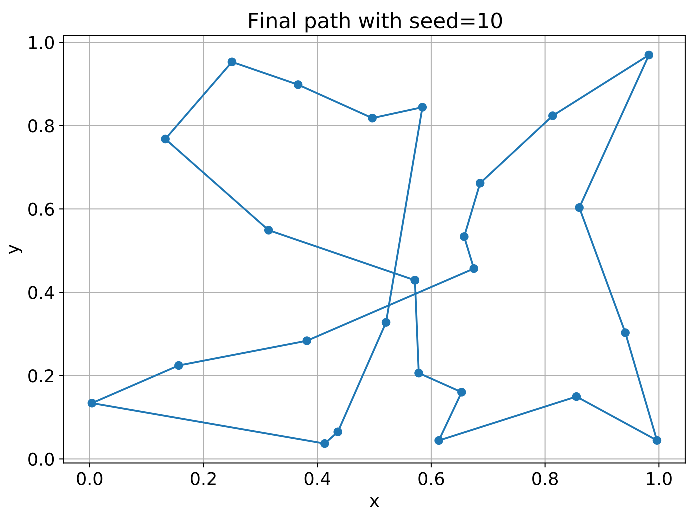


Figure 1: The plot showing the final path taken using seed=10 for the second loop. The total distance D = 5.64.

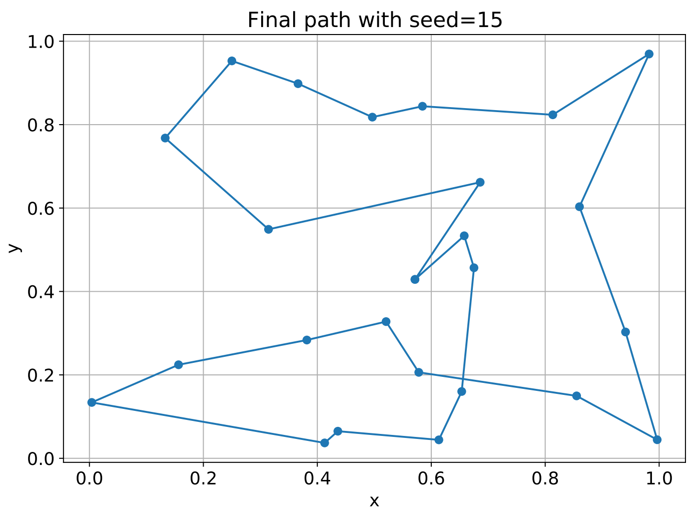


Figure 2: The plot showing the final path taken using seed=15 for the second loop. The total distance D = 5.36.

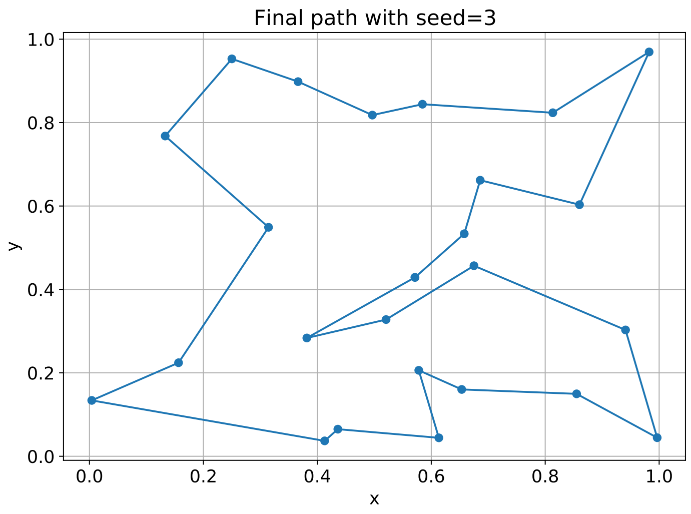


Figure 3: The plot showing the final path taken using seed=3 for the second loop. The total distance D = 5.13.

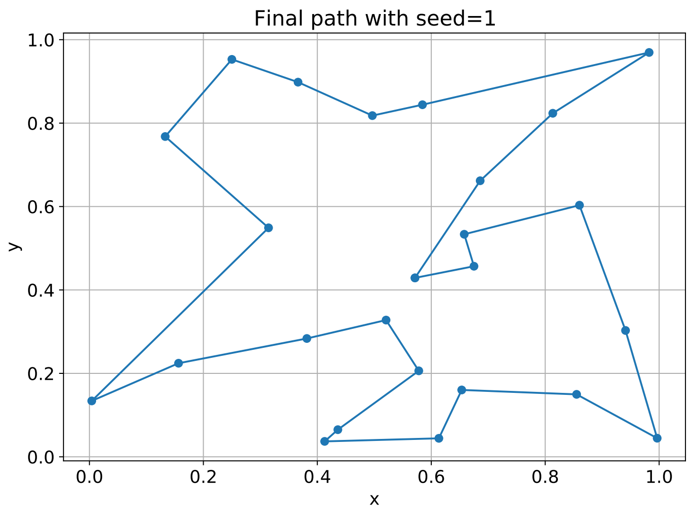


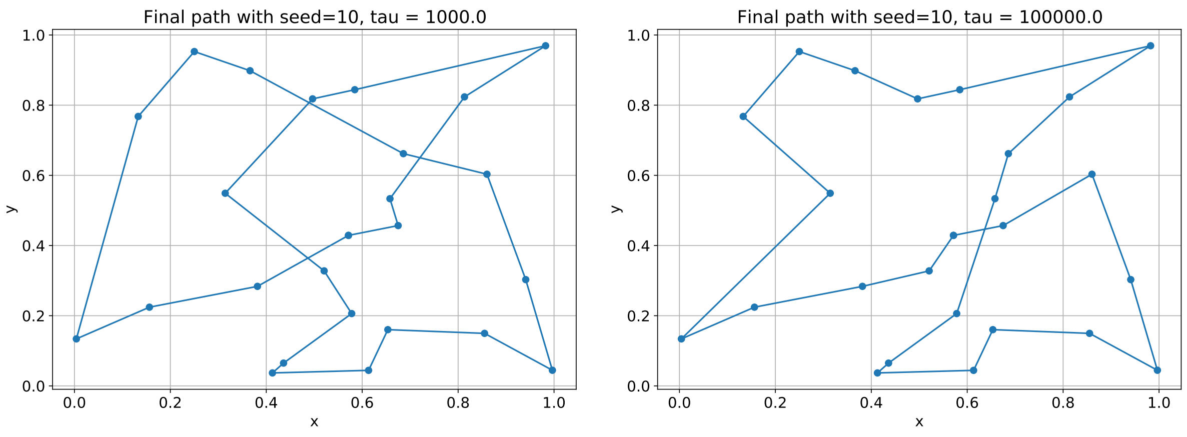
Figure 4: The plot showing the final path taken using seed=1 for the second loop. The total distance D = 5.10.

```
CODE OUTPUT

For seed = 10, the total distance D = 5.64.
For seed = 15, the total distance D = 5.36.
For seed = 3, the total distance D = 5.13.
For seed = 1, the total distance D = 5.10.
```

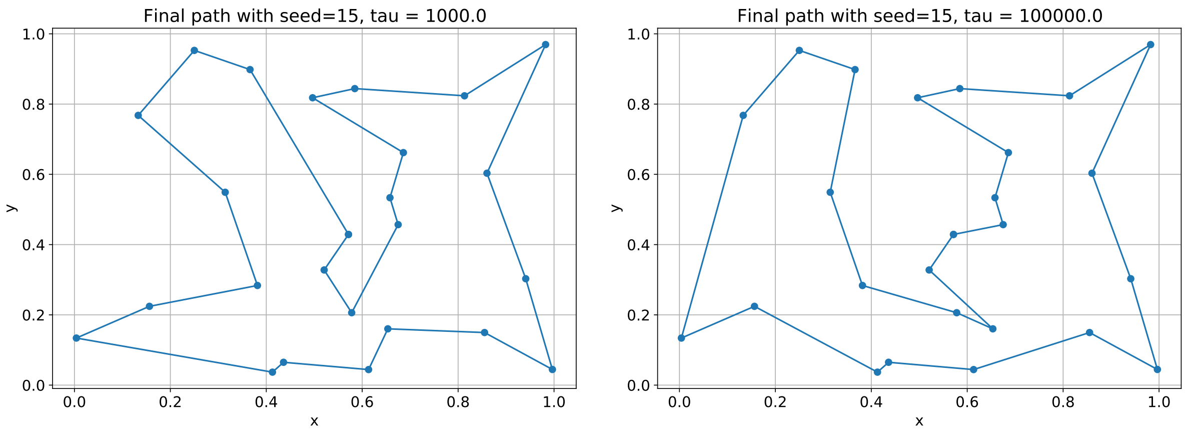
As shown above, the smallest value of the final value of the distance D was found to be 5.10 for seed = 1. As stated above, this value varied within a small range as a result of different paths taken.

Next, we vary the scheduling time constant τ by making it shorter and then longer than the default value of $\tau = 10^4$. The plots showing the path taken and the final value of D with this change for different values of seed is given below:



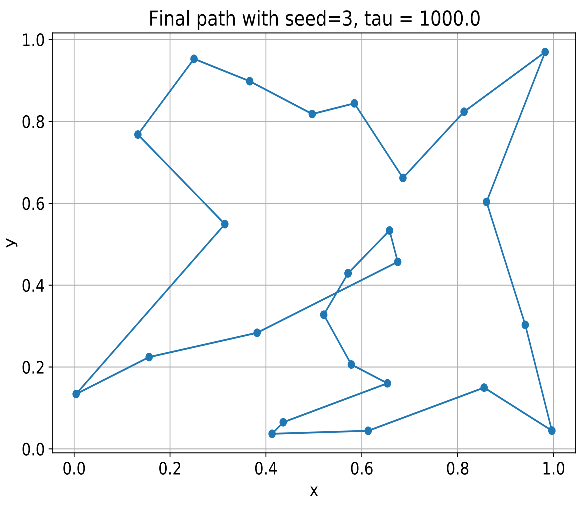
(a) (b)

Figure 6: The plots showing the final path taken using seed=10 for the second loop with (a) $\tau = 10^3$ and (b) $\tau = 10^5$. The final distance (a) $D = 5.75$ and (b) $D = 5.23$.

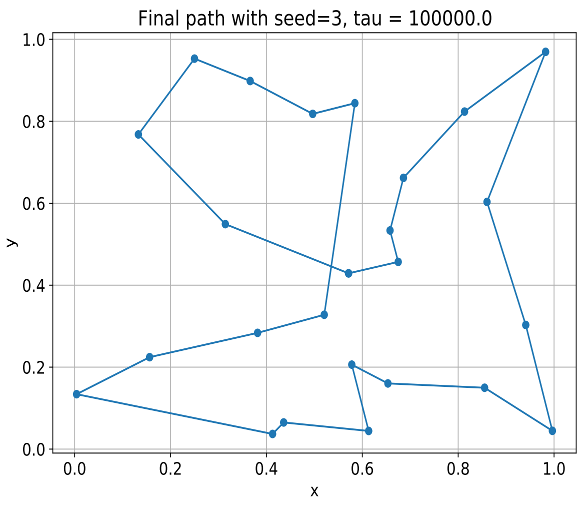


(a) (b)

Figure 8: The plots showing the final path taken using seed=15 for the second loop with (a) $\tau = 10^3$ and (b) $\tau = 10^5$. The final distance (a) $D = 5.44$ and (b) $D = 5.47$.

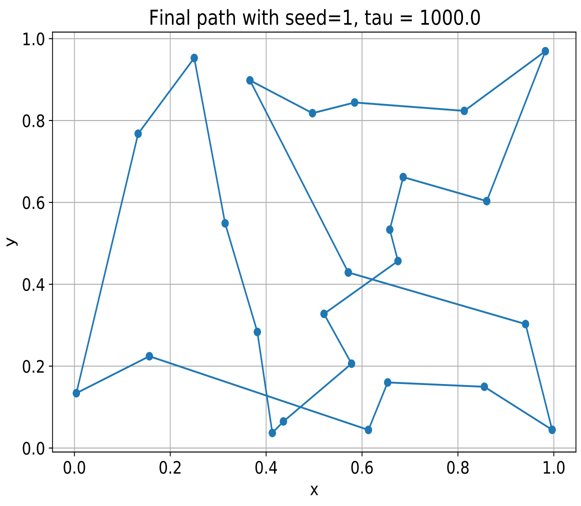


(a)

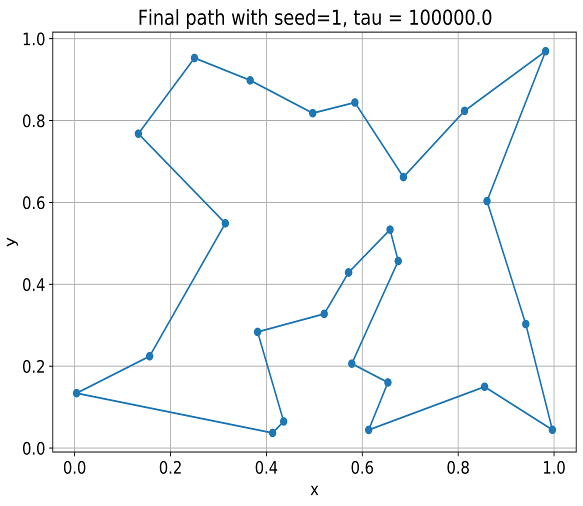


(b)

Figure 10: The plots showing the final path taken using seed=3 for the second loop with (a) $\tau = 10^3$ and (b) $\tau = 10^5$. The final distance (a) $D = 5.21$ and (b) $D = 5.21$.



(a)



(b)

Figure 12: The plots showing the final path taken using seed=1 for the second loop with (a) $\tau = 10^3$ and (b) $\tau = 10^5$. The final distance (a) $D = 6.18$ and (b) $D = 5.09$.

The impact of τ and the seed value on D is summarized in the table below:

seed value	τ	distance (D)
10	10^3	5.75
10	10^4	5.64
10	10^5	5.23
15	10^3	5.44
15	10^4	5.36
15	10^5	5.47
3	10^3	5.21
3	10^4	5.13
3	10^5	5.21
1	10^3	6.18
1	10^4	5.10
1	10^5	5.09

Table 1: Table showing the the final value of the distance D for varying the scheduling time constant τ and the seed value before the second while loop.

As shown in table 1, allowing the system to cool more slowly (i.e. higher value of τ) decreases the final value of distance D when the seed value used before the second while loop is 10 or 1 but in other cases D increases by a slight amount. Likewise, allowing the system to cool more quickly increases the final value of distance D in all cases.

(b) In this section we use simulated annealing to find the global minimum of the given functions. The algorithm starts from $(x, y) = (2, 2)$ with Monte Carlo moves of the form $(x, y) \rightarrow (x + \delta x, y + \delta y)$ where δx and δy are random numbers drawn from a Gaussian distribution with mean 0 and standard deviation 1. The scheduling time constant τ was set to 10^4 and the start and temperatures were set to 1 and 10^{-5} respectively. The final values of (x, y) and the plots of x and y as a function of time is shown below:

```
CODE OUTPUT:

The value of (x, y) for Qb(i) is (-0.001, 0.992).
The value of (x, y) for Qb(ii) is (15.958, 0.998).
```

(i) The plots of x and y as a function of time during the run for $f(x, y) = x^2 - \cos(4\pi x) + (y - 1)^2$ are shown below:

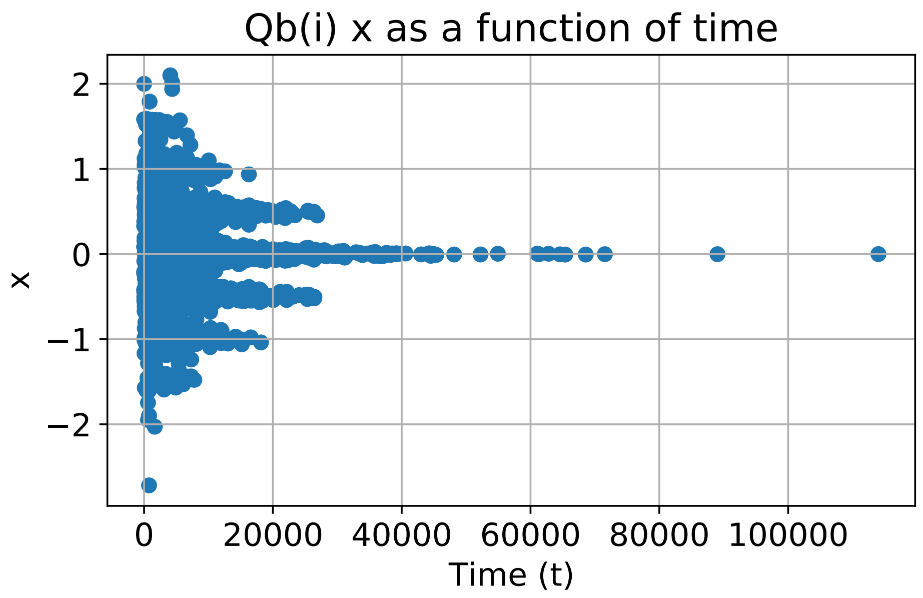


Figure 13: The plot of x as a function of time during the run.

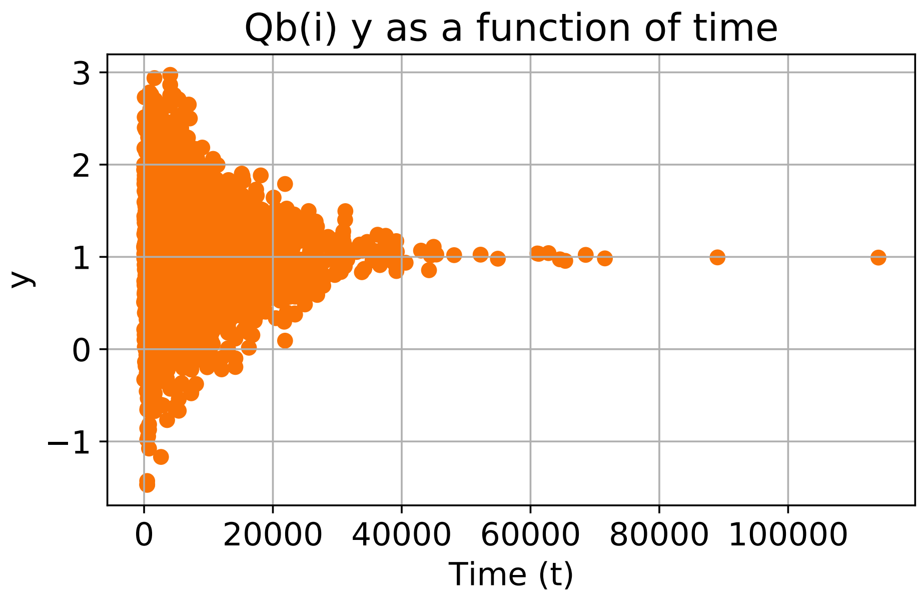


Figure 14: The plot of y as a function of time during the run.

As expected, the final value of (x, y) is approximately equal to $(0, 1)$.

(ii) The plots of x and y as a function of time during the run for $f(x,y) = \cos(x)+\cos(\sqrt{2}x)+\cos(\sqrt{3}x)+(y-1)^2$ are shown below:

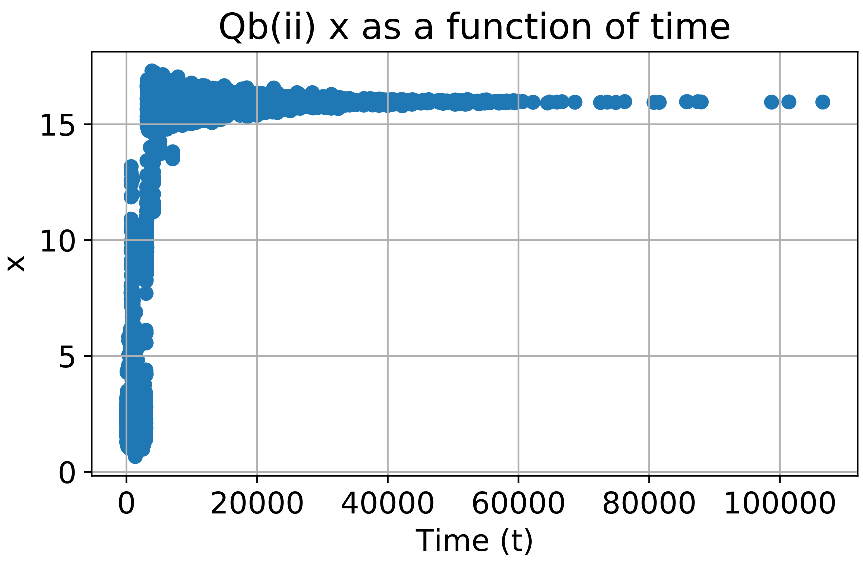


Figure 15: The plot of x as a function of time during the run.

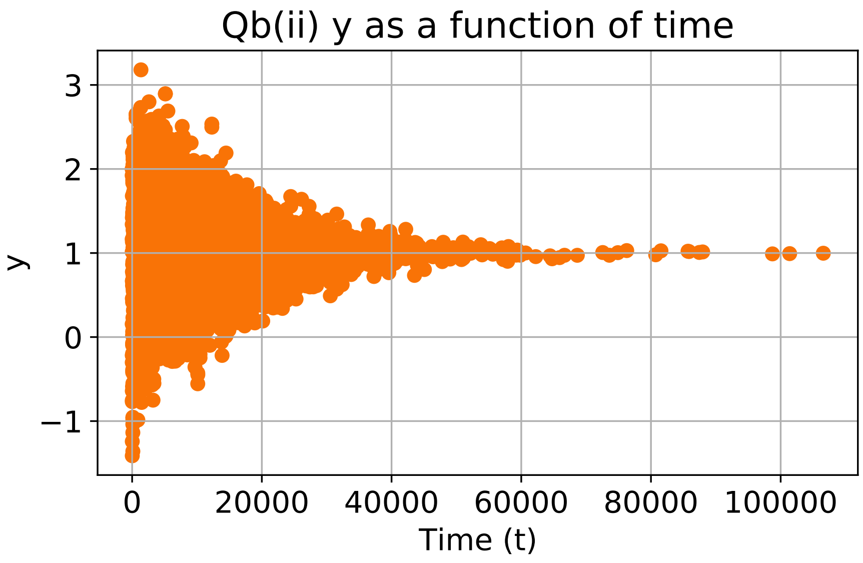


Figure 16: The plot of y as a function of time during the run.

Here, the final value was found to be approximately equal to $(16, 1)$ in this particular run. However, x varies between $16, 2, 42$ in different runs most of the times but might end up in an unexpected value such as 34 in some of the runs as the algorithm might get stuck in some local minimum.

Question 2: Ising model (Markov Chain Method)

In this question we will plot the Ising model for a 20 x 20 system and vary its spin using the Markov Chain Model.

(a) We wrote a function that calculated the total energy as required. We used computer optimization techniques to ensure that our program ran in a decent run time.

(b) We now used our function as a basis for a Metropolis-style simulation of the Ising model, using Markov Chain Models. We set $J = 1$, $T = 1$, and initially set the spin of the grid to ± 1 so that on average half of them would be up and the other half would be down. We then randomly flipped using the Metropolis acceptance formula from Eq. (10.60). If the move was rejected we replaced the object with its initial value and repeated this process for $N = 100,000$ steps.

```
PSEUDO-CODE

# Adapted from Prof Nicolas
# Initialize constants
# Initialize dipole array to store spins
# Define energy function as required in part a
# Define helper function for the acceptance
# Define acceptance function the same was as Prof Nicolas
# Initialize constants for part b
# Initialize empty arrays
# Start Markov Chain Model for simulation
    # Choose i and j at random
    # Compute old energy
    # Flip spin randomly
    # Calculate energy with flipped spin
    # Calculate the change in energy
    # Run acceptance limitation
        # Set new E if accepted and update energy
        # Re-flip dipole if no accepted
    # Finally update magnetic list.
# Plot figure
```

(c) We now use this simulation to generate a plot of the magnetization with time.

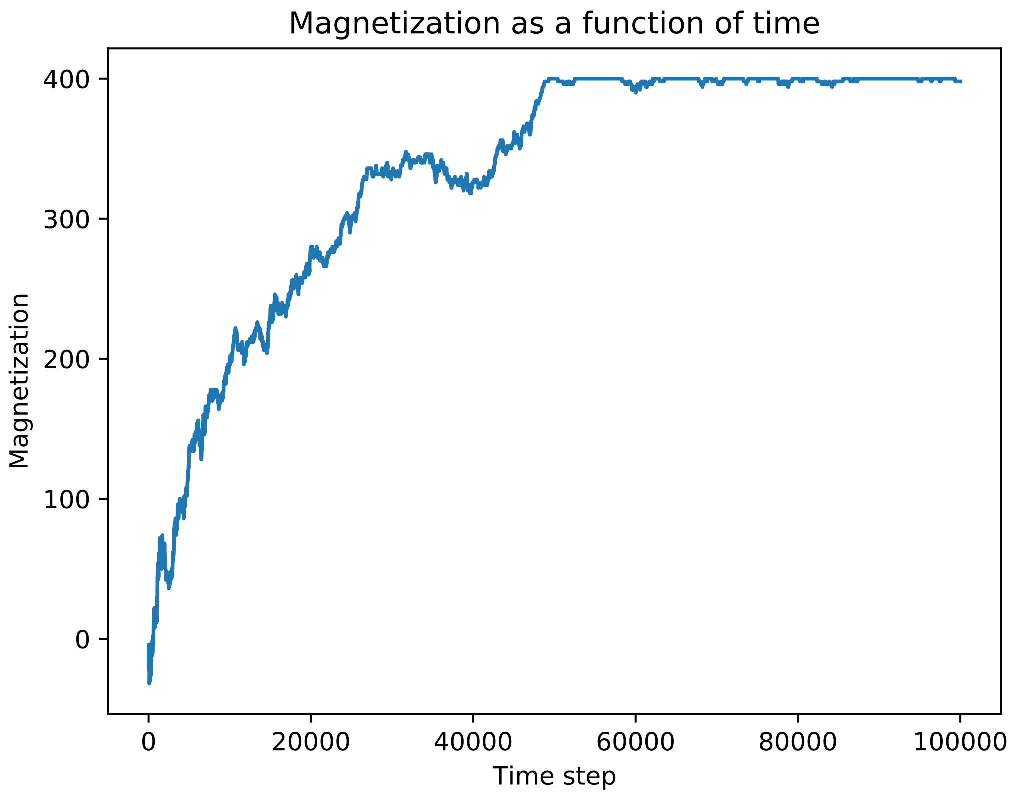


Figure 17: As the time increased we found the magnetization as a function of time tend towards 400 which is exactly equal to 20^2. This means that all the items in our Ising grid have a positive spin up orientation.

(d) We ran the program many times and notices that sometimes the program would go completely in the other direction to the figure above. We would find the magnetization tend towards -400 suggesting that all the particles are now spin-down. **The change always goes in the direction it starts off in.** This correlates with the nearest neighbour interactions the Ising model is well known for.

(e) We made a second version of our program which highlights the temperature varying between $T = 1.0, 2.0, 3.0$. **As the temperature increases the progression of the magnetization becomes less and less monotonic.** As shown in the graph below. As the temperature of the system is increased random excitation's start to play a smaller role in the change of an Ising model.

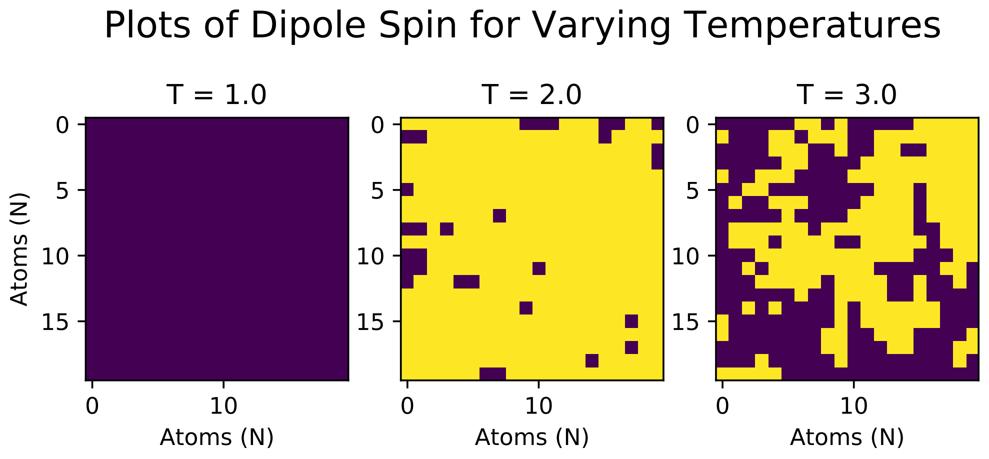


Figure 18: At $T = 1.0$ the Ising spin model was completely homogeneous. When the temperature was to $T = 2.0$, the Ising model was semi-homogeneous. As the temperature, increased to $T = 3.0$, the Ising model was similar to its random arrangement at the start. As the temperature increased, the final structure of the Ising model become less predictable and less monotonic.

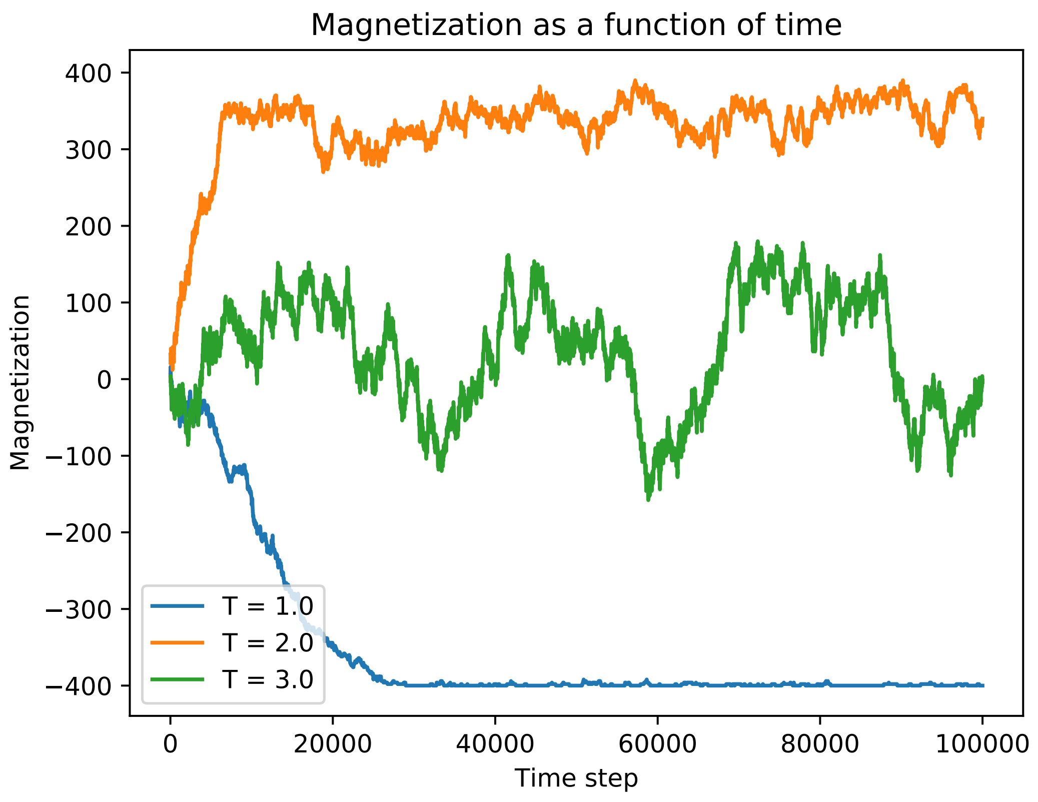
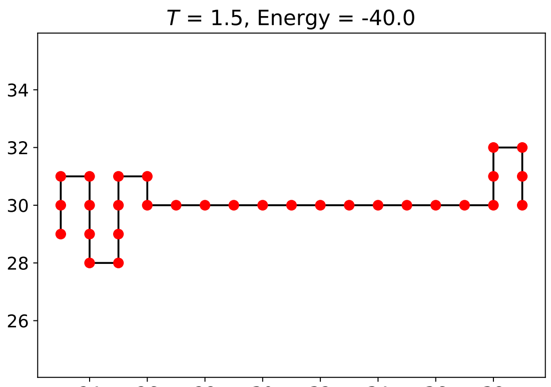


Figure 19: As the time increased we found the magnetization as a function of time tend towards -400 for $T = 1.0$, the magnetization tended closer and closer to 400 for $T = 2.0$. When $T = 3.0$, the magnetization was constantly fluctuation between ± 200.

Question 3: Protein Folding

(a) We ran the script as defined in the lab manual, first with $N = 30$, $T = 1.5$, $\epsilon = -5.0$ and $n = 10^5$ and observed the following figures.



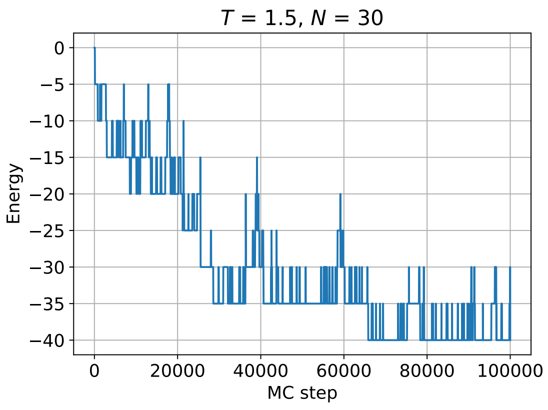
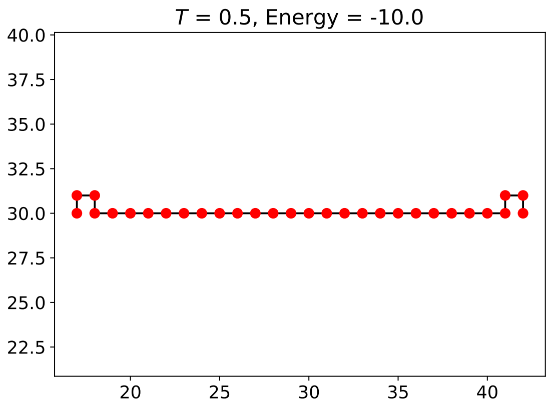


Figure 20: The protein chain is relatively long with a few folds at the ends of the structure.

Figure 21: We observe the energy steadily decreasing with each step of the Monte-Carlo simulation until it ultimately reached its relatively stable form at -40.

We now change the temperature to $T = 0.5$ We observed a much sharper change and a much straighter protein.



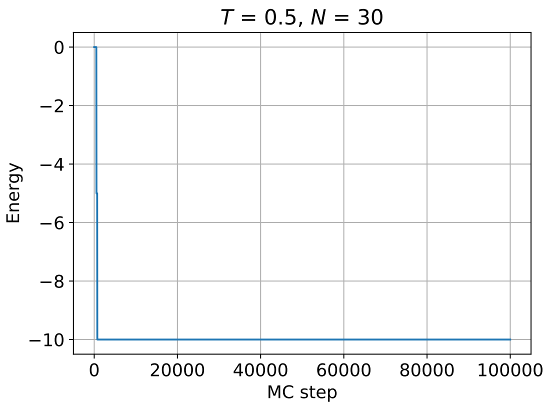
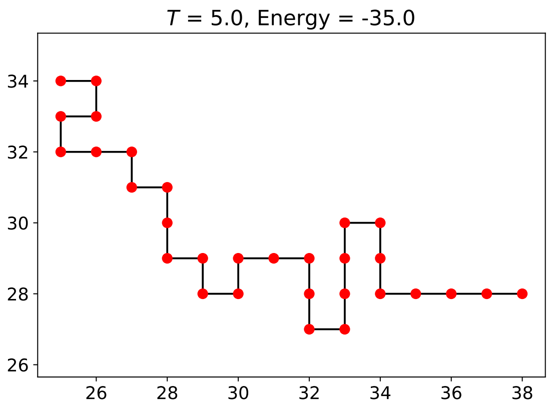


Figure 22: The protein chain is relatively long with little to no folds at the ends of the structure.

Figure 23: As a result of the little to no change of the structure we observed the energy sharply drop with the first few steps and then remain stable throughout the rest of the Monte =Carlo simulation. **Essentially this is telling us the simulation took less iterations to converge.**



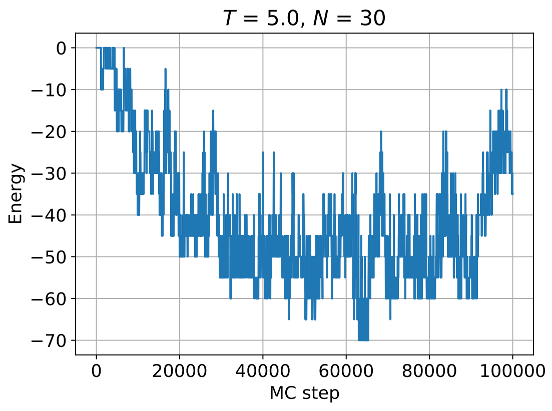
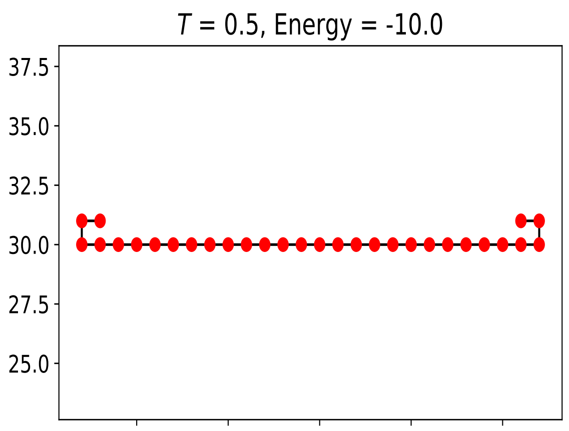


Figure 24: The protein chain was now observed to create more folds and turns in its structure. It is far less straight than the other sets of proteins formed.

Figure 25: The observed energy this time was very different from the two other iterations. We observed the energy to decrease, lowering its value to almost -70 before slowly starting to increase again. The folds of the protein chain heavily affected the energy.

(b) Now, we run the simulation for $T = 0.5$ and $T = 1.5$ cases for $n = 1,000,000$ steps by changing the parameters in the given code `L11-protein-start.py`. The plots showing the energy and the protein structure are given below:



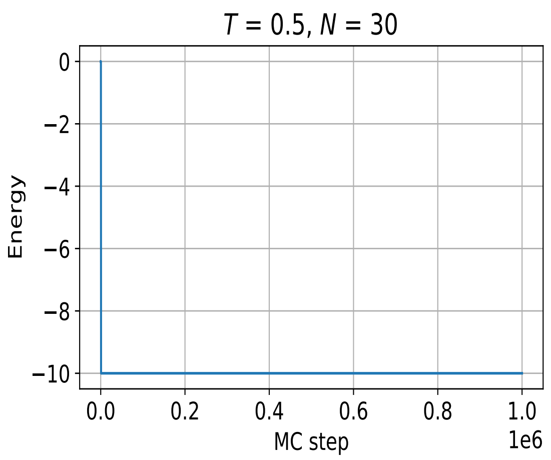
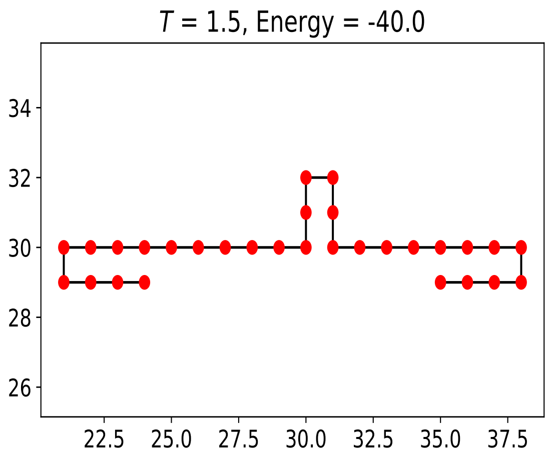


Figure 26: The protein structure for for $T = 0.5$ with $n = 1,000,000$ steps.

Figure 27: The plot showing the energy for corresponding Monte-Carlo steps for $T = 0.5$ with $n = 1,000,000$ steps.



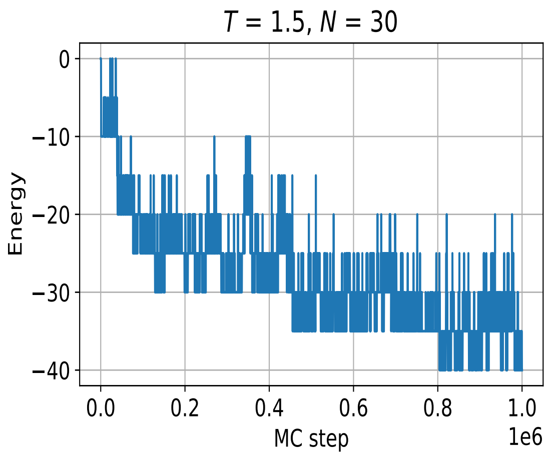


Figure 28: The protein structure for for $T = 1.5$ with $n = 1,000,000$ steps.

Figure 29: The plot showing the energy for corresponding Monte-Carlo steps for $T = 1.5$ with $n = 1,000,000$ steps.

```
CODE OUTPUT:
Energy averaged over last half of simulations for T=0.5 is: -10.00
Energy averaged over last half of simulations for T=1.5 is: -34.05
```

The typical energy of the protein (for the second half of the simulation) is lower in the case of $T = 1.5$ as stated above. Yes, this is as expected since the higher T allows the algorithm to have rapid movements and gives a chance to escape a local minimum and possibly find a global minimum. Looking at the final structure of the protein for $T = 0.5$ case given in *fig. 27*, we can see that the structure has not changed much from its initial structure of a straight horizontal line. This suggests that the algorithm might have been stuck in a local minimum and could not escape due to small T.

(c) Here we implemented the changing temperature system to simulate the folding of the protein. Other than updating the given code to the simulation we changed line 134 of our code:

```
if random() < np.exp(-(new_energy - energy) / T_array[j]):
```

After running the simulation we compared out values for the energy computed.

```
OUR CHANGED CODE OUTPUT:
Energy averaged over last quarter of simulations is: -48.90

OUR ORIGINAL CODE OUTPUT:
Energy averaged over last half of simulations is: -10.00
```

When we compare our results from part a to the value of energy we obtained from part c, we find that the energy of the new system in which temperature is incrementally decreased gives a smaller energy.

When the system is cooled very fast, it can tend to get caught in local minimums. But when it is cooled slowly through a staircase, it will **allow our system to take on as many configurations as possible before reaching our desired temperature $T = 0.5$,** this would give it the strongest chance at reaching the lowest configuration possible.

Explanation: If we start from a very low temperature, we are in the frozen state and not much is happening, since the T is very small resulting in a value very close to zero. Our random probability would have to pass this condition. There is a very low chance for a random number to be lower than this. If we start from a higher temperature, we are letting the system evolve and access more states before cooling down. This in turn results in a higher probability of getting a more stable low energy state. This makes better use of the Monte Carlo ”steps”.

(d) Finally, we quantitatively explore the temperature dependence of the energy of a particular protein by simulating the protein folding starting at $T = 10$ and stepping by $\delta T = 0.5$ until we reach $T = 0.5$. At each temperature the system is simulated for $500,000$ steps and the mean energy and the standard deviation in energy at that temperature is visualized in the plot with errorbar below:

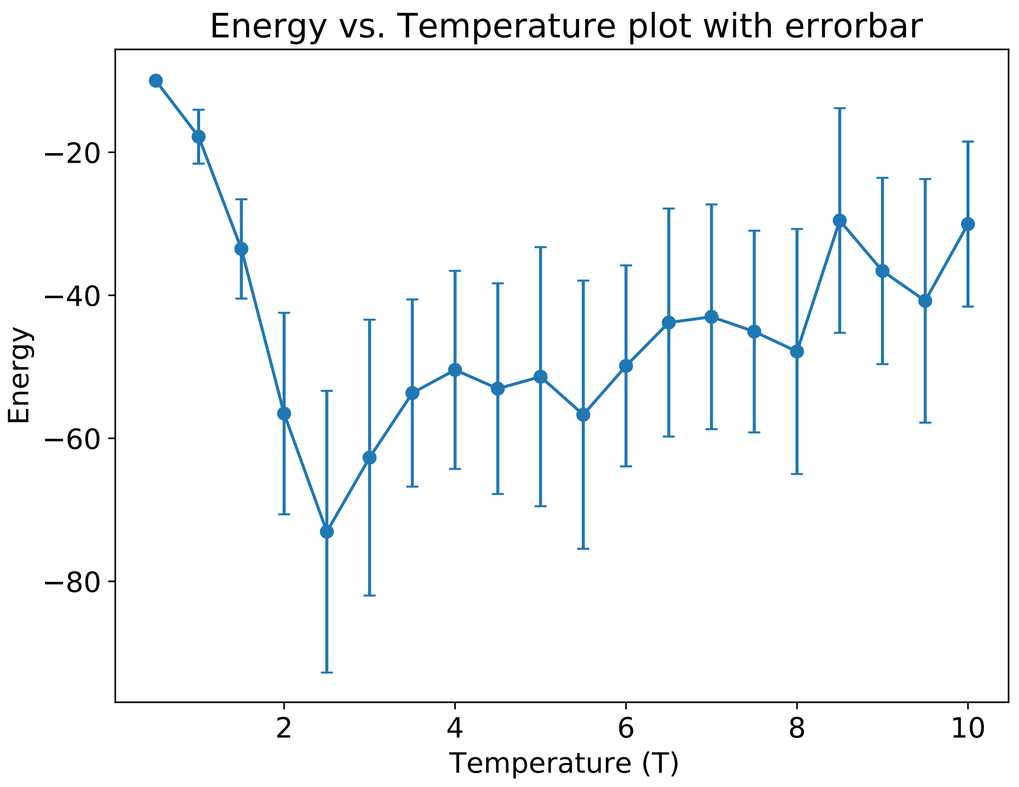


Figure 30: The plot showing the mean value of energy at the given temperature with the standard deviation visualized by using the errorbar.

From fig. 30 we find that the lowest average energy occurs at $T = 2.5$ while the standard deviation remains fairly similar across $T = 2$ and $T = 10$. For the lower temperatures, the algorithm might get stuck in a local minimum and fail to escape while for the higher temperatures, the algorithm might keep on jumping from one minimum to another and miss the global minimum. This explains the reason why energy decreases until $T = 2.5$ and starts increasing again. From fig. 30 there seems to be a couple of instances for a phase transition particularly from $T = 1.5$ to $T = 2.0$ and from $T = 8.0$ to $T = 8.5$ where there is a sharp change in energy over a small temperature range.