PHY407H F: Explanatory Notes For Lab 11

Idil Yaktubay and Souren Salehi

5 December 2022

1 Question 1 - By Souren Salehi

(a) In this part we used to code contained in `salesman.py` from the textbook and we looped it over different values of time constant to see the effect of speeding up and slowing down the cooling function. We also looped over different seeds to see the impact of taking different paths.

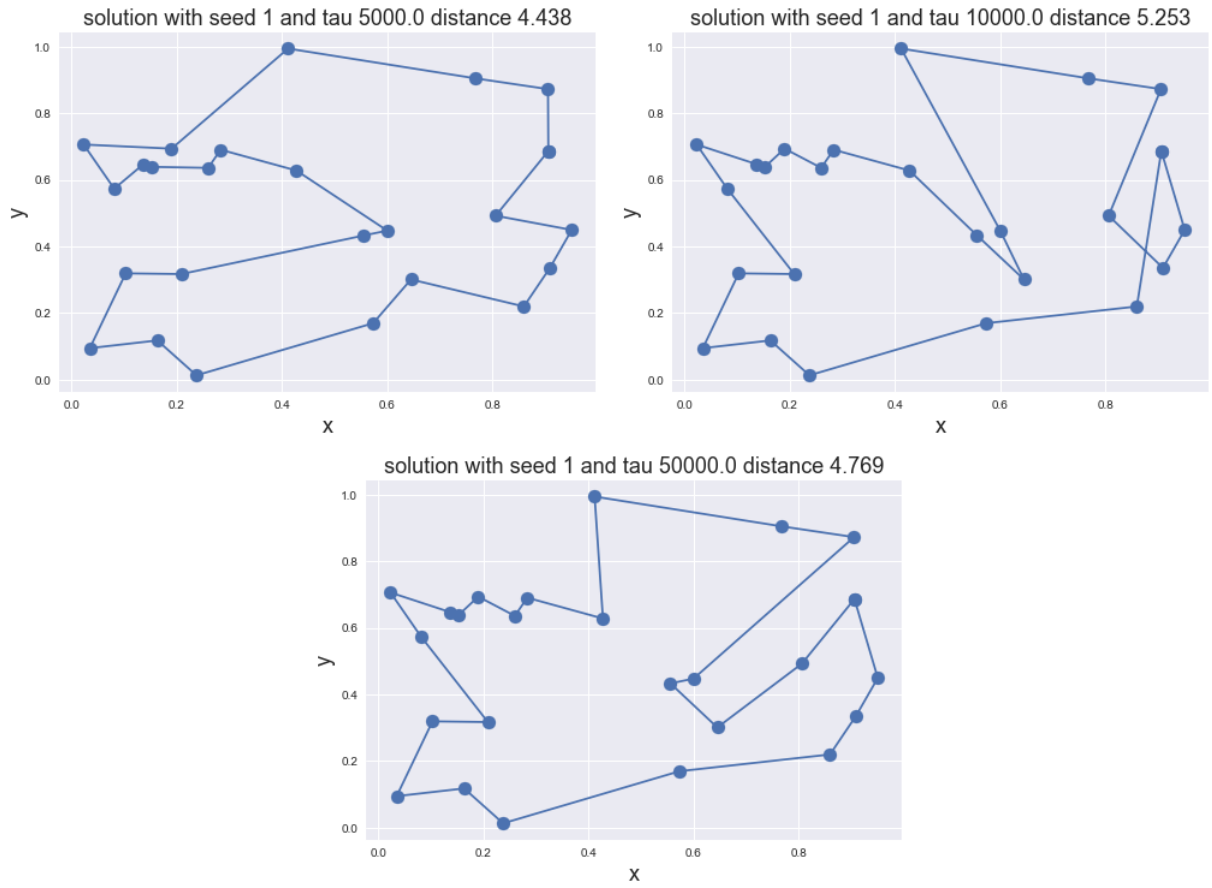


Figure 1: Plot of the solution to the salesman problem with seed equals set to 1 with different time constants of $5 \cdot 10^4$, $1 \cdot 10^5$ and $5 \cdot 10^5$ respectively

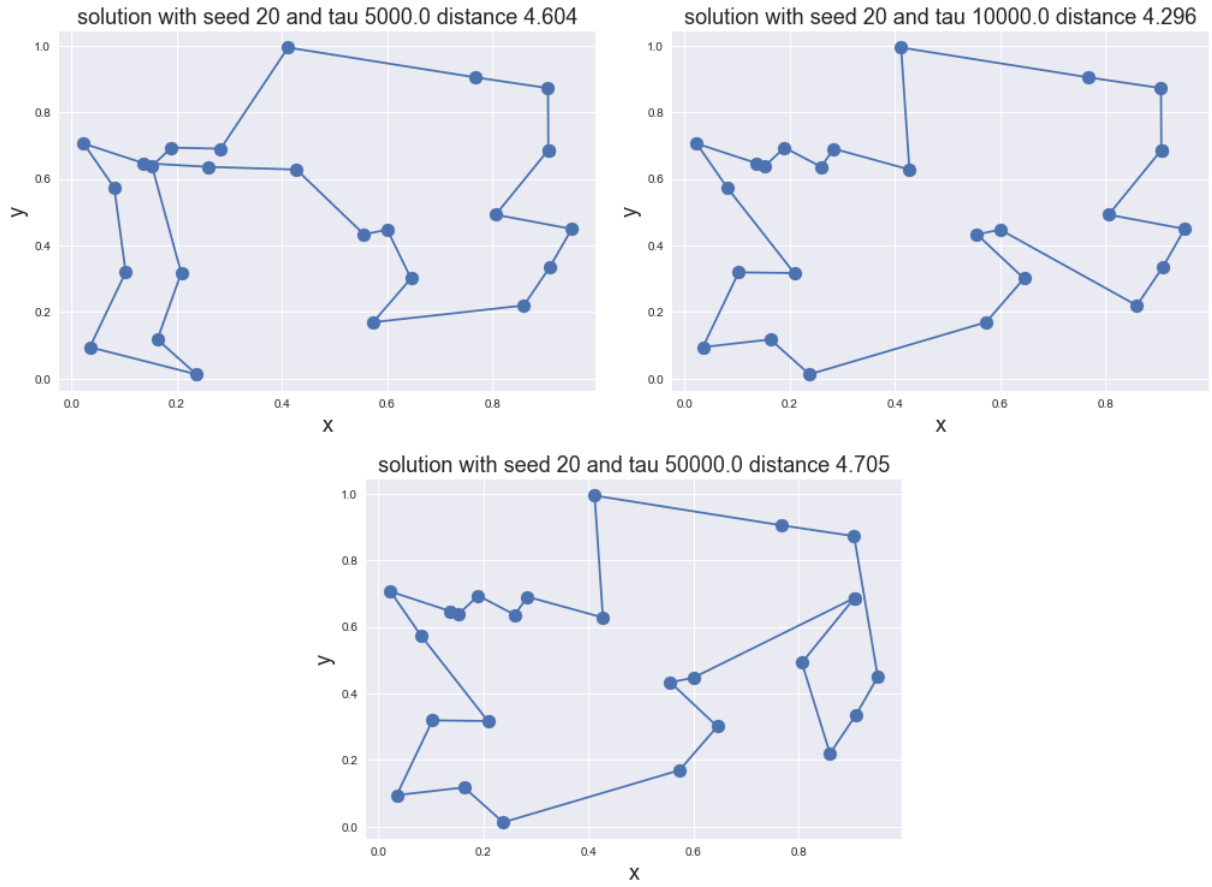
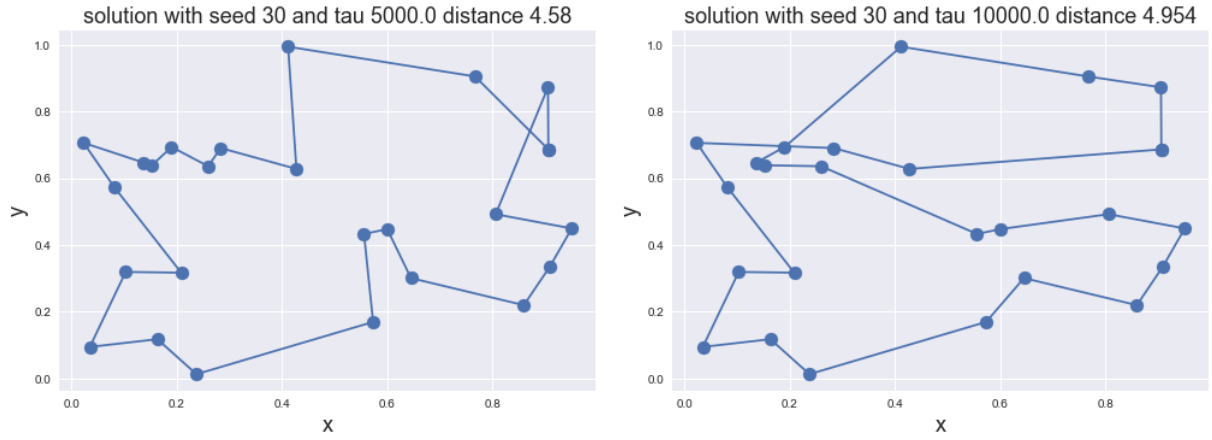


Figure 2: Plot of the solution to the salesman problem with seed equals set to 20 with different time constants of $5 \cdot 10^4$, $1 \cdot 10^5$ and $5 \cdot 10^5$ respectively



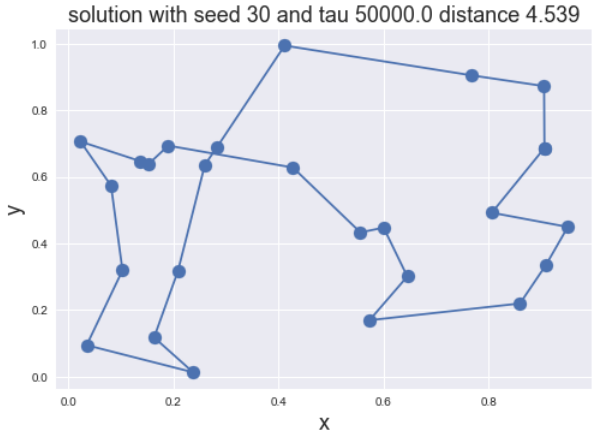


Figure 3: Plot of the solution to the salesman problem with seed equals set to 30 with different time constants of $5 \cdot 10^4$, $1 \cdot 10^5$ and $5 \cdot 10^5$ respectively

Comparing the distances for the same time constant and different paths taken, we can see that the path can have a noticeable impact on the total distance. Comparing the solutions with time constant 100000, for seeds 1 and 20, we see that it made a large difference as the distance changed from 5.253 to 4.296.

As we change the time constant however we see varying effects for the same seed numbers. On average the larger taus seems to be allowing for the shortest distance however that isn't a consistent result. For example the shortest distance for the 1 seed is the $5 \cdot 10^4$ time constant rather than the larger values. So we notice that on average the distances get larger with slower cooling, however it is not the case in every scenario.

(b and c) In parts b and c we modified the simulated annealing code for the traveling salesman problem to minimise the functions

$$f(x, y) = x^2 - cos(4\pi x) + (y - 1)^2, \tag{1}$$

for part b and

$$f(x, y) = cosx + cos(\sqrt{2}x) + cos(\sqrt{3}x) + (y - 1)^2, \tag{2}$$

for the range $0 < x < 50, -20 < y < 20$.

The solutions we obtained for b and c respectively are shown in figure 4 below.

```
for Q1b, the solutions are x= -2e-05 y= 0.9793
for Q1c, the solutions are x= 15.96072 y= 0.99511
```

Figure 4: Code output of file `lab11-Q1.py` for the solution to the minimisation of equation 1 and 2 using simulated annealing.

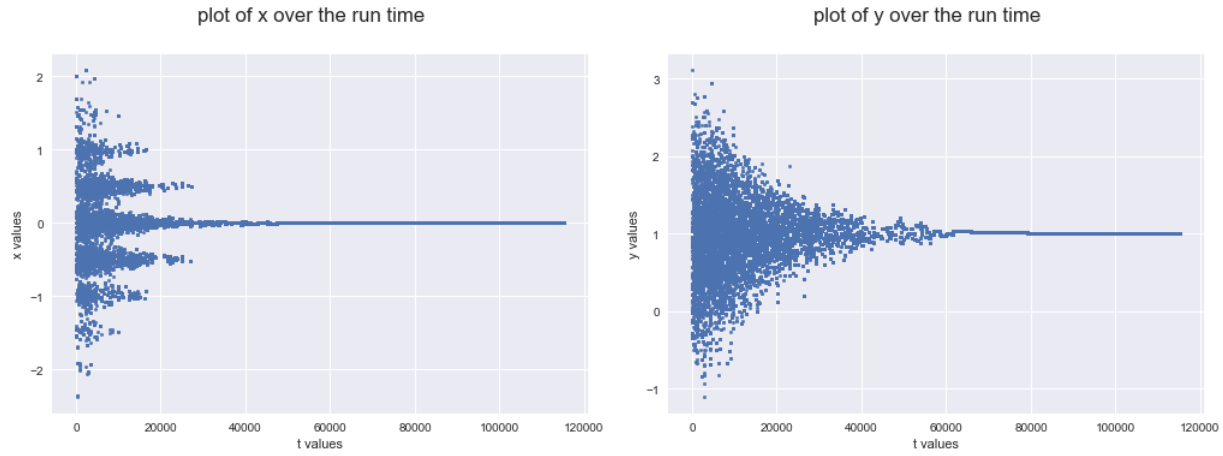


Figure 5: Spread of the values of x and y for the solution to the minimisation of equation 1 over the run time of the loop.

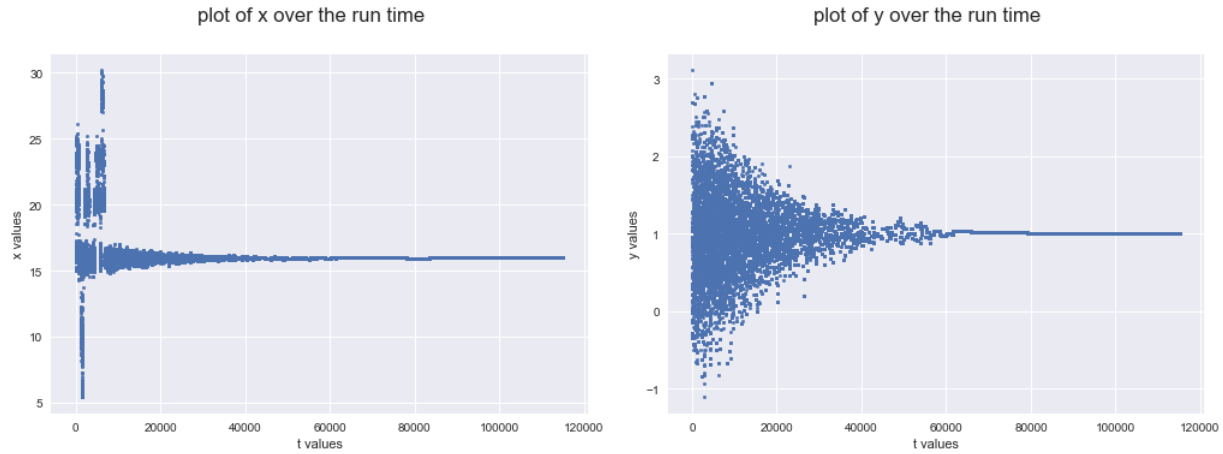


Figure 6: Spread of the values of x and y for the solution to the minimisation of equation 2 over the run time of the loop.

Figures 5 and 6 demonstrate how the simulated annealing allows us to converge to the answer to the minimisation as we take T to go to zero. For part b we obtained an x value of $-2 \cdot 10^{-5}$ and a y value of 0.9793 where we expected 0 and 1 for the values respectively. For part c we obtained values of 15.96072 and 0.99511 for x and y respetively with expected values of 16 and 1.

2 Question 2 - By Idil Yaktubay

(a) In this question, we have used `L11-protein-start.py` to generate Monte Carlo simulations of protein folding. In each case, we started with horizontal proteins of zero energy and used different temperatures. The generated protein molecules are two-dimensional and consist of a single type of amino acid. First, we found the optimal tertiary structure of a protein molecule with default parameters $N = 30$, $T = 1.5$, $\epsilon = -5$, and $n = 10^5$, where N is the number of amino acids in the protein molecule, T is the temperature, ϵ is the interaction energy between unconnected but adjacent amino acids, and n is the number of Monte Carlo steps. Figure 7 depicts the optimal tertiary structure of a protein molecule generated with these parameters, and Figure 8 depicts the energy of the same molecule at every Monte Carlo step.

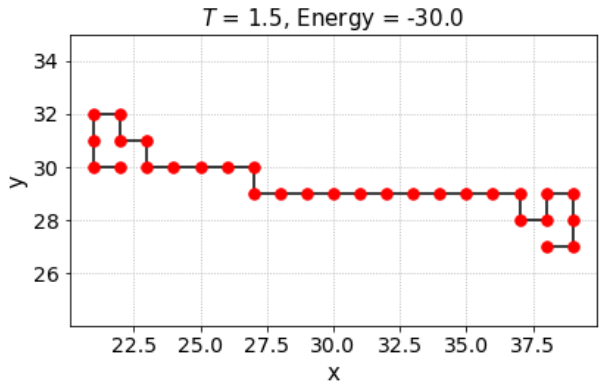


Figure 7: Optimal tertiary structure of a two-dimensional protein molecule with identical amino acids. This was generated using the Markov chain/Metropolis method with parameters $N = 30$, $T = 1.5$, $\epsilon = -5$, and $n = 10^5$. The energy of this molecule is -30.0. The code is contained in `L11-protein-start.py`.

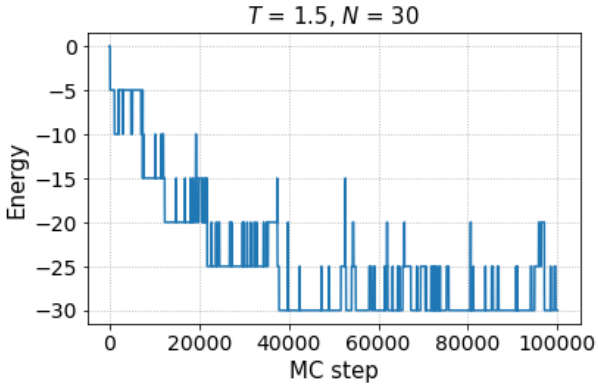


Figure 8: Energy at each Monte Carlo step during protein folding for the same parameters from Figure 7. The average energy in the last half of the plot is -28.78. The code is cotnained in `L11-protein-start.py`.

On Figure 8, we notice that the energy starts at 0.0 in the beginning of our calculations and gradually drops until it reaches -30.0. Roughly speaking, Figure 8 seems to show that the energy levels off, or equilibrates, at around the 38000th Monte Carlo step. The average energy of the last half of the simulation was -28.78. This means that the number of Monte Carlo steps we have used was fairly sufficient for us to observe a consistent levelling-off, though we could use more steps for a more accurate average energy. Also, on Figure 7, we see that the optimal tertiary structure has six adjacent but unconnected monomers. Given our interaction energy of $\epsilon = -5.0$, our calculated equilibrium energy of -30.0 makes sense.

We have generated new optimal tertiary structures with $T = 0.5$ first and with $T = 5$ second and by keeping the remaining parameters the same. Figures 9 and 10 depict the optimal tertiary structure and the energy at each Monte Carlo step for $T = 0.5$, respectively, and Figures 11 and 12 depict the same for $T = 5.0$, respectively. We notice that in Figure 9, the protein has three unconnected but adjacent monomers, hence the energy of -15.0. This shows that the protein has not changed significantly from its initial horizontal position. On Figure 10, the energy levels off at roughly around the 27000th Monte Carlo step, after which it does not fluctuate at all, making the average over the last half -15.0 as well. On Figure 11, the structure has eight unconnected but adjacent monomers and hence an energy of -40.0. In other words, the final structure is substantially different from the initial horizontal structure. Lastly, on Figure 12 we see that the energy values at each Monte Carlo step fluctuate a lot. Here, the average energy over the last half of the simulation is -44.31.

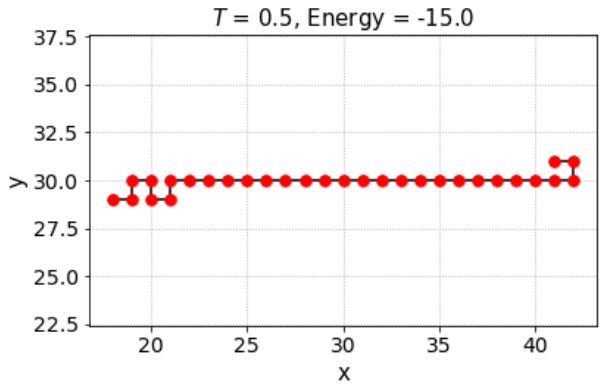


Figure 9: Optimal tertiary structure of a two-dimensional protein molecule with identical amino acids. This was generated using the Markov chain/Metropolis method with parameters $N = 30$, $T = 0.5$, $\epsilon = -5$, and $n = 10^5$. The energy of this molecule is -15.0. The code is contained in `L11-protein-start.py`.

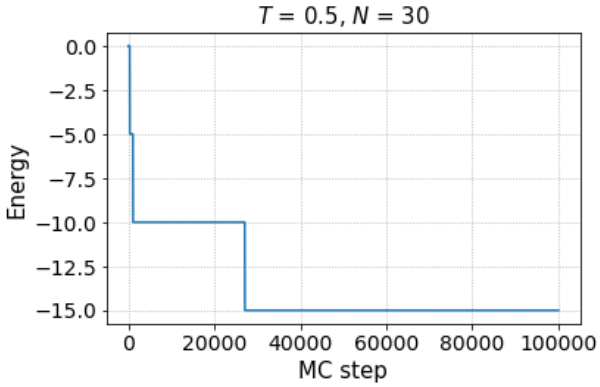


Figure 10: Energy at each Monte Carlo step during protein folding for the same parameters from Figure 9. The average energy in the last half of the plot is -15.0. The code is contained in `L11-protein-start.py`.

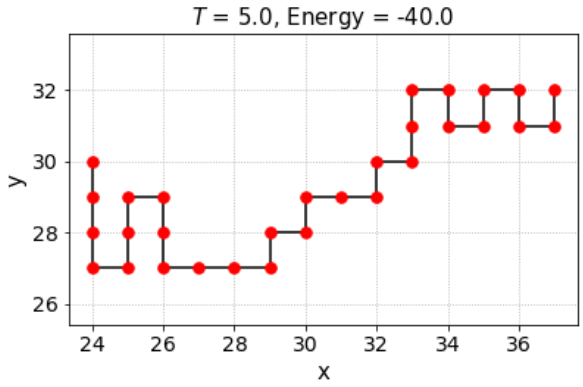


Figure 11: Optimal tertiary structure of a two-dimensional protein molecule with identical amino acids. This was generated using the Markov chain/Metropolis method with parameters $N = 30$, $T = 5.0$, $\epsilon = -5$, and $n = 10^5$. The energy of this molecule is -40.0. The code is contained in `L11-start-protein.py`.

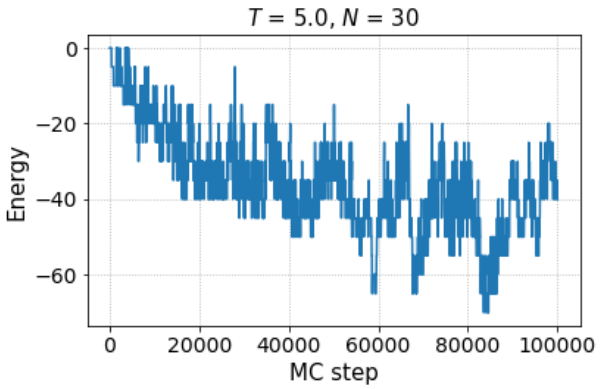


Figure 12: Energy at each Monte Carlo step during protein folding for the same parameters from Figure 11. The average energy in the last half of the plot is -44.31. The code is contained in `L11-protein-start.py`.

(b) Now, we have changed the number of Monte Carlo steps to $n = 10^6$ in `L11-protein-start.py` and generated the usual plots for $T = 0.5$ and $T = 1.5$ (Figures 13 and 14, and Figures 15 and 16 respectively). We observe that in the $T = 1.5$ case, the typical energy of the protein over the second half of the simulation is much lower. Specifically, the average energies in this portion are -10.0 and -63.66 for the $T = 0.5$ and $T = 1.5$ cases, respectively. This is expected for the following reason. In our Monte Carlo simulation, we first check how much energy it would take to make a move that does not stretch the protein, denoted ΔE. If this energy is positive, whether or not we make the move is determined with probability $e^{\frac{-\Delta E}{T}}$, allowing us to make a move even when we would be climbing up in energy (as long as T is nonzero). This probability decreases for smaller values of T, making it less likely for the amino acids to make new moves. Considering that our initial condition for the protein is just a straight horizontal line and the fact that probability of new moves decreases with T, it makes sense that our final tertiary structure is barely different from our initial horizontal structure for the lower temperature $T = 0.5$. Since the structure barely changes, the energy barely decreases after the Monte Carlo steps. In this case, we need a much larger number of Monte Carlo steps for the protein to reach equilibrium and thus a longer execution time.

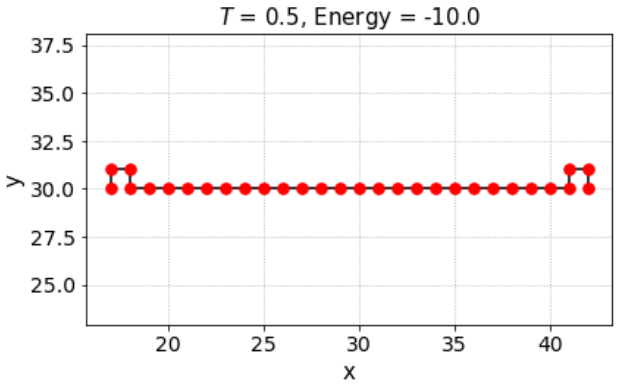


Figure 13: Optimal tertiary structure of a two-dimensional protein molecule with identical amino acids. This was generated using the Markov chain/Metropolis method with parameters $N = 30$, $T = 0.5$, $\epsilon = -5$, and $n = 10^6$. The energy of this molecule is -10.0. The code is contained in `L11-protein-start.py`

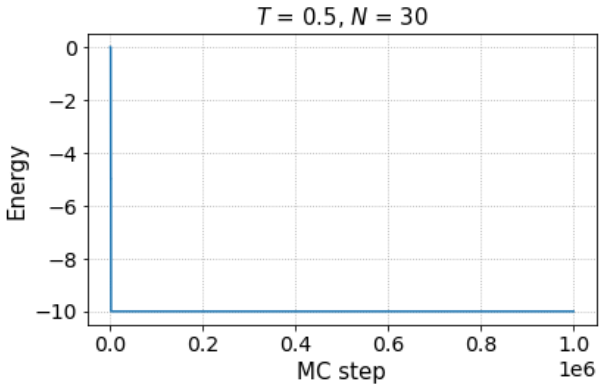


Figure 14: Energy at each Monte Carlo step during protein folding for the same parameters from Figure 13. The average energy in the last half of the plot is -10.0. The code is contained in `L11-protein-start.py`

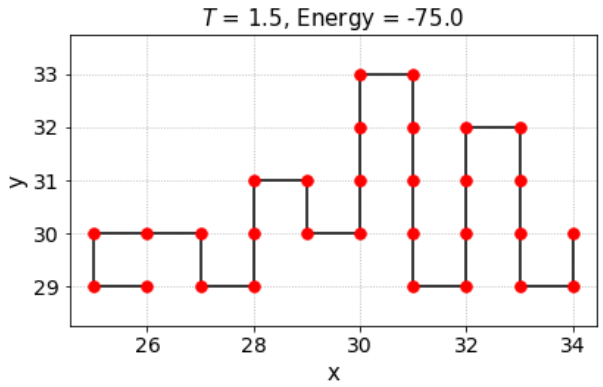


Figure 15: Optimal tertiary structure of a two-dimensional protein molecule with identical amino acids. This was generated using the Markov chain/Metropolis method with parameters $N = 30$, $T = 1.5$, $\epsilon = -5$, and $n = 10^6$. The energy of this molecule is -75.0. The code is contained in `L11-protein-start.py`

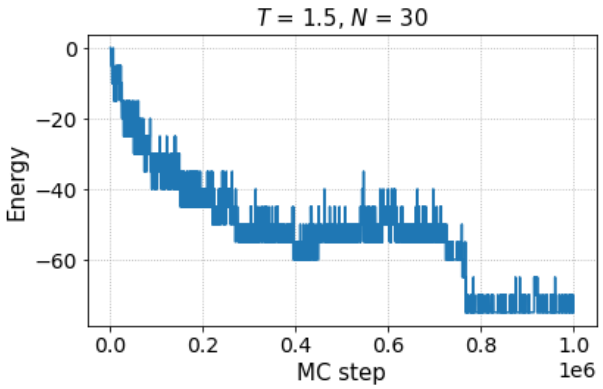


Figure 16: Energy at each Monte Carlo step during protein folding for the same parameters from Figure 15. The average energy in the last half of the plot is -63.66. The code is contained in `L11-protein-start.py`

(d) We can make sure that our protein molecule reaches equilibrium more quickly even when the temperature is low by starting the Monte Carlo simulation at a higher temperature and gradually decreasing the temperature at each step until the end of the simulation. We have done this by starting at $T = 3.5$ and decreasing the temperature by 1 over $T_{step} = 4$ steps until it reaches the lowest temperature $T_f = 0.5$ at the last step for $n = 2 \times 10^6$. With this method, we found the average protein energy over the last quarter of the simulation to be -71.98, which is significantly lower than the final value of -10.0 from part (b). Here, by allowing the protein to have higher probabilities to transition in cases where ΔE is positive, we made sure it reached equilibrium much quicker. Our final tertiary structure also looked substantially different from the starting horizontal position, unlike last time. To compare this method with the previous method more fairly, we also ran our program for the same number of Monte Carlo steps as last time ($n = 10^6$) and saw that the final energy was -65.0, which is again much lower than the value from part (b). In summary, this method allows the protein to reach equilibrium with fewer Monte Carlo steps and thus much more quickly. The code that implements this method is contained in `Lab11_Q2.py`.

(e) To quantitatively explore the temperature dependence of a protein, we have simulated protein folding for a set of temperatures starting at $T = 10$ and stepping by $\delta T = 0.5$ until $T = 0.5$ is reached. For each temperature, we have used 500 000 Monte Carlo steps. To induce equilibrium at low temperatures, we have implemented the cooling down technique from part (d) for $T \leq 1.5$. For each temperature, we have recorded the average energy of the protein and the corresponding standard deviation in the last half of the simulation. Figure 17 depicts the average energy versus temperature plot. Taking the error bars and the overall shape of the plot into account, it looks like there is a fairly smooth variation of energy with temperature, indicating no sharp jumps in energy. Based on our plot, there is no evidence for a phase transition. However, Finkelstein and Galzitskaya (2004) suggest that real proteins undergo phase transitions to "ensure robustness of protein structures" (23). Therefore, we can conclude that our computational model fails to display protein phase transitions. This is perhaps because we chose $N = 30$ when real proteins usually

have "50 to several thousand amino acids" (from lab manual). Our code for this part is contained in `Lab11_Q2.py`.

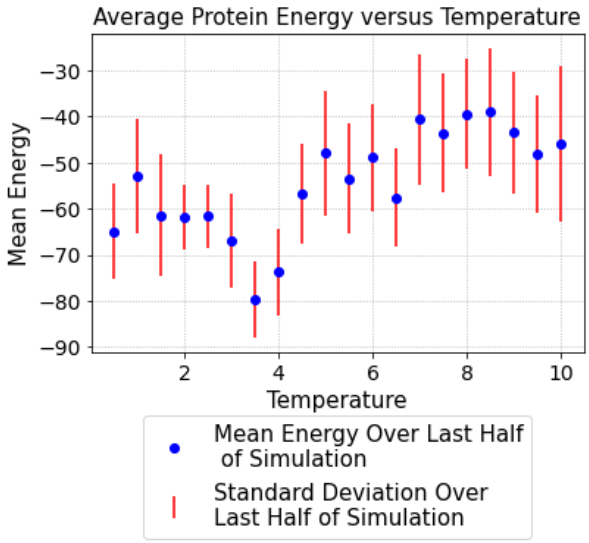


Figure 17: Average energy with respect to temperature for protein with 30 monomers. The energy value at each temperature was obtained by averaging over the last half of Monte Carlo simulations with $n = 5 \times 10^5$ steps. The temperature values were picked from a range of $T = 10.0$ to $T = 0.5$ with step sizes of $\delta T = 0.5$. For $T \leq 1.5$, the cooling down technique from part (d) was used. The interaction energy was $\epsilon = -5.0$. The code is contained in `Lab11_Q2.py`.

3 References

Finkelstein, A. V. and O.V. Galzitskaya. 2004. "Physics of Protein Folding." *Physics of Life Reviews* 1: 23-56. https://doi.org/10.1016/j.plrev.2004.03.001.