An Efficient Algorithm for Exploring Protein Configurations with Two and Three Polarities

Sofia Gil Rodrigo

Statistical Biophysics

June 10, 2024

1 Introduction

The folding of proteins is a fundamental process in biology that determines the three-dimensional structure of proteins, which is essential for their function. Understanding the mechanisms underlying protein folding has been a long standing goal in biophysics, yet with no final solution. However, several models that try to approximate the process have been developed and one of the simplest ones is the Hydrophobic-Polar (HP) model. This model classifies amino acids into two categories: hydrophobic (H) and polar (P), which facilitates the study of protein folding by focusing on the tendency of hydrophobic residues to cluster away from the aqueous environment.

In this project we explore the classical HP model together with a version that includes three different polarities. Our goal is to determine the optimal folding configuration without considering a specific protein sequence, with the objective of approximating structures that are frequent in real proteins. We do this in a regular two-dimensional grid of size 6x6.
Using Python and the *numpy* library for efficient matrix operations, we implement an algorithm that allow us to compute the energy of all the possible configurations for a given polarity sequence. We then iterate over a large number of random polarity sequences to identify the configurations with the lowest energy states.

2 Methods

We program all the algorithms using Python and mainly the package *numpy* for optimizing the matrix operations. The first step to obtain the optimal configuration is to give a polarity to the aminoacids. We generate a random list of 36 polarities, one for each aminoacid where we use -1, 0 and 1 to represent polar, hydrophobic and neutral aminoacids. For the case with two polarities we sample the polarities at random between -1 and 1.

Therefore, for each polarity sequence we calculate the energy of all the possible configurations and give each configuration a point if it has the lowest energy value. The energy is computed by using the equation in [1]:

$$H = \sum_{i<j} e_{\nu_i \nu_j} \Delta(r_i - r_j) \tag{1}$$

where $\Delta(r_i - r_j) = 1$ if r_i and r_j are next to each other in the lattice but not neighbors in the path and zero otherwise. The interaction energies are given by [1, 2], which are shown in Table 3 and Table 7.

In total, there are 2^{36} different polarity sequences and even though we put a lot of effort into optimizing the code, it is still computationally expensive to calculate the energy of all the configurations for all the sequences. For this reason we computed the energy of all configurations for 2^{17} random polarity sequences. As a next step, to be able to increase the number of sequences we selected the 1000 configurations with the highest count of lower energies and repeated the process for these 1000 configurations but with 2^{20} polarity sequences. Out of these results, we again reduced to the 100 configurations with the highest count of lower energies and, finally, repeated again the process for these 100 configurations, but with 2^{24} polarity sequences. We

did this for both two and three polarities configurations. The final results are the optimal paths for each case.

Efficient algorithm design

As mentioned before, iterating over a large number of polarity sequences and all the configurations for each sequence is computationally expensive. To make this process as efficient as possible we developed an approach based on adjacency matrices of different useful networks.

The first step is to load the dataset, which is an array of size (28728, 36), containing the 28728 different configurations of the protein in a lattice of size 6x6 and 36 aminoacids. Once this is set, the three functions that we will need to create and optimize are:

- Function to generate a random polarity sequence
- Function to load a given configuration
- Function to compute the energy of a given configuration

The function to generate a random polarity sequence is straightforward, we just need to sample the polarities between -1 and 1, for the two polarities case:

```
def get_polarity_sequence(n):
# n is the number of aminoacids
polarity = [1, -1]
polarity_sequence = [random.choice(polarity) for i in range(n)]
return np.array(polarity_sequence)
```

The next step is the function to load a given configuration. Here we need to previously think of the form in which we want to store the configuration information. Because of the adjacency matrix approach that we will use to compute the energy, we need to prepare two matrices: one representing the base network and the other representing the path network.

The base network is a network where the nodes are each point in the 6x6 lattice and the edges are the connections between the points in order to form the regular grid (Fig. 1). This network has 36 nodes (aminoacids) and therefore can be represented by a 36x36 adjacency matrix, that we name base adjacency matrix (Fig. 2a). Notice that we are not interested in working with networks but rather with their adjacency matrices, due to their optimized Python libraries. Then, we also need the weights network, where the nodes also represent the aminoacids but the edges represent the connections between the aminoacids in the given configuration that we are loading. Its corresponding adjacency matrix will also be of size 36x36 and an example for a specific configuration is shown in Fig. 2b.

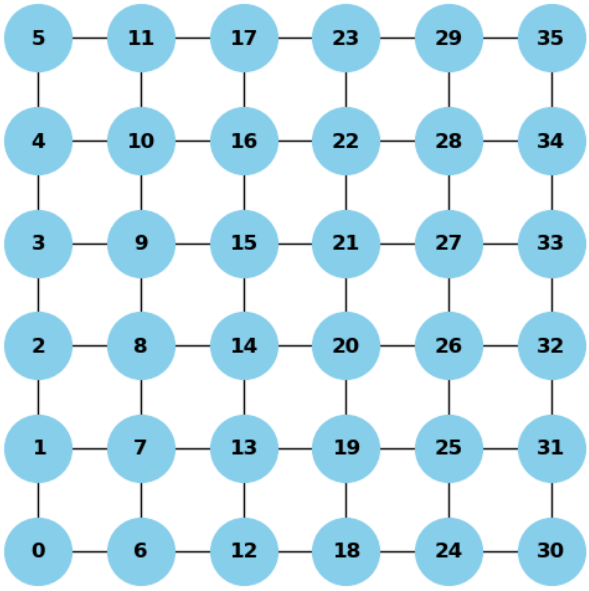


Figure 1: Base network for a 6x6 lattice.

The base adjacency matrix is the same for all the configurations, so we can compute it once and use it for all the configurations. The weights adjacency matrix is the one that we will need to load for each configuration. The function that returns the weights adjacency matrix for a given configuration is the following:

```
def get_weights_matrix(index, grid_paths, shape):
# index: index of the configuration
# grid_paths: array with all the configurations
# shape: shape of the lattice

path = grid_paths[index] - 1

# make path element integers
path = path.astype(np.int32)

# initialize the weights matrix
weights = np.zeros(shape)

# set the weights
for i in range(len(path) - 1):
    weights[path[i], path[i + 1]] = 1
    weights[path[i + 1], path[i]] = 1

return weights
```

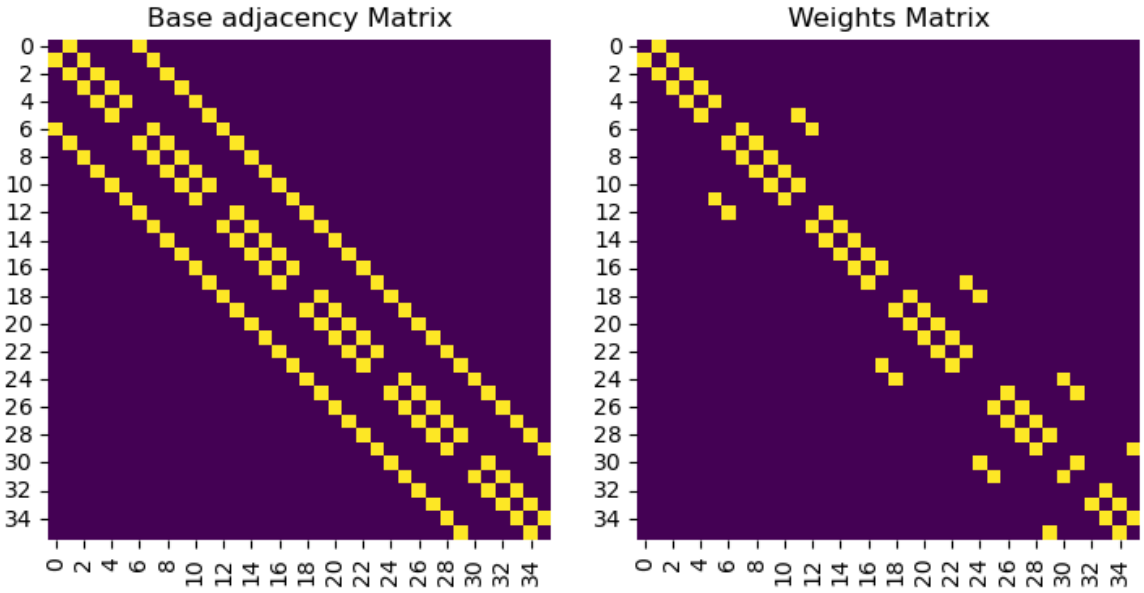


Figure 2: Base adjacency matrix (a) and weights adjacency matrix for a given configuration (b).

Finally, the function to compute the energy of a given configuration is the most important one. We need to compute the energy using the equation (1) and the interaction energies given in [1, 2]. In the approach we develop we try to avoid using if statements, which are computationally expensive.

For the two polarities case, we have 1 and -1 as polarities, so we can use the possible outcomes of the sum of these values as indices for an array containing in the appropriate position the interaction energies. This would help us to avoid using if statements to check each pair of aminoacids polarities. It is shown in Tables 2, 3 and 4. Notice that it is essential that the sum of the polarities is not repeated.

Once we have the array with the interaction energies, we can compute the energy without if statements by doing the operation:

$$energy\ configuration = \sum_{i>j} e_{p_i+p_j} \cdot (A_{ij}^{base} \cdot (1 - W_{ij})) \tag{2}$$

Notice that this is equivalent to the equation (1). The $e_{p_i+p_j}$ is the interaction energy which is indexed by the sum of the polarities of the aminoacids (Tab. 4). The other part of the equation is the equivalent to the $\Delta(r_i - r_j)$. We need an operation that gives 1 if the aminoacids are next to each other in the lattice but not neighbors, in our case this is the equivalent to the element of the base adjacency matrix to be 1 and the element of the weights adjacency matrix to be 0. Therefore, the operation $A_{ij}^{base} \cdot (1 - W_{ij})$ will give 1 if the aminoacids are next to each other in the lattice but not neighbors and 0 otherwise.

Table 1: Polarity values for two polarities.

p	h
1	-1

	p	h
p	2	0
h	0	-2

Table 2: Sum of the polarities for all the possible pairs.

	p	h
p	0	-1
h	-1	-2.3

Table 3: Interaction energies for two polarities.

i	0	1	2	-2	-1
e	-1	0	0	-2.3	0

Table 4: Interaction energies indexed by the sum of the polarities.

For the case of three polarities, we use the same approach. The only difference that we need to take into account is that, when generating the random polarity sequence, we need to sample between three values whose sum cannot be repeated, so that we can use them as indices for the array with the interaction energies. We find that a possibility is to sample between 2, -1 and 0, corresponding to polar, hydrophobic and neutral aminoacids. The sum of the polarities is shown in Table 6 and as we can observe there are no repeated sums. The interaction energies are given in Table 7 and the array with the energies indexed by the sum of the polarities is shown in Table 8.

Table 5: Polarity values for three polarities.

p	h	n
2	-1	0

	p	h	n
p	4	1	2
h	1	-2	-1
n	2	-1	0

Table 6: Sum of the polarities for all the possible pairs.

	p	h	n
p	-3	-1	-2
h	-1	-4	-2
n	-2	-2	-3

Table 7: Interaction energies for three polarities.

i	0	1	2	3	4	-2	-1
e	-3	-1	-2	0	-3	-4	-2

Table 8: Interaction energies indexed by the sum of the polarities.

This approach is very efficient because of the use of matrices and the absence of if statements. In addition, one of the main reasons to structure the code with *numpy* arrays is that we can use the *numba* package to compile the functions and make them faster. We use the *jit* decorator from *numba* to compile the functions and we can see a significant improvement in the speed of the code.

3 Results

By using all the methods explained in the previous section we were able to obtain the optimal paths for the two and three polarities cases in a reasonable amount of time. The first simulations done for all the configurations with 2^{17} polarity sequences took around 6 hours to compute. The second and third simulations with 2^{20} and 2^{24} polarity sequences took around 4 hours each.

The results of the optimal paths for the two and three polarities cases are shown in Fig. 3. In the paper we can see that the optimal paths resemble the alpha and beta sheets of proteins, and we made an interpretation of this resemblance in Fig 4. We discussed that the optimal path for the two polarities case resembles the alpha loop followed by a beta sheet and the optimal

path for the three polarities case resembles a beta sheet between two alpha loops.

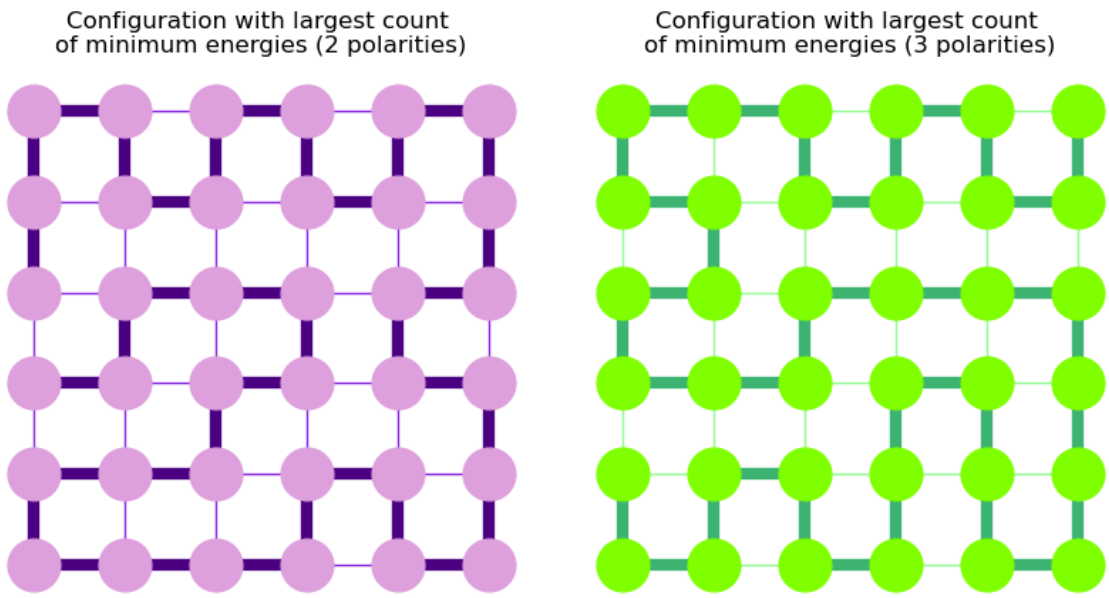


Figure 3: Configurations that give the maximum count of lower energies for the two and three polarities cases.

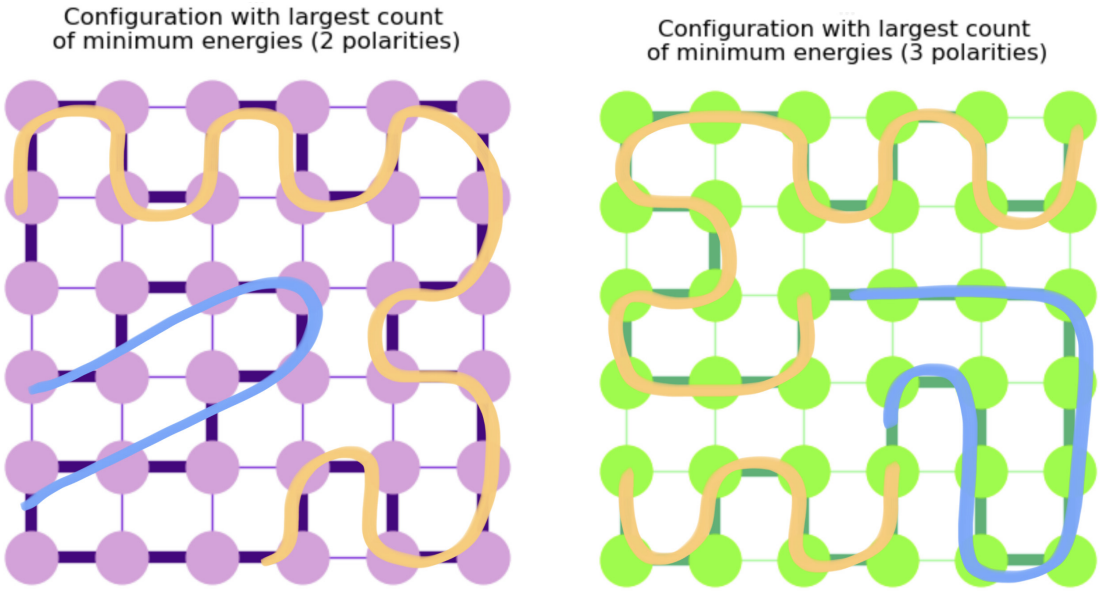
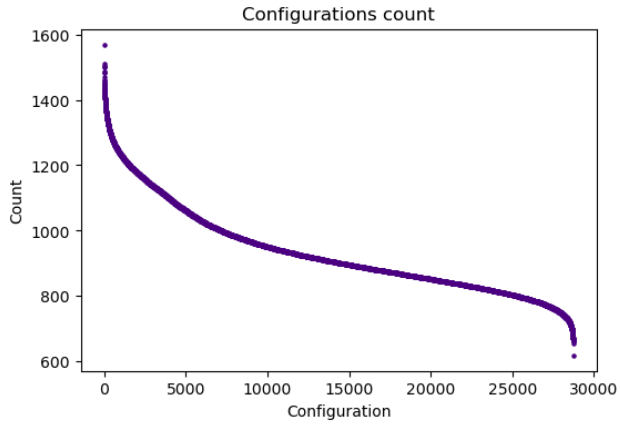
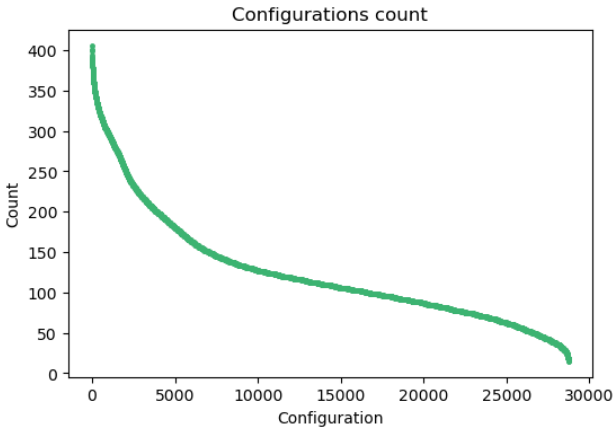


Figure 4: Interpretation of the beta sheet and alpha loop resemblance of the optimal paths. Alpha loops are represented in orange and beta sheets in blue.

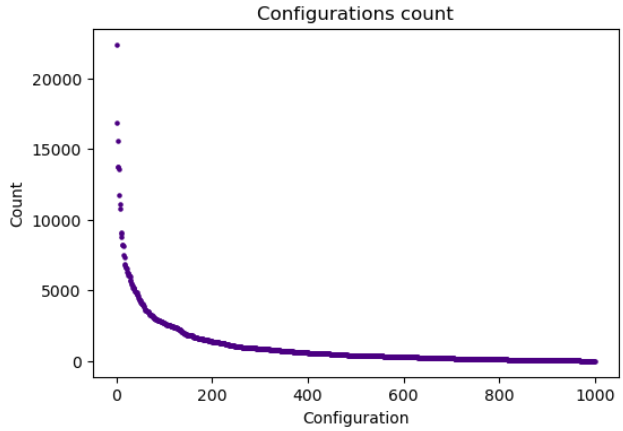
The rank of configurations with the lowest energy can be seen for the different intermediate steps in Fig. 5. In all the cases, we observe an exponential decay in the ocurrences of each configuration when ranked. This gives robustness to the fact that the configuration with the maximum ocurrence of lower energies is the optimal path.



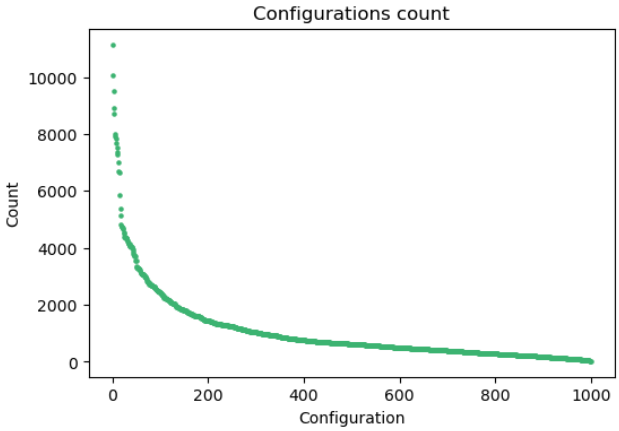
(a) Two polarities, all configurations, 2^{17} sequences.



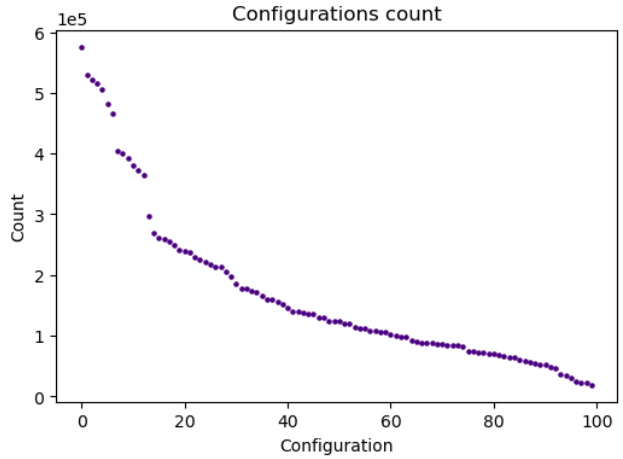
(b) Three polarities, all configurations, 2^{17} sequences.



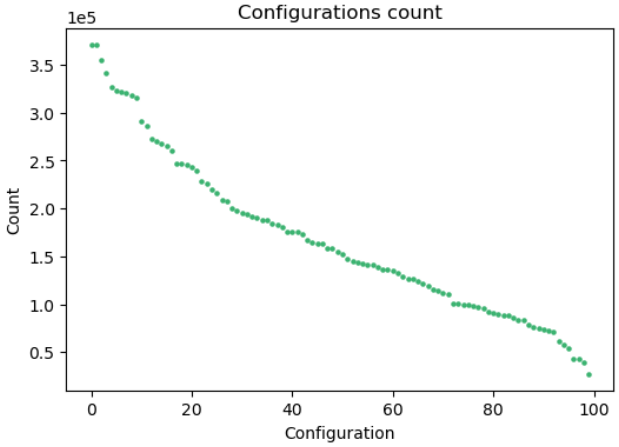
(c) Two polarities, 1000 configurations, 2^{20} sequences.



(d) Three polarities, 1000 configurations, 2^{20} sequences.



(e) Two polarities, 100 configurations, 2^{24} sequences.



(f) Three polarities, 100 configurations, 2^{24} sequences.

Figure 5: Count of lowest energy ocurrences for different configurations ranked from more common to less common.

Finally, we show the mean energy of each configuration in Fig. 6. It is important to take

into account that me mean value of the energy is not the most relevant measure, because the property that will make that configuration the optimal path for a given protein is having the lowest energy, not the lowest mean energy across different polarity sequences. However, it is still interesting to see that the mean energy of all configurations have a very small standard deviation. Furthermore, we also notice that there seems to be some correlation between the mean energy and the configuration index, which is expected because configurations with closer indices are more similar to each other.

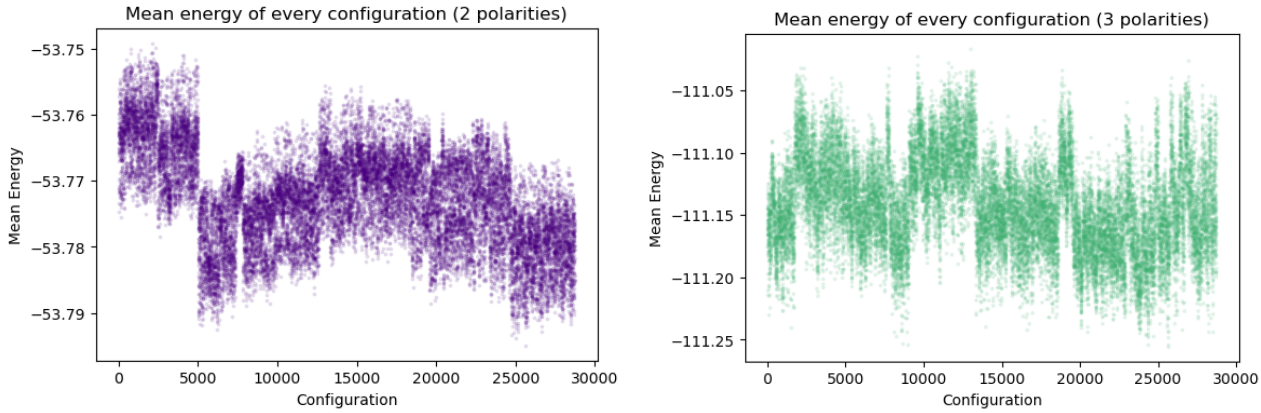


Figure 6: Mean energy of each configuration for the two and three polarities cases. In the x-axis we have the index of the configuration as they appear in the original dataset.

4 Conclusions

In this project, we developed a computational framework to identify the optimal configurations of amino acid sequences within a simplified lattice model. Using the HP model, we assigned polarities to amino acids and computed the energy of all possible configurations for a series of randomly generated polarity sequences. This process was iteratively refined to focus on the most probable configurations, which at the same time reduced the computational cost of the simulations.

We found that, as exposed in references [1, 2], the optimal configurations for both two-polarity and three-polarity cases keep some resemblance to the secondary structures of proteins, alpha helices and beta sheets. This resemblance suggests that the simplified HP model, despite its abstraction, captures key elements of protein folding mechanisms.

References

[1] Chao Tang. "Simple models of the protein folding problem". In: *Physica A: Statistical Mechanics and its Applications* 288.1 (2000). Dynamics Days Asia-Pacific: First International Conference on NonLinear Science, pp. 31–48. ISSN: 0378-4371. DOI: https://doi.org/10.1016/S0378-4371(00)00413-1. URL: https://www.sciencedirect.com/science/article/pii/S0378437100004131.

[2] C. Rebecca Locker and Rigoberto Hernandez. "A minimalist model protein with multiple folding funnels". In: *Proceedings of the National Academy of Sciences* 98.16 (2001), pp. 9074–9079. DOI: 10.1073/pnas.161438898. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.161438898. URL: https://www.pnas.org/doi/abs/10.1073/pnas.161438898.