Protein folding prediction by Genetic Algorithm in a HP model FCC lattice

Sofia Torres[1]

sofiasousatorres@gmail.com

Introduction

Since Anfinsen's time it is thought that the native 3d (or tertiary) structure of a protein in its standard physiological environment is determined only by its amino acid sequence (primary structure). Anfinsen Dogma also concludes that the native structure should be unique such that the native structure should have the lowest free energy possible for a conformation with that sequence. [1] However, with more or less 20 different possible amino acids, the search space for such a conformation is astronomically large, which not only makes it impossible for a cell to explore all possible conformations in order to find it but also makes it impossible to solve using a brute force algorithm. [2]

This need to reduce the search space with the goal of better understanding how proteins folds combined with the knowledge that hydrophobic interactions between amino acids is the main force driving protein folding**,** the HP model was born [3**]**. The HP model reduces all 20 amino acids with a binary question: is the amino acid hydrophobic or not? Then represents the amino acids in a 2D or 3D lattice. The free energy of a conformation is as low as the number of noncovalent contacts (a HH contact) between two amino acids that are adjacent in space but not in sequence. [4]

Although this is a greatly simplification of the mechanics of protein folding, it is helpful in understanding the basics principles of the mechanism and it still constitutes a NP-hard problem both in 2D and 3D.

This problem has been subject to many approaches, such as ions motion optimization [5], CSP [6] and many others.

Genetic algorithms have also been chosen to tackle this problem by some authors before, mostly notably [7].

Mostly authors have been comparing their solution with some benchmarking sequences (of real proteins), where the native HP free energy is known.

Approach

The solution consists of applying a genetic algorithm to evolve the fitness of a population of individuals that have its conformation restrained by a 3D Face centred cubic (FCC) lattice. The FCC lattice was chosen because it is the lattice that can compact the greatest number of (in this case) monomers (amino acids).

In this solution the genotype is not represented as a bit array but as a peptide object – a sequence of amino acids objects. This is because firstly, of the multiple difficulties of encoding the monomers multiple features – such as: a) if it is hydrophobic (H) or polar (P) b) localization in space. Second, the individuals are free to explore all the continuous space (in the lattice) making it hard to estimate the number of possible conformations it may have. Finally, both the crossover and mutation operators need a lot of restrictions in order to produce valid conformations, each would be difficulted by the bit array representation and would also entail a lot of computational heavy encoding and decoding. [8]

A population is composed of n peptides, each has a length of l – is composed of l amino acids. Each amino acid occupies one point of the FCC lattice. No point can be attributed to two different monomers making this problem a self-avoiding walk problem

In an FCC lattice each point has 12 neighbours. Two monomers (i,j) are neighbours if:

$|xi - xj| \leq 1$, $|yi - yj| \leq 1$, $|zi - zj| \leq 1$ and $|xi - xj| + |yi - yj| + |zi - zj| = 2$.

Such that each monomer is distanced $\sqrt{2}$ from each its neighbours. If a point is a vertice then it has 12 face points as neighbours and if it is a face, then it has 4 vertices neighbours and 8 face neighbours. Figure 1. shows the 12 neighbours of a vertice point.

It is important to notice that if two monomers are consecutive, that is $|i-j|=1$ then they also must be neighbours in order to be considered valid.

The **fitness function** counts how many HH contacts there is in an individual. The contact is established between two monomers (i,j) if they are both hydrophobic, are neighbours and are not consecutive in the peptide sequence: $|i - j| > 1$. Therefore, fitness of conformation c:

Fitness (c) = $\sum_{i=1}^{n-2} \sum_{j=i+2}^{n} HH(i,j)$ **,**
where HH is a contact.

The **initialization** of the population is in a stochastically way in order to introduce genetic variability to the initial population. The peptide – or chromosome is generated by initializing the first aa at the (0,0,0) position and adding the next aa in a neighbouring position of the previous aa without disobeying the self-avoiding walk principle.

Selection is made by k-tournament, k individuals are chosen and the one with the highest fitness is chosen to the next phase (crossover).

Crossover is implemented as single-point crossover. Where two chromosomes parents swap their conformation after a point, resulting in two children each with half of the parent's conformations. However, because of the self-avoiding walk principle the resulting children may not result in valid

conformations. **Figure 2.** Shows an example of crossing two individuals.

Mutation works by pull-move [9]. One random aa (i) is dislocated to a free neighbouring position, that new position needs to be neighbouring either the i+1 aa or the i-1 aa (one of the consecutive aas to i). If the random aa is the first or the last in the chromosome then the new position doesn't need to be neighbouring its consecutive aa.

When $i \neq 0$ and $i \neq l$ (where l is the length of the chromosome), for a move to be considered valid, the new location of i must be neighbouring at least either i+1 or i-1. If the new position of i neighbours both i+1 and i-1 then the position is already valid.

When say, the new i is neighbour of i+1 then i-1 will take the old position of i, this pulling will happen from i-1 to 0 unless one of those pulls produces an aa (i-n) that is already neighbour of its both consecutive aas, if i-n is neighbour of i-n+1 and i-n-1. If new i neighbours i-1 the events are symmetrical.

In the case that i=0 or i=l then the new position is already valid (as long as it is free) and the other aas will just take the old position of i-1 or i+1, respectably. This last detail differs from typical application of pull-moves.

Replacement applies selection, crossover and mutation several times such that the new population has the same number of individuals as the previous one.

Implementation

The project was implemented with python and can be access through: https://github.com/fistorres/FCC-folding-GA. It is also in the same file as is this report.
It can be ran through the command line by writing:
Python pop.py <iterations> <fitness> <sequence> <size><k> <cross> <t>. Where iterations: maximum number of iterations that we want to run, fitness: maximum fitness we want to reach, sequence: sequence of aas we want as chromosomes "HPHP…", size: size of population, k: number of individuals chosen for k-tournament, cross: rate of crossover and t:rate of mutation.

The prototype is composed of two main modules and a third module (benchmark.py, that has only some testing functions as well as the benchmarking sequences): main.py and Ga.py. Main.py.

The main module has the implementation of the Aa and Pep classes, where a pep is a peptide or chromosome, and Aa is an amino acid or "bit" in the chromosome. The pep class besides having the crossover and mutation methods also has a plotting method and the fitness calculation method. The Aa class is very simple only having the position and H/P as the two mains attributes, its methods consist of checking if two aas are consecutive or neighbors and return a list a free neighbor.

The Ga.py module consists of the Pop class and Ga class, with the only important methods being selection (k tournament) and replacement.

The GA class is a very simple class runs one iteration. Finally, to run the Ga one function runGa() is called.

Running the algorithms yields two plots, one with the meanest and highest fitness at each generation and the conformation with the highest fitness found. Figure 4 and 5 are examples of this.

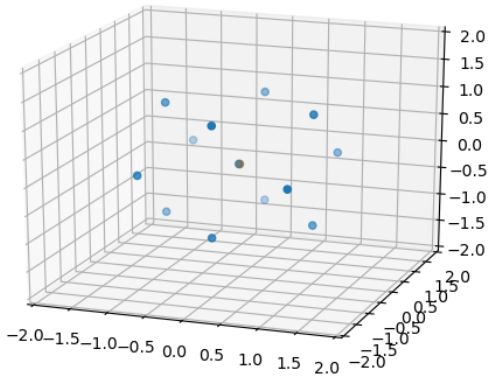

Figure 1: The twelve neighbors of the center point.

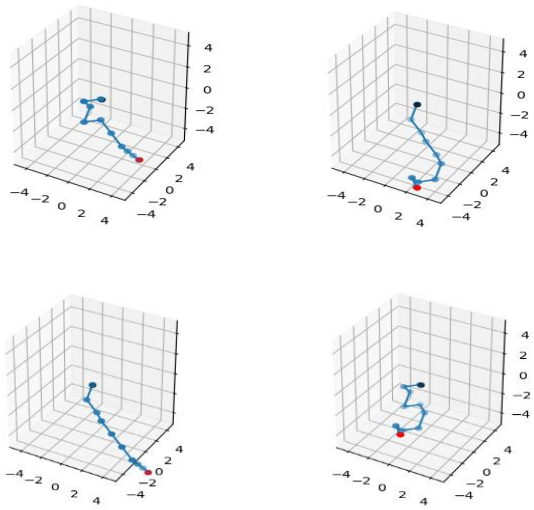

Figure 2: An example of crossover. The two conformations in the top are parents while the two conformations in the bottom are the result of a crossover at the sixth amino acid (counting from the red (1st one).

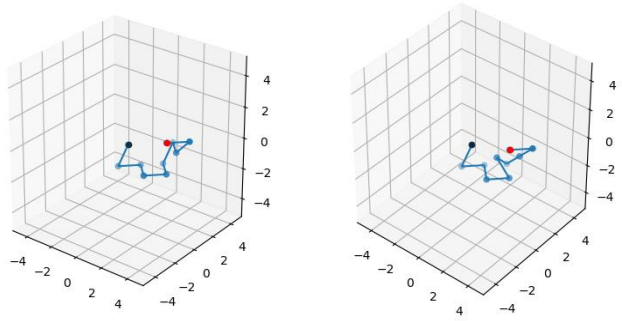

Figure 3: An example of mutation. The image on the left is the before and the right the after mutation. A mutation displaced the 5th monomer counting from the red one. The 6th monomer is still a neighbor. The 4th monomer took the old position of the 5th, however the 3rd didn't take the 4th old position since it is already a neighbor of the 4th new position.

Results

This implementation was tested with the following sequences:

	Length	Native HP energy	
S1	20	23	"HPHPPHHPHPPHPHHPPHPH"
H1	48	69	"HPHHPPHHHHPHHHPPHHPPHPHHHPHPHHPPHHPPPHPPPPPPPPHH"
F90_1	90	169	"PPHHHPPPHHPPPPHHPHHHHHHPHPHPHHPHHHHHPHHHPHPHHHHPHHPPPPHHH PHPHPPHHHPHHPHPHPHPPHHHPPPPHHPPHPPP"
S1-2	135	357	"HHHHPHHHHHHPPHHPHHHHHHHHHPHHPHHHHHHHHHHHPPHHPPPPPHHPHHHHH HHHPHPPHHPPPHHHHHHHHHPHHHHHHPPHHHHHHHHPHPPHHHHHHHHHHPPHHPPPHH HHHHHPHHPHHHHHHHHPPHHHH"
R1	200	384	"PPPHPHHPHHPPPHPHPPPPHPHHPPHPHHHHHPPHHPPHHHHHHPPHPPHHPPHPHP HHHHHPHHPHHHPPPHHHPHHPPHPHPPHPPPHPPHPPHPPHHHPHHHPHPPHPHHPHHH HPHPHHHPHHHPPPPPPHHHHHHPPPPPPPPHHHPPHPHPPPHPHPHPHHPPHHPPPPHHH HHHPPPHHPPPPPHPPPHHPP"

Table 1: List of sequences tested.

S1, H1 and **F90_1** were runned with the following parameters: Maximum iterations=100, number of individuals=100, k= 3 crossover rate=0.85, mutation rate = 0.2.

S1-2 and **R1** had the following parameters: Maximum iterations = 40, number of individuals = 80, k=3, crossover rate=0.65, mutation rate = 0.2.

These parameters were the best for this implementation. The parameter that seem to most contribute to finding an optimum has the crossover rate and the population size, since this allowed avoiding getting stuck at other local optimums, which appeared to happen very often for other parameters.

S1 reached 32 bonds at around the 20th iteration.

H1 reached 53 in at 100 iterations and 59 at 200 iterations, it is unclear if with more time the protein would reach a peak of 69 bons but by looking at figure 6A. it seems that although the fitness development slows, it doesn't reach a plateau yet making it possible to reach in future iterations.

S1-2 reached around 130 bonds, one thirds of the supposed maximum.

The values for **R1** weren't very interesting and made the algorithm very slow.

These results leave space for improvement.

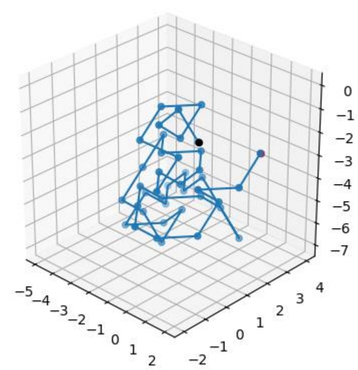

Figure 4: Solution found for sequence S1 with 32 bounds

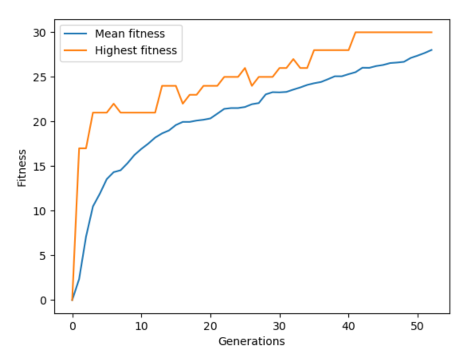

Figure 5. mean and highest fitness plotted against generation – same run as figure 4.

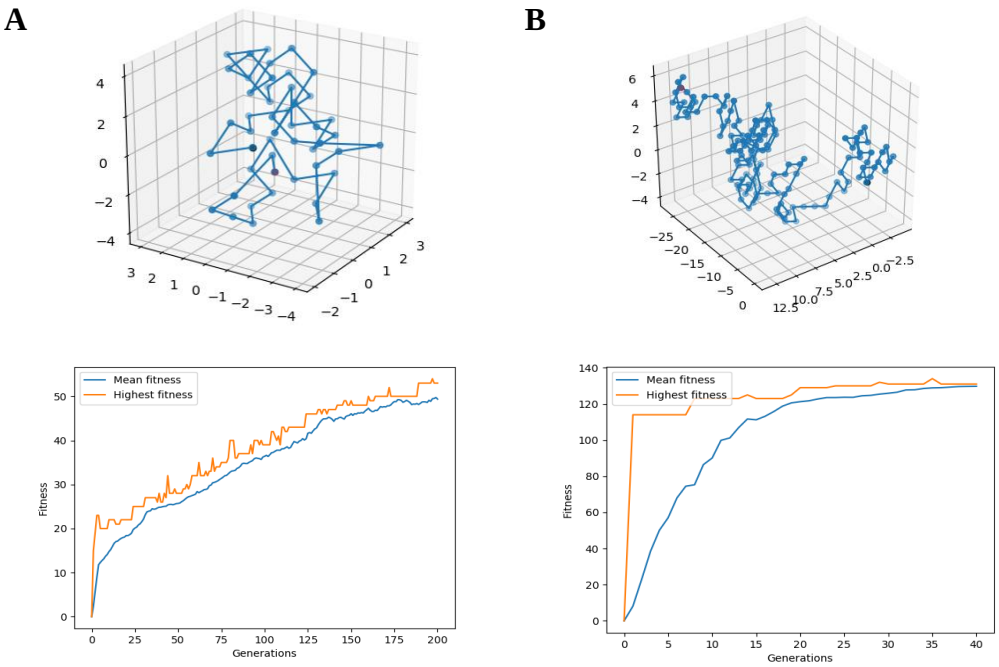

Figure 6: A corresponds to the H1 sequence with 53 bonds and B corresponds to the S1-2 sequence with ~130 bonds.

Comments

Unfortunately, this implementation doesn't perform perfectly with the benchmarking examples although I would say that the scores reached are not terrible either. Most of the times the algorithm gets stuck at a local optimum never finding the global optimum probably due to the lack of diversity. Since one of the features that seemed to improve this was the population size, this implementation would need to be rewritten in order to make it faster since right now it is not very practical with population size of 200 or above. Likewise, crossover should be improved to raise the number of viable children. By looking at the conformations alone it is clear that mutation and crossover operators don't seem to have been able to explore all the conformations leaving space from future improvements (which is made visible by the lack of compactness of some of the solutions).

Also, this prototype structure could be easily modified to allow for other than H/P amino acid classifications, for example: charged, polar, amphipathic and hydrophobic. Then the fitness function would need to be altered to account for more interaction, this approach however would not be in accordance to the HP model and the protein free energy calculated might not be very representative of the real phenomenon but it might be interesting to study the dynamics of the different interactions.

Nevertheless, this algorithm can find some conformations that appear to have biological meaning. In figure 6 we can see the packing of monomers inside a core – resembling a hydrophobic protein core.

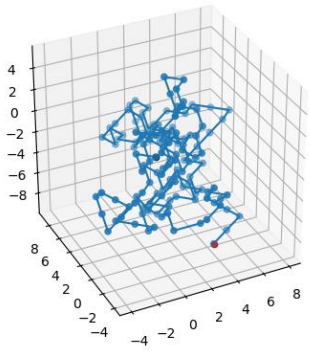

Figure 7: Example of how this algorithm can replicate some of the structures we see in real life.

References

[1] Anfinsen, Christian B.. "Principles that govern the folding of protein chains." Science 181 4096 (1973): 223-30.

[2] Zwanzig R, Szabo A, Bagchi B. Levinthal's paradox. Proc Natl Acad Sci U S A. ;89(1):20–22. doi:10.1073/pnas.89.1.20 (1992)

[3] Dill KA. Theory for the folding and stability of globular proteins. Biochemistry. 24(6):1501–1509. (1985)

[4] Protein Actions: Principles and Modeling, by Ivet Bahar, Ken A. Dill, and Robert Jernigan

[5] Biochemistry 1985, 24, 6, 1501-1509 (1985)

[6] Yang, CH., Wu, KC., Lin, YS. et al. BioData Mining (2018) 11: 17

[7] Tsay and Su: An effective evolutionary algorithm for protein folding on 3D FCC HP model by lattice rotation and generalized move sets. Proteome Science (2013)

[8] An Improved Real-Coded Genetic Algorithm Using the Heuristical Normal Distribution and Direction-Based Crossover. Jiquan Wang ,1 Mingxin Zhang,1 Okan K. Ersoy,2 Kexin Sun,1 and Yusheng Bi1 (2019)

[9] “Pull moves” for rectangular lattice polymer models are not fully reversible Dániel Györffy, Péter Závodszky and András Szilágyi 9(6):1847-1849 (2012)