HP model for protein folding: an overview and some applications

Gregorio Berselli, Isacco Faglioni

February 16, 2023

https://github.com/Grufoony/HP_model

Abstract

Purpose of this project is to provide the reader with a general view of the HP model for protein folding. Firstly, the report will focus on an overview of proteins and, in particular, why it's important to study how they fold. Next, the HP model will be presented in its mathematical form, introducing the two main approaches to this problem: the PERM algorithm and the Reinforcement Learning technique. Then, applications of the PERM algorithm to some real world sequence of amino acids will prove to be capable of obtaining outcomes compatible with the well known benchmarks. The comparison of these results to the ones produced by Deep Reinforced Learning will display the computational efficiency and precision of the latter algorithms. Lastly, a brief discussion of the results is given, together with a global resume on the whole topic.

Contents

Introduction

Chain molecules are a typical problem of statistical mechanics [1]. The main mathematical approach applied to chain molecules is the statistical mechanics framework, which usually applies statistical methods and probability theory to large groups of microscopic entities. In our case, the chain represents the system, and we're interested in the interactions between its beads. If we set up the problem considering the system in a canonical ensemble, which means that it can only exchange energy with the universe, we can write the *partition function* as

$$Z = \sum_i e^{-\beta \epsilon_i}$$

where ϵ_i is the energy of the i-th microstate and $\beta = \frac{1}{k_B T}$. From that we can define the *Helmholtz' free energy* of the system

$$F = -\frac{1}{\beta} \ln Z$$

and we know that the stable conformation of the chain will minimize this energy. The consequence of that approach is that if we knew all the microstates (possible conformations of the chain) of the system and their energies, we would be able to compute all thermodynamic potentials. Unfortunately, we are not able to compute every single microstate with its energy for long protein: the best we can do, as we will see, is to estimate the number of possible folds or try to find the global minima (solution) of the free energy.

In particular, this project will focus on the protein folding problem. How protein folds is a vital process that can be useful for a lot of purposes, from bioengineering to the medical applications. Understanding how proteins fold may help us to cure diseases like Alzheimer or anemia, in which proteins start to not fold correctly, treat virus infections and design more efficient drugs [2]. Moreover, having a tool which allows us to simulate the folding process can be useful in order to understand what happens when a mutation occurs in a protein. There are many reasons for mutations, including the radiation exposure and the environment itself [3]. A mutation is not always a bad thing because evolution starts from them, and we may also force some mutations in laboratory in order to get specific results.

The aim of this project is to report the main results in the developing of the HP model for protein folding. In order to do so, we'll first illustrate what the HP model is. Next, we will analyze two main approaches to this problem. The first one is a *brute force* approach, that allows us to understand the properties of a protein by computing all its possible configurations. However, a brute force approach is typically the last thing one would like to do. An alternative method is that of using Reinforcement Learning algorithms. This method constitutes an improvement of the brute force methods since it uses artificial intelligence to save computational time. Unfortunately the topic of RL is far too wide and complex for this report, for this reason the section on this method will cover extensively its most basic implementation and only describe qualitatively the state of the art models which are Deep Reinforcement Learning algorithms.

1 Proteins

Proteins are large biomolecules formed by one or more chains of amino acids. The genetic information contained in the DNA only specifies the sequence of these amino acids, the so-called *primary structure* of the protein. It has been experimentally observed that purified proteins, i.e. proteins purified away from other cellular components, tends to spontaneously refold after being completely unfolded [4]. Understanding why and how protein folds may be a vital task for biological researches. In fact, in order to be biologically active, a protein must adopt specific folded three-dimensional structures. The main observable to look for in order to extract some information about protein folding is the free energy of the protein itself. We can distinguish between three main conformational states of a protein.

1.1 Unfolded state

The first state we're going to discuss is the *unfolded* state. The ideal unfolded protein is the random coil, where all conformations have comparable free energies except when atoms of the polypeptide chain come into proximity. The number of possible conformations for a protein grows exponentially with its length: it would therefore be impossible for a fully unfolded protein to encounter on a finite timescale all its possible conformations. Consequently, the native conformation is unlikely to be found by a totally random process.

1.2 Intermediate state

It has been experimentally observed that many proteins exist, under certain conditions, in stable conformations that are not completely folded nor completely unfolded. We call these states *intermediate* states. This may be caused by the free-energy landscape of the protein itself, that contains many local minima, as we can see from Fig. 1.

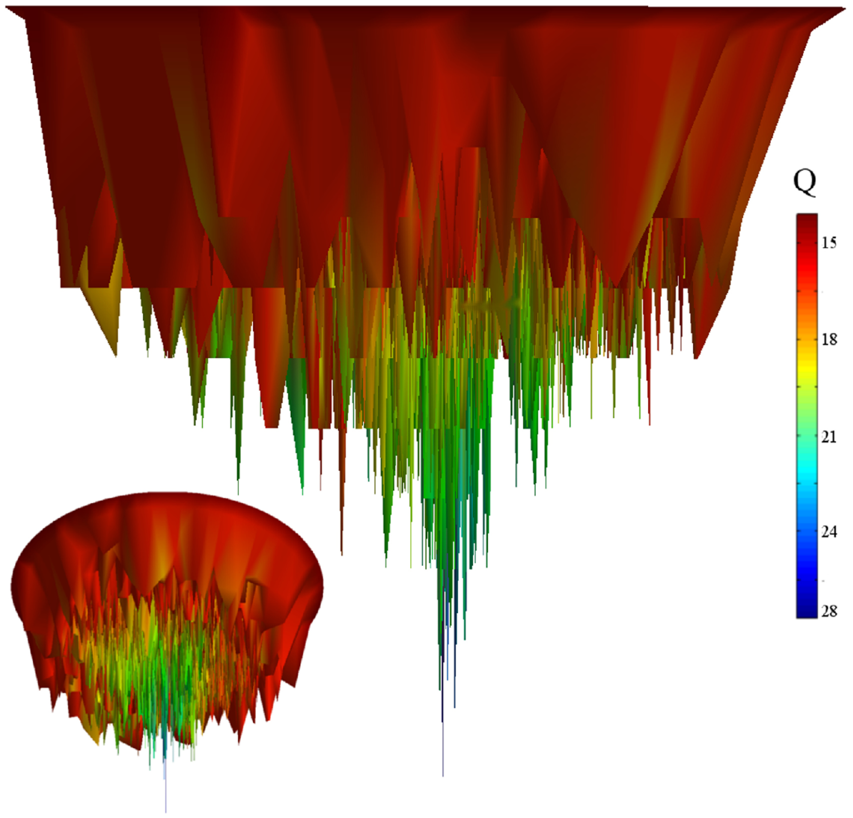


Figure 1: *Example of a protein's energy landscape taken from [5]. The third axis (depth) of the funnel is associated with the energy of the local minima, and the color map represents the reaction coordinate Q. It's possible to see a lot of minima that are not the global one.*

1.3 Native state

We call *native* state the conformational state in which the protein is fully folded, i.e. in its free-energy minimum. The native conformations of proteins are known in great detail from the structures determined by X-ray crystallography. How much alteration is necessary before a protein no longer folds to its normal conformation, either remaining unfolded or in an intermediate state, is not certain. Proteins have been found to be surprisingly adaptive to mutations that would be expected to be disruptive, but the hydrophobic core seems to be the most critical aspect for stability of the normal folded state. This criticality is the basis of HP model that we'll discuss later.

1.4 Stability of the folded state

Calling [U] the concentration of unfolded proteins and [N] the concentration of proteins in their native state, from the detailed balance we can say that

$$[\mathrm{U}]\, k_{\mathrm{N}} = [\mathrm{N}]\, k_{\mathrm{U}}$$

Then, it must exist an equilibrium constant $k_{\text{eq}} = \frac{[\text{N}]}{[\text{U}]} = \frac{k_{\text{N}}}{k_{\text{U}}}$ such that the free energy at equilibrium is

$$\Delta G = G_{\text{N}} - G_{\text{U}} = k_B T \ln k_{\text{eq}} \tag{1}$$

The large heat capacity change upon protein unfolding causes there to be a temperature at which stability of the folded state is at a maximum: the stability of the folded state decreases at both higher and lower temperatures.

Despite this definition, it is clear that the folding problem has an aspect more similar to

$$\text{U} \leftarrow fast \rightarrow \text{I} \leftarrow slow \rightarrow \text{N}$$

where we've also to consider the intermediate state. Assuming that we start from the unfolded state, there will be only one fast process from that state to an intermediate or the native one. In this case, we can neglect the fast process and consider only the slower one and the detailed balance becomes

$$[\text{I}]\, k_{\text{N}} = [\text{N}]\, k_{\text{I}}$$

with a free energy analog to the Eq. (1).

2 The Hydrophobic-Polar model

The Hydrophobic-Polar model is a very simple model used to analyze the protein folding phenomenon [2]. The goal is, like in the complex systems field, to simplify as much as possible the problem maintaining consistency with experimental data. In this model, the 20 types of amino acids present in nature are divided into 2 classes: the hydrophobic ones, labelled by *H*, and the hydrophilic (or polar) ones, labelled by *P*. So, taking a sequence of amino acids of length k, we need to convert it in an *HP* sequence, i.e.

$$s = (s_1, \dots, s_k) \ , \ s_i \in \{H, P\} \quad \forall i \in [1, \dots, k]$$

Once obtained the sequence we need to define in which space $\mathbb{G}$ we want to fold it. To simplify the problem, we may think the protein folds on a square cubic lattice $\mathbb{G} = \mathbb{Z}^2$ or a three-dimensional cubic lattice $\mathbb{G} = \mathbb{Z}^3$. We can now define a *fold* as an injective mapping $f : [1, \dots, k] \to \mathbb{G}$ and denote with $\mathcal{Z}_k$ the set of all folds with length k. It is clear that at each point $f(i)$ is assigned an amino acid s_i. In order to define a way to express the energy of a fold we need to choose a function that differentiates between adjacent and connected amino acids

$$\Delta\left(f(i), f(j)\right) = \begin{cases} 1 \ , \ \|f(i) - f(j)\|_{\mathbb{G}} = 1 \land i \pm 1 \neq j \\ 0 \ , \ \text{otherwise} \end{cases}$$

Now, we've seen experimentally that folded proteins tends to form hydrophobic nuclei, so we set to -1 the energy value of a single *H*-*H* bond, and more in general

$$\epsilon\left(s_i, s_j\right) = \begin{cases} -1 \ , \ s_i = s_j = H \\ 0 \ , \ \text{otherwise} \end{cases}$$

So the total energy of the fold f is

$$\mathcal{E}(s, f) = \sum_{0 \leq i \leq j \leq k} \epsilon\left(s_i, s_j\right) \Delta\left(f(i), f(j)\right)$$

The goal of the *HP* model is then, like most physical problems, to minimize the energy in order to find the global minima, as we have seen in Fig. 1

$$\mathcal{E}_{\text{stable}} = \min_{f \in \mathcal{Z}_k} \mathcal{E}(s, f)$$

3 PERM approach

It has been shown that the *HP* model is a *NP hard* problem, so it's very difficult to fold efficiently longer protein sequences. We will now discuss the Rosenbluth sampling method [2], which involves drawing successive steps of a random walk only from among acceptable points, which are points previously not visited.

3.1 Self-Repelling Chains

In a random walk process on $\mathbb{G} = \mathbb{Z}^3$, each iteration consist in a step among one out of six equally likely directions. An analogue result is valid for $\mathbb{G} = \mathbb{Z}^2$ where we can choose one out of four equally likely directions. Our goal is to simulate the folding of a protein of length k in the space $\mathbb{G}$ by doing a random walk of k steps. The first thing we can notice is that this cannot be a regular random walk process because the fold is not allowed to cross itself or back up on itself at any iteration. The direct consequence is that each iteration of the random walk will have some constraints, e.g. all steps after the first one will have at least one forbidden direction (because they cannot back up). At this point, we can define as *Self-Avoiding Walk* (SAW) a lattice path that does not visit the same point more than once.

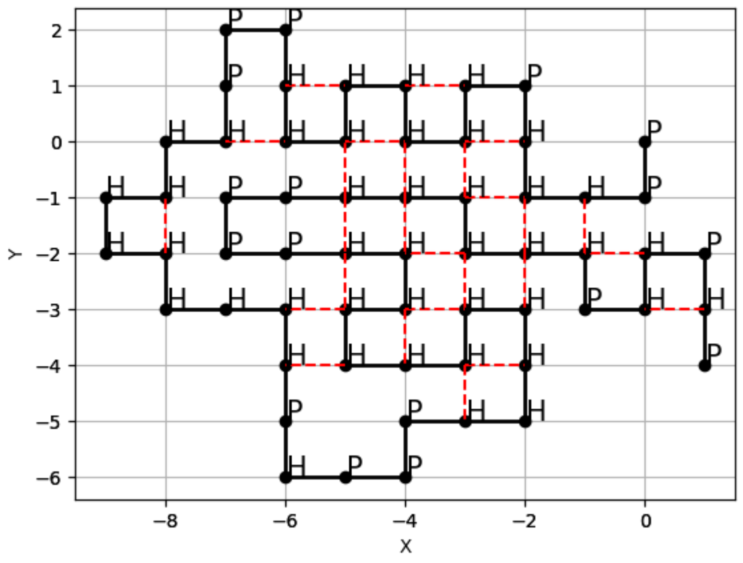


Figure 2: *Example of a self repelling chain starting from the origin of a square cubic lattice. Each node represents an amino acid. The red dashed lines represents the H-H contacts.*

It is known that SAWs are fractals and their number on a given lattice will increase exponentially with the length [6]. However, there's a quite important difference between a "true" SAW and the problem we're interested in. It has been proven [7] that a SAW differentiates from a Self-Repelling Chain (polymer problem, see Fig. 2) by a statistical point of view. In fact, in an SRC, every possible configuration of the chain is equiprobable while this statement is not true for a SAW. The result is that, since the statistics of the SRC is equivalent to the $n \to 0$ limit of an $O(n)$ symmetric ϕ^4 field theory, the upper critical dimensionality for it turns out to be 4 vs 2 for a SAW. We won't go deeper in this aspect because it is beyond the scope of this project. The solution of an SRC is based on a *constraint programming* approach, in which we try to solve a Constraint Satisfaction Problem like that.

3.2 Estimate the number of folds

For simplicity, let's assume that our SRC starts form the origin of our lattice. At any iteration m we will have only w_m possible moves, where w represents a sort of weight of our chain. Let's call W the weight of the whole fold: a fold of length k will have a weight

$$W_k = \prod_{m=1}^{k} w_m$$

It is not excluded that at a certain step $m < k$ we may have no further possibilities for continuation: we then say that a *non-extendable* fold of length m has been formed. Let's denote with $\mathcal{K}$ the set of all folds with length $m \leq k$. Clearly $\mathcal{Z}_k \subset \mathcal{K}$. We notice that the probability of picking a random fold $f_i \in \mathcal{K}$ of length k is

$$\mathbb{P}\left(f_i \in \mathcal{Z}_k\right) = \prod_{j=1}^{k} \frac{1}{w_j} = \frac{1}{W_k}$$

Let's now denote with n_y the number of folds f_i for which $W\left(f_i\right) = y$ and the set of these elements $\mathcal{W}_y = \{f \in \mathcal{K} \mid W\left(f_i\right) = y\}$. Then, for the law of large numbers

$$\frac{n_s}{n} \approx \mathbb{P}\left(\mathcal{W}_y\right)$$

In the same way we can notice that

$$\langle W \rangle = \frac{1}{n} \sum_{i=1}^{n} W\left(f_i\right) \approx \sum_{f} \mathbb{P}\left(f\right) W\left(f\right) = \sum_{f \in \mathcal{Z}_k} 1 = |\mathcal{Z}_k|$$

So we've found an estimator for the number of folds

$$\hat{\mathcal{Z}}_k = \langle W \rangle \approx |\mathcal{Z}_k| \tag{2}$$

Notice that this estimator does not give the number of *unique* folds. Using the *batch mean method* we are also able to estimate the variance of our estimator. Subdividing the sequence of weights $W\left(f_1\right), \ldots, W\left(f_n\right)$ into j blocks of length l each, so $n = jl$. Defining the mean of the b-th block as

$$\mu_b = \frac{1}{l} \sum_{i=(b-1)l+1}^{bl} W\left(f_i\right)$$

we can define our estimator for the variance as

$$\hat{\sigma} = \sqrt{\frac{1}{j} \sum_{b=1}^{j} \left(\mu_b - \langle W \rangle\right)^2} \tag{3}$$

4 Reinforcement learning

Between the plethora of algorithms used to approximate the solutions of the protein folding problem, one of the most promising is Reinforcement Learning.

4.1 Basics of Reinforcement Learning

RL is a kind of algorithm that is particularly effective when the program is able to interact with the analyzed system. The main structure of a RL algorithm is that of an *agent* which interacts with an *environment* and on the base of its actions gets as feedback a *reward* and the new state of the environment. The goal of the model is to find the best *policy* which is a function that returns the action that maximizes the reward at each state.

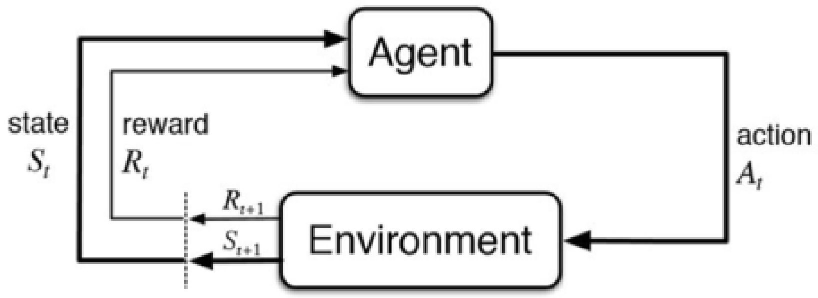


Figure 3: *Basic architecture of a Reinforcement Learning algorithm.*

4.1.1 Markov Decision Processes

MDPs are at the core of RL algorithms, since they describe a discrete-time stochastic control process. The name comes from the fact that they are an extension of Markov's chains, which are systems in which the evolution of a state depends only on the current state and not on its past. For more on the topic see [8].

It is then necessary to describe in Markov terms Fig. 3. S is the set of all possible states and A is the set of all possible actions. The distribution of all possible rewards R, can be obtained by every couple of states and actions $(s,\ a)$ and for each possible adjacent couple of states there is a transition probability P associated.

In the following paragraph we will use a common notation in RL which is that of using the capital letters to denote sets and the lowercase ones to denote single elements of the set

4.1.2 Choosing optimal policies

For each discrete instant t the system moves from a state s_0 to a state s_1 choosing an action a. After each action the agent receives a reward r. In order to choose the optimal policy the algorithm must maximize the cumulative reward starting from a state s_0, defined as:

$$\pi^{opt} = \max \left\{ E \left(\sum_{t>0} r^{t*r|\pi} \right) \right\} \tag{4}$$

It is worth mentioning that the expectation value has an additional constant called discount factor (γ). The expectation value for a single trajectory takes the form: $E(r) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$. From this expression it is clear that the role of the discount factor is that of giving more weight to closer rewards. This is justified due to intrinsic uncertainty that future rewards hold as a part of an MDP.

To choose in which way the algorithm should try to reach the optimal policy, the designer can choose between two different approaches:

- learning through *State Value* function: this function returns the sum of all rewards received starting from an input state following a fixed policy. Using this function one can choose the best route through states;
- learning through *Action Value* function: this function returns the expected reward of taking a given action at a given state. If an algorithm uses the action value function it's called a *Q-learning algorithm.*

4.1.3 Q-Learning

Q-learning is an extremely useful tool to solve the problem of non-deterministic MDPs. The algorithm exploits the Q-function to associate to each state action pair a scalar value (*Q-values*). Q-values are defined as the sum of all discounted rewards assuming the Agent is in state s and performs action a, and then continues playing until the end of the episode following some policy π. An optimal Q-value is a Q-value that follows the optimal policy. All Q-values are stored in the so called *Q-table.*

4.1.4 Bellman Equation and Optimal Q-values

The Bellman equation for Q-learning gives the conditions for equilibrium at correct Q-values, setting therefore a constraint to optimize the policy.

$$Q(s_0, a) = r(s_0, a) + \gamma \max_{a'} \{Q(s_1, a')\} \tag{5}$$

$Q(s_0, a)$ is the Q-value associated with the couple (s_0, a), $r(s_0, a)$ is the reward associated with the same couple and the term

$$\gamma \max_{a'} \{Q(s_1, a')\}$$

is the discounted maximum Q-value for the state-action pair at the successive instant.

4.1.5 General Q-learning algorithm

Q-learning algorithms follow more or less the same general structure

- For each pair (s, a) initialize $Q(s, a)$ to 0.

- Repeat for each episode (complete iteration that stops at the terminal state)
- Select the initial state s.
- Choose a from s using policy derived from Q
- Repeat (for each step of the episode)
- Take action a, observe r, s_0
- Update table entry:

$$Q(s,a) \leftarrow r(s,a) + \gamma \max_{a_0} \{Q(s_0, a_0)\}$$

- $s \to s_1$ until new s is terminal
- Until the maximum number of episodes reached or the Q-values do not change

An additional parameter might be given to fix how fast the Q-values should change for each iteration. This parameter is called the *Learning Rate* $\alpha \in [0,1]$. In this case the Bellman equation becomes:

$$Q(s_0,a) = (1-\alpha)Q(s_0,a) + \alpha\left(r(s_0,a) + \gamma \max_{a'} \{Q(s_1,a')\}\right) \tag{6}$$

4.2 Reinforcement Learning applied to the HP model

The model described is able to solve bidimensional folds for a protein approximated with the HP model. Its general structure can be generalized to tridimensional folds with very few adjustments, making it a useful tool suited for this specific problem. This model was first proposed by [9].

Consider a protein s in the k-dimentional HP model

$$s = (s_1, \ldots, s_k) \ , \ s_i \in \{H, P\} \quad \forall i \in [1, \ldots, k]$$

in order to use RL to solve the problem we need to specify the various components of the RL algorithm: state space, action space, transition function and reward function.

The state space S (the agent's environment) is made of $\frac{4^k-1}{3}$ elements. A state is terminal (the process stops) if the number of state visited is $k-1$, so we expect it to visit each possible state.

The action space A is four dimensional and sanctions that from each state the system can transition between four other states (Up, Down, Right, Left or a_0, a_1, a_2, a_3).

The transition function δ returns the stochastic outcomes of taking each action in a specific state. So for each state is given the accessible sequences. δ is defined as:

$$\delta(s_i, a_l) = s_{4i-3+l} \quad l \in [1,4] \quad \forall i \in \left[1, \frac{4^k-1}{3}\right] \tag{7}$$

Since there is no reason for initial bias between states the neighbors are all equiprobable with probability of transition $\frac{1}{4}$.

Before defining the reward function it is appropriate to spend a few words to visualize how the RL algorithm described so far models the protein and its folds. Consider a complete episode and its associated the path $\varsigma = (\varsigma_0, \varsigma_1, \ldots, \varsigma_{k-1})$ ς_i. To this path there will be an action sequence associated $a_\varsigma = (a_{\varsigma_0}, a_{\varsigma_1}, \ldots, a_{\varsigma_{k-2}})$. This sequence of actions is linked to the path since each element of both sequences appears in Eq. (7). a_ς can be viewed as a possible bidimensional structure of the protein sequence. Therefore, we can now associate to a certain action sequence, and by extension to a path ς, the energy defined in section 2.

We can now define the reward function $r(\varsigma_n | s_1, \varsigma_1, \ldots, \varsigma_{l-1})$:

- if a_ς is not valid $\rightarrow$ 0.01
- if the system reached a terminal state $(l = k - 1)$ $\rightarrow$ -E
- otherwise ("most common case") $\rightarrow$ 0.1

By defining the reward function in this way, it returns the reward received by the agent in the state ς_l after it had visited all the previous states. With the reward function it is now clear that the goal of maximizing the rewards on a path from the starting state to the terminal one is equivalent of that of finding self-avoiding random walks that minimize the associated energy.

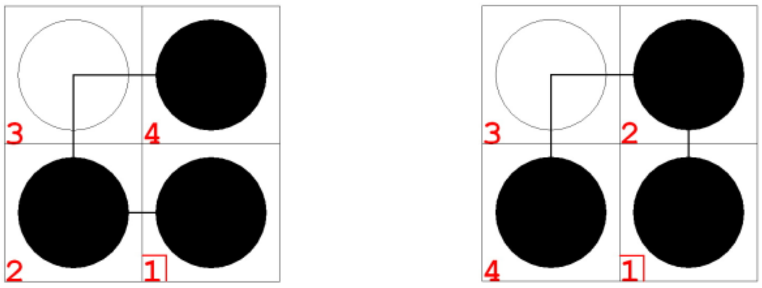


Figure 4: *Visualization of the link between actions and configuration. In the figure on the right the actions are: Up, Left, Down. In the figure on the right the actions are: Left, Up, Right [9].*

4.3 Learning Phase

For learning, it is convenient to use Q-learning. It can be proven that during the training process the values of the Q-table converge to the true values (for the proof see [9]), so it is guaranteed that at the end of the training process the values found will be close to the true values. After the training the agent acts with a greedy mechanism until it reaches a solution. The adjective *greedy* in this context means that at each step the system transitions to a neighbor state with maximum Q-value.

It is now important to notice that this process might produce more than one optimal policies. In this case, as far as the bidimensional folding goes, the predicted structures will have the same free energy.

4.4 Beyond Reinforcement Learning

From what seen so far in this section Q-learning looks like a good way to approach protein folding in the HP model approximation. Despite this, simple Q-learning algorithms such as the one described in section 4.1.5 are inadequate compared to the state-of-the-art model.

4.4.1 Deep Q-Learning

The subject of *Deep Q-learning* is far too complicated and wide to be treated to its full extend in this article, therefore this section contains only its intuitive idea and the main reasons why it outperforms simple Q-learning.

As we have seen in section 4.1.3 a Q-learning algorithm learns through the process of optimizing the Q-values, which are the best action for a given state given all the previous actions. Deep Q-learning exploits the Neural Networks to estimate cluster of Q-values. In this case there no longer is a Q-table, but a Deep Neural Network estimating many Q-values at the same time, saving resources.

Since Deep Neural Networks are currently the best statistical estimators available, this kind of algorithm performs extremely well. The price to pay for such improvements is that Deep Neural Networks operate as *black boxes*, meaning that it is often impossible to understand the optimization process for complex problems such as this one.

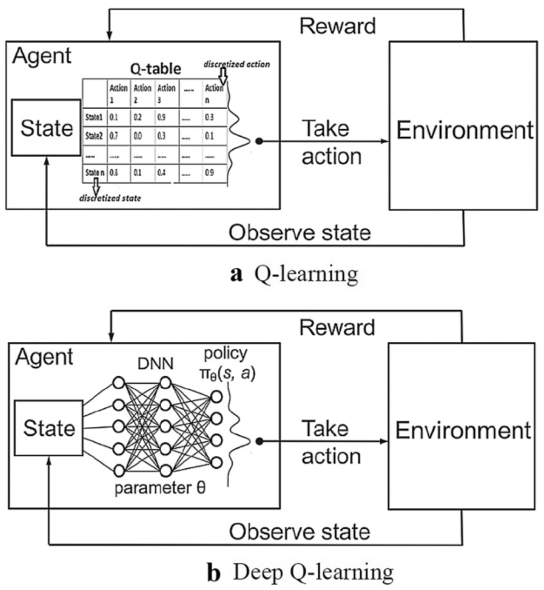


Figure 5: *Difference between Q-learning and Deep Q-learning visualized [10].*

5 Results

Our test were performed mainly on two machines: one with an Intel® Core i7-7700 @ 3.6 GHz with 16 GB of RAM and one on an AMD® Ryzen 7 @ 3.6 GHz with 32 GB of RAM.

5.1 Monte Carlo PERM

We want now to report an application of the PERM algorithm, made following the article's recipe [2]. The goal of this algorithm is to estimate the total number of fold and then simulate a finite number of SRCs hoping to find the energy minimum. In the article the number of iterations is set to be 10^5 for all proteins and the result is that they found only local minima, which are intermediate states. Our goal is to improve these 2D simulations by increasing the number of iterations, hoping to find the real minimum reported in a famous benchmark [11].

The first protein for which we've improved the result is $HH(P)_5HH(P)_3H(P)_3HP$, that has a length of 18 amino acids (see Fig. 6). In particular, we've found $(1.25 \pm 0.35) \times 10^8$ folds with an energy minimum of -4 (a.u.), equal to the benchmark's one.

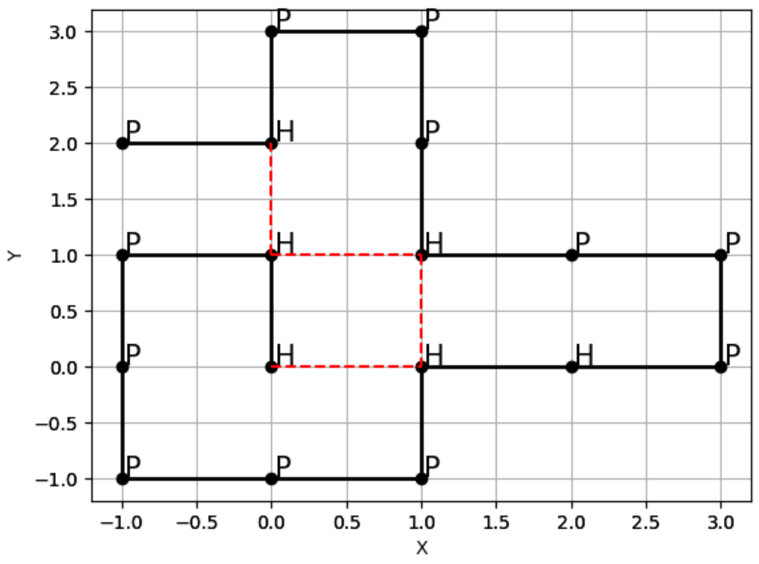


Figure 6: *The protein $HH(P)_5HH(P)_3H(P)_3HP$ has reached its energy minimum at -4 (a.u.) after 5×10^6 iterations (~ 20 minutes).*

Another protein for which we've improved the result is $HHHP(PH)_3PP(HP)_3PH$, that has a length of 20 amino acids (see Fig. 7). In this case, we've found $(8.9688 \pm 0.0033) \times 10^{10}$ folds with an energy minimum of -9 (a.u.), one unit more than the benchmark's one.

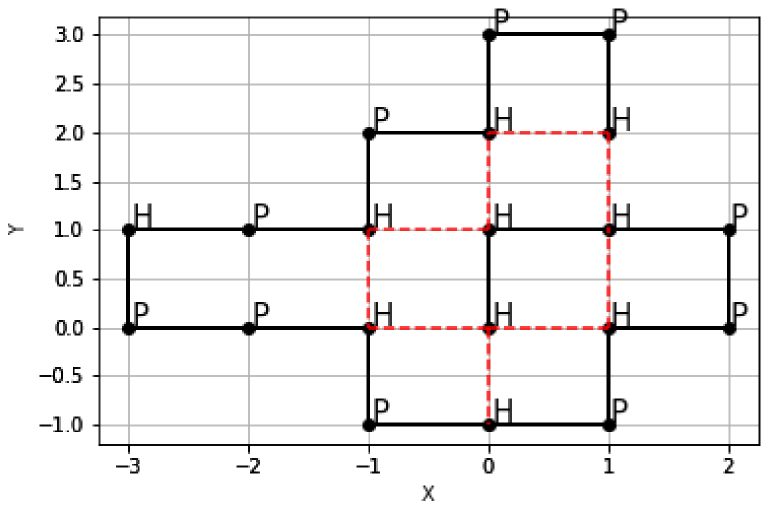


Figure 7: *The protein* $HHHP(PH)_3PP(HP)_3PH$ *has reached an energy minimum of -9 (a.u.) after* 10^7 *iterations (*~ 45 *minutes).*

We've also tried to improve the protein $HPHPHHH(P)_3HHHH(P)_2HH$, that has a length of 20 amino acids (see Fig. 8). However, after about 5 hours of running time it has folded with an energy of -7 (a.u.), like in the article, with an estimated number of folds equal to $(1.24 \pm 0.43) \times 10^8$.

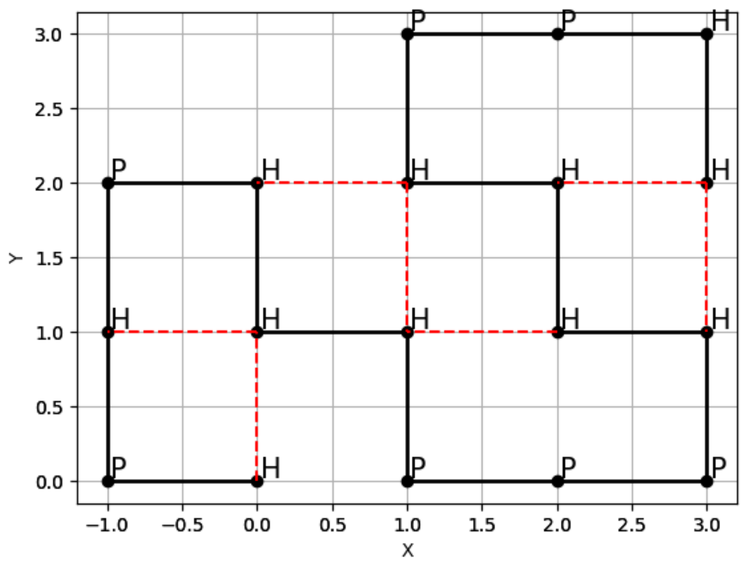
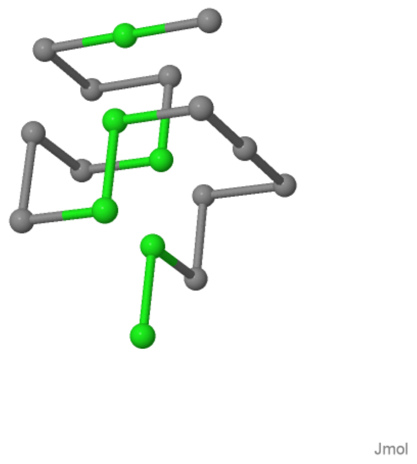


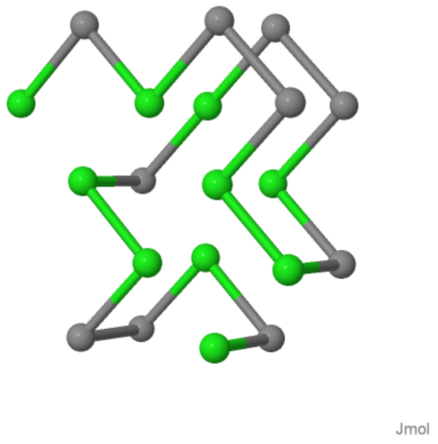
Figure 8: *The protein* $HPHPHHH(P)_3HHHH(P)_2HH$ *has reached an energy minimum of -7 (a.u.) after* 10^8 *iterations (*~ 5 *hours).*

5.2 Constraint-based Protein Structure Prediction PERM

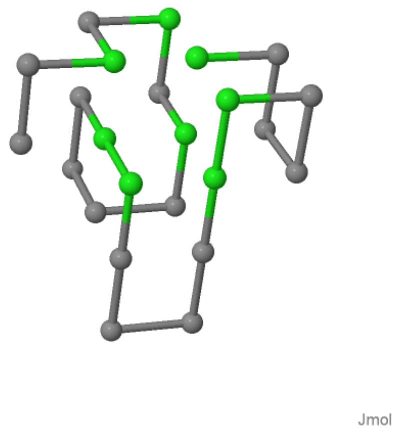
In the 2D case we showed that increasing the number of iterations resulted in better estimate of the energy. However, the previous algorithm is not able to give significant outputs in the 3D case. We tried so to estimate the energy in the 3D case with another algorithm, which is also based on constraint programming. This is implemented by *CPSP-tools* [12] and has revealed to be very powerful, folding proteins of length ~ 200 in few seconds. Furthermore, this program is able to fold proteins in more complex lattices, like the 3D-face-centered-cubic one. In our case, we wanted to improve the results of article [2] so, in order to keep consistency, we opted for a 3D-cubic lattice. All proteins tested converged to their minima, as we can see in Fig. 9.



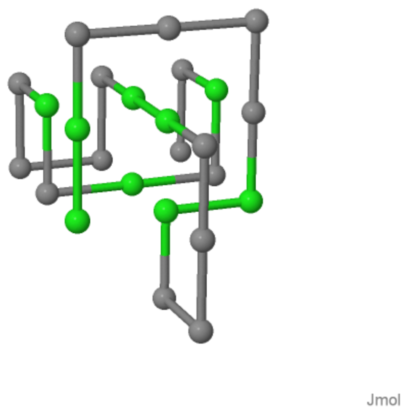
(a) *Energy* -4 *(a.u.) for the sequence* $HH(P)_5HH(P)_3H(P)_3HP$.



(b) *Energy* -11 *(a.u.) for the sequence* $(HP)_2PH(HP)_2(PH)_2HP(PH)_2$.



(c) *Energy* -8 *(a.u.) for the sequence* $P(PH)_3(P)_4(HH(PP)_2)_2H$.



(d) *Energy* -8 *(a.u.) for the sequence* $P(PH)_3(P)_4(HH(PP)_2)_2HH$.

Figure 9: *Four proteins of length 18 (a), 20 (b), 24 (c) and 25 (d).*

5.3 Reinforcement Learning

Due to the fact that Reinforcement Learning code is extremely hard to write and optimize performance-wise, this section will only report in Tab. 1 the main results obtained with DRL algorithms by [10]. These results are enough to show that DRL outperforms PERM and more in general brute force methods.

Performance		
Sequence	PERM	DRL
$(HP)_2PH_2PHP_2HPH_2P_2HPH$	< 1s	< 1s
$P_2HP_2H_2P_4H_2P_4H_2P_4H_2$	6s	< 1s
$P_3H_2P_2H_2P_5H_7P_2H_2P_4H_2P_2HP_2$	< 1s	< 1s
$P_2HP_2H_2P_2H_2P_5H_{10}P_6H_2P_2H_2P_2HP_2H_5$	3m	25s
$H_2(PH)_3PH_4PHP_3HP_3HP_4$ $HP_3HP_3HPH_4(PH)_3PH_2$	3s	< 1s
$P_2H_3PH_8P_3H_{10}PHP_3H_{12}P_4H_6PH_2PHP$	7s	< 1s
$H_{12}(PH)_2(P_2H_2)_2P_2H(P_2H_2)_2P_2HPHPH_{12}$	78h	48m
$H_4P_4H_{12}P_6(H_{12}P_3)_3HP_2(H_2P_2)_2HPH$	64s	1s
$P_3H_2P_2H_4P_2H_3(PH_2)_3H_2P_8H_6P_2H_6P_9$ $HPH_2PH_{11}P_2H_3PH_2PHP_2HPH_3P_6H_3$	1h	4m
$P_6HPH_2P_5H_3PH_5PH_2(P_2H_2)_2PH_5PH_{10}$ $PH_2PH_7P_{11}H_7P_2HPH_3P_6HPHP_2$	9m	1m

Table 1: *Performance of Perm and Deep Reinforced Learning on the same protein sequence, the energy was omitted because for each sequence both model found the same energy of the folded state.*

6 Discussion

As we saw, protein folding is actually a hard task. The application of PERM algorithm gave better results by incrementing the number of iteration, but required a lot of computational time, e.g. 5 hours for 10^8 iterations of Fig. 8. Furthermore, the application of the *CPSP-tools* library managed to compute the energies in the 3D case, where the other algorithm failed. However, this approach is not sustainable because the result's uncertainty will grow up as the protein length increases and even well-optimized brute force algorithm will suffer the curse of dimensionality.

In sec **??** we explained how simple Reinforcement Learning constitutes an *smart* (AI-wise) version of the PERM algorithm, but it still falls short from a computational point of view. By comparing performances it emerged that the best models for predicting protein structure in the HP approximation are Deep Q-Networks, which are a particular kind of Reinforcement Learning algorithms that exploit neural networks. It is important to underline that despite this kind of models have already outperformed classical algorithms their research field is still thriving and there still is room for improvement

Therefore, a solution to the HP model should be searched into the Artificial Intelligence field. Solutions based on deep reinforcement learning, and in general in the A.I. field, are more sustainable and give better results. Bioinformatics is conceptualizing biology in terms of molecules and applying "informatics techniques" to understand and organize the information associated with these molecules, on a large scale [13]. The aims of bioinformatics are multiple. Firstly, well-organized data allow researches to access existing information and to submit new entries as they are produced. Secondly, it develops tool and resources useful for the analysis of these data. The development of these tools requires a lot of knowledge in biology, informatic and also physics and mathematics.

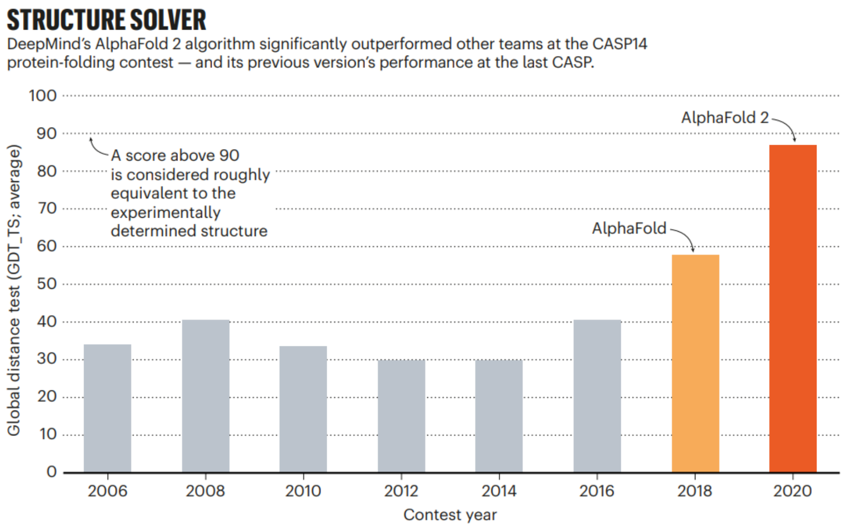


Figure 10: *Protein folding algorithms' accuracy through time [14].*

Two example of intelligences that are giving many good results are AlphaFold and AlphaFold2 by Google. In particular, the second one reached a precision level comparable to the experimental one, as wee can see from Fig 10. The future of this topic seems to be linked to research whose goal is to expand the protein databases underlying these intelligences.

References

[1] T. Lezon, "Statistical mechanics of chain molecules: An overview,"

[2] M. Wierzbiński and A. Crimi, "The rosenbluth sampling calculation of hydrophobic-polar model," *bioRxiv*, 2022. DOI: 10.1101/2022.05.02.490260. eprint: https://www.biorxiv.org/content/early/2022/05/02/2022.05.02.490260.full.pdf. [Online]. Available: https://www.biorxiv.org/content/early/2022/05/02/2022.05.02.490260.

[3] D. Sadava, S. Hacker, H. C. Heller, and D. M. Hillis, *La nuova biologia blu: Genetica, DNA, evoluzione, biotech.* Zanichelli, 2020.

[4] T. E. Creighton, "Protein folding," *Biochemical Journal*, vol. 270, no. 1, pp. 1–16, Aug. 1990, ISSN: 0264-6021. DOI: 10.1042/bj2700001. eprint: https://portlandpress.com/biochemj/article-pdf/270/1/1/602339/bj2700001.pdf. [Online]. Available: https://doi.org/10.1042/bj2700001.

[5] A. B. Oliveira Jr, F. M. Fatore, F. V. Paulovich, O. N. Oliveira Jr, and V. B. Leite, "Visualization of protein folding funnels in lattice models," *PloS one*, vol. 9, no. 7, e100861, 2014.

[6] N. Madras and A. D. Sokal, "The pivot algorithm: A highly efficient monte carlo method for the self-avoiding walk," *Journal of Statistical Physics*, vol. 50, pp. 109–186, 1988.

[7] D. J. Amit, G. Parisi, and L. Peliti, "Asymptotic behavior of the" true" self-avoiding walk," *Physical Review B*, vol. 27, no. 3, p. 1635, 1983.

[8] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[9] G. Czibula, M. Bocicor, and I.-G. Czibula, "A reinforcement learning model for solving the folding problem," *International Journal of Computer Technology and Applications*, vol. 2, no. 01, 2011.

[10] R. Jafari and M. M. Javidi, "Solving the protein folding problem in hydrophobic-polar model using deep reinforcement learning," *SN Applied Sciences*, vol. 2, pp. 1–13, 2020.

[11] B. University. "The ProFOLDING Project." (2011), [Online]. Available: https://www.brown.edu/Research/Istrail_Lab/hp2dbenchmarks.html (visited on 02/07/2023).

[12] M. Mann, S. Will, and R. Backofen, "Cpsp-tools–exact and complete algorithms for high-throughput 3d lattice protein studies," *BMC bioinformatics*, vol. 9, pp. 1–8, 2008.

[13] N. M. Luscombe, D. Greenbaum, and M. Gerstein, "What is bioinformatics? an introduction and overview," *Yearbook of medical informatics*, vol. 10, no. 01, pp. 83–100, 2001.

[14] E. Callaway, "'it will change everything': Deepmind's ai makes gigantic leap in solving protein structures," *Nature*, vol. 588, no. 7837, pp. 203–205, 2020.