ISSN : 2229-6093

A Reinforcement Learning Model for Solving the Folding Problem

Gabriela Czibula, Maria-Iuliana Bocicor and Istvan-Gergely Czibula
Babeş-Bolyai University
Department of Computer Science
1, M. Kogalniceanu Street, 400084, Cluj-Napoca, Romania
{gabis, iuliana, istvanc}@cs.ubbcluj.ro

Abstract

In this paper we aim at proposing a reinforcement learning based model for solving combinatorial optimization *problems. Combinatorial optimization problems are hard to solve optimally, that is why any attempt to improve their solutions is beneficent. We are particularly focusing on the bidimensional* protein folding *problem, a well known NP-hard optimizaton problem important within many fields including bioinformatics, biochemistry, molecular biology and medicine. A reinforcement learning model is introduced for solving the problem of predicting the bidimensional structure of proteins in the hydrophobic-polar model. The model proposed in this paper can be easily extended to solve other optimization problems. We also give a mathematical validation of the proposed reinforcement learning based model, indicating this way the potential of our proposal.*

Keywords: *Bioinformatics, Reinforcement Learning, Protein Folding.*

1 Introduction

Combinatorial optimization is the seeking for one or more optimal solutions in a well defined discrete problem space. In real life approaches, this means that people are interested in finding efficient allocations of limited resources for achieving desired goals, when all the variables have integer values. As workers, planes or boats are indivisible (like many other resources), the Combinatorial Optimization Problems (COPs) receive today an intense attention from the scientific community.

The current real-life COPs are difficult in many ways: the solution space is huge, the parameters are linked, the decomposability is not obvious, the restrictions are hard to test, the local optimal solutions are many and hard to locate, and the uncertainty and the dynamicity of the environment must be taken into account. All these characteristics, and others more, constantly make the algorithm design and implementation challenging tasks. The quest for more and more efficient solving methods is permanently driven by the growing complexity of our world.

Reinforcement Learning (RL) [17] is an approach to machine intelligence in which an agent can learn to behave in a certain way by receiving punishments or rewards on its chosen actions.

In this paper we aim at proposing a reinforcement learning based model for solving *combinatorial optimization* problems. We are particularly focusing on a well known problem within bioinformatics, the *protein folding* problem, which is an *NP*-complete problem that refers to predicting the structure of a protein from its amino acid sequence. Protein structure prediction is one of the most important goals pursued by bioinformatics and theoretical chemistry; it is highly important in medicine (for example, in drug design) and biotechnology (for example, in the design of novel enzymes).

We are addressing in this paper the bidimensional protein structure prediction, but our model can be easily extended to the problem of predicting the three-dimensional structure of proteins. Moreover, the proposed model can be generalized to address other optimization problems. We also give a mathematical validation of the proposed reinforcement learning based model, indicating this way the potential of our proposal. To our knowledge, except for the ant [8] based approaches, the bidimensional *protein folding* problem has not been addressed in the literature using reinforcement learning, so far.

The rest of the paper is organized as follows. Section 2 presents the main aspects related to the topics approached in this paper, the *protein folding* problem and *reinforcement learning*. The reinforcement learning model that we propose for solving the bidimensional protein folding problem is introduced in Section 3. Section 4 presents the mathematical validation of our approach, containing the convergence proof for the proposed Q-learning algorithm. Computational experiments are given in Section 5 and in Section 6 we provide an analysis of the proposed reinforcement model, emphasizing its advantages and drawbacks. Section 7 contains some conclusions of the paper and future development of our work.

2 Background

In this section we briefly review the fundamentals of the *protein folding* problem and *reinforcement learning*.

2.1 The Protein Folding Problem

2.1.1 Problem relevance

Proteins are one of the most important classes of biological macromolecules, being the carriers of the message contained in the DNA. They are composed of amino acids, which are arranged in a linear form and fold to form a three-dimensional structure. Proteins have very important functions in the organism, like structural functions in the muscles and bones, catalytic functions for all biochemical reactions that form the metabolism and they coordinate motion and signal transduction.

Therefore, proteins may be considered the basic units of life and a good understanding of their structure and functions would lead to a better understanding of the processes that occur in a living organism. As soon as it is synthesized as a linear sequence of amino acids, a protein folds, in a matter of seconds, to a stable three-dimensional structure, which is called the protein's native state. It is assumed that the information for the folding process is contained exclusively in the linear sequence of amino acids and that the protein in its native state has a minimum free energy value. Once in its stable three-dimensional state, a protein may perform its functions - three-dimensional interactions with other proteins, interactions that mediate the functions of the organism.

The determination of the three-dimensional structure of a protein, the so called *protein folding* problem, using the linear sequence of amino acids is one of the greatest challenges of bioinformatics, being an important research direction due to its numerous applications in medicine (drug design, disease prediction) and genetic engineering (cell modelling, modification and improvement of the functions of certain proteins).

Moreover, unlike the structure of other biological macromolecules (e.g., DNA), proteins have complex structures that are difficult to predict. That is why different computational intelligence approaches for solving the protein folding problem have been proposed in the literature, so far.

In the folowing we are addressing the *Bidimensional Protein Folding Problem* ($BPFB$), more exactly the problem of predicting the bidimensional structure of proteins, but our model can be easily extended to the three-dimensional protein folding problem.

2.1.2 The Hydrophobic-Polar Model

An important class of abstract models for proteins are lattice-based models - composed of a lattice that describes the possible positions of amino acids in space and an energy function of the protein, that depends on these positions. The goal is to find the global minimum of this energy function, as it is assumed that a protein in its native state has a minimum free energy and the process of folding is the minimization of this energy [1].

One of the most popular lattice-models is Dill's Hydrophobic-Polar (HP) model [6].

In the folding process the most important difference between the amino acids is their hydrophobicity, that is how much they are repelled from water. By this criterion the amino acids can be classified in two categories:

- *hydrophobic* or *non-polar* (H) - the amino acids belonging to this class are repelled by water
- *hydrophilic* or *polar* (P) - the amino acids that belong to this class have an affinity for water and tend to absorb it

The HP model is based on the observation that the hydrophobic forces are very important factors in the protein folding process, guiding the protein to its native three dimensional structure.

The primary structure of a protein is seen as a sequence of n amino acids and each amino acid is classified in one of the two categories: hydrophobic (H) or hydrophilic (P):

$$\mathcal{P} = p_1 p_2 ... p_n, \text{ where } p_i \in \{H, P\}, \ \forall 1 \leq i \leq n$$

A conformation of the protein $\mathcal{P}$ is a function *C*, that maps the protein sequence $\mathcal{P}$ to the points of a two-dimensional cartesian lattice.

If we denote:

$$\mathcal{B} = \{\mathcal{P} = p_1 p_2 ... p_n |\, p_i \in \{H, P\}, \forall 1 \leq i \leq n, n \in \mathcal{N}\}$$

$$\mathcal{G} = \{G = (x_i, y_i) |\, x_i, y_i \in \Re, 1 \leq i \leq n\}$$

then a conformation C is defined as follows:

$$C : \mathcal{B} \rightarrow \mathcal{G}$$

$$\mathcal{P} = p_1 p_2 ... p_n \mapsto \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

(x_i, y_i) - represents the position in the two-dimensional lattice to which the amino acid p_i is mapped by the function $C, \forall 1 \leq i \leq n$

The mapping C is called a *path* if:

$$\forall 1 \le i, j \le n, \text{ with } |i-j| = 1 \Rightarrow |x_i - x_j| + |y_i - y_j| = 1$$

In fact, this definition states that the function C is a path if any two consecutive amino acids in the primary structure of the protein are neighbors (horizontally or vertically) in the bidimensional lattice. It is considered that any position of an amino acid in the lattice may have a maximum number of 4 neighbors: up, down, left, right.

A path C is *self-avoiding* if the function C is an injection:

$$\forall 1 \le i, j \le n, \text{ with } i \ne j \Rightarrow (x_i, y_i) \ne (x_j, y_j)$$

This definition affirms that the mapped positions of two different amino acids must not be superposed in the lattice.

A configuration C is *valid* if it is a *self avoiding path*.

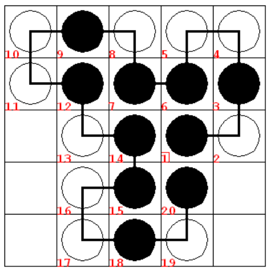

Figure 1. A protein configuration for the sequence $\mathcal{P} = HPHPPHHPHPPHPHHPPHPH$, of length 20. Black circles represent hydrophobic amino acids, while white circles represent hydrophilic ones. The configuration may be represented by the sequence $\pi = RUULDLULLDRDRDLDRRU$. The value of the energy function for this configuration is -9.

Figure 1 shows a configuration example for the protein sequence $\mathcal{P} = HPHPPHHPHPPHPHHPPHPH$, of length 20, where the hydrophobic amino acids are represented in black and the hydrophilic ones are in white.

The energy function in the HP model reflects the fact that hydrophobic amino acids have a propensity to form a hydrophobic core. Consequently the energy function adds a value of -1 for each two hydrophobic amino acids that are mapped by C on neighboring positions in the lattice, but that are not consecutive in the primary structure $\mathcal{P}$. Such two amino acids are called topological neighbors. Any hydrophobic amino acid in a valid conformation C can have at most 2 such neighbors (except for the first and last aminoacids, that can have at most 3 topological neighbors).

If we define the function I as:

$$I : \{1, \dots, n\} \times \{1, \dots, n\} \rightarrow \{-1, 0\}$$

where $\forall 1 \le i, j \le n$, with $|i-j| \ge 2$

$$I(i,j) = \begin{cases} -1 & \text{if } p_i = p_j = H \text{ and } |x_i - x_j| + |y_i - y_j| = 1 \\ 0 & \text{otherwise} \end{cases}$$

then the energy function for a valid conformation C is defined as follows:

$$E(C) = \sum_{1 \le i \le j-2 \le n} I(i,j) \qquad (1)$$

The protein folding problem in the HP model is to find the conformation C whose energy function $E(C)$ is minimum. The energy function for the example configuration presented in Figure 1 is -9: for each pair of hydrophobic amino acids that are neighbors in the lattice, but not in the primary structure of the protein a value of -1 is added. These pairs are: (1,6), (1,14), (1,20), (3,6), (7,12), (7,14), (9,12), (15,18), (15,20).

A solution for the bidimensional HP protein folding problem, corresponding to an n-length sequence $P \in \mathcal{B}$ could be represented by a $n-1$ length sequence $\pi = \pi_1 \pi_2 ... \pi_{n-1}$, $\pi_i \in \{L, R, U, D\}$, $\forall 1 \le i \le n-1$, where each position encodes the direction of the current amino acid relative to the previous one (L-left, R-right, U-up, D-down). As an example, the solution configuration corresponding to the sequence presented in Figure 1 is $\pi = RUULDLULLDRDRDLDRRU$.

2.1.3 Literature review

Berger and Leighton [2] show that the protein folding problem in the HP model is NP-complete, therefore various approximation and heuristics methods approach this problem, including Growth algorithms, Contact Interactions method and general optimization techniques like Monte Carlo methods, Tabu Search, Evolutionary and Genetic algorithms and Ant Optimization algorithms.

Beutler and Dill [3] introduce a chain-growth method - the Core-directed chain Growth method (CG), that uses a heuristic bias function in order to assemble a hydrophobic core. This method begins by counting the total number of hydrophobic amino acids in the sequence and constructing a core, as square as possible, that should contain all H amino acids. Then, the CG method grows a chain conformation by adding one “segment” at a time - a segment is a string of a few amino acids, of a predetermined length.

Shmygelska and Hoos [16] present an Ant Colony Optimization Algorithm that iteratively undergoes three phases: the construction phase - each ant constructs a candidate solution by sequentially growing a conformation of the given primary sequence of the protein, starting from a randomly chosen position in the sequence; the local search phase - the protein conformations are further optimized by the ants; update phase - the ants update the pheromone matrix based on values of the energy function obtained after the first 2 phases.

Another algorithm, based on Ant Colony Optimization was proposed by Talheim, Merkle and Middendorf [18], who develop a hybrid population based ACO algorithm. Instead of keeping and using pheromone information, as in traditional ACO algorithms, the population based ACO - P-ACO transfers a population of solutions from one iteration to another. The hybrid P-ACO algorithm that the authors describe is called PFold-P-ACO and it consists of a P-ACO part and a branch-and-bound part. The latter uses the pheromone information from the P-ACO and it starts when the former has found an improvement over a certain number of iterations.

Unger and Moult [20] were the first ones to apply genetic algorithms for the problem of protein structure prediction. Their technique evolves a population of valid conformations for a given protein sequence, using operations like mutation - in the form of conventional Monte Carlo steps and crossover - selected sequences are cut and rejoined to other sequences, at the same point. To verify the validity of each new conformation, Metropolis-type criteria are used. This method proved to find better solutions for the protein folding problem in the bidimensional HP lattice model than the traditional Monte Carlo methods.

A hybrid algorithm, which combines genetic algorithms and tabu search algorithms is proposed in [22]. The authors introduce the tabu search in the crossover and mutation operations, as they believe this strategy can improve the local search capability. The algorithm adopts a variable population size to maintain the diversity of the population. A new ranking selection strategy is used, which can accept inferior solutions during the search process, thus having stronger hill-climbing capabilities.

Chira has introduced in [5] a new evolutionary model with hill-climbing genetic operators for the HP model of the protein folding problem. The crossover operator is defined specifically for this problem and it ensures an efficient exploration of the search space. The hill climbing mutation is based on the pull move operation, which is applied within a steepest-ascent hill climbing procedure. To ensure the diversity of the genetic material, the algorithm explicitly reinforces diversity after a certain number of iterations.

There are also approaches in the literature in the directin of using supervised machine learning techniques for protein fold prediction. Support Vector Machine and the Neural Network learning methods are used by Ding and Dubchak in [7] as base classifiers for the protein fold recognition problem.

2.2 Reinforcement learning

The goal of building systems that can adapt to their environments and learn from their experiences has attracted researchers from many fields including computer science, mathematics, cognitive sciences [17].

Reinforcement Learning (RL) [11] is an approach to machine intelligence that combines two disciplines to solve successfully problems that neither discipline can address individually:

- *Dynamic programming* - a field of mathematics that has traditionally been used to solve problems of optimization and control.

- *Supervised learning* - a general method for training a parametrized function approximator to represent functions.

Reinforcement learning is a synonym of learning by interaction [14]. During learning, the adaptive system tries some actions (i.e., output values) on its environment, then, it is reinforced by receiving a scalar evaluation (the reward) of its actions. The reinforcement learning algorithms selectively retain the outputs that maximize the received reward over time. Reinforcement learning tasks are generally treated in discrete time steps. In RL, the computer is simply given a goal to achieve. The computer then learns how to achieve that goal by trial-and-error interactions with its environment.

Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical *reward* signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the highest reward by trying them. In a reinforcement learning problem, the agent receives from the environment a feedback, known as *reward* or *reinforcement*; the reward is received at the end, in a terminal state, or in any other state, where the agent has correct information about what he did well or wrong. The agent will learn to select actions that maximize the received *reward*.

The agent's goal, in a RL task is to maximize the sum of the reinforcements received when starting from some initial state and proceeding to a terminal state.

A reinforcement learning problem has three fundamental parts [17]:

- *The environment* - represented by "states". Every RL system learns a mapping from situations to actions by trial-and-error interactions with the environment.

- *The reinforcement function* - the "goal" of the RL system is defined using the concept of a reinforcement function, which is the exact function of future reinforcements the agent seeks to maximize. In other words, there exists a mapping from state/action pairs to reinforcements. After performing an action in a given state, the RL agent will receive some reinforcement (reward) in the form of a scalar value. The RL agent learns to perform actions that will maximize the sum of the reinforcements received when starting from some initial state and proceeding to a terminal state.

- *The value (utility) function* - explains how the agent learns to choose "good" actions, or even how we might measure the utility of an action. Two terms are defined: a policy determines which action should be performed in each state. The value of a state is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state. The value (utility) function would therefore be the mapping from states to actions that maximizes the sum of the reinforcements when starting in an arbitrary state and performing actions until a terminal state is reached.

At each time step, t, the learning system receives some representation of the environment's state s, it takes an action a, and one step later it receives a scalar reward r_t, and finds itself in a new state s'. The two basic concepts behind reinforcement learning are trial and error search and delayed reward [4]. The agent's task is to learn a control policy, $\pi : S \to A$, that maximizes the expected sum E of the received rewards, with future rewards discounted exponentially by their delay, where E is defined as $r_0 + \gamma \cdot r_1 + \gamma^2 \cdot r_2 + \ldots$ ($0 \leq \gamma < 1$ is the discount factor for the future rewards).

One key aspect of reinforcement learning is a trade-off between *exploitation* and *exploration* [19]. To accumulate a lot of reward, the learning system must prefer the best experienced actions, however, it has to try (to experience) new actions in order to discover better action selections for the future. There are two basic RL designs to consider:

- The agent learns a *utility function* (U) on states (or states histories) and uses it to select actions that maximize the expected utility of their outcomes.

- The agent learns an *action-value function* (Q) giving the expected utility of taking a given action in a given state. This is called *Q-learning*.

2.2.1 Q-learning

Q-learning [17] is another extension to traditional dynamic programming (value iteration) and solves the problem of the *non-deterministic* Markov decision processes, in which a probability distribution function defines a set of potential successor states for a given action in a given state.

Rather then finding a mapping from states to state values, Q-learning finds a mapping from state/action pairs to values (called *Q-values*). Instead of having an associated value function, Q-learning makes use of the Q-function. In each state, there is a Q-value associated with each action. The definition of a Q-value is the sum of the (possibly discounted) reinforcements received when performing the associated action and then following the given policy thereafter. An *optimal Q-value* is the sum of the reinforcements received when performing the associated action and then following the optimal policy thereafter.

If $Q(s,a)$ denotes the value of doing the action a in state s, $r(s,a)$ denotes the reward received in state s after performing action a and s' represents the state of the environment reached by the agent after performing action a in state s, the Bellman equation for Q-learning (which represents the constraint equation that must hold at equilibrium when the Q-values are correct) is the following [15]:

$$Q(s,a) = r(s,a) + \gamma \cdot \max_{a'} Q(s',a') \qquad (2)$$

where γ is the discount factor for the future rewards.

The general form of the $Q - learning$ algorithm is given in Figure 2.

```
For each pair $(s,a)$ initialize $Q(s,a)$ to 0.
Repeat (for each episode)
  Select the initial state $s$.
  Choose $a$ from $s$ using policy derived from $Q$.
        ($\epsilon$-Greedy, SoftMax [17])
  Repeat (for each step of the episode)
    Take action $a$, observe $r$, $s'$.
    Update the table entry $Q(s,a)$ as follows
      $Q(s,a) \leftarrow r(s,a) + \gamma \max_{a'} Q(s',a')$.
    s → s'
  until $s$ is terminal
Until the maximum number of episodes reached
or the $Q$-values do not change
```

Figure 2. The Q-learning algorithm

The method for updating the Q-values estimates given in Equation (2) can be modified in order to consider a learning rate $\alpha \in [0,1]$ [21], and consequently the Q-values are adjusted according to (3):

$$Q(s,a) = (1-\alpha)\cdot Q(s,a)+\alpha\cdot(r(s,a)+\gamma\cdot\max_{a'} Q(s',a')) \qquad (3)$$

3 Our Approach

In this section we introduce our reinforcement learning model proposal for solving the bidimensional *protein folding* problem. More exactly, we are focusing on predicting the bidimensional structure of a protein sequence. The model that we propose can be easily extended to adress the three-dimensional protein folding problem.

Let us consider, in the following, that $\mathcal{P} = p_1p_2...p_n$ is a protein HP sequence consisting of n amino acids, where $p_i \in \{H, P\},\ \forall 1 \leq i \leq n$. As we have indicated in Subsection 2.1.2, the bidimensional structure of $\mathcal{P}$ will be represented as an $n-1$-dimensional sequence $\pi = \pi_1\pi_2...\pi_{n-1}$, where each element π_k $(1 \leq k \leq n)$ encodes the direction (L, U, R or D) of the current amino acid location relative to the previous one.

A general RL task is characterized by four components:

1. a *state space* $\mathcal{S}$ that specifies all possible configurations of the system;
2. the *action space* $\mathcal{A}$ that lists all available actions for the learning agent to perform;
3. the *transition function* δ that specifies the possibly stochastic outcomes of taking each action in any state;
4. a *reward function* that defines the possible reward of taking each of the actions.

3.1 The *RL* task

Let us consider in the following an n-dimensional HP protein sequence, $\mathcal{P} = p_1p_2...p_n$. We assume that $n \geq 3$.

The RL task associated to the *BPFP* is defined as follows:

- The state space $\mathcal{S}$ (the agent's environment) will consist of $\frac{4^n-1}{3}$ states, i.e $\mathcal{S} = \{s_1, s_2, ..., s_{\frac{4^n-1}{3}}\}$. The *initial state* of the agent in the environment is s_1. A state $s_{i_k} \in \mathcal{S}(i_k \in [1, \frac{4^n-1}{3}])$ reached by the agent at a given moment after it has visited states $s_1, s_{i_1}, s_{i_2}, ...s_{i_{k-1}}$ is a *terminal* (final or goal) state if the number of states visited by the agent in the current sequence is $n-1$, i.e. $k = n-2$. A path from the initial to a final state will represent a possible bidimensional structure of the protein sequence $\mathcal{P}$.

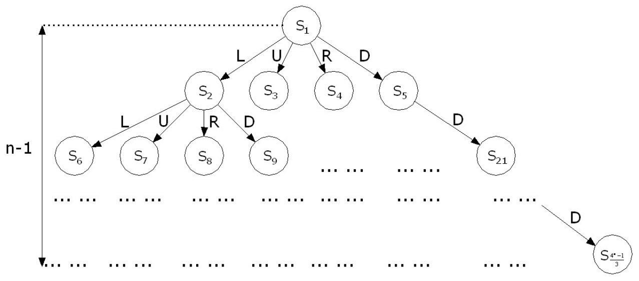

Figure 3. The states space.

- The action space $\mathcal{A}$ consists of 4 actions available to the problem solving agent and corresponding to the 4 possible directions $L(Left)$, $U(Up)$, $R(Right)$, $D(Down)$ used to encode a solution, i.e $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, where $a_1 = L$, $a_2 = U$, $a_3 = R$ and $a_4 = D$.
- The transition function $\delta : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ between the states is defined as in Formula 4.

$$\delta(s_j, a_k) = s_{4\cdot j-3+k}\ \forall k \in [1,4],\ \forall j, 1 \leq j \leq \frac{4^{n-1}-1}{3} \qquad (4)$$

  This means that, at a given moment, from a state $s \in \mathcal{S}$ the agent can move in 4 successor states, by executing one of the 4 possible actions. We say that a state $s' \in \mathcal{S}$ that is accessible from state s, i.e $s' \in \bigcup_{a\in\mathcal{A}} \delta(s,a)$, is the *neighbor* (*successor*) state of s.

  The transitions between the states are equiprobable, the transition probability $P(s, s')$ between a state s and each neighbor state s' of s is equal to 0.25 .
- The reward function will be defined below (Formula (5)).

A graphical representation of the states space for $BPFP$ associated to an n-dimensional HP protein sequence is given in Figure 3. The circles represent states and the transitions between states are indicated by arrows labeled with the action that leads the agent from a state to another.

Let us consider a path π in the above defined evironment from the initial to a final state, $\pi = (\pi_0\pi_1\pi_2\cdots\pi_{n-1})$, where $\pi_0 = s_1$ and $\forall 0 \leq k \leq n-2$ the state π_{k+1} is a *neighbor* of state π_k. The sequence of actions obtained following the transitions between the successive states from path π will be denoted by $a_\pi = (a_{\pi_0}a_{\pi_1}a_{\pi_2}\cdots a_{\pi_{n-2}})$, where $\pi_{k+1} = \delta(\pi_k, a_{\pi_k}),\ \forall 0 \leq k \leq n-2$. The sequence a_π will be refered as the *configuration* associated to the path π and it can be viewed as a possible bidimensional structure of the protein sequence $\mathcal{P}$. Consequently we can associate to a path π a value denoted by E_π representing the energy of the bidimensional configuration a_π of protein $\mathcal{P}$(Subsection 2.1.2).

The *BPFP* formulated as a RL problem will consist in training the agent to find a path π from the initial to a final state that will corespond to the bidimensional structure of protein $\mathcal{P}$ given by the coresponding configuration a_π and having the minimum associated energy.

It is known that the estimated utility of a state [15] in a reinforcement learning process is the estimated *reward-to-go* of the state (the sum of rewards received from the given state to a final state). So, after a reinforcement learning process, the agent learns to execute those transitions that maximize the sum of rewards received on a path from the initial to a final state.

As we aim at obtaining a a path π having the minimum associated energy E_π, we define the reinforcement function as follows (Formula (5)):

- if the transition generates a configuration that is not *valid* (i.e self-avoiding) (see Section 2.1.2) the received reward is 0.01.
- the reward received after a transition to a non terminal state is τ, where $\tau > 0.01$ is a small positive constant (e.q 0.1);
- the reward received after a transition to a final state π_{n-1} after states $s_1, \pi_1, \pi_2, ...\pi_{n-2}$ were visited is minus the energy of the bidimensional structure of protein $\mathcal{P}$ corresponding to the configuration a_π.

$$r(\pi_k|s_1, \pi_1, \pi_2, ...\pi_{k-1}) = \begin{cases} 0.01 & if \;\; a_\pi \; is \; not \; valid \\ -E_\pi & if \;\; k = n-1 \\ 0.1 & otherwise \end{cases}, \qquad (5)$$

where by $r(\pi_k|s_1, \pi_1, \pi_2, ...\pi_{k-1})$ we denote the reward received by the agent in state π_k, after it has visited states $\pi_1, \pi_2, ...\pi_{k-1}$.

Considering the reward defined in Formula (5), as the learning goal is to maximize the total amount of rewards received on a path from the initial to a final state, it can be easily shown that the agent is trained to find a self avoiding path π that minimizes the associated energy E_π.

3.2 The learning process

During the training step of the learning process, the agent will determine its *optimal policy* in the environment, i.e the *policy* that maximizes the sum of the received rewards.

For training the BPF (*Bidimensional Protein Folding*) agent, we propose the Q-learning approach that was described in Subsubsection 2.2.1). As we will prove in Section 4, during the training process, the Q-values estimations converge to their exact values, thus, at the end of the training process, the estimations will be in the vicinity of the exact values.

After the training step of the agent has been completed, the solution learned by the agent is constructed by starting from the initial state and following the $Greedy$ mechanism until a solution is reached. From a given state i, using the $Greedy$ policy, the agent transitions to a neighbor j of i having the maximum Q-value. Consequently, the solution of the *BPFP* reported by the RL agent is a path $\pi = (s_1\pi_1\pi_2 \cdots \pi_{n-2})$ from the initial to a final state, obtained following the policy described above. We mention that there may be more than one optimal policy in the environment determined following the $Greedy$ mechanism described above. In this case, the agent may report a single optimal policy of all optimal policies, according to the way it was designed. Considering the general goal of a RL agent, it can be proved that the configuration a_π corresponding to the path π learned by the BPF agent converges, in the limit, to the sequence that corresponds to the bidimensional structure of protein $\mathcal{P}$ having the minimum associated energy.

4 Mathematical validation. The convergence proof

In this section we give a mathematical validation of the approach proposed in Section 3 for solving the bidimensional protein folding problem. More exactly, we will prove that the Q-values learned by the BPF agent converge to their optimal values (i.e the values that lead to the policy corresponding to the bidimensional structure of protein $\mathcal{P}$ having the minimum associated energy) as long as all state-action pairs are visited an infinite number of times.

Let us consider, in the following, the RL task defined in Subsection 3.1. We denote by Q^* the exact evaluation function and by Q the estimate (hypothesis) of the Q^* function computed during the training step of the agent, as indicated in Figure 2.

We mention that the value of Q^* is the reward received immediately upon executing action a from state s, plus the value (discounted by γ) of following the optimal policy thereafter (Formula (6))[12].

$$Q^*(s, a) = r(s, a) + \gamma \cdot \max_{a'} Q^*(\delta(s, a), a') \qquad (6)$$

We mention that the reward function r is the one defined in Formula (5).

We want to prove that the Q-values learned after applying the Q-learning algorithm (Figure 2) to the RL task associated with the $BPFP$ (Section 3) converge to the exact Q^* values.

Let us denote by $Q_n(s,a)$ the agent's estimate of $Q(s,a)$ at the n-th training episode. We will prove that $lim_{n\to\infty}Q_n(s,a) = Q^*(s,a),\ \forall s \in \mathcal{S},\ a \in \delta(s,a)$.

First, we have to prove some additional lemmas.

Lemma 1 *Let us consider the n-dimensional HP protein sequence, $\mathcal{P} = p_1p_2...p_n$. The immediate reward values defined as given in Formula (5) are bounded, i.e*

$$0 \le r(s,a) \le \frac{(n-1)\cdot(n-2)}{2},\ \forall s \in \mathcal{S},\ a \in \delta(s,a). \tag{7}$$

Proof

Considering the Formula (1) that indicates how the energy associated to a bidimensional structure of a n-dimensional protein sequence is computed, it is obvious that

$$0 \ge E \ge \sum_{i=1}^{n-2}\sum_{j=i+2}^{n}(-1) = -\frac{(n-1)\cdot(n-2)}{2} \tag{8}$$

As $0 \le r(s,a) \le -E\ \forall s \in \mathcal{S},\ a \in \delta(s,a)$, using inequality (8) Lemma 1 is proved.

□

Lemma 2 *For each state action pair ($\forall s \in \mathcal{S},\ a \in \delta(s,a)$), the estimates $Q(s,a)$ increase during the training process, i.e*

$$Q_{n+1}(s,a) \ge Q_n(s,a),\ \forall n \in N^* \tag{9}$$

Proof

We prove by mathematical induction.

Below we use s' to denote $\delta(s,a)$.

First, we prove that Inequalities (9) hold for $n = 0$.

From (2), as the initial Q-values are 0, i.e $Q_0(s,a) = 0$, we obtain $Q_1(s,a) = r(s,a)$. Since all rewards are positive, it follows that $Q_1(s,a) \ge Q_0(s,a)$ and the first step of the induction is proven.

Now, we have to prove the induction step. Assuming that inequalities (9) hold for a given $n \le 2$, i.e

$$Q_n(s,a) \ge Q_{n-1}(s,a), \tag{10}$$

we want to prove that Inequalities (9) hold for $n+1$, also, i.e

$$Q_{n+1}(s,a) \ge Q_n(s,a), \tag{11}$$

Using (2) and the fact that rewards are bounded (Lemma 1), we have that

$$Q_{n+1}(s,a) - Q_n(s,a) = \gamma\cdot(max_{a'}Q_n(s',a')- \tag{12}$$
$$-max_{a'}Q_{n-1}(s',a'))$$

From (12), using (10) and the fact that $\gamma > 0$, it follows that

$$Q_{n+1}(s,a) - Q_n(s,a) \ge \gamma\cdot(max_{a'}Q_{n-1}(s',a')- \tag{13}$$
$$-max_{a'}Q_{n-1}(s',a')) = 0$$

Consequently, (13) proves the induction step.

So, Lemma 2 is proven.

□

Lemma 3 *For each state action pair ($\forall s \in \mathcal{S},\ a \in \delta(s,a)$), the estimates $Q(s,a)$ are upper bounded by the exact Q^* values, i.e*

$$Q_n(s,a) \le Q^*(s,a),\ \forall s \in \mathcal{S},\ a \in \delta(s,a) \tag{14}$$

Proof

We know that $Q^*(s,a)$ is the discounted sum of rewards obtained when starting from s, performing action a and following an optimal policy to a final state. Because all rewards are positive, it is obvious that $Q^*(s,a) \ge 0$.

We prove Inequalities (14) by induction.

First, we prove that Inequalities (9) hold for $n = 0$.

Since $Q_0(s,a) = 0$, and $Q^*(s,a) \ge 0$ we obtain that $Q_0(s,a) \le Q^*(s,a)$. So, the first step of the induction is proven.

Now, we have to prove the induction step. Assuming that inequalities (14) hold for a given $n \le 1$, i.e

$$Q_n(s,a) \le Q^*(s,a), \tag{15}$$

we want to prove that Inequalities (14) hold for $n+1$, also, i.e

$$Q_{n+1}(s,a) \le Q^*(s,a), \tag{16}$$

Using (2) and (6 we have that

$$Q_{n+1}(s,a) - Q^*(s,a) = \gamma\cdot(max_{a'}Q_n(s',a')- \tag{17}$$
$$-max_{a'}Q^*(s',a'))$$

From (17), using (15) and the fact that $\gamma > 0$, it follows that

$$Q_{n+1}(s,a) - Q^*(s,a) \ge \gamma\cdot(max_{a'}Q^*(s',a')- \tag{18}$$
$$-max_{a'}Q^*(s',a')) = 0$$

Consequently, (18) proves the induction step.

So, Lemma 3 is proven.

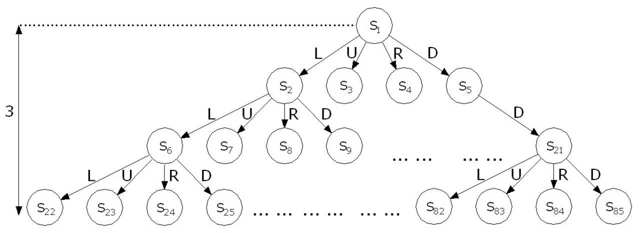

Figure 4. The environment.

□

Now, we can give and prove the theorem that assures the convergence.

Theorem 1 *Let us consider the RL task associated to the $BPFP$ given in Section 3. The BPF agent is trained using the algorithm indicated in Figure (2). If each state action pair is visited infinitely often during the training, then $Q_n(s,a)$ converges to $Q^*(s,a)$ as $n \to \infty$, for all s, a.*

Using Lemmas 2 and 3, we have that the array $Q_n(s,a)$ increases and is upper bounded, which means that it is convergent. Moreover, it follows that the (superior) limit of $Q_n(s,a)$ is $Q^*(s,a)$.

So, Theorem 1 is proven.

□

5 Computational experiments

In this section we aim at providing the reader with an easy to follow example illustrating how our approach works.

Let us consider a HP protein sequence $\mathcal{P} = HHPH$, consisting of four amino acids, i.e $n = 4$. As we have presented in Section 3, the states space will consist of 85 states, i.e $\mathcal{S} = \{s_1, s_2, ..., s_{85}\}$. The states space is illustrated in Figure 4. In the figure the circles represent states and the transitions between states are indicated by arrows labeled with the action that leads the agent from a state to another.

5.1 Experiment 1

First, we have trained the BPF agent as indicated in Subsection 3.2, using Formula (2) to update the Q-values estimates. We remark the following regarding the parameters setting:

- the discount factor for the future rewards is $\gamma = 0.9$;
- the number of training episodes is 36 training episodes;
- the ϵ-Greeedy action selection mechanism was used;

State	**Action**	**Action**	**Action**	**Action**
	$a_1 = L$	$a_2 = U$	$a_3 = R$	$a_4 = D$
1	1.00000000	1.00000000	0.91900000	1.00000000
2	0.10900000	1.00000000	0.91000000	1.00000000
3	1.00000000	0.10900000	1.00000000	0.91000000
4	0.91000000	0.10900000	0.10000000	0.10900000
5	1.00000000	0.91000000	1.00000000	0.10900000
6	0.00000000	0.00000000	0.01000000	0.00000000
7	0.00000000	0.00000000	1.00000000	0.01000000
8	0.01000000	1.00000000	1.00000000	1.00000000
9	0.00000000	0.01000000	1.00000000	0.00000000
10	0.00000000	0.00000000	0.01000000	1.00000000
11	0.00000000	0.00000000	0.00000000	0.01000000
12	0.01000000	0.00000000	0.00000000	1.00000000
13	1.00000000	0.01000000	1.00000000	1.00000000
14	1.00000000	1.00000000	0.01000000	1.00000000
15	1.00000000	0.00000000	0.00000000	0.01000000
16	0.01000000	0.00000000	0.00000000	0.00000000
17	1.00000000	0.01000000	0.00000000	0.00000000
18	0.00000000	1.00000000	0.01000000	0.00000000
19	1.00000000	1.00000000	1.00000000	0.01000000
20	0.01000000	1.00000000	0.00000000	0.00000000
21	0.00000000	0.01000000	0.00000000	0.00000000
22	0.00000000	0.00000000	0.00000000	0.00000000
....	....	....	....	
85	0.00000000	0.00000000	0.00000000	0.00000000

Table 1. The Q-values after traning was completed.

Using the above defined parameters and under the assumptions that the state action pairs are equally visited during training and that the agent explores its search space (the ϵ parameter is set to 1), the Q-values indicated in Table 1 were obtained.

Three optimal solutions were reported after the training of the BFP agent was completed, determined starting from state s_1, following the $Greedy$ policy (as we have indicated in Subsection 3). All these solutions correspond, having a minimum associated energy of -1.

The optimal solutions are:

1. The path $\pi = (s_1 s_2 s_7 s_{28})$ having the associated *configuration* $a_\pi = (LUR)$ (Figure 5).

2. The path $\pi = (s_1 s_3 s_{10} s_{41})$ having the associated *configuration* $a_\pi = (ULD)$ (Figure 6).

3. The path $\pi = (s_1 s_3 s_{12} s_{49})$ having the associated *configuration* $a_\pi = (URD)$ (Figure 7).

4. The path $\pi = (s_1 s_5 s_{18} s_{71})$ having the associated *configuration* $a_\pi = (DLU)$ (Figure 8).

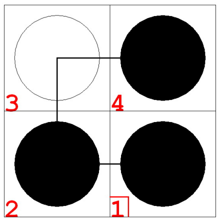

Figure 5. The configuration learned is LUR. The value of the energy function for this configuration is -1.

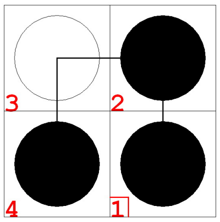

Figure 6. The configuration learned is ULD. The value of the energy function for this configuration is -1.

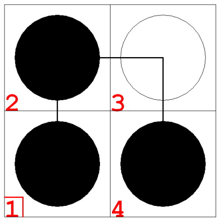

Figure 7. The configuration learned is URD. The value of the energy function for this configuration is -1.

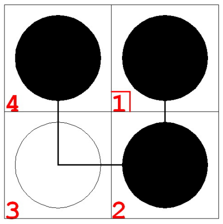

Figure 8. The configuration learned is DLU. The value of the energy function for this configuration is -1.

State	Action $a_1 = L$	Action $a_2 = U$	Action $a_3 = R$	Action $a_4 = D$
1	0.01532343	0.01521037	0.01530453	0.01531066
2	0.00394219	0.00412038	0.00066135	0.00411950
3	0.00420771	0.00394307	0.00420771	0.00066135
4	0.00066135	0.00394307	0.00394040	0.00394130
5	0.00403129	0.00057402	0.00403040	0.00394130
6	0.00000000	0.00000000	0.00010000	0.00000000
7	0.00000000	0.00000000	0.01000000	0.00010000
8	0.00010000	0.01000000	0.01000000	0.01000000
9	0.00000000	0.00010000	0.01000000	0.00000000
10	0.00000000	0.00000000	0.00010000	0.01000000
11	0.00000000	0.00000000	0.00000000	0.00010000
12	0.00010000	0.00000000	0.00000000	0.01000000
13	0.01000000	0.00010000	0.01000000	0.01000000
14	0.01000000	0.01000000	0.00010000	0.01000000
15	0.01000000	0.00000000	0.00000000	0.00010000
16	0.00010000	0.00000000	0.00000000	0.00000000
17	0.01000000	0.00010000	0.00000000	0.00000000
18	0.00000000	0.01000000	0.00010000	0.00000000
19	0.01000000	0.01000000	0.01000000	0.00010000
20	0.00010000	0.01000000	0.00000000	0.00000000
21	0.00000000	0.00010000	0.00000000	0.00000000
22	0.00000000	0.00000000	0.00000000	0.00000000
....	....	....	....	
85	0.00000000	0.00000000	0.00000000	0.00000000

Table 2. The Q-values after traning was completed.

Consequently, the BPF agent learns a solution of the bidimensional protein folding problem, i.e a bidimensional structure of the protein $\mathcal{P}$ that has a minimum associated energy (-1).

5.2 Experiment 2

Secondly, we have trained the BPF agent as indicated in Subsection 3.2, but using for updating the Q-values estimates Formula (3). As proven in [21], the Q-learning algorithm converges to the real Q^*-values as long as all state-action pairs are visited an infinite number of times, the learning rate α is small (e.q 0.01) and the policy converges in the limit to the Greedy policy. We remark the following regarding the parameters setting:

- the learning rate is $\alpha = 0.01$ in order to assure the convergence of the algorithm;
- the discount factor for the future rewards is $\gamma = 0.9$;
- the number of training episodes is 36 training episodes;
- the ϵ-Greeedy action selection mechanism was used;

Using the above defined parameters and under the assumptions that the state action pairs are equally visited during training and that the agent explores its search space (the ϵ parameter is set to 1), the Q-values indicated in Table 2 were obtained.

The solution reported after the training of the BFP agent was completed is the path $\pi = (s_1 s_2 s_7 s_{28})$ having the associated *configuration* $a_\pi = (LUR)$, determined starting from state s_1, following the $Greedy$ policy (as we have indicated in Subsection 3).

The solution learned by the agent is represented in Figure 9 and has an energy of -1.

Consequently, the BPF agent learns the optimal solution of the bidimensional protein folding problem, i.e the bidimensional structure of the protein $\mathcal{P}$ that has a minimum associated energy (-1).

6 Disscussion

Regarding the Q-learning approach introduced in Section 3 for solving the bidimensional protein folding problem, we remark the following:

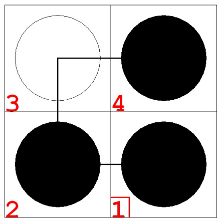

Figure 9. The learned solution is LUR. The value of the energy function for this configuration is -1.

- The training process during an episode has a time complexity of $\theta(n)$, where n is the length of the HP protein sequence. Consequently, assuming that the number of training episodes is k, the overall complexity of the algorithm for training the BPF agent is $\theta(k \cdot n)$.
- If the dimension n of the HP protein sequence is large and consequently the state space becomes very large (consisting of $\frac{4^n-1}{3}$ states), in order to store the Q values estimates, a neural network should be used.

In the following we will briefly compare our approach with some of the existing approaches. The comparison is made considering the computational time complexity point of view. Since for the most of the existing approaches the authors do not provide the asymptotic analysis of the time complexity of the proposed approaches, we can not provide a detailed comparison.

Hart and Belew show in [9] that an asymptotic analysis of the computational complexity for evolutionary algorithms (EAs) is difficult, and is done only for particular problems. He and Yao analyse in [10] the computational time complexity of evolutionary algorithms, showing that evolutionary approaches are computationally expensive. The authors give conditions under which EA need at least exponential computation time on average to solve a problem, which leads us to the conclusion that for particular problems the EA performance is poor. Anyway, a conclusion is that the number of generations (or equivalently the number of fitness evaluations) is the most important factor in determining the order of EA's computation time. In our view, the time complexity of an evolutionary approach for solving the problem of predicting the structure of an n-dimensional protein is at least $noOfRuns \cdot n \cdot noOfGenerations \cdot populationLength$. Since our RL approach has a time complexity of $\theta(n \cdot noOfEpisodes)$, it is likely (even if we can not rigurously prove) that our approach has a lower computational complexity.

Neumann et al. show in [13] how simple ACO algorithms can be analyzed with respect to their computational complexity on example functions with different properties, and also claim that asymptotic analysis for general ACO systems is difficult. In our view, the time complexity of an ACO approach for solving the problem of predicting th structure of an n-dimensional protein is at least $noOfRuns \cdot n \cdot noOfIterations \cdot noOfAnts$. Since our RL approach has a time complexity of $\theta(n \cdot noOfEpisodes)$, it is likely (even if we can not rigurously prove) that our approach has a lower computational complexity.

Compared to the supervised classification approach from [7], the advantage of our RL model is that the learning process needs no external supervision, as in our approach the solution is learned from the rewards obtained by the agent during its training. It is well known that the main drawback of supervised learning models is that a set of inputs with their target outputs is required, and this can be a problem.

We can conclude that a major advantage of the RL approach that we have introduced in this paper is the fact that the convergence of the Q-learning process was mathematically proven, and this confirms the potential of our proposal. We think that this direction of using reinforcement learning techniques in solving the protein folding problem is worth being studied and further improvements can lead to valuable results.

The main drawback of our approach is that, as we have shown in Section 4, a very large number of training episodes has to be considered in order to obtain accurate results and this leads to a slow convergence. In order to speed up the convergence process, further improvements will be considered.

7 Conclusions and Further Work

We have proposed in this paper a reinforcement learning based model for solving the bidimensional protein folding problem. To our knowledge, except for the ant based approaches, the bidimensional *protein folding* problem has not been addressed in the literature using reinforcement learning, so far. The model proposed in this paper can be easily extended to solve the three-dimensional protein folding problem, and moreover to solve other optimization problems.

We have emphasized the potential of our proposal by giving a mathematical validation of the proposed model, highlighting its advantages.

We plan to extend the evaluation of the proposed RL model for some large HP protein sequences, to further test its performance. We will also investigate possible improvements of the RL model by analyzing a temporal difference approach [17], by using different reinforcement functions and by adding different local search mechanisms in order to increase the model's performance. An extension of the

BPF model to a distributed RL approach will be also considered.

ACKNOWLEDGEMENT

This work was supported by CNCSIS - UEFISCSU, project number PNII - IDEI 2286/2008.

References

[1] C. B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.

[2] B. Berger and T. Leighton. Protein folding in hp model is np-complete. *Journal of Computational Biology*, 5:27–40, 1998.

[3] T. Beutler and K. Dill. A fast conformational search strategy for finding low energy structures of model proteins. *Protein Science*, 5:2037–2043, 1996.

[4] D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: an algorithm and performance comparisons. In *Proceedings of the 12th international joint conference on Artificial intelligence - Volume 2*, pages 726–731, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[5] C. Chira. Hill-climbing search in evolutionary models for protein folding simulations. *Studia*, LV:29–40, 2010.

[6] K. Dill and K. Lau. A lattice statistical mechanics model of the conformational sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989.

[7] C. H. Q. Ding and I. Dubchak. Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, 17:349–358, 2001.

[8] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.

[9] W. E. Hart and R. K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190–195. Morgan Kaufmann, 1991.

[10] J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57 – 85, 2001.

[11] L. J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[12] T. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997.

[13] F. Neumann, D. Sudholt, and C. Witt. Computational complexity of ant colony optimization and its hybridization with local search. In *Innovations in Swarm Intelligence*, pages 91–120. 2009.

[14] A. Perez-Uribe. Introduction to reinforcement learning, 1998. http://lslwww.epfl.ch/∼anperez/RL/RL.html.

[15] S. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall International Series in Artificial Intelligence. Prentice Hall, 2003.

[16] A. Shmygelska and H. Hoos. An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. *BMC Bioinformatics*, 6, 2005.

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[18] T. Thalheim, D. Merkle, and M. Middendorf. Protein folding in the hp-model solved with a hybrid population based aco algorithm. *IAENG International Jurnal of Computer Science*, 35, 2008.

[19] S. Thrun. The role of exploration in learning control. In *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, Florence, Kentucky, 1992.

[20] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Mol. Biol.*, 231:75–81, 1993.

[21] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[22] X. Zhang, T. Wang, H. Luo, Y. Yang, Y. Deng, J. Tang, and M. Q. Yang. 3d protein structure prediction with genetic tabu search algorithm. *BMC Systems Biology*, 4, 2009.