Solving the Protein Folding Problem Using a Distributed Q-Learning Approach

Gabriela Czibula, Maria-Iuliana Bocicor, and Istvan-Gergely Czibula

***Abstract*—**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). We are introducing in this paper a distributed reinforcement learning based approach for solving the *bidimensional protein folding* problem, an *NP*-complete problem that refers to predicting the bidimensional structure of a protein from its amino acid sequence. Our model is based on a distributed $Q - learning$ approach. The experimental evaluation of the proposed system has provided encouraging results, indicating the potential of our proposal. The advantages and drawbacks of the proposed approach are also emphasized.

Keywords – Bioinformatics, Distributed Reinforcement Learning, Protein Folding.

I. INTRODUCTION

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.

This work was possible with the financial support of the Sectorial Operational Programme for Human Resources Development 2007-2013, co-financed by the European Social Fund, under the project number POS-DRU/107/1.5/S/76841 with the title "Modern Doctoral Studies: Internationalization and Interdisciplinarity".

Gabriela Czibula, Maria-Iuliana Bocicor and Istvan-Gergely Czibula are with Department of Computer Science, Babeş-Bolyai University, Cluj-Napoca, 1, M. Kogălniceanu street, 40084, Cluj-Napoca, Romania. Phone: +40-264-405.327. E-mail:{gabis, iuliana, istvanc}@cs.ubbcluj.ro

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 [1], 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.

Reinforcement Learning (RL) [5] 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 distributed reinforcement learning based model for solving the *bidimensional 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). The model proposed in this paper extends the reinforcement learning model that we have previously introduced in [6], [7] for solving the problem.

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. To our knowledge, except for the ant [8] based approaches, the bidimensional *protein folding* problem has not been addressed in the literature using distributed reinforcement learning, so far.

The rest of the paper is organized as follows. Section II presents the *protein folding* problem and Section III briefly describes existing approaches in solving the *protein folding problem*. The fundamentals of reinforcement learning are given in Section IV-B. Section V introduces the distributed reinforcement learning model that we propose for solving the bidimensional protein folding problem. An experimental evaluation of the proposed approach is given in Section VI, as well as an analysis of the proposed model, emphasizing its advantages and drawbacks. Section VII contains some conclusions of the paper and future development of our work.

II. The Protein Folding Problem. 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 [9].

One of the most popular lattice-models is Dill's Hydrophobic-Polar (HP) model [10].

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):

$$\mathcal{P} = p_1p_2...p_n, \text{ where } p_i \in \{H, P\}, \ \forall 1 \le i \le n$$

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:

$$\mathcal{B} = \{\mathcal{P} = p_1p_2...p_n | \ p_i \in \{H, P\}, \forall 1 \le i \le n, n \in \mathcal{N}\}$$

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

then a conformation C is defined as follows:

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

$$\mathcal{P} = p_1p_2...p_n \mapsto \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

(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, $\forall 1 \le i \le n$

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:

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

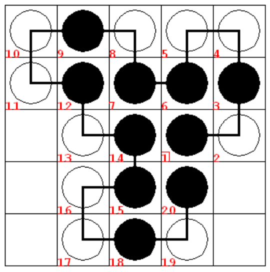

Fig. 1. A protein configuration for the sequence $\mathcal{P} = HPHPPHHPHPPHPHHPPHPH$, of length 20. Black circles represent hydrophobic amino acids, while white circles represent hydrophilic ones. The configuration may be represented by the sequence $\pi = RUULDLULLDRDRDLDRRU$. The value of the energy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

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 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:

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

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 $P \in \mathcal{B}$ can 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 $\pi = RUULDLULLDRDRDLDRRU$.

III. Related work

In this section we briefly present several approaches existing in the literature for solving the *protein foding* problem.

A. Protein Folding

The protein structure prediction problem is one of the most important problems in computational biology. Many approximation and heuristics methods, that use different abstract models tackle it.

Berger and Leighton [11] 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 [12] 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 [13] 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 [14], 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 [15] 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 [16]. 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 [17] 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 direction 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 [18] as base classifiers for the protein fold recognition problem.

Researchers have also been focused on solving problems that help solving the protein folding problem, such as the prediction of the location of disulfide bridges [2].

We have previously introduced in [6], [7] a reinforcement learning based model for solving the problem of predicting the bidimensional structure of proteins in the hydrophobic-polar model.

B. Distributed Protein Folding

The literature offers few approaches that make use of multi-agent systems to the purpose of proposing solutions to this problem. We will briefly present some of these approaches.

Bortolussi, Dovier and Fogolari present in [19] a high-level agent-based framework for protein structure prediction. The agents are separated on three layers, according to their knowledge and power. The first layer contains agents that explore the configuration space. At this level each amino acid is modeled as an independent agent, that moves in the state space (using a simulated annealing scheme), interacts and communicates with its spatial neighbors in order to minimize the energy function. As each agent in the first layer can explore a relatively small neighborhood and the search space is very complex, a good coordination between these agents is considered to be necessary. This is achieved by some higher-level agents - the strategic agents, who have a global knowledge of the current configuration and whose role is to coordinate the search agents (according to a global strategy), as well as to control the cooling strategy for the temperature in the simulated annealing technique. Last, but not least, the third layer is composed of cooperative agents, which implement a basic form of cooperation, based on secondary structure information, between lower-level agents. The cooperative agents control the activity of the search and the strategic agents and are also responsible for the termination of the simulation.

In [20], the authors present a bioinformatics framework, called Evolution, which is based on a multi-agent system and uses a blackboard architecture. There are three types of agents: model agents, algorithm agents and interface agents and the communication and coordination among them is achieved by a blackboard mechanism, which also allows data sharing. The blackboard is composed of several levels, each one recording the solution elements needed for the resolution of the problem: the amino acid sequence level - deals with the input amino acids sequences; the HP sequence level - contains the HP translations (of the input sequences), generated by the model generator agent; the initial conformational space level - contains the initial bidimensional or tridimensional conformations, generated by the conformation generator agent (that is considered a model agent); the algorithm workspace level - records all the conformations that were generated in the processes of heuristic search and optimization, which are achieved by the genetic agents (algorithm agents); the plausible conformation level - contains the optimal conformations found by the genetic agents as a result of the heuristic search and the optimization. Finally, there are also the interface agents, that are used for the implementation of numerous graphic user interfaces and graphic tools that Evolution provides for the user-system interaction.

In Artificial Intelligence, an agent is an autonomous entity that receives perceptions and performs actions upon an environment in order to achieve a certain goal. Therefore, an Ant Colony Optimization (ACO) method can be regarded as a multi-agent system, where ants are simple agents which indirectly communicate through interaction with the environment, by depositing pheromone on the paths they follow. Consequently, the ACO algorithm introduced in [13] by Shmygelska and Hoos for the Protein Folding Problem can be viewed as a multi-agent based approach.

IV. 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 [3], [4].

Reinforcement Learning (RL) [21] 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 [22]. 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 [5]:

- *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 [23]. 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 $r_0 + \gamma \cdot r_1 + \gamma^2 \cdot r_2 + ...$ ($0 \leq \gamma < 1$ is the discount factor for the future rewards).

One key aspect of reinforcement learning is a trade-off between *exploitation* and *exploration* [24]. 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*.

A. Q-learning

Q-learning [5] 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 [25]:

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

where γ is the discount factor for the future rewards.

The general form of the $Q - learning$ algorithm is given below.

```
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 [5])
  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 $\rightarrow$ s'
    until $s$ is terminal
Until the maximum number of episodes reached or the $Q$-values do not change
```

The method for updating the Q-values estimates given in Equation (2) can be modified in order to consider a learning rate $\alpha \in [0,1]$ [26], and consequently the Q-values are adjusted according to (3):

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

B. Distributed Reinforcement Learning

Reinforcement learning (RL) is an approach to machine intelligence, in which an autonomous agent learns to behave in an environment, by receiving rewards - for its successes or punishments - for its failures. By trial and error, the agent must find an optimal policy, that selects the most appropriate action, in each state, in order to achieve the goal. In recent years there has been great interest in distributed reinforcement learning multi-agent systems (MASs) consisting of agents that work together in order to optimize a joint performance measure.

Distributed reinforcement learning problems may be modeled in different ways, one of the most frequently used being the Multi-Agent Markov Decision Process (MAMDP), which is an extension of the single-agent Markov Decision Process [25]. In an MAMDP each agent has a finite state space and a finite action space and thus the joint state space will be the Cartesian product of all state spaces of all the agents, while the joint action space will be, similarly, the Cartesian product of all action spaces, corresponding to all the agents. Therefore, an MAMDP can be modeled as a 5-tuple (N, S, A, δ, R), where N is the set of agents, S is the joint state space, A is the joint action space, δ is the transition probability and R is the reward. These last two functions - δ and R are similar to those in the standard MDP, except that they are defined over the joint state and joint action spaces, being independent of the time step. Consequently, the Markov property still holds within this model.

One of the most frequently used single agent RL algorithms is Q-Learning [25], which finds a mapping from state/action pairs to values (called *Q-values*). An optimal Q-value is the sum of reinforcements received when performing the associated action and following the optimal policy thereafter. Q-values were proven to converge to their optimal values within the Q-Learning algorithm if all the state/action pairs are visited an infinite number of times. Still, in MASs, the convergence of Q-values cannot be guaranteed, as each agent is simultaneously learning its own actions and the environment becomes non-stationary.

In the literature, there are several approaches to the problem of applying Q-Learning in MASs. Claus and Boutilier present in [27] two ways in which Q-Learning could be used in MASs: the *Independent Learners (IL) Algorithm* - agents ignore other agents' actions and each one learns its Q-values independently; the *Joint Action Learners (JAL)* - agents that learn Q-values for the joint state/action space, rather than the individual state/action space. Each JAL maintains information about the strategies of the other agents and chooses its actions according to the expected value based on this information.

In [28] the author studies cooperative agents and independent agents for the purpose of concluding whether the former outperform the latter. The author states that several independent RL agents will surely outperform one single RL agent, due to the fact that they have more resources and better chances of receiving rewards and then he proposes to study the results obtained by several agents that cooperate. Three ways of cooperation are identified: agents can communicate instantaneous information such as perceptions, actions or rewards; they can communicate experienced episodes (sequences of perceptions, actions and rewards); or they can communicate learnt decision policies. The conclusion of the paper is that cooperative RL agents that share episodes or policies learn faster and converge sooner than independent agents, but coordination also has some drawbacks, as sharing knowledge comes with a communication cost and cooperative behavior for joint tasks automatically implies a larger state space.

In [29], Lauer and Riedmiller propose an algorithm which finds optimal policies for distributed Q-learning in deterministic environments. As it is considered impossible for each agent to distinguish between different joint action vectors that contain the same individual action for an agent, the individual agents are not capable of computing Q-functions defined on the current state of the whole environment and the joint vector of actions. Therefore, the authors propose projecting the large Q-table in smaller Q-tables that compress the information of the large one, specific for each agent. A projection based on the *optimistic assumption* is introduced: each agent assumes that the other agents will behave optimally, i.e. the joint action vector composed of all the individual actions of the agents represents an optimal action for the system. In this case, the Q-values of each agent are chosen as the optimal values of the large Q-table. Still, the optimal assumption can be violated in the case of several optimal joint actions in a single state, when the optimal behavior of each agent is not guaranteed. For this reason, an additional mechanism for coordination between the agents is introduced: the learning algorithm of each agent updates the current policy only if there was an improvement in its own Q-value.

Marino and Morales describe in [30] a new distributed Q-Learning algorithm - DQL: the agents find a common policy in a distributed environment by transmitting information about how optimal each action is using traces (of all the other agents) generated from transition between states. The final policy is based on the most frequently selected actions.

V. Our Distributed Reinforcement Learning Model Proposal

A. Background

In this section we present the reinforcement learning model that we have previously introduced in [6] for solving the bidimensional *protein folding* problem. More exactly, we are focusing on predicting the bidimensional structure of a protein sequence. The RL model introduced in [6] will be extended in this section towards a distributed architecture.

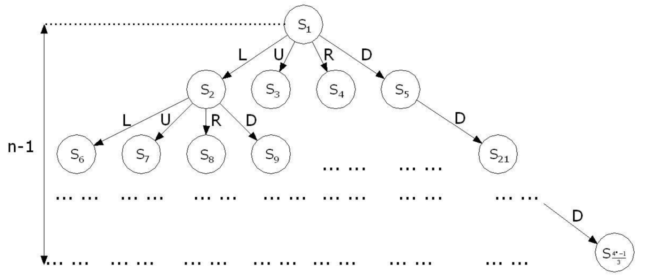

Fig. 2. The states space.

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 II, 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.

The RL task associated to the *BPFP* is defined as follows [6]:

- 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 $\mathcal{P}$.
- 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 $\delta : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ between the states is defined as in Formula 4.

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

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 2. 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 II).

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 [25] 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 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 II) 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.

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

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.

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, a Q-learning approach was proposed. We have proven in [6] that, 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.

B. The proposed Distributed Approach

As we have shown in [6], a very large number of training episodes is required in order to obtain an accurate solution using the Q-learning approach presented in Subsection V-A. That is why, in order to speed up the training process, we extend the proposed approach towards a distributed one, in which multiple cooperative agents learn to coordinate in order to find the optimal policy in their environment. The approach proposed in the section is a kind of concurrent Q-learning approach. Q-learning is an approach to *reinforcement learning* that finds a mapping from state/action pairs to values (called *Q*-values), rather then finding a mapping from states to state 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.

We have two types of agents in our architecture:

- $BPFA$ (*Bidimensional Protein Folding Agents*). Each $BPFA$ agent runs in a separate process or thread and is trained using the Q-learning algorithm [6]. Each local agent performs local Q-values estimations updates from its own point of view.
- a $BPFS$ (*Bidimensional Protein Folding Supervisor*) agent wich supervises the learning process and synchronizes the computations of the individual $BPFA$ agents. It keeps a blackboard [31] which stores the global Q-values estimations. The local $BPFA$ agents use the global Q-values estimations stored in the blackboard and comunicate to the $BPFS$ agent their intention to update a Q-value estimation. If a local agent tries to update a certain Q-value, the $BPFS$ agent will update the

global Q-value estimation only if the new estimation received from the local agent is greater than the Q-values estimations existing in the blackboard. This updating strategy is based on our previous result from [6] in which we have proven that in a non-distributed RL scenario the Q-values estimates for each state-action pair increase during the training process and are upper bounded by the exact values.

We have two possible architectures for the proposed multi-agent system:

1) The $BPFA$ agents are running in the same process with the supervisor agent $BPFS$. In this case each $BPFA$ agent has an instance of the supervisor $BPFS$ agent, and this will reduce the cost of communication (messages exchanges) between the agents. The blackboard stored by $BPFS$ allows a kind of indirect data communication between the local agents and the global supervising agent.
2) The $BPFA$ agents are distributed across multiple processes/machines. In this case network communication is involved when the $BPFA$ agents ask the supervisor for Q-values or request a Q-value update. In order to reduce the number of messages exchanged between $BPFA$ agents and the supervisor $BPFS$, the frequency for updating the Q-values in the blackboard can be decreased if the $BPFA$ agents will synchronize their Q-values with the supervisor only after several training epochs.

The training process consists of three phases and will be briefly described in the following.

Phase 1. Initial phase

The $BPFS$ supervisor agent initializes with 0 the Q-values from the blackboard.

Phase 2. Training phase of each $BPFA$ agent

During some training episodes, the individual $BPFA$ agents will experiment (using the ϵ-Greedy action selection mechanism) some (possible optimal) paths from the initial to a final state, updating the Q-values estimations according to the $Q-learning$ algorithm [32].

The general form of the $Q-learning$ algorithm is given in Figure 3. We denote in the following by $Q(s,a)$ the Q-value estimate associated to the state s and a, as stored by the blackboard of the $BPFS$ agent.

```
Repeat (for each episode)
  Select the initial state $s$.
  Choose action $a$ from $s$ using policy derived from $Q$.
     ($\epsilon$-Greedy, SoftMax [5])
  Repeat (for each step of the episode)
    Take action $a$, observe the reward $r(s,a)$ and the next state $s'$.
    $BPFA$ agent asks $BPFS$ agent for $Q(s,a)$.
    $BPFS$ agent retrieves $Q(s,a)$ from the blackboard.
    $BPFS$ sends the retrieved $Q(s,a)$ to $BPFA$.
    $BPFA$ agent updates the table entry $Q(s,a)$ as follows
      $Q(s,a) = (1-\alpha)\cdot Q(s,a) + \alpha\cdot(r(s,a) + \gamma\cdot\max_{a'} Q(s',a'))$
    where $\alpha \in [0,1]$ is a learning rate [26].
    $BPFA$ sends the new $Q(s,a)$ to $BPFS$.
    $BPFS$ updates $Q(s,a)$ in the blackboard if needed
    s $\rightarrow$ s'
  until $s$ is terminal
Until the maximum number of episodes is reached or the $Q$-values do not change
```

Fig. 3. The Q-learning algorithm.

Phase 3. Final phase

After the training of the multiagent system has been completed, the solution learned by the $BPFS$ supervisor agent is constructed by starting from the initial state and following the $Greedy$ mechanism until a solution is reached.

The architecture that we propose for the distributed $Q-learning$ system for solving the *bidimensional protein folding problem* is presented in Figure 4. The message exchanges between the agents, as highlighted in Figure 3, are illustrated in Figure 4 by the labeled arrows.

VI. Computational experiment

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 V-A, the states space will consist of 85 states, i.e $\mathcal{S} = \{s_1, s_2, ..., s_{85}\}$.

We have applied the distributed RL approach introduced in Subsection V-B with the following settings:

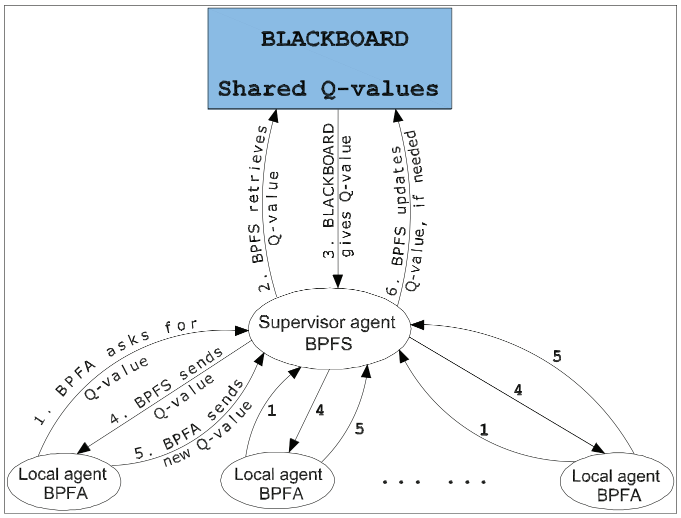

Fig. 4. The distributed $Q-learning$ architecture.

State	**Action** $a_1 = L$	**Action** $a_2 = U$	**Action** $a_3 = R$	**Action** $a_4 = D$
1	0.01532343	0.01286043	0.00835149	0.01531066
2	0.00394219	0.00412038	0.00066135	0.00411950
3	0.00420771	0.00394307	0.00420771	0.00039671
4	0.00039671	0.00039671	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	0.00010000	0.00010000
14	0.00010000	0.00010000	0.00010000	0.00010000
15	0.00010000	0.00010000	0.00010000	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 I
THE Q-VALUES STORED IN THE BLACKBOARD.

- two local $BPFA$ agents were used;
- the number of training episodes for each local $BPFA$ agent is 40;
- both local $BPFA$ agents have the same behaviour in the $Q - learning$ scenario: they use the ϵ-Greeedy action selection mechanism, a learning rate $\alpha = 0.01$ in order to assure the convergence of the algorithm and a discount factor for the future rewards $\gamma = 0.9$.

Using the above defined settings and under the assumptions that the state action pairs are equally visited during the training and that each local $BPFA$ agent explores its search space (the ϵ parameter is set to 1), the Q-values indicated in Table I were obtained by the supervisor $BPFS$ agent.

After the training of the $BPFA$ agents was completed, the solution reported by the $BPFS$ agent 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.

The solution learned by the $BPFS$ agent is represented in Figure 5 and has an energy of -1.

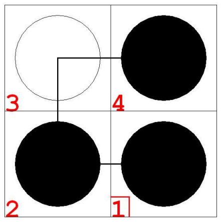

Fig. 5. The learned solution is LUR. The value of the energy function for this configuration is -1.

Consequently, the $BPFS$ 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).

Regarding the RL approach previously introduced in [6], [7] for solving the bidimensional protein folding problem, approach that has been extended in this paper towards a distributed one, we remark the following:

- 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 a $BPFA$ 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.

The main drawback of the non-distributed learning approach is that a very large number of training episodes has to be considered in order to obtain accurate results and this leads to a slow convergence.

It is obvious that the distributed RL approach presented in this paper, by using multiple agents during the training step reduces the overall computational time. The problem that has to be further investigated is how to preserve the accuracy of the results in the distributed approach.

VII. CONCLUSIONS AND FURTHER WORK

We have proposed in this paper a distributed 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 distributed reinforcement learning, so far.

We plan to extend the evaluation of the proposed distributed RL model for some large HP protein sequences, to further test its performance. We will also investigate possible improvements of the distributed RL model by improving the behavior of the local $BPFA$ agents, by using different reinforcement functions and by adding different local search mechanisms in order to increase the agents' performance.

REFERENCES

[1] T. F. Gharib, "A hybrid approach for indexing and searching protein structures," *WSEAS TRANSACTIONS on COMPUTERS*, vol. 8, pp. 966–975, 2009.
[2] H.-H. Lin and L.-Y. Tseng, "Prediction of disulfide bonding pattern based on support vector machine with parameters tuned by multiple trajectory search," *WSEAS TRANSACTIONS on COMPUTERS*, vol. 8, pp. 1429–1439, 2009.
[3] J. Wu, "An advanced hybrid machine learning approach for assessment of the change of gait symmetry," *WSEAS TRANSACTIONS on COMPUTERS*, vol. 8, pp. 1522–1532, 2009.
[4] Y. C. Alicia Tang, S. M. Zain, and N. A. Rahman, "Design and development of a qualitative simulator for learning organic reactions," *INTERNATIONAL JOURNAL OF COMPUTERS*, vol. 3, pp. 96–103, 2009.
[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
[6] G. Czibula, M. Bocicor, and I. Czibula, "A reinforcement learning model for solving the folding problem," *International Journal of Computer Technology and Applications*, vol. 2, pp. 171–182, 2011.

[7] G. Czibula, M. Bocicor, and I. G. Czibula, “An experiment on protein structure prediction using reinforcement learning,” *Studia Babes-Bolyai Informatica*, vol. LVI, p. to appear, 2011.

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

[9] C. B. Anfinsen, “Principles that govern the folding of protein chains,” *Science*, vol. 181, pp. 223–230, 1973.

[10] K. Dill and K. Lau, “A lattice statistical mechanics model of the conformational sequence spaces of proteins,” *Macromolecules*, vol. 22, pp. 3986–3997, 1989.

[11] B. Berger and T. Leighton, “Protein folding in HP model is NP-complete,” *Journal of Computational Biology*, vol. 5, pp. 27–40, 1998.

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

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

[14] 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*, vol. 35, 2008.

[15] R. Unger and J. Moult, “Genetic algorithms for protein folding simulations,” *Mol. Biol.*, vol. 231, pp. 75–81, 1993.

[16] W. Zhang and T. G. Dietterich, “Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling,” *Journal of Artificial Intelligence Reseach*, vol. 1, pp. 1–38, 2000.

[17] C. Chira, “Hill-climbing search in evolutionary models for protein folding simulations,” *Studia Informatica*, vol. LV, pp. 29–40, 2010.

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

[19] L. Bortolussi, A. Dovier, and F. Fogolari, “Agent-based protein structure prediction,” *Multiagent and Grid Systems*, pp. 183–197, 2007.

[20] P. P. G. Perez, H. I. Beltran, and A. Rojo-Dominguez, “Multi-agent systems applied in the modeling and simulation of biological problems: A case study in protein folding,” *World Academy of Science, Engineering and Technology*, pp. 128–138, 2009.

[21] L. J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” *Machine Learning*, vol. 8, pp. 293–321, 1992.

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

[23] 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*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 726–731.

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

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

[26] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] C. Claus and C. Boutilier, “The dynamics of reinforcement learning in cooperative multiagent systems,” *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 746–752, 1998.

[28] M. Tan, “Multi-agent reinforcement learning: Independent vs. cooperative agents,” pp. 487–494, 1997.

[29] M. Lauer and M. Riedmiller, “An algorithm for distributed reinforcement learning in cooperative multi-agent systems,” *Proceedings of International Conference on Machine Learning (ICML)*, pp. 535–542, 2000.

[30] C. E. Mariano and E. Morales, “A new distributed reinforcement learning algorithm for multiple objective optimization problems,” *Lecture Notes In Artificial Intelligence*, pp. 290–299, November 2000.

[31] T. Gonzaga, C. Bentes, R. Farias, M. C. Castro, and A. C. Garcia, “Using distributed-shared memory mechanisms for agents communication in a distributed system,” in *Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 39–46.

[32] P. Dayan and T. Sejnowski, “TD(Lambda) converges with probability 1,” *Mach. Learn.*, vol. 14, pp. 295–301, 1994.