A Distributed Q-Learning Approach to Fragment Assembly

Maria-Iuliana Bocicor, Gabriela Czibula, István Gergely Czibula

Babeş-Bolyai University,
1, M. Kogălniceanu Street, Cluj-Napoca, 400084, Romania,
{iuliana, gabis, istvanc}@cs.ubbcluj.ro

Abstract: The process of DNA sequencing has nowadays become of great importance in basic biology research, as well as in various fields such as medicine, biotechnology or forensic biology. The fragment assembly problem is a very complex optimization problem that deals with sequencing of DNA, and many computational techniques including computational intelligence algorithms were designed for finding good solutions for this problem. Since DNA fragment assembly is a crucial part of any sequencing project, researchers are still focusing on developing better assemblers. We are introducing in this paper a distributed reinforcement learning based approach for solving the fragment assembly problem, an NP-complete optimization problem that attempts to reconstruct the original DNA sequence from a large number of fragments, each several hundred base-pairs long. 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, DNA fragment assembly.

1. Introduction

The process of DNA sequencing has nowadays become of great importance in basic biology research, as well as in various fields such as medicine, biotechnology or forensic biology. Several techniques have been developed to achieve the DNA sequencing, but the main problem with the current technology is that it cannot read an entire genome at once, not even more than 1000 bases.

The DNA fragment assembly (FA) is a technique that attempts to reconstruct the original DNA sequence from a large number of fragments, each several hundred base-pairs long [1]. It is an NP-hard combinatorial optimization problem [2] which is growing in importance and complexity as more research centers become involved on sequencing new genomes [3]. Various heuristics, including computational intelligence algorithms, have been designed for solving the fragment assembly problem, but since this problem is a crucial part of any sequencing project, better assemblers are needed [3].

In this paper we aim at proposing a distributed reinforcement learning based model for solving the DNA Fragment Assemby problem. Reinforcement Learning (RL) [4] is an approach to machine intelligence in which an agent [5] can learn to behave in a certain way by receiving punishments or rewards on its chosen actions.

The model proposed in this paper extends toward a distributed approach the reinforcement learning based model that we have previously introduced in [6] for solving the FA problem. To our knowledge, except for the ant [7] based approaches, the DNA Fragment Assembly problem has not been addressed in the literature using distributed reinforcement learning, so far.

The rest of the paper is organized as follows. Section 2 presents the DNA fragment assembly problem and Section 3 briefly describes existing approaches in solving the considered problem. The fundamentals of distributed reinforcement learning are given in Section 4. Section 5 introduces the distributed reinforcement learning model that we propose for solving the fragment assembly problem. An experimental evaluation of the proposed approach is given in Section 6, and Section 7 provides an analysis of the introduced distributed model, emphasizing its advantages and drawbacks. Section 8 contains some conclusions of the paper and future development of our work.

2. The Fragment Assembly Problem

The Fragment Assembly (FA) problem deals with sequencing of DNA [3]. In order to sequence larger strands of DNA, they are first broken into smaller pieces. The FA problem is then to reconstruct the original molecules sequence from the smaller fragment sequences [3].

The genome of all living organisms, encoded in the DNA, represents the totality of their hereditary information. DNA, or the

deoxyribonucleic acid is composed of complex organic molecules and is the information-bearing molecule in the cell – it stores information that contains the instructions needed to construct other components of the cell such as proteins and RNA, and eventually, entire organisms.

Determining the order of nucleotide bases, or the process of DNA sequencing, has nowadays become of great importance in basic biology research, as well as in various fields such as medicine, biotechnology or forensic biology. Several techniques have been developed to achieve the DNA sequencing, but the main problem with the current technology is that it cannot read an entire genome at once, not even more than 1000 bases. As even the simplest organisms (such as viruses or bacteria) have much longer genomes, the need to develop methods that would overcome this limitation arose. One of these, called shotgun sequencing was introduced in 1982, by Fred Sanger [8] and it consists of the next steps: first, several copies of the DNA molecule are created; then each of the copies is cut at random sites in order to obtain molecules short enough to be sequenced directly - fragments; the last and most difficult step involves assembling these molecules back into the original DNA molecule, based on common subsequences of fragments.

The DNA Fragment Assembly Problem specifically refers to this last step. Usually, there are three phases that must be followed in order to obtain a solution for this problem [1]:

- *Overlap Phase* - consists of finding the longest match between the prefix of a fragment and the suffix of another. All possible pairs of fragments are compared, in order to determine their similarity.
- *Layout Phase* - refers to finding the order of fragments. This is the most difficult step, because of various challenges:

1. **Unknown orientation**: the fragments' orientation is lost, so if for a specific fragment there are no overlapping fragments, there might be some that are similar with the fragment's reverse.
2. **Base call errors**: these are errors that might appear during the experimental phase. There are three types of such errors: substitutions (a certain nucleotide base is subtituted by another), insertions (a new nucleotide base is inserted to a fragment it does not belong to) and deletions (certain nucleotide bases are deleted from fragments. These errors affect the overlaps between the fragments).
3. **Incomplete coverage**: The given set of fragments cannot form the orginial DNA.
4. **Repeated regions**: some subsequences may be repeated several times in the original DNA. These repeats can cause serious problems in the assembly process.
5. **Chimeras**: these occur when two fragments that are not adjacent in the original molecule join together in one fragment.

- *Consensus Phase* - consists of determining the original DNA molecule from the layout obtained in the previous phase.

Next, we will illustrate the assembly process, by using a simple example, taken from [9]. Let us assume that, for the DNA sequence *TTACCGTGC* we are given the set of fragments:

F_1 = ACCGT
F_2 = CGTGC
F_3 = TTAC
F_4 = TACCGT

First we need to determine the overlap of each fragment with the other three fragments. This is usually done using an alignment algorithm and a similarity measure. Then, we need to find the order of these fragments, based on the computed similarities. The order is: $F_3F_4F_1F_2$.

$F_3 \rightarrow$	T	T	A	C	–	–	–	–	–
$F_4 \rightarrow$	–	T	A	C	C	G	T	–	–
$F_1 \rightarrow$	–	–	A	C	C	G	T	–	–
$F_2 \rightarrow$	–	–	–	–	C	G	T	G	C
	T	T	A	C	C	G	T	G	C

3. Literature Review

Many algorithms have already been developed in the literature to solve the *DNA fragment assembly* problem.

The FA problem is known to be NP-complete [10], therefore exact solutions for this problem are very difficult to obtain. For this reason, various heuristic methods have been applied: genetic algorithms, clustering algorithms, ant colony optimization algorithms.

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

3.1 Fragment assembly

Kikuchi and Chakraborty approach the Fragment Assembly Problem using an improved genetic algorithm [11]. A chromosome is represented as a permutation of the DNA fragments' labels (numbers from 1 to N, where N is the total number of fragments), while the fitness function is represented as the sum of similarity measures of all adjacent fragments. The authors add two new methods to their genetic algorithm, in order to improve its efficiency: The Chromosome Reduction Step - used to make the search more efficient (the fragments which are contained within contigs formed in the best chromosome are deleted from all the other chromosomes, thus decreasing their lengths); The Chromosome Refinement Step - greedy heuristic to locally improve the fitness of chromosomes (if the beginning of fragment *i* overlaps with the end of fragment *i*+1 in the chromosome, then these fragments are swapped).

A four-phase approach is proposed in [12] in order to reconstruct a DNA sequence from fragments. The authors firstly construct an overlap graph, whose vertices contain the fragments and whose edges contain the approximate overlaps between every pair of fragments. Secondly, they try to find an oriented subgraph, by assigning an orientation to each fragment, in order to eliminate some overlaps and retain the possibility of using others. As this phase is NP-complete, the authors proposed a new greedy approximation algorithm that computes an optimal orientation. The third phase consists of selecting a set of edges from an oriented subgraph determined in the previous phase to induce a consistent layout of the oriented fragments. The final step is to determine the original DNA sequence by merging the selected overlaps into a multiple sequence alignment and voting on a consensus.

Parsons, Forrest and Burks [13] argue that the FA problem is quite similar to the Travelling Salesman Problem (TSP) and that genetic algorithms could provide an efficient approach. A solution is represented as a permutation of integer numbers, which represent fragments, and where each two successive fragments overlap. The authors use two fitness functions, one that simply sums the overlap scores over all adjacent fragments - which has to be maximized and another one, that considers the overlap strength among all possible pairs, penalizing layouts in which fragments with strong overlap are far apart – which has to be minimized. New genetic operators that try to preserve or extend existing contigs are defined: order crossover, edge recombination, inversion and transposition.

Angeleri et al. [14] presents a supervised learning approach to the fragment assembly problem. The idea is to train a recurrent neural network to track a sequence of bases constituting a given fragment and to assign to the same cluster all sequences which are well tracked by this network.

3.2 Distributed fragment assembly

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.

Luque, Alba and Khuri approach the DNA Fragment Assembly problem using a sequential and also a distributed genetic algorithm [15]. The distributed method is implemented using a parallel genetic algorithm, which has multiple component genetic algorithms, each of these having a single population and being able to communicate results to the other component algorithms. The authors used a descentralized distributed search, by developing a parallel implementation, where separate subpopulations evolve independently and exchange individuals among them, with a certain frequency and following certain rules. Compared to the sequential approach, the distributed one obtained high accuracy solutions for large DNA sequences, in lower waiting times.

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. An ant colony system algorithm was used by Meksangsouy and Chaiyaratana [16] in order to solve the FA problem. The solution is cooperatively generated by all ants in the colony and the

performance measur is determined as the sum of the overlap scores calculated for each successive pair of adjacent fragments in the final layout. The authors investigated two types of assembly problems: single-contig and multiple-contig problems, obtaining very good results, especially for the latter.

Another approach that uses an ant colony system combined with a nearest neighbor search heuristic is proposed in [17]. The assembly process is composed of two phases: fragment assembly and contig assembly. During the first stage, the ant colony optimization algorithm is used to determine alignments between fragments and subsequently contiguous sequences (contigs), using possible orders of fragments. The second phase consists of assembling together the contigs obtained in the previous stage, using the nearest neighbor rule.

4. Distributed Reinforcement Learning

Reinforcement learning is a synonym of learning by interaction [18]. 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.

Recently 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 [19]. 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 [19], 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 [20] 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 [21] 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 somedrawbacks, as sharing knowledge comes with a communication cost and cooperative behavior for joint tasks automatically implies a larger state space.

In [22], 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 [23] 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.

5. A Distributed Reinforcement Learning Model for Solving the DNA Fragment Assembly Problem

5.1 Background

In this section we present the reinforcement learning model that we have previously introduced in [6] for solving the DNA Fragment Assembly problem.

Let us consider, in the following, that *Seq* is a DNA sequence and $F_1, F_2, ..., F_n$ $(n > 1)$ is a set of fragments. As indicated in Section 2, the *FA* problem consists of determining the order in which these fragments have to be assembled back into the original DNA molecule, based on common subsequences of fragments. Consequently, the *FA* problem can be viewed as the problem of generating a permutation σ of $\{1,2,...,n\}$ that optimizes the performance of the alignment $F_\sigma = (F_{\sigma_1}, F_{\sigma_2}, ..., F_{\sigma_n})$. The performance measure *PM* we consider in this paper is one of the fitness functions defined in [13], which sums the overlap scores over all adjacent fragments and which has to be maximized.

According to [13], the performance measure PM for the sequence of fragments $F_\sigma = (F_{\sigma_1}, F_{\sigma_2}, ..., F_{\sigma_n})$ is defined as in Equation (1):

$$PM(F_\sigma) = \sum_{i=1}^{n-1} w(F_{\sigma_i}, F_{\sigma_{i+1}}) \quad (1)$$

where *w(a, b)* denotes the similarity measure between sequences *a* and *b*.

We define the RL task associated to the FA problem as follows:

- The state space *S* (the agent's environment) will consist of $\frac{n^{n+1}-1}{n-1}$ states, i.e. $S = \{s_1, s_2, ..., s_{\frac{n^{n+1}-1}{n-1}}\}$. The initial state of the agent in the environment is s_1. A state $s_{i_k} \in S\,(i_k \in \left[1, \frac{n^{n+1}-1}{n-1}\right])$ reached by the agent at a given moment after it has visited states $s_1, s_{i_1}, s_{i_2}, ..., s_{i_n}$ and has selected

actions $a_{i_1}, a_{i_2}, ..., a_{i_n}$ is a terminal (final or goal) state if the number of states visited by the agent in the current sequence is $n + 1$ and all the actions are distinct, i.e $a_{i_j} \neq a_{i_k}, \forall 1 \leq j,k \leq n, j \neq k$.

- The action space A consists of n actions available to the problem solving agent and corresponding to the n possible values $1,2,...,n$ used to represent a solution (permutation of $\{1,2,...,n\}$), i.e. $A = \{a_1, a_2, ..., a_n\}$, where $a_i = i$, $\forall 1 \leq i \leq n$.
- The transition function $\delta : S \to P(S)$ between the states is defined as in Formula (2).

$$\delta(s_j) = \bigcup_{k=1}^{n} \Delta(s_j, a_k) \; \forall j, 1 \leq j \leq \frac{n^n - 1}{n-1}, \qquad (2)$$

where $\Delta(s_j, a_k) = s_{n \cdot j - n + 1 + k} \; \forall k \in [1, n]$.

This means that, at a given moment, from a state $s \in S$ the agent can move in n successor states, by executing one of the n possible actions. We say that a state $s' \in S$ that is accessible from state s, i.e. $s' \in \bigcup_{a \in A} \Delta(s, a)$, 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 (3)).

Let us consider a path π in the above defined environment from the initial to a final state, $\pi = (\pi_0 \pi_1, \pi_2, ..., \pi_n)$, where $\pi_0 = s_1$ and $\forall 0 \leq k \leq n-1$ 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 $a_\pi = (a_{\pi_0} a_{\pi_1}, a_{\pi_2}, ..., a_{\pi_{n-1}})$, where $\pi_{k+1} = \Delta(\pi_k, a_{\pi_k}), \forall 0 \leq k \leq n-1$. The sequence a_π will be referred as the *action configuration* associated to the path π. A path π is called *valid* if all the actions within its *action configuration* are distinct, i.e $a_{\pi_j} \neq a_{\pi_k}, \forall 1 \leq j,k \leq n, j \neq k$. The action configuration a_π associated to a valid path π can be viewed as a possible order for the fragments assembly process, i.e a permutation that gives the assembly order (the fragments are assembled in the order $a_{\pi_0} a_{\pi_1}, a_{\pi_2}, ..., a_{\pi_{n-1}}$). Consequently we can associate to a valid path π a value denoted by $PM(F_{a_\pi})$ representing the performance measure (see Equation (1)) of the alignment $F_{a_\pi} = (F_{a_{\pi_0}} F_{a_{\pi_1}}, F_{a_{\pi_2}}, ..., F_{a_{\pi_{n-1}}})$.

The FA problem formulated as a RL problem will consist in training the agent to find a path π from the initial to a final state having the maximum associated performance measure $PM(F_{a_\pi})$. It is known that the estimated utility of a state [19] 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 *valid* path π having the maximum associated performance measure, we define the reinforcement function as follows (Formula (3)):

- the reward received after a transition to a non terminal state is τ, where τ is a small positive constant (e.q 0.1);
- the reward received after a transition to a final state π_n after states $\pi_0 = s_1, \pi_1, \pi_2, ..., \pi_{n-1}$ were visited is the performance measure *PM* of the alignment $F_{a_\pi} = (F_{a_{\pi_0}} F_{a_{\pi_1}}, ..., F_{a_{\pi_{n-1}}})$ (see Equation (1)).

$$r(\pi_k \mid s_1, \pi_1, \pi_2, ..., \pi_{k-1}) = \begin{cases} PM(F_{a_\pi}), k = n \\ \tau \quad otherwise \end{cases} \qquad (3)$$

where by $r(\pi_k \mid s_1, \pi_1, \pi_2, ..., \pi_{k-1})$ we denote the reward received by the agent in state π_k, after its history in the environment is $\pi_0 = s_1, \pi_1, \pi_2, ..., \pi_{k-1}$.

Considering the reward defined in Formula (3), 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 path π that maximizes the performance of the associated alignment.

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 *FA* (*Fragment Assembly*) agent, we propose a Q-learning approach [4]. The idea of the training process is the following:

- The Q-values are initialized with 0.
- During some training episodes, the agent will experiment (using the ε*-Greedy* action selection mechanism) some (possible optimal) valid paths from the initial to a final state, updating the *Q*-values estimations according to the *Q-learning* algorithm [24].
- During the training process, the *Q*-values estimations converge to their exact values, thus, at the end of the training rocess, 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 nitial state and following the *Greedy* mechanism until a solution is reached. From a given state *i*, using the *Greedy* policy, he agent transitions to a neighbor *j* of *i* having the maximum *Q*-value. Consequently, the solution of the *FA* problem reported y the RL agent is a path $\pi = (s_1\pi_1, \pi_2, ..., \pi_n)$ 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 or all optimal policies, according to the way it was designed.

It is proven in [25] that the learned *Q*-values converge to their optimal values as long as all state-action pairs are visited an infinite number of times. Consequently, the action configuration a_π corresponding to the path π learned by the *FA* agent converges, in the limit, to the optimal order in which the fragments have to be assembled, indicating the alignment $F_{a_{\pi_0}} F_{a_{\pi_1}}, ..., F_{a_{\pi_{n-1}}}$ having the maximum associated performance measure.

5.2 Our distributed approach proposal

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 5.1. 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 [24].

We have two types of agents in our distributed architecture:

- *FAA* (*Fragment Assembly Agents*). Each FAA agent runs in a separate process or thread and is trained using the *Q*-learning algorithm [26]. Each local agent performs local *Q*-values estimations updates from its own point of view.
- a *FAS* (*Fragment Assembly Supervisor*) agent wich supervises the learning process and synchronizes the computations of the individual *FAA* agents. It keeps a blackboard [27] which stores the global *Q*-values estimations. The local *FAA* agents use the global *Q*-values estimations stored in the blackboard and comunicate to the *FAS* agent their intention to update a *Q*-value estimation. If a local agent tries to update a certain *Q*-value, the *FAS* agent will update the global *Q*-value estimation only if the new estimation received from the local agent is greater than the *Q*-value estimation existing in the blackboard. This updating strategy is based on our previous result from [26] 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 multiagent system:

1. The *FAA* agents are running in the same process with the supervisor agent *FAS*. In this case each *FAA* agent has an instance of the supervisor *FAS* agent, and this will reduce the cost of communication (messages exchanges) between the agents. The blackboard stored by *FAS* allows a kind of indirect data communication between the local agents and the global supervising agent.

2. The *FAA* agents are distributed across multiple processes/machines. In this case network communication is involved when the *FAA* agents ask the supervisor for *Q*-values or request a *Q*-value update. In order to reduce the number of messages exchanged between *FAA* agents and the supervisor *FAS*, the frequency for updating the *Q*-values in the blackboard can be decreased if the *FAA* 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 *FAS* supervisor agent initializes with 0 the *Q*-values from the blackboard.

Phase 2. Training phase of each *FAA* agent

During some training episodes, the individual FAA agents will experiment (using the ε-Greedy action selection mechanism) some (possible optimal) valid paths from the initial to a final state, updating the *Q*-values estimations according to the *Q-learning* algorithm [24].

Repeat (for each episode)
  Select the initial state *s*.
  Choose action *a* from *s* using policy
    derived from *Q* (ε -Greedy, SoftMax [4])
  Repeat (for each step of the episode)
    Take action *a*, observe the reward *r(s, a)*
      and the next state *s'*.
    FAA **agent asks** *FAS* **agent for** *Q(s, a)*.
    FAS **agent retrieves** *Q(s, a)* **from the blackboard**.
    FAS **sends the retrieved** *Q(s ,a)* to *FAA*
    FAA agent updates the table entry *Q(s, a)*
      as follows

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

    FAA **sends the new** *Q(s, a)* **to** *FAS*.
    FAS **updates** *Q(s, a)* **in the blackboard if needed**
    $s \leftarrow s'$
  until s is terminal
Until the maximum number of episodes is
  reached or the Q-values do not change

Figure 1. The distributed *Q*-learning algorithm

The general form of the distributed *Q-learning* algorithm is given in Figure 1. We denote in the following by *Q(s, a)* the *Q*-value estimate associated to the state *s* and action *a*, as stored by the blackboard of the *FAS* agent.

Phase 3. Final phase

After the training of the multiagent system has been completed, the solution learned by the *FAS* 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 fragment assembly problem is presented in Figure 2. The message exchanges between the agents, as highlighted in Figure 1, are illustrated in Figure 2 by the labeled arrows.

6. Computational Experiment

In this section we aim at providing the reader with an easy to follow example illustrating how our approach works. The example is taken from [9] and was described in Section 2.

6.1 Example

In the following, we will present all possible permutations of fragments from the example given in the Section 2, and their associated performance measures, in order to show that our distributed approach obtains the optimal results.

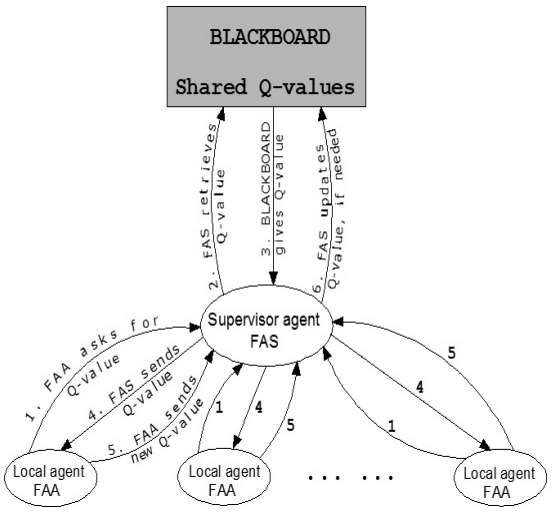

Figure 2. The distributed *Q*-learning architecture

Table 1 and Table 2 illustrate, respectively, the overlap (similarity) scores and the sub-sequence alignments for all possible pairs of fragments that can be obtained. These similarity measures and local alignments

between two fragments are obtained using the Smith-Waterman algorithm that detects a local alignment by dynamic programming (as parameters for the scoring matrix of the Smith-Waterman algorithm [28], we used: *match*=1, *mismatch*=-0.33 and *gap*=1.33).

Table 1. The similarity scores for the fragments, obtained by the Smith-Waterman algorithm [28].

	F_1	F_2	F_3	F_4
F_1	–	3.00	2.00	5.00
F_2	3.00	–	1.67	3.00
F_3	2.00	1.67	–	3.00
F_4	5.00	3.00	3.00	–

Table 2. The alignments for the fragments, obtained by the Smith-Waterman algorithm [28].

	F_1	F_2	F_3	F_4
F_1	–	CGT	AC	ACCGT
F_2	CGT	–	TGC-TAC	CGT
F_3	AC	TGC-TAC	–	TAC
F_4	ACCGT	CGT	TAC	–

Table 3 presents all the possible permutations of the 4 given fragments and the performance measure *PM* (Equation (1)) for each permutation. It can be seen that the maximum values for the performance measure *PM* are obtained in two cases: for the alignments $F_2F_1F_4F_3$ and $F_3F_4F_1F_2$. The original DNA is the one indicated by the alignment $F_2F_1F_4F_3$.

6.2 Distributed RL model and results

Let us consider the example mentioned in the previous subsection, the DNA sequence *Seq* = *TTACCGTGC* and four fragments F_1 = *ACCGT*; F_2 = *CGTGC*; F_3 = *TTAC*; F_4 = *TACCGT*, i.e. n = 4. We aim at identifying the most appropriate order in which the fragments have to be assembled back into the original sequence *Seq*. As we have presented in Section 5, the states space will consist of 341 states, i.e $S = \{s_1, s_2, ..., s_{341}\}$.

We have applied the distributed RL approach introduced in Subsection 5.1 with the following settings:

- two local *FAA* agents were used;
- the number of training episodes for each local *FAA* agent is 106;
- both local *FAA* agents have the same behaviour in the *Q*-learning scenario: they use the ε-Greeedy action selection mechanism, 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 *FAA* agent explores its search space (the ε parameter is set to 1). After the training of the *FAA* agents was completed, two solutions are reported by the *FAS* agent. The solutions are determined starting from state s_1, following the Greedy policy. Both of them are optimal, having the the maximum associated performance of 11 (see Table 3).

The learned optimal solutions are:

1. The path $\pi = (s_1 s_3 s_{10} s_{41} s_{164})$ having the associated action configuration $a_\pi = (2143)$.
2. The path $\pi = (s_1 s_4 s_{17} s_{66} s_{263})$ having the associated action configuration $a_\pi = (3412)$.

7. Discussion

Regarding the *Q*-learning approach presented in Subsection 5.1 and previously introduced in [6] for solving the DNA fragment assembly problem, we remark the following:

- The training process during an episode has a time complexity of $\theta(n)$, where n is the number of fragments considered in the assembly process. Consequently, assuming that the number of training episodes is k, the overall complexity of the algorithm for training the *FA* agent is $\theta(k \cdot n)$.
- If the number n of the fragments considered in the assembly problem is large and consequently the state space becomes very large, 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.

Table 3. All the possible permutations, with their associated performance measure *PM*

No.	Alignment	Similarity score	Similarity score	Similarity score	Scores' sum
1	$F_1F_2F_3F_4$	$w(F_1,F_2)=3.00$	$w(F_2,F_3)=1.67$	$w(F_3,F_4)=3.00$	7.67
2	$F_1F_2F_4F_3$	$w(F_1,F_2)=3.00$	$w(F_2,F_4)=3.00$	$w(F_4,F_3)=3.00$	9.00
3	$F_1F_3F_2F_4$	$w(F_1,F_3)=2.00$	$w(F_3,F_2)=1.67$	$w(F_2,F_4)=3.00$	6.67
4	$F_1F_3F_4F_2$	$w(F_1,F_3)=2.00$	$w(F_3,F_4)=3.00$	$w(F_4,F_2)=3.00$	8.00
5	$F_1F_4F_2F_3$	$w(F_1,F_4)=5.00$	$w(F_4,F_2)=3.00$	$w(F_2,F_3)=1.67$	9.67
6	$F_1F_4F_3F_2$	$w(F_1,F_4)=5.00$	$w(F_4,F_3)=3.00$	$w(F_3,F_2)=1.67$	9.67
7	$F_2F_1F_3F_4$	$w(F_2,F_1)=3.00$	$w(F_1,F_3)=2.00$	$w(F_3,F_4)=3.00$	8.00
8	$F_2F_1F_4F_3$	$w(F_2,F_1)=3.00$	$w(F_1,F_4)=5.00$	$w(F_4,F_3)=3.00$	11.00
9	$F_2F_3F_1F_4$	$w(F_2,F_3)=1.67$	$w(F_3,F_1)=2.00$	$w(F_1,F_4)=5.00$	8.67
10	$F_2F_3F_4F_1$	$w(F_2,F_3)=1.67$	$w(F_3,F_4)=3.00$	$w(F_4,F_1)=5.00$	9.67
11	$F_2F_4F_1F_3$	$w(F_2,F_4)=3.00$	$w(F_4,F_1)=5.00$	$w(F_1,F_3)=2.00$	10.00
12	$F_2F_4F_3F_1$	$w(F_2,F_4)=3.00$	$w(F_4,F_3)=3.00$	$w(F_3,F_1)=2.00$	8.00
13	$F_3F_1F_2F_4$	$w(F_3,F_1)=2.00$	$w(F_1,F_2)=2.00$	$w(F_2,F_4)=3.00$	8.00
14	$F_3F_1F_4F_2$	$w(F_3,F_1)=2.00$	$w(F_1,F_4)=5.00$	$w(F_4,F_2)=3.00$	10.00
15	$F_3F_2F_1F_4$	$w(F_3,F_2)=1.67$	$w(F_2,F_1)=2.00$	$w(F_1,F_4)=5.00$	8.67
16	$F_3F_2F_4F_1$	$w(F_3,F_2)=1.67$	$w(F_2,F_4)=3.00$	$w(F_4,F_1)=5.00$	8.67
17	$F_3F_4F_1F_2$	$w(F_3,F_4)=3.00$	$w(F_4,F_1)=5.00$	$w(F_1,F_2)=3.00$	11.00
18	$F_3F_4F_2F_1$	$w(F_3,F_4)=3.00$	$w(F_4,F_2)=3.00$	$w(F_2,F_1)=3.00$	9.00
19	$F_4F_1F_2F_3$	$w(F_4,F_1)=5.00$	$w(F_1,F_2)=3.00$	$w(F_2,F_3)=1.67$	9.67
20	$F_4F_1F_3F_2$	$w(F_4,F_1)=5.00$	$w(F_1,F_3)=2.00$	$w(F_3,F_2)=1.67$	8.67
21	$F_4F_2F_1F_3$	$w(F_4,F_2)=3.00$	$w(F_2,F_1)=3.00$	$w(F_1,F_3)=2.00$	8.00
22	$F_4F_2F_3F_1$	$w(F_4,F_2)=3.00$	$w(F_2,F_3)=1.67$	$w(F_3,F_1)=2.00$	6.67
23	$F_4F_3F_1F_2$	$w(F_4,F_3)=3.00$	$w(F_3,F_1)=2.00$	$w(F_1,F_2)=3.00$	8.00
24	$F_4F_3F_2F_1$	$w(F_4,F_3)=3.00$	$w(F_3,F_2)=1.67$	$w(F_2,F_1)=3.00$	7.67

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.

8. Conclusions and Further Work

We have proposed in this paper a distributed reinforcement learning based model for solving the DNA fragment assembly problem. To our knowledge, except for the ant based approaches, the fragment assembly problem has not been addressed in the literature using distributed reinforcement learning, so far. We have emphasized the potential of our proposal by evaluating it on a simple case study, highlighting its advantages and drawbacks.

We plan to extend the evaluation of the proposed distributed RL model for some larger instances, to further test its performance.

We will also investigate possible improvements of the distributed RL model by improving the behavior of the local *FAA* agents, by using different reinforcement functions and by adding different local search mechanisms in order to increase the agents' performance.

Acknowledgements

This work was partially supported by CNCSIS - UEFISCDI, project number PNII - IDEI 2286/2008. The work was also possible with the partial financial support of the Sectorial Operational Programme for Human Resources Development 2007-2013, co-financed by the European Social Fund, under the project number POSDRU/107/1.5/S/ 76841 with the title "Modern Doctoral Studies: Internationalization and Interdisciplinarity".

REFERENCES

1. LI, L., S. KHURI, **A Comparison of DNA Fragment Assembly Algorithms**, in Proc. of the Intl. Conf. on Mathematics and Engineering Techniques in Medicine and Biological Sciences. CSREA Press, 2004, pp. 329-335.
2. TALMACIU, M., E. NECHITA, **Some Combinatorial Optimization Problems for Weak-bisplit Graphs**, Studies in Informatics and Control, vol. 19, no. 4, 2010, pp. 427-434.
3. HASSANIEN, A. E., M. G. MILANOVA, T. G. SMOLINSKI, A. ABRAHAM, **Computational Intelligence in Solving Bioinformatics Problems: Reviews,**

Perspectives, and Challenges, in Computational Intelligence in Biomedicine and Bioinformatics, 2008, pp. 3-47.

4. SUTTON, R. S., A. G. BARTO, **Reinforcement Learning: An Introduction,** MIT Press, 1998.
5. SUSNEA, I., G. VASILIU, A. FILIPESCU, A. RADASCHIN, **Virtual Pheromones for Real-time Control of Autonomous Mobile Robots**, Studies in Informatics and Control, vol. 18, no. 3, 2009, pp. 233-240.
6. BOCICOR, M., G. CZIBULA, I. G. CZIBULA, **A Reinforcement Learning Approach for Solving the Fragment Assembly Problem**, SYNASC 2011 - 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. IEEE Computer Society, 2011, submitted.
7. DORIGO, M., T. STŰTZLE, **Ant Colony Optimization**, Scituate, MA, USA: Bradford Company, 2004.
8. SANGER, F., A. COULSON, G. HONG, D. F. HILL, G. PETERSEN, **Nucleotide Sequence of Bacteriophage Lambda DNA**, J. Molecular Biology, vol. 162, no. 4, 1982, pp. 729-773.
9. KOSTERS, W., **Bioinformatics: Fragment Assembly**, IPA–Algorithms and Complexity - course, 2007. [Online]. Available: http://www.liacs.nl/kosters/bio/bio.pdf.
10. PEVZNER, P. A., **Computational Molecular Biology: An Algorithmic Approach**, MIT Press, 2000.
11. KIKUCHI, S., G. CHAKRABORTY, **Heuristically Tuned GA to Solve Genome Fragment Assembly Problem**, IEEE CEC, 2006, pp. 1491-1498.
12. KECECIOGLU, J. D., E. W. MYERS, **Combinatiorial Algorithms for DNA Sequence Assembly**, Algorithmica, vol. 13, no. 1/2, 1995, pp. 7–51.
13. PARSONS, R. J., S. FORREST, C. BURKS, **Genetic Algorithms, Operators, and DNA Fragment Assembly**, in Machine Learning. Kluwer Academic Publishers, 1995, pp. 11-33.
14. ANGELERI, E., B. APOLLONI, D. DE FALCO, L. GRANDI, **DNA Fragment Assembly using Neural Prediction Techniques**, Intl. Journal Neural Systems, vol. 9, no. 6, 1999, pp. 523-544.
15. LUQUE, G., E. ALBA TORRES, S. KHURI, **Assembling DNA Fragments with a Distributed Genetic Algorithm**, Parallel Computing for Bioinformatics and Computational Biology, 2006, pp. 285-302.
16. MEKSANGSOUY, P., N. CHAIYARATANA, **DNA Fragment Assembly using an Ant Colony System Algorithm**, in Proceedings of CEC'03 - vol.3. IEEE Press, 2003, pp. 1756-1763.
17. WETCHARAPORN, W., N. CHAIYARATANA, S. TONGSIMA, **DNA Fragment Assembly by Ant Colony and Nearest Neighbour Heuristics**, Artificial Intelligence and Soft Computing ICAISC 2006, pp. 1008-1017.
18. PEREZ-URIBE, A., **Introduction to Reinforcement Learning**, 1998, http://lslwww.epfl.ch/~anperez/RL/RL.html.
19. RUSSELL, S., P. NORVIG, **Artificial Intelligence - A Modern Approach**, ser. Prentice Hall International Series in Artificial Intelligence. Prentice Hall, 2003.
20. CLAUS, C., C. BOUTILIER, **The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems**, In Proceedings of the Fifteenth National Conference on Artificial Intelligence, 1998, pp. 746-752.
21. TAN, M., **Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents**, 1997, pp. 487-494.
22. LAUER, M., M. RIEDMILLER, **An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-agent Systems**, Proceedings of International Conference on Machine Learning (ICML), 2000, pp. 535-542.
23. MARIANO, C. E., E. MORALES, **A New Distributed Reinforcement Learning Algorithm for Multiple Objective Optimization Problems**, Lecture Notes In Artificial Intelligence, November 2000, pp. 290-299.

24. DAYAN, P., T. SEJNOWSKI, **TD(Lambda) Converges with Probability 1**, Machine Learning., vol. 14, 1994, pp. 295-301.

25. WATKINS, C. J. C. H., P. DAYAN, **Q-learning**, Machine Learning, vol. 8, no. 3-4, 1992, pp. 279-292.

26. CZIBULA, G., M. BOCICOR, I. CZIBULA, **A Reinforcement Learning Model for Solving the Folding Problem**, International Journal of Computer Technology and Applications, vol. 2, 2011, pp. 171-182.

27. GONZAGA, T., C. BENTES, R. FARIAS, M. C. CASTRO, A. C. GARCIA, **Using Distributed-shared Memory Mechanisms for Agents Communication in a Distributed System**, in Proc. of the 7th Intl. Conf. on Intelligent Systems Design and Applications. USA, IEEE Comp. Society, 2007, pp. 39-46.

28. SMITH, T., M. WATERMAN, **Identification of Common Molecular Subsequences**, Journal of Molecular Biology, vol. 147, no. 1, 1981, pp. 195-197.