STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 2, 2013

A COMPARISON OF REINFORCEMENT LEARNING BASED MODELS FOR THE DNA FRAGMENT ASSEMBLY PROBLEM

GABRIELA CZIBULA, ISTVAN-GERGELY CZIBULA, AND IULIANA M. BOCICOR[(1)]

Abstract. The *DNA fragment assembly* is a very complex optimization problem important within many fields, such as bioinformatics, computational biology or medicine. The problem is *NP-hard*, that is why many computational techniques, including computational intelligence algorithms, were designed to find good solutions for this problem. This paper is intended to present and investigate two reinforcement learning based models for solving the *DNA fragment assembly* problem. We provide an experimental comparison of these two models, that will study the obtained performances of the reinforcement learning based approaches, by using different action selection policies, with variable parameters.

1. Introduction

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. The main problem with the current sequencing technology is that it cannot read an entire genome at once, not even more than 1000 nucleobases.

The *DNA fragment assembly (FA)* refers to reconstructing the original DNA sequence from a large number of fragments, each several hundred nucleobases long, based on common subsequences of fragments. It is an NP-hard combinatorial optimization problem, growing in importance and complexity as more research centers become involved in sequencing new genomes [5].

Received by the editors: April 12, 2013.

2010 *Mathematics Subject Classification.* 68P15, 68T05.

1998 *CR Categories and Descriptors.* I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*; I.2.8[**Computing Methodologies**]: Problem Solving, Control Methods, and Search – *Heuristic methods*.

Key words and phrases. Bioinformatics, DNA fragment assembly, reinforcement learning, Q-learning.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

Reinforcement Learning (RL) [14] is an approach to machine intelligence in which an agent [13] can learn to behave in a certain way by receiving punishments or rewards for its chosen actions. 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 this paper we aim at investigating two reinforcement learning based models for solving the problem of DNA fragment assembly. One of these models, which was previously proposed in [2], is improved in this study. Moreover, this paper introduces a second reinforcement learning based model for the same problem and compares the two approaches.

The rest of the paper is organized as follows. Section 2 introduces the *DNA fragment assembly problem* as well as some main aspects related to *reinforcement learning*. The reinforcement learning models that we propose for solving the FA problem are detailed in Section 3. Experimental evaluations, analysis and comparisons of the algorithms are given in Section 4. Section 5 outlines some conclusions of the paper and indicates future research directions.

2. Background

This section briefly presents the *DNA FA problem* as well as some fundamental aspects related to *reinforcement learning*.

2.1. The DNA Fragment Assembly Problem.

Determining the order of nucleotide bases (molecules composing the DNA), or the process of DNA sequencing, has nowadays become of great importance in basic biology research, as well as in fields such as medicine, biotechnology or forensic biology. The main problem with the current sequencing 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 [11] 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, based on common subsequences of fragments. The DNA FA problem specifically refers to this last step. For more details, we refer the reader to [2].

2.2. Reinforcement Learning. Background.

Reinforcement Learning [8] is an approach to machine intelligence that combines two disciplines to solve successfully problems that neither discipline can address individually: *Dynamic programming* and *Supervised learning*. RL is a synonym of learning by

interaction [10]. 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.

An important aspect in reinforcement learning is maintaining an equilibrium between *exploitation* and *exploration* [15]. The agent should accumulate a lot of reward, by choosing the best experienced actions, but at the same time it must explore its environment, by trying new actions. In this study we use three policies for selecting actions: the *ϵ-greedy policy* [14], the *softmax policy* [14] and an *intelligent ϵ-Greedy based action selection mechanism*, that we have introduced in [3], which uses a one step look-ahead procedure in order to better guide the exploration of the agent through the search space.

3. Reinforcement Learning Based Models for the DNA Fragment Assembly Problem

In this section we introduce the two reinforcement learning models proposed for solving the DNA FA problem. First, we present some concepts and notations that will apply to both models.

A general reinforcement learning task is characterized by four components: a *state space* $\mathcal{S}$ specifying all possible configurations of the system; an *action space* $\mathcal{A}$ listing all available actions for the learning agent; a *transition function* δ specifying the possibly stochastic outcomes of taking each action in any state; a *reward function* defining the possible reward of taking each of the actions.

Let us consider that $\mathcal{S}eq$ is a DNA sequence and $F_1, F_2, \ldots, F_n$ is a set of fragments. As indicated in Subsection 2.1, 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, from a computational perspective, as the problem of generating a permutation σ of $\{1, 2, \ldots, n\}$ that optimizes the performance of the alignment $F_\sigma = (F_{\sigma_1}, F_{\sigma_2}, \ldots, F_{\sigma_n})$ $(n > 1)$. The performance measure PM we consider in this paper is one of the fitness functions defined in [9], which sums the overlap scores over all adjacent fragments and has to be maximized. According to [9], the performance measure PM for the sequence of fragments $F_\sigma = (F_{\sigma_1}, F_{\sigma_2}, \ldots, F_{\sigma_n})$ is defined as in Equation (1):

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

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

3.1. **Path Finding Model.** We have previously introduced a reinforcement learning based model for the FA problem [2]. In the rest of the paper, it will be referred to as the *path finding model*. The RL task associated to the FA problem consists in training the agent to find a path from the initial to a final state having the maximum associated overall similarity. During the training step of the learning process the learning agent determines its *optimal policy* in the environment, i.e. the mapping from states to actions that maximizes the sum of the received rewards. The equivalent *action configuration* is viewed as a permutation that gives the optimal alignment of the DNA fragments. As the goal is to find a path having the maximum value of the performance measure, the reinforcement function rewards the agent with a small value (e.g. 0.1) for each transition to an non terminal state and with the performance measure of the found alignment, after a transition to a final state [2]. For training the FA agent [2] we used a Q-learning approach [14], in conjunction with an ϵ-Greedy action selection policy. For more details about the definitions of the state and action spaces or reward and transition functions, we refer the reader to [2].

Here we propose a modification of the reward function for this model, in order to better guide the agent towards good solutions. The reward function, as defined in [2], illustrates situations when feedback is given at the end of each trial episode. It will be modified so as to give feedback to the agent after each transition to a new state, even if the state is not final. The moment of rewarding is important in the learning process, as learning happens faster if feedback is being given after each transition from one state to another. If we denote by π a path from the initial to a final state, $\pi = (\pi_0\pi_1\pi_2 \dots \pi_n)$, where $\pi_0 = s_1$ (s_1 is the initial state), by $a_\pi = (a_{\pi_0} a_{\pi_1} a_{\pi_2} \dots a_{\pi_{n-1}})$ - the sequence of actions obtained following the transitions between the successive states from the path π, which gives the alignment of fragments $F_{a_\pi} = (F_{a_{\pi_0}}, F_{a_{\pi_1}}, \dots, F_{a_{\pi_{n-1}}})$ (see [2]), then the new definition of the reward function is given in Formula 2:

$$r(\pi_k | s_1, \pi_1, \pi_2, \dots, \pi_{k-1}) = \begin{cases} 0 & \text{if } \ k = 1 \\ w(F_{a_{\pi_{k-1}}}, F_{a_{\pi_{k-2}}}) & \text{otherwise} \end{cases} \tag{2}$$

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

Therefore, after each transition to a new state, the FA agent receives as reward the value of the similarity measure between the fragment corresponding to the current action and the fragment corresponding to the previously taken action. As the learning goal is to maximize the total amount of rewards received on a path from the initial to a final state, the FA agent is actually

trained to find a path π that maximizes the overall similarity of the associated alignment.

3.2. **Permutation Model.** In the following, we introduce a second reinforcement learning based model for the DNA FA problem, considering the general aspects introduced at the beginning of this section. In the rest of the paper, this model will be referred to as the *permutation model*.

The components of the reinforcement learning task are:

- The state space $\mathcal{S}$ (the agent's environment) will consist of $n!$ states, i.e $\mathcal{S} = \{s_1, s_2, \ldots, s_{n!}\}$. Each state $s_i, i = \overline{1, n!}$ in the environment will represent a permutation of all the elements from the fragment set $\{F_1, F_2, \ldots, F_n\} : F_{\sigma^i} = (F_{\sigma_1^i}, F_{\sigma_2^i}, \ldots, F_{\sigma_n^i})$. The initial state will be represented by the identical permutation $s_0 = (F_1, F_2, \ldots, F_n)$. A state s reached by the agent at a given moment will be considered a *terminal (final) state* if the associated performance measure is sufficiently close to a goal value. However, in order to be able to define a final state, apriori knowledge is needed to determine an upper bound of the set of values for the performance measure of all possible permutations.
- The action space consists of $\binom{n}{2} = \frac{n(n-1)}{2}$ actions available to the problem solving agent. Let us denote by $N_a = \frac{n(n-1)}{2}$ the number of actions. An action will be a pair of distinct indices $(i, j), i, j \in \{1, \cdots, n\}, i \neq j$ specifying that the fragments located at indices i and j in the current state will be interchanged in order to make a transition to the following state. Therefore, the action space may be represented as $\mathcal{A} = \{a_1, a_2, \ldots, a_{N_a}\}$, where $a_k \in \{(i, j) | i, j \in \{1, \cdots, n\}, i \neq j\}, \forall 1 \leq k \leq N_a$.
- The transition function $\delta : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ between the states is defined as:

$$\delta(s_j) = \bigcup_{k=1}^{N_a} \Delta(s_j, a_k) \; \forall j \in \{1, \cdots, n!\}, \tag{3}$$

where $\Delta(s_j, a_k) = s_l, \; \forall j, \; l \in \{1, \cdots, n!\}, \; l \neq j$ and s_l is the state resulted from s_j, by interchanging the elements on the positions specified by a_k. This means that, at a given moment, from a state $s \in \mathcal{S}$ the agent can move in N_a successor states, by executing one of the N_a possible actions. 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 $1/N_a$.

- The reward function will also be based on the performance measure for a sequence of fragments, as defined in Formula (1). In this case, the

reinforcement function associated to a transition from a state $s_j = F_{\sigma^j}$ to a state $s_l = F_{\sigma^l}$ $(j,\ l \in \{1, \cdots, n!\},\ l \neq j)$, by executing action $a_k, 1 \leq k \leq N_a$, will be:

(4)
$$r(s_l = F_{\sigma^l} | s_j, a_k) = \begin{cases} PM(F_{\sigma^l}) - PM(F_{\sigma^j}) & \text{if } s_l \text{ is non-terminal} \\ PM(F_{\sigma^l}) - PM(F_{\sigma^j}) + PM(s_0) & \text{otherwise} \end{cases}$$

where s_0 is the identical permutation.

The *FA* problem formulated as a RL problem will consist in training the agent to find a sequence of actions from the initial to the final state so as to maximize the total amount of received rewards, which equates to finding a permutation having the associated performance measure sufficiently close to a given maximum (goal) value.

4. Experiments

This section aims to comparatively evaluate the two reinforcement learning based approaches described in Section 3.

4.1. **Case Study.** The tests are performed on a small section of DNA belonging to the bacterium *Escherichia coli (E. coli)*. The DNA sequence contains 25 bases: $TACTAGCAATACGCTTGCGTTCGGT$. Using the Perl scripts that the authors of [16] produced to generate fragments from a given DNA reference, for the above mentioned *E. Coli* sequence, we obtained 8 fragments, each having a length of 10 bases: $F_1 = TGCGTTCGGT$, $F_2 = TACGCTTGCG$, $F_3 = ATACGCTTGC$, $F_4 = TACTAGCAAT$, $F_5 = GCAATACGCT$, $F_6 = CAATACGCTT$, $F_7 = CTAGCAATAC$, $F_8 = AATACGCTTG$.

These fragments are ordered in the following way to form the original DNA sequence: $F_4F_7F_5F_6F_8F_3F_2F_1$. The maximum value for the performance measure (Equation 1) is 56.01 and it is obtained for two cases: the above alignment, indicating the original DNA sequence, as well as its reverse. The overlap similarity scores for all possible pairs of fragments are obtained using the Smith-Waterman algorithm [12].

For applying the RL models to solve the FA problem, we used a software framework that we have previously introduced in [4] for solving combinatorial optimization problems using reinforcement learning techniques.

We compare the two models by running the corresponding implementations, using the Q-Learning algorithm [14], in conjunction with the three action selection mechanisms we mentioned in Subsection 2.2. Regarding the parameter setting, for all tests we used the following values: the discount factor for the future rewards is $\gamma = 0.9$; the learning rate is $\alpha = 0.95$; the number of

training episodes is $4 \cdot 10^5$. In the case of the permutation model, the maximum value of the performance measure was defined to be 57 and a state was considered final if its performance measure was at most 1 unit away from this maximum. For each of the three action selection policies, tests were made for different values of the policy parameter (ϵ - in the case of ϵ-greedy and the one step look-ahead procedure and τ - in the case of softmax): $\{0.2, 0.4, 0.6, 0.8\}$. Each algorithm was run five times, for each case of selection policy and each value of the policy parameter and the shown results are averaged over these runs. We mention that the experiments were carried out on a PC with an Intel Core i3 Processor at 2.4 GHz, with 3 GB of RAM.

4.2. **Comparative Results.** Below we present the results obtained by the Q-learning based algorithms implementing the models described in Section 3.

The path finding model proves to obtain the correct alignment of fragments with all three action selection mechanisms, the difference being the number of training epochs needed to converge to the optimal solution. The left-hand side of Figure 1 illustrates the performance measure of the solutions obtained during the training process. For all three action selection policies, the algorithm converges to the optimal solution, the one having a performance measure of 56.01. We note that the algorithm using the one step look-ahead procedure achieves the fastest convergence, reaching the correct solution after only 10^3 training epochs, for $\epsilon = 0.8$. The next best, in terms of training epochs until convergence, is the algorithm using the softmax policy, the last one being the ϵ-greedy algorithm. In all three cases, as the values of the policy parameter increase, thus leading to more exploration of the states space, the convergence is achieved more rapidly: on average, for $\epsilon = 0.2$ and $\tau = 0.2$, the algorithm converges (to the optimal or near-optimal solution) after $153 \cdot 10^3$ epochs, while for $\epsilon = 0.8$ and $\tau = 0.8$, the maximum performance measure is reached, on average, after only $25 \cdot 10^3$ epochs.

In terms of accuracy, the permutation model performs equally well, determining the correct alignment for all the tests. In what concerns the number of training epochs, this second approach outperforms the first, for the case study we considered. As can be seen on the right-hand side of Figure 1, the permutation model needs fewer epochs to reach the right solution: on average, for all three action selection policies, convergence is reached after $\sim 6 \cdot 10^3$ epochs, as opposed to $\sim 84 \cdot 10^3$ epochs, which are needed on average for the path finding model. In this case, all three action selection policies find the solution equally fast, but for different values of the policy parameter. Generally, as the value of the policy parameter increases, the convergence is achieved sooner.

In the following, we will also offer an analysis of the computational time needed for the proposed RL algorithms in order to reach the correct solution.

Table 1 illustrates the average time (in seconds) needed for the two algorithms implementing the models that we presented in Section 3. Here, p represents the

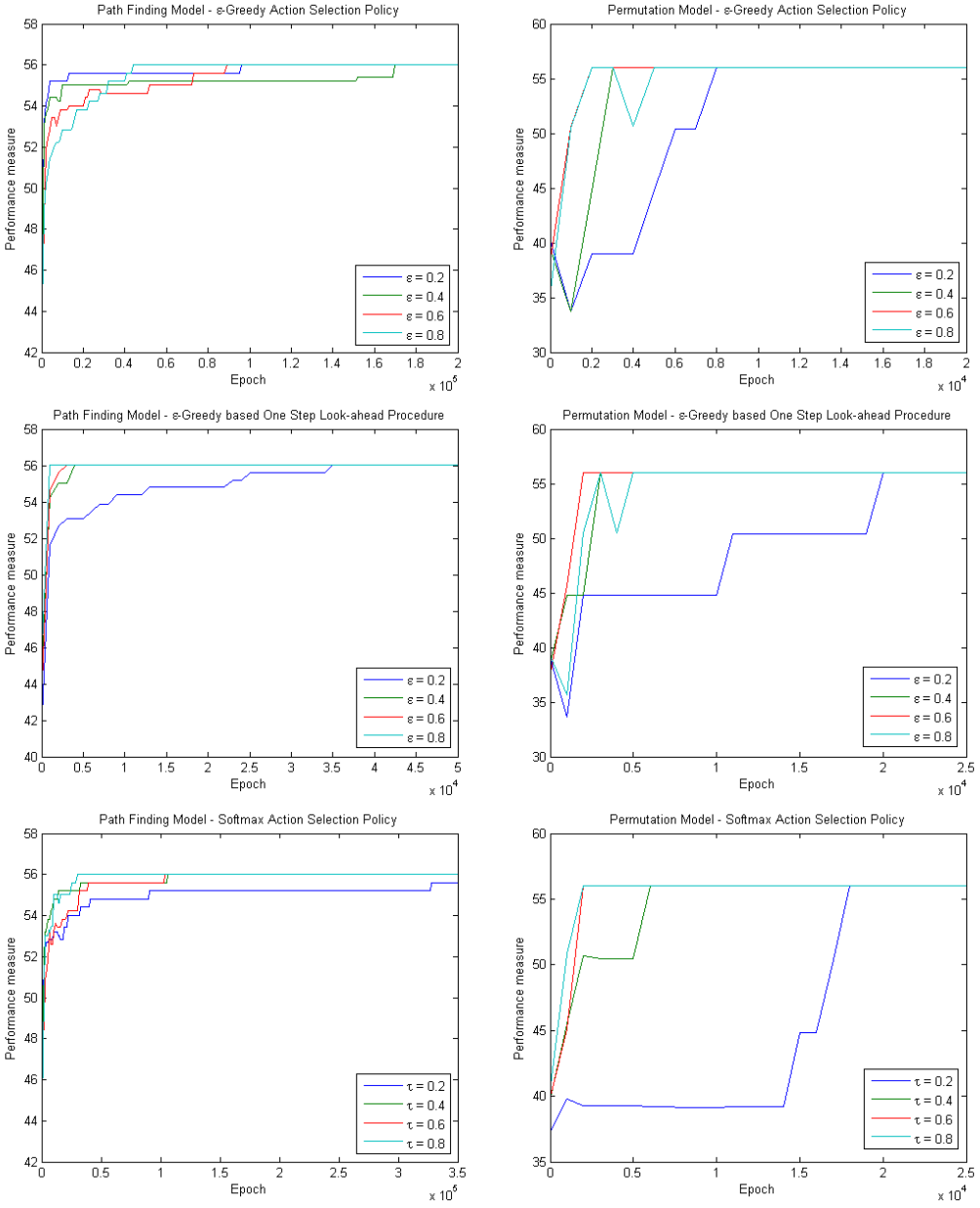

FIGURE 1. Illustration of the learning process for the two RL models.

	Path Finding Model				**Permutation Model**			
	$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$	$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$
ϵ-Greedy policy	2.8	9.2	14.8	11.4	< 2	< 2	< 2	< 2
Look-ahead procedure	2.2	< 2	< 2	< 2	4	< 2	< 2	< 2
Softmax policy	29.6	17.4	40.6	10.2	705.8	44.6	35.6	26.8

TABLE 1. Average computational time (in seconds) needed for each of the presented algorithms to reach the correct solution.

policy parameter, referring to ϵ, in the case of the ϵ-greedy based mechanisms and to τ, for the softmax action selection policy.

Even though intuitively the time is directly correlated with the number of training epochs, between the two models there are some major differences, which lead to different durations for epochs. For instance, in the case of the path finding model, the agent executes exactly 8 actions, to reach a final configuration. For the permutation model, however, the number of actions is different in each epoch, as the agent keeps trying actions until it reaches a state having a performance measure as close as possible to the maximum defined value. Hence, according to the way the agent explores the search space and the way it exploits existing knowledge, an epoch in the permutation model could last less or longer than one in the path finding model. From Table 1 we notice that for the ϵ-greedy mechanism, the solution is reached in less time using the permutation model. Even if an epoch in the permutation model would last longer, the number of epochs needed until convergence is more than 26 times less than the number of epochs until convergence in the path finding approach. Another important observation concerns the softmax policy. An agent guiding its search by softmax must rank all the possible actions from a state, which requires extra computations that are not necessary for the other two policies. Therefore, as shown in Table 1, the algorithms using softmax need more time for each epoch, consequently more time, in total. Futhermore, regarding the permutation model, from any given state, there are 28 possible actions (compared to only 8, for the other model) and then the computations needed for softmax take even longer, as can be seen from Table 1.

4.3. **Discussion and Comparison to Related Work.** We have experimented with two reinforcement learning based models, proposed for the problem of DNA fragment assembly, used in conjuction with three different action selection mechanisms: ϵ-greedy [14], an intelligent ϵ-Greedy based look-ahead action selection mechanism that we have previously introduced [3] and the softmax selection policy [14]. The *path finding model* was introduced in a different study [2] and uses a Q-learning algorithm, with an ϵ-greedy action

selection policy. In this paper we have improved the path finding model and we have introduced the *permutation model*, which is also based on Q-learning.

Both algorithms demonstrated to find the correct alignment of fragments, for the DNA sequence that we considered for the experiments. The difference between them lies in the number of training epochs and the computational time each one needs to achieve convergence.

For the considered case study and parameter setting, the permutation model proved to outperform the path finding approach, in all cases, when inspecting the number of epochs. In what concerns the computational time, the situation is the same, for the ϵ-greedy policy, when the permutation based algorithm finds the correct solution in less than 2 seconds. Still, for the softmax action selection policy, the required time is considerably higher. We remark that in both cases, the algorithms using the intelligent action selection procedure [3] converge, on average, in less epochs than those using ϵ-greedy or softmax, as this procedure efficiently guides the exploration of the search space. We will further investigate how an intelligent action selection mechanism, based on softmax instead of ϵ-Greedy, could influence the outcome.

Regarding the permutation model, we note the following. As it is difficult to determine a final state, we need apriori information about the possible values of the performance measure for permutations. Another option would be to consider a state terminal if the number of actions that were performed from the initial state equals a given maximum number of steps. However, the problem with this course of action could be the fact that the agent would not aim to maximize the total sum of rewards, but the sum of rewards obtained after the given number of maximum steps. A second observation concerns the action space. In this study we use a fairly simple definition of an action, but further work will be done to investigate new types of actions, such as operators inspired from the field of genetic algorithms (e.g. crossover, mutation).

We remark that for the permutation model the number of states of the environment is smaller than for the path finding approach. However, an important drawback of the permutation model is the fact that apriori knowledge about the problem is needed in order to define a final state and this information is not available in all types of situations. Therefore, the path finding approach is more general and, as the results it obtains are good both in terms of accuracy and in terms of computational time, we conclude that it is better.

In the following, we will briefly compare our models with other approaches existing in the literature for the FA problem. Since the used data sets are different from one study to another, we cannot offer a very detailed comparison.

Kikuchi and Chakraborty [6] present an improved genetic algorithm to approach the FA problem. To improve the efficiency of the simple genetic algorithm (regarding both speed and solution quality), the authors introduce

two new ideas, one of which implies manually combining fragments at certain generations. Compared to the approach from [6], our RL based methods do not need user intervention during the training process of the agent.

A comparison of four heuristic DNA FA algorithms is provided by Li and Khuri in [7]: a genetic algorithm, a greedy algorithm, a clustering heuristic algorithm and one using structured pattern matching. All algorithms are experimentally evaluated on several data sets, with the number of fragments ranging from 39 to 773. The authors determined that the running times of all four algorithms ranged from a few seconds to several hours. Even though the data set we use contains a smaller number of fragments, the maximum amount of time needed by the RL approaches is less than 12 minutes, while the minimum is less than 2 seconds. Therefore, we believe that even for larger instances it is likely that the necessary time for the RL models to converge to the correct solution is of the order of minutes, rather than hours.

Angeleri et al. [1] introduce a supervised approach, based on a recurrent neural network to solve the FA problem. In the case of supervised models, a set of inputs with their target outputs is required. The advantage of our RL methods is that the learning process needs no external supervision, as in our approach the solution is learned from the rewards obtained by the agent during its training. Still, as mentioned before, the permutation model requires apriori knowledge about the problem.

5. Conclusions and Further Work

In the present study we investigated two reinforcement learning based models (the *permutation model* and the *path finding model*) for an important problem in bioinformatics, namely the DNA fragment assembly problem. The algorithms implementing both approaches, using three different action selection policies, have been experimentally evaluated and compared.

We plan to extend the evaluation of both Q-learning based algorithms for larger DNA sequences, and implicitly greater number of fragments, to further develop the analysis. We will also investigate possible improvements of these models by adding various local search mechanisms, by combining the softmax policy with the intelligent action selection procedure introduced in [3], by decreasing the action selection parameters (ϵ and τ) during the training process or by extending the permutation model to a distributed RL approach.

ACKNOWLEDGEMENT

This work was partially supported by the Sectoral 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] E. Angeleri, B. Apolloni, D. de Falco, and L. Grandi. DNA fragment assembly using neural prediction techniques. *Int. J. Neural Syst.*, 9(6):523–544, 1999.

[2] M. Bocicor, G. Czibula, and I. G. Czibula. A Reinforcement Learning Approach for Solving the Fragment Assembly Problem. In *Proceedings of the 3th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '11)*, pages 191–198. IEEE Computer Society, 2011.

[3] G. Czibula, I. M. Bocicor, and I. G. Czibula. Temporal Ordering of Cancer Microarray Data through a Reinforcement Learning Based Approach. *PLoS ONE*, 8(4):e60883, 2013.

[4] I. G. Czibula, G. Czibula, and M. I. Bocicor. A Software Framework for Solving Combinatorial Optimization Tasks. *Studia Universitatis "Babes-Bolyai", Informatica*, Special Issue, LVI(3):3–8, 2011.

[5] A. E. Hassanien, M. G. Milanova, T. G. Smolinski, and A. Abraham. Computational intelligence in solving bioinformatics problems: Reviews, perspectives, and challenges. In *Computational Intelligence in Biomedicine and Bioinformatics*, pages 3–47. 2008.

[6] S. Kikuchi and G. Chakraborty. Heuristically tuned GA to solve genome fragment assembly problem. *IEEE CEC*, pages 1491–1498, 2006.

[7] L. Li and S. Khuri. A comparison of DNA fragment assembly algorithms. In *Proc. of the Int?l Conf. on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pages 329–335. CSREA Press, 2004.

[8] L. J. Lin. Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8:293–321, 1992.

[9] R. J. Parsons, S. Forrest, and C. Burks. Genetic algorithms, operators, and DNA fragment assembly. In *Machine Learning*, pages 11–33. Kluwer Academic Publishers, 1995.

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

[11] F. Sanger, A. Coulson, G. Hong, l. D. Hil, and G. Petersen. Nucleotide sequence of bacteriophage Lambda DNA. *J. Molecular Biology*, 162(4):729–773, 1982.

[12] T. Smith and M. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[13] I. Susnea, G. Vasiliu, A. Filipescu, and A. Radaschin. Virtual pheromones for real-time control of autonomous mobile robots. *Studies in Informatics and Control*, 18(3):233–240, 2009.

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

[15] S. Thrun. The Role of Exploration in Learning Control. In *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, Florence, Kentucky, 1992.

[16] W. Zhang, J. Chen, Y. Yang, Y. Tang, J. Shang, B. Shen, and I. K. Jordan. A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies. *PLoS ONE*, 6(3):e17915, 2011.

(1) Babeş-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania
E-mail address: {gabis, istvanc, iuliana}@cs.ubbcluj.ro