STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVI**, Number 1, 2021
DOI: 10.24193/subbi.2021.1.03

CONSTRUCTING UNROOTED PHYLOGENETIC TREES WITH REINFORCEMENT LEARNING

PANNA LIPTÁK AND ATTILA KISS

Abstract. With the development of sequencing technologies, more and more amounts of sequence data are available. This poses additional challenges, such as processing them is usually a complex and time-consuming computational task. During the construction of phylogenetic trees, the relationship between the sequences is examined, and an attempt is made to represent the evolutionary relationship. There are several algorithms for this problem, but with the development of computer science, the question arises as to whether new technologies can be exploited in these areas of computational biology.

In the following publication, we investigate whether the reinforced learning model of machine learning can generate accurate phylogenetic trees based on the distance matrix.

1. Introduction

In phylogenetics, the evolutionary relationships among biological entities are examined. These entities can mean species, individuals but also genes. This paper will focus on the relationship between genes. Trying to identify the inheritance and mutation processes is an important challenge. It can help biologists to refine their understanding of how evolution works and by that further develop the models of evolution or a current example of its usefulness: defining relationships can help to see the geographical distribution of certain subspecies/mutations. A phylogenetic tree is a branching diagram that represents the lineage relationships between genes. It reflects how the examined samples evolved from a common ancestor which is in the root of the tree. Each internal node splits apart a single group into two descendant groups. The genes of interest can be found in the leaves of the tree. It is possible to create an unrooted tree; in this case, the ancestral root is not defined, only

Received by the editors: 24 April 2021.

2010 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* code [**Artificial Intelligence**]: Applications and Expert Systems – *Medicine and science*.

Key words and phrases. Bioinformatics, Reinforcement Learning, Machine Learning Algorithms.

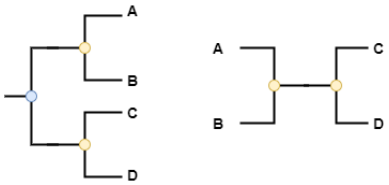


FIGURE 1. Example of a rooted (left) and an unrooted (right) phylogenetic tree. The leaves represent the taxon (A, B, C, D) and the yellow points are the internal nodes. For example, the most recent common ancestor of A and B is at their branch point. The blue point represents the root of the tree, which on the left example is the most recent common ancestor of A, B and C, D nodes.

the relatedness of the leaf nodes. In this paper, we will concentrate on these unrooted trees.

There are many mathematical and algorithmic approaches to construct the tree of given genes. For distance-based algorithms - like UPGMA and Neighbor join – first we need to perform a multiple sequence alignment to compute pairwise distances. This data is stored in the distance matrix. In these approaches, we try to generate a tree where sequences with shorter pairwise distances are closer to each other.

Another type of algorithm is the character-based approach. The Maximum Parsimony method implies an implicit model of evolution. It tries to find a tree with a minimal number of evolutionary steps required to explain the input data. The Maximum Likelihood method uses probability calculations based on a given model of evolution. It considers all possible trees and therefore it is computationally intense, more precisely it is an NP-hard problem [2].

We will discuss the Neighbor join (NJ) [16] algorithm in more detail in the Related Works section. We investigate how an unrooted tree that was constructed based on the distance matrix could be constructed by a reinforcement learning model. Even though NJ is considered a fast algorithm (there are heuristic-accelerated versions, see Related Works), implementing the original method leads to an algorithm of $O(n^3)$ time complexity [20], which is not ideal for large data sets.

Reinforcement learning (RL) is one of the three paradigms of machine learning [22]. It is life-like in the sense that learning is based on experiences. Given an environment and an agent within, the agents goal is to find a series of actions with the maximum reward. The agent receives a reward after every action/step it makes, and the purpose is to learn what decision to make at

each state to earn the highest reward at the end of the episode, that is, to determine an optimal policy.

In this paper, we examine how the RL agent can construct an accurate phylogenetic tree by making decisions in the environment described in the Main contribution section. During the training phase, the agent tries to find the most suitable policy which would be used on the test data set to determine the accuracy of the algorithm. Our experiments have shown that with this approach, it is possible to create sufficiently accurate unrooted phylogenetic trees based on the distance matrices.

2. Related Works

The Neighbor join algorithm was first introduced by Saitou and Nei [16] in 1987. The main principle is to minimize the total branch length at each stage; therefore, it is a greedy algorithm. For choosing the two taxa to merge, each step a Q matrix has to be calculated, shown in Formula 1.

$$(1)\quad Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n}(d(i,k)) - \sum_{k=1}^{n}(d(k,j)) \quad (\forall i,j : 1 \leq i < j \leq n)$$

The two sequences with the smallest Q value will be joined. After that, we have to recalculate each taxon's distance to the new u node. In 1988, Studier and Keppler [20] published an improved version of NJ, correcting the way of inferring the distances, as shown in Formula 2 (considering i, j is joined and k is every other taxon). In the RL approach, we eliminate the Q matrix and let the RL agent learn a policy to decide in each state which two taxa to choose for the next join.

$$(2)\qquad d(u,k) = \frac{1}{2}[d(i,k) + d(k,j) - d(i,j)]$$

As mentioned before, this algorithm has $O(n^3)$ time complexity [20], which is not ideal for large data sets. There are several works in the literature with heuristic-accelerated versions. In this section, we discuss some of these approaches.

QuickJoin [10] introduces an algorithm with $\Theta(n^2)$ complexity, although the worst-case remains $O(n^3)$. It uses a quad-tree to find the lower bounds of $Q(i,j)$ values, therefore there is no need to calculate the whole Q matrix: the algorithm can skip when the lower bound is higher than one of the known $Q(i,j)$ value. This prunes the search for the minimal $Q(i,j)$ value. For building the quad-tree they use a linear function, which only depends on $d(i,j)$.

RapidNJ [18] uses the observation that in the formula of the Q matrix (shown in Formula 1), the sum is constant in the context of row i. Therefore, it can be used as an upper bound for each row in Q, reducing the search space. This approach has a worst-case $O(n^3)$ complexity, but in the paper, they showed that in practice it has a better performance. NINJA [24] is based on the same idea: dramatically reducing the viewed candidates at each step, but it improves the results of RapidNJ, while still offering $O(n^3)$ worst-case time complexity.

There are two other methods worth mentioning: Relaxed neighbor join [4] and Fast neighbor join [3] to improve the speed by choosing the taxon to join from a subset. In the relaxed version, a transformed distance is calculated for the sequences and two taxa are joined if they are they are the minimum transformed distance of each other. Fast neighbor join offers $O(n^2)$ time complexity, using the visible set as the candidate set for choosing two taxa to join. These two methods are proven fast, but at the cost of the phylogenetic tree they construct provides only an approximate solution if the pairwise distances are not nearly-additive.

These works are just some of the more important milestones, but it also shows how important the improvement of the time complexity of the NJ is in phylogenetics. With the development of artificial intelligence in recent years, there has been a tendency to take advantage of the opportunities offered by machine learning in other fields as well, such as phylogenetics. We would like to present some of these approaches.

Works using machine learning for phylogenetic tree construction already exists. In [1], they introduced an approach to the case where the distance matrix is incomplete. By using deep architectures, they could eliminate the need for a molecular clock assumption, representing a real-world occurrence of the problem.

Multiple sequence alignment is also a challenging problem of bioinformatics. In [11] a reinforcement learning-based approach was introduced. They found that the RL approach outperformed (in most cases) other methods, whilst decreasing the computational time. The training process used the Q-learning algorithm. Another solution for this problem uses a deep RL algorithm and a long short-term memory network, introduced in [8]. Their experiments show, that this version not only outperforms canonical multiple sequence aligner tools but other RL approaches too.

In [23] a convolutional network was used to infer the topology of an unrooted tree by classification. They experimented on simulated data sets and the results showed that this model has great potential. It was not only faster than other methods but it was highly accurate and the accuracy of the classification

could be further improved by the number of training data sets. The limitation of this approach is the number of sequences, it was constructed to classify the topology of four sequences (quartet topologies).

An example of a more specific use-case of phylogenetic trees was presented in [9]: an inverse reinforcement learning approach to model a cancer cell's genetic evolutionary process. In this method, the optimal policy and the reward function were reverse engineered to reach interpretable biological conclusions.

In this section, the variants and complexity of NJ algorithms were presented, as well as examples of how artificial intelligence has been used in bioinformatics. The question arose as to whether the NJ problem could be solved by reinforcement learning if we consider each joining operation as a step that also influences the possible future operations. Thus, it is up to the RL agent to decide which two taxon's merging will lead to an optimal solution. At the time of writing, to the knowledge of the authors, no attempt has been made to use the RL method to determine the topology of phylogenetic trees.

3. Main contribution

In this section, we introduce the proposed model of reinforcement learning to solve the problem of constructing unrooted phylogenetic trees based on distance matrices.

The scene of learning is the environment where the agent performs actions that cause the state of the environment to change and the next step takes place in this new state. If the agent reaches the predefined stop condition, the episode ends and the process starts all over again. The agent's knowledge of the environment is called observation and along these, it tries to find connections between the decisions and the reward it receives at the end of each episode. The goal of the model is to learn a policy, to be able to make good (rewarding) decisions in any given state of the environment.

3.1. **Environment.** The input of the proposed model is an $(n+1) \times (n+1)$ $(n > 3)$ dimensional matrix. The first row and column contain the sequence label, which will symbolize the particular taxa in the completed phylogenetic tree. Omitting the elements containing the labels, we get an $n \times n$ dimensional matrix, which is the distance matrix of the sequences.

A distance matrix is symmetric and the diagonal only contains zeros, therefore it is enough to store the upper triangular of it as a vector for the model. Since labels must also be stored for the tree, we create two vectors from every distance matrix. The *state* space $\mathcal{S} = \{s_0, s_1, ..., s_{\frac{n*(n-1)}{2}}\}$ contains the distance values, and the *labels* indicate which two sequence labels belong to that distance. The length of $\mathcal{S}$ is the size of the upper triangular matrix, which is

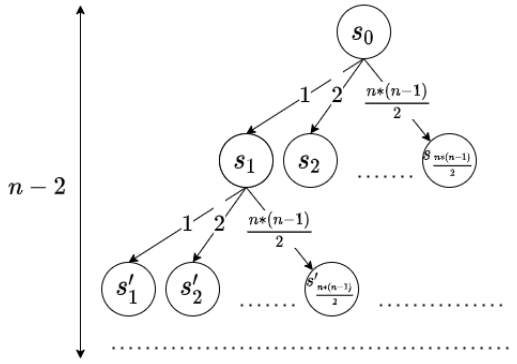


FIGURE 2. Schematic model of the $\mathcal{S}$ state space

$\frac{n*(n-1)}{2}$. If we assume that the i. sequence label is *"i"* ($\forall i : 1 \leq i \leq n$) then the initialization of the state vector should follow Formula (3).

$$p = \sum_{k=1}^{i-2}(n-k) + j - i \quad (\forall i, j : 1 \leq i < j \leq n) \quad s_p = d(i,j) \tag{3}$$

The *action* space $\mathcal{A}$ is defined as choosing a state from $\mathcal{S}$. More specifically, $\mathcal{A} = \{a_1, ..., a_{n-2}\}$ where $1 \leq a_r \leq \frac{n*(n-1)}{2} (\forall 1 \leq r \leq n-2)$. Every action represents a merge of two sequences in the phylogenetic tree, therefore the length of the *action* space is $n-2$: given n sequences we define a step as arbitrary choosing two sequences to merge, after 1 step there are $n-1$ sequences left, after $n-2$ steps there are $n-(n-2) = 2$ left, at this point we do not need to take more steps, because the only choice we have is merging the last two taxon which results in a phylogenetic tree.

After every step, the environment has to be updated according to the chosen action. This consists of updating the distance values, which are done according to the NJ algorithm, and of modifying the labels. A new distance must be calculated for every position where at least one of the corresponding sequences contains a taxon that was chosen for being merged. For example, if in a given step *"i"* and *"j"* are the two taxon that will be merged to a common ancestor, then we have to update every distance that contains either *"i"* or *"j"* and in the corresponding label change *"i"* or *"j"* to *"ij"*. As a result, the constructed tree will contain *(i,j)* (according to the Newick format [13]). Note that after the first step, a merging can consist of an already existing subtree. For this model, branch lengths are not calculated only the topology of the tree but future work will consider extending the model.

As Figure 2 shows, the environment itself is a tree graph with a depth of $n-2$. The tree contains invalid states: if the algorithm starts with n taxon then after the first step there are $n-1$ taxon left and the number of possible taxa pairs for the next step is $\frac{(n-1)(n-2)}{2}$; thus, in this episode we define the following states $s'_{\frac{(n-1)*(n-2)}{2}+1}, s'_{\frac{(n-1)*(n-2)}{2}+2}, ..., s'_{\frac{n*(n-1)}{2}}$ as invalid states.

The RL agent starts in the s_0 state, in each step it chooses a number between 1 and $\frac{n(n-1)}{2}$ and moves to the corresponding state. If the state is valid, then the two nodes can be merged and the affected distances have to be recalculated, hence the change in the state's notation system (s'). If the chosen state was invalid then in this episode the agent failed to construct a phylogenetic tree and the episode ends. Figure 2 is an example where the agent's first choice is 1 and the other not chosen nodes are not further explored in this episode.

3.2. **Reward function.** If the agent chooses an action that corresponds to an invalid state then the episode ends with -1 as the reward. If the RL agent reaches a leaf of the tree with a valid state then the episode also ends and the reward will be proportional to the symmetric difference [15] between the RL constructed tree and the phylogenetic tree calculated by the aforementioned NJ algorithm. Let π be the sequence of decisions made by the agent, where $\pi = (\pi_0, .., \pi_k)$ and $\forall k \in \{1, ..., n-2\}, \pi_k$ is an element of the state space. Furthermore, we define symdiff(t_1, t_2) as the symmetric difference between phylogenetic trees t_1 and t_2. In this case, the *reward* function can be defined as shown in Formula (4) where $tree_c$ is the tree constructed by the RL agent and $tree_{NJ}$ was calculated by the original NJ algorithm.

(4)

$$r(\pi_k|\pi_{k-1}, ..., \pi_1) = \begin{cases} -1 & , \pi_k \text{ is an invalid state} \\ \frac{(3n-6)-\text{symdiff}(tree_c, tree_{NJ})}{(3n-6)} * 10 & , k = n-2 \\ 0 & , \text{otherwise} \end{cases}$$

In Formula (4), $3n-6$ is the maximum symmetric difference between two trees with n nodes based on Robinson-Foulds (1981) [15]. With the given function, we transform the symmetric difference, which is in $[0, 3n-6]$ - where 0 means the two trees are identical - to $[0, 10]$ where 10 means that they are identical. This is necessary because the agent tries to maximize the reward, and the goal is to construct phylogenetic trees similar to the tree that was produced by the NJ algorithm. The third case of a step is when the agent chose a valid state but it was not a leaf node. In this case, the model rewards this step with 0, because in this state it cannot determine whether this step was optimal for the phylogenetic tree that will be ideally constructed at the end of the episode.

Algorithm 1: One step in the RL environment

Input: action

if *episodeEnded* **then**
  $resetEnvironment()$;
end
if *state[action] == -1* **then**
  $episodeEnded = True$;
  $terminate : state, reward = -1$;
else
  $firstNode$ = get first node from labels[action];
  $secondNode$ = get second node from labels[action];
  $newNode = firsNode + secondNode$;
  $newLabels = []$;
  $newDistances = []$;
  for *label in labels* **do**
    if *label contains firstNode* **then**
      $distFirst = state$[index of label];
      get otherNode from label;
      for *otherLabel in labels* **do**
        if *otherLabel contains secondNode and otherNode* **then**
          $distSecond = state$[index of otheLabel];
        end
      end
      $newDistance = \frac{1}{2} * (distFirst + distSecond - state[action])$;
      add newNode + otherNode to newLabels;
      add newDistance to newDistances;
    end
  end
  delete affected states and labels;
  add $newLabels$ to $labels$ and $newDistances$ to $state$;
  add -1 to state and empty label to labels for every invalid state;
  $addNewNodeToNewickTree(firstNode, secondNode)$;
  if *there is only one valid state left* **then**
    $addNewNodeToNewickTree$(remaining nodes);
    $constructedTree = treePieces[0]$;
    $episodeEnded = True$;
  end
  if *episodeEnded* **then**
    $diff = symmetricDifference(goalTree, constructedTree)$;
    $terminate : state, reward = ((3 * n - 6) - diff)/(3 * n - 6) * 10$;
  else
    $transition : state, reward = 0, discount$;
  end
end

Algorithm 1 presents the pseudo-code of a step in the proposed RL environment. The input is the chosen action and at the end of the step the environment either transitions into a new state (see line 40 of Algorithm 1) or the episode ends because of an invalid state (see lines 4-6 of Algorithm 1) or because it finished the construction of the phylogenetic tree (see lines 31-34 of Algorithm 1).

3.3. **Policy.** The policy is the strategy that the agent uses to reach its goal in a given environment. It is the function of the current state of the environment. In this model, the RL agent starts exploring the states with a random policy and the goal is to find the optimal policy by the end of the training phase. Updating the policy by determining the optimum next action is based on the Q values, the values of specific actions. The Q value is the function of the current state s and the action a made in that state. As shown in Formula (5), the Q value consists of the immediate reward received by taking action a in state s and the maximum reward that could be earned in the following state (s'), multiplied by the discount factor (γ). It is a recursive equation as s' will depend on s'' and so on. In an optimal policy, the agent chooses an action with the maximum Q value. The equation for updating the Q values is shown in Formula (6). For calculating the new Q' value for a given state and action the formula contains the learning rate α which determines to what extent should new information overwrite old Q values: set to 0 means only the old information is taken to account, and 1 means only the most recent information.

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a) \tag{5}$$

$$Q'(s_t,a_t) = Q(s_t,a_t) + \alpha * (r(s_t,a_t) + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)) \tag{6}$$

Calculating all the Q values for a given state is complex, both computationally and in many cases storage-wise. In these cases, deep Q learning [12] could be a solution, because it uses a neural network to estimate the Q values for each state-action pair and therefore approximates the optimal Q function. The input of the deep Q network is a state and the outputs are estimated Q values for each possible action from the given state. The proposed model used a deep Q network with 100 layers supplemented with a replay buffer of a maximum size of 10000. A replay buffer consists of the previous step's transition data and the deep Q network samples a small batch of transitions for its calculations because it is a more stable approach to use uncorrelated samples than using only the latest transitions.

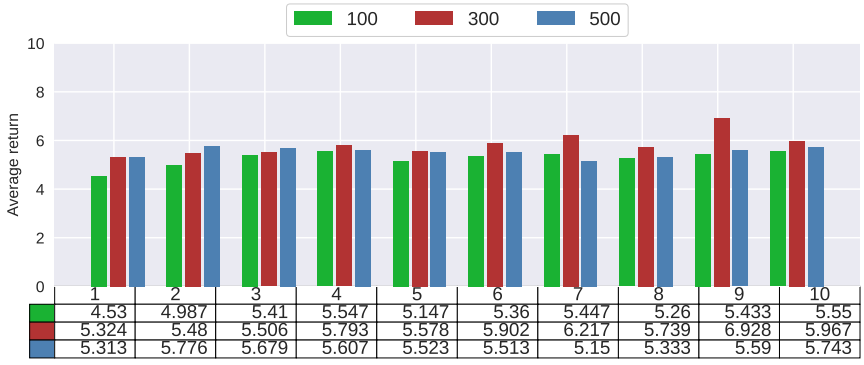
4. Experimental Results

For the experiments, we simulated data sets with different tree topologies using Rose [19] which is a tool that implements the probabilistic model of evolution. As an input of this algorithm, we created several trees in Newick [13] format and set 50 as the average length of the sequences. The trees can be divided into three categories according to their topology: balanced, pectinate, and random. The output of Rose is the generated alignment. From the alignments, we calculate the distance matrices with Emboss's [14] *distmat* module. The reinforcement learning algorithm can be parameterized to work with any constant number of sequences and the training has to run on a data set that contains alignments with the same number of sequences. To create the tree we wanted to resemble the one we constructed, we used the DendroPy library's [21] NJ algorithm and also to calculate the symmetric difference. We implemented the proposed approach in Python using Tensorflow's reinforcement learning library, TF-Agents [6].

We trained the proposed RL algorithm on data sets of different sizes: 100, 300, and 500 distance matrices, each containing 3 types of topologies. $25\% - 25\%$ of the distance matrices was based on pectinate and balanced trees and the rest 50% was based on random trees. For these experiments, each data set had 6 sequences, and for each type of topology, we assigned randomized branch lengths to have a diverse training set.

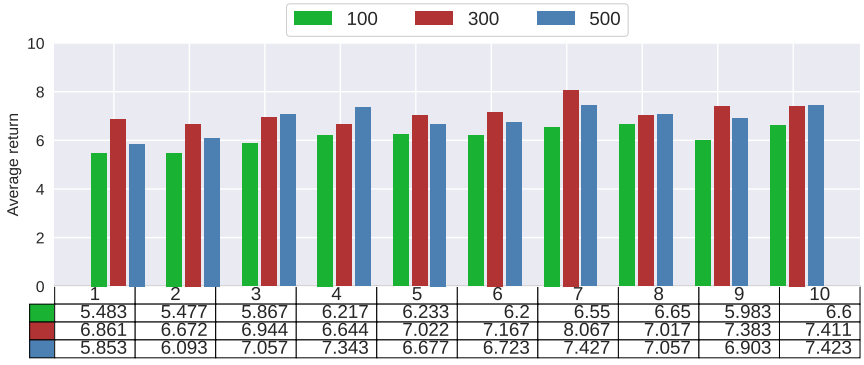
Figure 3 shows how the average return changes as the number of training sessions increases. One training session means going through the training data set and trying to construct the phylogenetic tree for the given distance matrix and repeating on each one 10 times to encourage exploitation. Every test case had the same parameters except the discount factor. Each time the average return was calculated using the trained policy on a different evaluation environment which also contained the training data set, thereby we can determine whether the policy became more accurate or not on the training data set.

We designed this experiment to determine the discount factor which is an important training parameter. The discount factor is between 0 and 1 and it means whether the agent should prioritize immediate rewards (0) or prefer potential future rewards (higher values) the agent expects to receive. In the experiment shown in Figure 3a, we used 0.05 as the discount factor and the diagrams show what we would expect: in this environment, immediate rewards have smaller significance than future rewards. Higher discount factor values showed more suitable tendencies. In this algorithm, the RL agent has to focus on maximizing what he receives at the end of the episode but also consider immediate rewards due to avoiding invalid states. The results of the experiments



	1	2	3	4	5	6	7	8	9	10
100	4.53	4.987	5.41	5.547	5.147	5.36	5.447	5.26	5.433	5.55
300	5.324	5.48	5.506	5.793	5.578	5.902	6.217	5.739	6.928	5.967
500	5.313	5.776	5.679	5.607	5.523	5.513	5.15	5.333	5.59	5.743

(A) 0.05 discount factor



	1	2	3	4	5	6	7	8	9	10
100	5.483	5.477	5.867	6.217	6.233	6.2	6.55	6.65	5.983	6.6
300	6.861	6.672	6.944	6.644	7.022	7.167	8.067	7.017	7.383	7.411
500	5.853	6.093	7.057	7.343	6.677	6.723	7.427	7.057	6.903	7.423

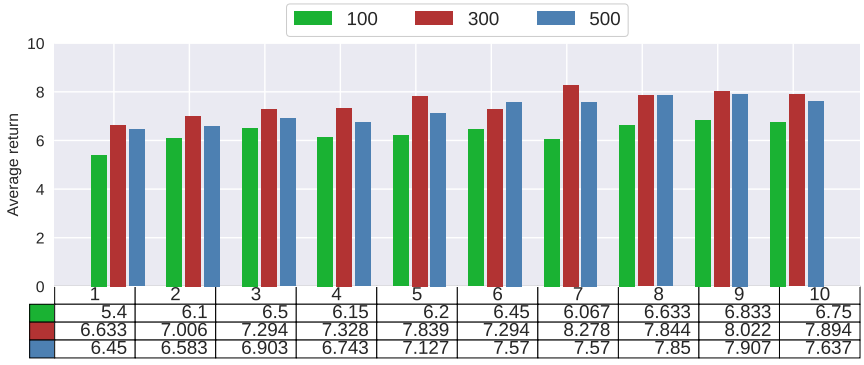
(B) 0.5 discount factor

FIGURE 3. Average return of the reward function at the end of each session with different discount factor parameters

that were performed to determine the discount factor are shown in Figure 3, and amongst them, the most promising test case is shown in Figure 3d.

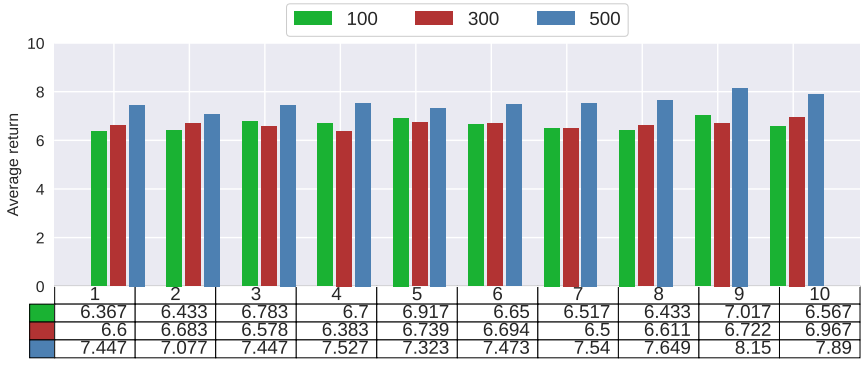
Based on these results, for the upcoming experiments, we used 0.95 as the discount factor.

For determining the accuracy, we experimented with different topology ratios in the training data set and examined how these differences affected the outcome. In the experiment shown in Figure 4, we trained every model during 10 sessions (e.g. for a data set containing 500 distance matrices this means 50000 episodes). According to the previous experiment, the larger data sets could reach better results after several episodes of training, therefore we only examined the cases where the training sets contained 300 and 500 distance



	1	2	3	4	5	6	7	8	9	10
100	5.4	6.1	6.5	6.15	6.2	6.45	6.067	6.633	6.833	6.75
300	6.633	7.006	7.294	7.328	7.839	7.294	8.278	7.844	8.022	7.894
500	6.45	6.583	6.903	6.743	7.127	7.57	7.57	7.85	7.907	7.637

(C) 0.75 discount factor

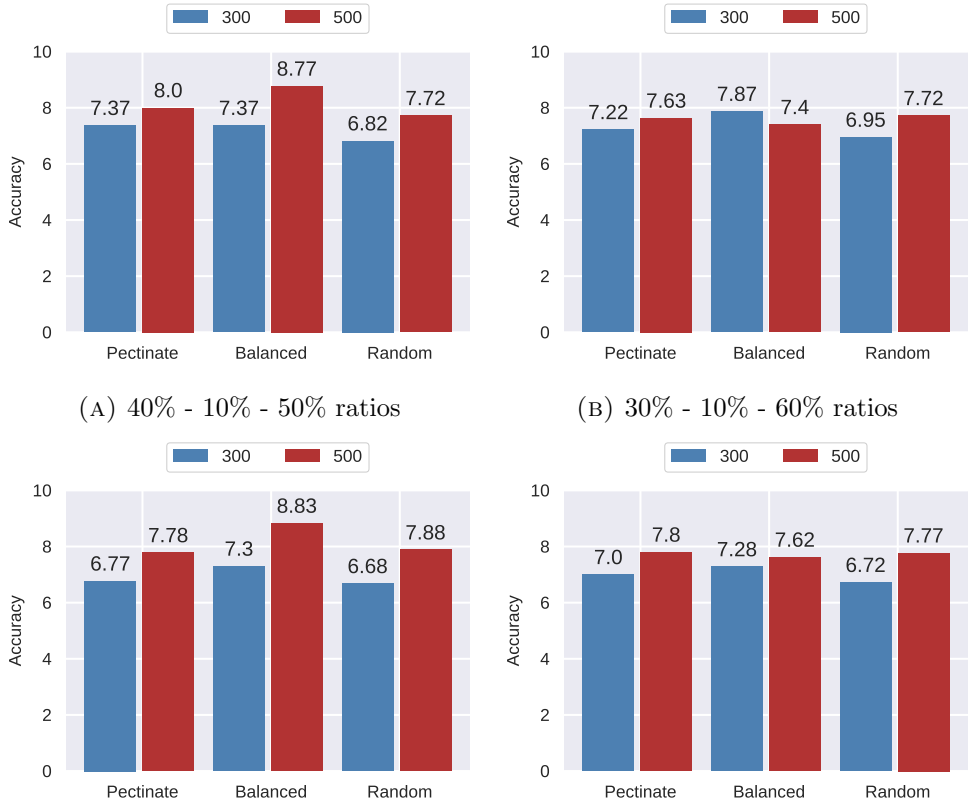


	1	2	3	4	5	6	7	8	9	10
100	6.367	6.433	6.783	6.7	6.917	6.65	6.517	6.433	7.017	6.567
300	6.6	6.683	6.578	6.383	6.739	6.694	6.5	6.611	6.722	6.967
500	7.447	7.077	7.447	7.527	7.323	7.473	7.54	7.649	8.15	7.89

(D) 0.95 discount factor

FIGURE 3. Average return of the reward function at the end of each session (cont.)

matrices. For this experiment, first, we had to create a test data set in the same way as the training data set and separate the trees by their topology, so we can evaluate the differences between them. For every topology, the test data set on which we evaluated the accuracy, contained 100 distance matrices. The result of the training is a policy by which the RL agent makes decisions about the next step in the algorithm. Using this policy, we constructed trees from the test data set and used the end reward as the accuracy (as mentioned before if the algorithm constructed a tree, then the end reward is between 0 and 10 where 10 means the tree is identical to the goal tree). In this evaluation phase, we still used the DendroPy library's NJ algorithm to determine the symmetric difference.



(A) 40% - 10% - 50% ratios

(B) 30% - 10% - 60% ratios

(C) 40% - 20% - 40% ratios

(D) 30% - 20% - 50% ratios

FIGURE 4. Accuracy on test data set with different topology ratios (balanced - pectinate - random) in the training set

Figure 4 shows how the different topology ratios in the training data set affected the accuracy of the model. In the cases of the smaller data set, the model could reconstruct a balanced tree with the highest accuracy, even if it was mostly trained on trees with other topologies. Therefore the training data set could contain balanced trees in a small ratio (10%) without compromising the accuracy of this topology. The overall highest accuracy was detected if the policy was trained on the data set containing 500 distance matrices and the proportion of trees was as follows: 40% pectinate, 10% balanced and 50% random. Upon these results we examined whether a longer training session could improve the policy's accuracy.

TABLE 1. Accuracy results of the trained policies on different topologies. Each policy was trained on the data set containing 500 distance matrices: 10% pectinate, 40% balanced and 50% random trees. The columns associated with the topologies contain the average return of the model

Episodes	Pectinate	Balanced	Random	Average accuracy
50 000	8.0	8.77	7.72	81.63%
100 000	8.48	8.55	8.28	84.36%
150 000	8.15	8.53	8.02	82.33%

Table 1 shows how the accuracy increases as we double the number of episodes in the training session in the case when the training data set contained 500 distance matrices. With 100000 episodes of training the average accuracy was 84.36%, but the further increase of the episodes resulted in the decrease of the accuracy, possibly due to overfitting, which means the model became too specific to the training data. For the model to work on different data as well, the training has to stop when the prediction has sufficient accuracy but it does not yet become too precise to the data it was trained on, because that would impact the generalization negatively. Our experiments showed that the model on the aforementioned training data set was the most accurate when it was trained for 100000 episodes.

Both the training and test data set were created with Rose [19] using the same parameters and therefore having the same model of evolution. It is expected that this evolutionary model affects the accuracy of the algorithm. To verify it we tested the model on data with a different model of evolution. We selected a 6-element subset of the Sarich data set [17] [5] and constructed the phylogenetic tree with the trained model. The result showed 50% accuracy, which means that indeed the RL model approximates the model of evolution of the training data set. To have a more generalized model further experiments have to be performed by expanding the training data set with different types of evolution models.

5. Conclusions and Future Work

In this study, we examined how a phylogenetic tree constructed by NJ could be recreated with a type of artificial intelligence, the reinforced learning model. The essence of the proposed model is that the RL agent moves in a tree graph where each descendant indicates which two individuals are joined in the constructed tree. At the end of the episode, the agent is rewarded for how well

he managed to recreate a tree similar to the one constructed by the traditional NJ algorithm. The model uses a deep Q network to find the optimal policy.

Our experiments showed that the proposed model has the possibility to produce an accurate phylogenetic tree based on the distance matrix, and this accuracy could be further improved by refining the training data set. Although the model has great potential for constructing accurate phylogenetic trees, it has its limitation. For each number of sequences that have to be joined, a unique RL model has to be trained because a model only works for a constant number of sequences that the training session's data sets contained.

This model outlines a generalized approach to the problem and in this form is a model illustrating artificial intelligence rather than an algorithm for producing real phylogenetic trees. However, we believe that it could serve as a useful basis on which to build a solution to the real problem.

In the future, we want to supplement the model with a long short-term memory network [7]. It may be worthwhile to implement this addition to the deep Q learning phase because it has the advantage of being able to memorize long/hidden dependencies and thus make more accurate decisions.

Besides, further experiments could be performed to improve the presented results by expanding the training data set ($10 - 100$ times and with different models of evolution) and by increasing the number of episodes in the training phase accordingly.

The presented results, although promising, were still generated on simulated data. For the RL approach to provide a solution in real cases, the model should be supplemented to work with incomplete or inaccurate data structures, or even starting one step further, where the sequences serve as input, thus providing more information, not just a statistic representation about the sequences. Also, when calculating trees, it would be worthwhile to calculate the length of the branches as well, not just the topology. In this case, the comparison of trees in the reward function would not be done according to the symmetric difference, but according to the Robinson-Foulds algorithm [15]. Furthermore, the case where a rooted phylogenetic tree has to be constructed could be examined.

Our proposed model is an example of how artificial intelligence and deep learning can be applied in bioinformatics, where many computationally complex problems possibly could be solved more effectively by the application of these technologies.

6. Acknowledgements

The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] Bhattacharjee, A., and Bayzid, M. S. Machine learning based imputation techniques for estimating phylogenetic trees from incomplete distance matrices. *BMC genomics 21*, 1 (2020), 1–14.

[2] Chor, B., and Tuller, T. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics 21*, suppl_1 (2005), i97–i106.

[3] Elias, I., and Lagergren, J. Fast neighbor joining. In *International Colloquium on Automata, Languages, and Programming* (2005), Springer, pp. 1263–1274.

[4] Evans, J., Sheneman, L., and Foster, J. Relaxed neighbor joining: a fast distance-based phylogenetic tree construction method. *Journal of molecular evolution 62*, 6 (2006), 785–792.

[5] Felsenstein, J., and Felenstein, J. *Inferring phylogenies*, vol. 2. Sinauer associates Sunderland, MA, 2004.

[6] Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., and Brevdo, E. TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018. [Online; accessed 06-April-2021].

[7] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[8] Jafari, R., Javidi, M. M., and Rafsanjani, M. K. Using deep reinforcement learning approach for solving the multiple sequence alignment problem. *SN Applied Sciences 1*, 6 (2019), 1–12.

[9] Kalantari, J., Nelson, H., and Chia, N. The unreasonable effectiveness of inverse reinforcement learning in advancing cancer research. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 437–445.

[10] Mailund, T., and Pedersen, C. N. Quickjoin—fast neighbour-joining tree reconstruction. *Bioinformatics 20*, 17 (2004), 3261–3262.

[11] Mircea, I.-G., Bocicor, I., and Czibula, G. A reinforcement learning based approach to multiple sequence alignment. In *International Workshop Soft Computing Applications* (2016), Springer, pp. 54–70.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature 518*, 7540 (2015), 529–533.

[13] Olsen, G. The "newick's 8: 45" tree format standard. *World-Wide-Web Reference. http://evolution.genetics.washington.edu/phylip/newick_doc.html* (1990).

[14] Rice, P., Longden, I., and Bleasby, A. Emboss: the european molecular biology open software suite. *Trends in genetics 16*, 6 (2000), 276–277.

[15] Robinson, D. F., and Foulds, L. R. Comparison of phylogenetic trees. *Mathematical biosciences 53*, 1-2 (1981), 131–147.

[16] Saitou, N., and Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution 4*, 4 (1987), 406–425.

[17] Sarich, V. M. Pinniped phylogeny. *Systematic Zoology 18*, 4 (1969), 416–422.

[18] Simonsen, M., Mailund, T., and Pedersen, C. N. Rapid neighbour-joining. In *International Workshop on Algorithms in Bioinformatics* (2008), Springer, pp. 113–122.

[19] Stoye, J., Evers, D., and Meyer, F. Rose: generating sequence families. *Bioinformatics (Oxford, England) 14*, 2 (1998), 157–163.
[20] Studier, J. A., and Keppler, K. J. A note on the neighbor-joining algorithm of saitou and nei. *Molecular biology and evolution 5*, 6 (1988), 729–731.
[21] Sukumaran, J., and Holder, M. T. Dendropy: a python library for phylogenetic computing. *Bioinformatics 26*, 12 (2010), 1569–1571.
[22] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.
[23] Suvorov, A., Hochuli, J., and Schrider, D. R. Accurate inference of tree topologies from multiple sequence alignments using deep learning. *Systematic biology 69*, 2 (2020), 221–233.
[24] Wheeler, T. J. Large-scale neighbor-joining with ninja. In *International Workshop on Algorithms in Bioinformatics* (2009), Springer, pp. 375–389.

ELTE Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary
Email address: ie33ou@inf.elte.hu, kiss@inf.elte.hu