The Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University
הפקולטה למדעים מדויקים ע"ש ריימונד ובברלי סאקלר
אוניברסיטת תל אביב

Tel Aviv University
The Raymond and Beverly Sackler Faculty of Exact Sciences
Blavatnik School of Computer Science

The tree reconstruction game: phylogenetic reconstruction using reinforcement learning

A thesis submitted toward the degree of Master of Science in Computer Science

By
Oz Granit

This research was carried out in The School of Computer Science
Under the supervision of **Prof. Yishay Mansour**

This study was jointly conducted by me, by Michael Alburquerque (Tel-Aviv University), and by Dana Azouri (Tel-Aviv University). We jointly designed the work including programming the algorithm and performing the analyses.
Prof Yishay Mansour, Prof. Tal Pupko and Prof. Itay Mayrose jointly supervised this work.

October 2023

Acknowledgments

I am deeply grateful to my advisor, Prof. Yishay Mansour, for his guidance and support during my time as a researcher at Tel Aviv University. From my early days as an undergraduate working on the workshop project that later became this thesis, to the completion of my Master's degree, Prof. Mansour's unwavering commitment and constructive feedback have been instrumental in shaping my growth in the school of computer science.

I would also like to express my appreciation for the collegiality and collaboration of my fellow researchers, Michael Alburquerque and Dr. Dana Azouri. Together, we have turned our years of work into a memorable and meaningful experience with cherished friendships.

Furthermore, I extend my gratitude to Professor Itay Mayrose and Professor Tal Pupko for their insightful critiques, valuable insights, and helpful recommendations that have greatly enhanced my research pursuits.

Finally, I owe a debt of gratitude to my family and friends for their unwavering love, support, and faith in my abilities throughout my academic journey at Tel Aviv University. Their encouragement and belief in me have been a boundless source of motivation and inspiration.

Abstract

The computational search for the maximum-likelihood phylogenetic tree is an NP-hard problem. As such, current tree-search algorithms might result in a tree that is the local optima, not the global one. Here we introduce a paradigm shift for predicting the maximum-likelihood tree, by approximating long-term gains of likelihood rather than maximizing likelihood gain at each step of the search. Our proposed approach harnesses the power of reinforcement learning to learn an optimal search strategy, aiming at the global optimum of the search space. We show that our reinforcement learning-based method obtains similar likelihood scores to those obtained by state-of-the-art techniques when analyzing empirical data containing dozens of sequences. Notably, this performance is attained without the need to perform costly likelihood optimizations apart from the training process, thus potentially allowing for an exponential increase in runtime. Our findings highlight the promising direction provided by reinforcement learning in addressing the challenges of phylogenetic tree reconstruction.

Table of Contents

Contents

Introduction

A phylogenetic tree is a hypothesis regarding the evolutionary relations among the studied sequences or organisms. Reconstructing a phylogenetic tree for a group of organisms is a fundamental challenge in evolutionary research since Darwin's time. Inferred phylogenies hold a great amount of information regarding the underlying evolutionary process, and their accurate inference is critical for numerous downstream analyses spanning molecular evolution, ecology, and genomics. Leading approaches for phylogeny reconstruction rely on probabilistic evolutionary models that describe the stochastic processes of nucleotide, amino-acid, and codon substitutions[1]. Under the maximum-likelihood paradigm of phylogeny reconstruction, the tree topology, its associated branch lengths, as well as parameters that dictate the substitution rates and the site-specific evolutionary rates are optimized for a given multiple sequence alignment (MSA). Notably, the number of possible tree topologies increases super-exponentially with the number of sequences. When only a few dozen of sequences are analyzed, there are already billions of alternative phylogenetic tree topologies that could potentially describe their evolutionary relationships, rendering the search for the best tree algorithmically challenging.

The computational search for the maximum-likelihood tree topology was previously shown to be NP-hard[2]. Thus, tree-search methodologies rely on a specified heuristic strategy, which must balance accuracy and running time. At present, heuristics employed by the community depend on the intuition of the algorithm developers and their expert knowledge regarding the phylogenetic search space. These are usually based on a hill-climbing rearrangement algorithm that defines neighboring trees[3]. Typically, a search algorithm begins from an initial tree and iteratively replaces the current one via rearrangement to a neighbor with a higher likelihood, until no better neighbor can be found. This procedure results in a tree that is locally better than all its neighbors, but this tree might not be the global optimum. In order to increase the probability of finding the global optimum tree, several techniques have been considered, e.g., initiating the search from multiple starting points, applying simulated annealing (that accepts suboptimal moves with a certain probability)[4], and employing genetic algorithms (in which different areas of the search space are being explored through the use of crossover, mutation and selection operators on a population of candidate solutions)[5]. Our aim here is not to devise a specific novel search strategy, but rather, devise an artificial intelligence (AI) framework, which automatically searches among alternative strategies (policies hereafter) for an optimal one. This framework views the strategy as an evolving entity, which is continuously optimized based on experience, i.e., training data.

Machine learning has been applied to multiple tasks in biology, including molecular-biology, evolutionary, and ecology research[6–15]. Reinforcement learning (RL) is a subfield of machine learning that is focused on learning to optimize long-term goals. Over the years RL has had many successes, from playing backgammon (in the 1990's) to playing Go and Attari games (in recent years). RL applications are usually modeled as a Markov decision process, in which an *agent* (an RL learner) interacts with its *environment* (the representation of the problem space) in discrete time steps. In each step, the agent chooses an *action* (a move) to be taken given the current *state*. The transition between the current state and the next state is influenced by the agent's action, which can be deterministic or stochastic. Another component of RL is a numeric feedback termed 'the reward', which depends on the current state and on the action taken. Given a sequence of rewards, the return function aggregates multiple rewards to one objective criterion, which implicitly defines the

goal of the learner (to maximize 'the return'). Popular *returns* include the finite horizon return, which considers only the first H returns, for some parameter H, and the discounted return, which weights the reward at time t by γ^t for some discount factor $\gamma < 1$.

The learner's main task is to select actions that would maximize the return. This is done through a *policy*, which is a mapping from states to actions, such that given a current state the policy selects an action. For any given policy, a *value function* can be computed, which is the expected return from each state. Our task is to identify the optimal policy, which induces an optimal value function. It is well known that there always exists a deterministic optimal policy. We will consider *terminating environments* in which the interaction terminates eventually. The sequence of states, actions, and rewards from the start state until termination is called an episode.

The RL characteristics of exploratory search and delayed reward, together with the capability of optimizing a sequence of actions (i.e., a policy) for the task of interest, distinguishes RL from other domains of machine learning[16,17]. Accordingly, RL is beneficial in environments where the task is aimed at reaching a winning state at the end of a procedure, rather than aimed at optimizing some myopic (or immediate) reward. When applied to the task of phylogeny inference, RL should allow taking non-greedy steps, which nevertheless, should allow reaching the optimal tree in the fewest number of steps. This is equivalent, for example, to allowing a chess player to make apparently suboptimal moves, such as sacrificing the queen, in order to win the game in the following several moves. Generally, a low (immediate) reward may still lead to an optimal terminal state. Thus, to develop and characterize a full RL algorithm for phylogenetic tree inference, it is necessary to design the algorithm based on a long-term plan by taking into account not only the immediate rewards, but more importantly the future ones.

Setting an RL representation of the phylogenetic tree search dynamics, requires a tailored representation of both the tree topology with branch lengths estimates and the possible actions to be taken, as well as deriving a meaningful immediate reward function. In the phylogeny context, the reward is based on the likelihood change resulting from a local modification of the tree, i.e., the log-likelihood difference between the next and current state (termed hereafter 'likelihood score'). Likewise, a value function considers the entire search path, namely the estimated likelihood scores of subsequent moves.

The RL-based algorithm introduced in this study is based on optimizing a policy for phylogenetic-tree search, with the aim to identify the optimal tree for a given MSA in terms of its likelihood score, relying on previous AI techniques to estimate the likelihood function without actually calculating it[7]. We modeled the phylogenetic tree search problem in a similar way as RL navigation to the highest point on a grid. Defining the grid (environment) as all possible topologies, the state of the agent is the current location, and the action is the move to a new location. The transitions in the environment are deterministic. An optimal policy of an agent would therefore be to reach the topology with the maximum likelihood-score, while taking the minimal number of steps. To account for the length of the path the agent takes till reaching the optimal topology, a discount factor γ is set (see Equation 1 in Methods). The discount factor in RL determines the weight the agent assigns to rewards in the distant future relative to those in the immediate future. If $\gamma = 0$, the agent will be completely myopic and only optimize an immediate reward. If $\gamma = 1$, the agent will evaluate each of its actions equally, based on the sum of all its future rewards, thus aiming to

reach the optimal configuration but disregarding the number of steps. In the implementation described here, a discount factor $0 < \gamma < 1$ was used as a hyper parameter, which provides an algorithmic incentive to reach the higher likelihood topologies earlier in the trajectory, such that the optimal policy (given a certain γ value) corresponds to finding the shortest path from the initial topology to the final topology. The discounted cumulative reward is updated recursively according to Bellman's equation[18] and is estimated using a Q-network[19].

The Q-network, and thus the proposed RL framework, should receive as input a vectorized representation of the states and actions in order to estimate the value function. The representation we designed to suit these requirements is based on a set of tree features that were previously shown to effectively characterize a state-action pair in the context of a likelihood-based phylogeny reconstruction[7]. Specifically, we represented each state-action pair the agent came across during training and testing by calculating 27 features that are based on: the current tree topology, its branch-lengths estimates, and a certain subtree pruning and regrafting (SPR) move[20] to a neighboring phylogenetic tree, by pruning a subtree from the current tree and regrafting it to the remaining tree (see Methods). After each move, i.e., an SPR modification to the current tree, the agent arrives to a new location in the tree space until reaching a predefined end of an episode (see Fig. 1 for a schematic flowchart of the RL framework applied in this study).

The goal of an RL agent is to learn a policy that would make optimal decisions in any given state of the environment. The optimization is performed during a training phase in which an agent plays numerous episodes, allowing it to collect relevant observations (i.e., transitions). That is, by exploring the dynamics of the environment the agent learns the optimal mapping between states and actions for maximizing the long term reward signal[17]. An important issue that needs to be tackled when developing an RL algorithm, as opposed to other types of learning, is how to balance the known trade-off between exploration and exploitation during the training phase. That is, in order to reach beneficial surfaces of high likelihood, the agent has to exploit the good transitions in the tree space it had already experienced. On the same time, it also has to explore unseen transitions, perhaps some that decrease the immediate likelihood gain, in order to make better selection of actions in the future. This was tackled using a known RL technique to sample an action based on its predicted benefit, while allowing some exploration.

Here, we developed an RL strategy for the task of searching for the maximum-likelihood phylogeny. Our method introduces novel approaches for tackling the NP-hard problem of maximum-likelihood tree search by optimizing the exploration strategy itself, which inherently considers suboptimal steps to be taken if they are expected to be beneficial in the long run. Additionally, our method does not require the direct time-consuming calculation of the likelihood function in order to predict an optimal tree. Furthermore, the computational resources needed for using this approach for phylogeny prediction are hardly influenced by the input sequence length. In the following, we first study the potential benefit of looking beyond a single step when using the classic hill-climbing optimization strategy, and demonstrate that taking suboptimal moves can regularly lead to better trees in a subsequent step. Then, a framework based on deep-Q-learning[19] for predicting the optimal tree for a given MSA is introduced. We demonstrate the application of the developed method on a set of unseen data, i.e., on unseen RL environments defined by nucleotide MSAs of up to 20 sequences. Importantly, both training and testing rely on empirical data, which were previously shown to be more challenging for phylogeny reconstruction compared

to simulated data[21–24]. Our results show that for this search space, the likelihood scores of the inferred phylogenies are comparable to those obtained from widely-used methods. We then explore the feasibility of applying an agent that was trained on a certain data size on different sizes of the search spaces.

Results

The potential benefit of a non-greedy search strategy for phylogenetic reconstruction

The strength of RL lies in its ability to take actions that are suboptimal in the short term for optimizing a long-term reward. To assess the potential benefit of RL in the context of phylogeny-tree search we examined a large number of two-step trajectories and computed the percentage of moves in which choosing two consecutive greedy moves (i.e., as in the greedy approach) would lead to lower likelihood score than choosing a non-greedy move, followed by a greedy one. To this end, for a set of 13,200 starting trees, we generated all possible 1,082,400,000 two-step trajectories. For each starting tree, we located the best tree (i.e., the best two-steps neighbor) in terms of the likelihood score. We then quantified the fraction of starting trees for which the greedy approach was not optimal. This analysis revealed that the greedy approach was suboptimal in 33% and 41% of the cases for datasets of size 7 and 12, respectively. Interestingly, some of the intermediate moves that led to trees with higher likelihood than the greedy approach were among the worst possible first moves (Fig. 2). Although the analysis was not prolonged for more than two steps ahead, this result implies that the strict stepwise greedy optimization is not necessarily the best strategy to traverse the tree topology space, even when the search space is rather limited.

Performance evaluation of the proposed RL framework

We developed a tree-search framework that is entirely based on RL, and tested its performance. For training the RL model, we assembled a large collection of transition data from a database of empirical MSAs[7] (see Methods). We defined transition data as all the observed shifts from a certain state-action pair to an adjacent state-action pair, together with the corresponding likelihood estimates. We first focused on relatively small datasets, containing at most 12 sequences (i.e., a space size of up to ca. 10^9 topologies). For these datasets, for each state the agent came across, we explored all possible immediate SPR moves during training and testing. For larger datasets that contained 15 and 20 sequences (i.e., a space size of up to ca. 10^{20} topologies), we restricted the range of possible actions from each state in order to make the training of agents feasible for the scope of this proof-of-concept study (see Methods).

The RL algorithm aims to optimize the entire search path from the starting tree to the global maximum. Therefore, throughout this study we measured the agent's performance at the end of an episode according to the improvement the agent achieved relative to improvement obtained by RaxML-NG[25] from the same starting tree (see Methods). Thus, an agent that achieved the maximal observed improvement received a score of 1, while an agent that achieved an improvement of 150 likelihood points relative to the starting tree, but 50 likelihood points less than the estimated global maximum, received a score of 0.75.

Typical examples of the likelihood improvement as the agent progresses in the search space is shown in Fig. 3. In these examples, we compared the trajectory of a trained agent to a hill-climbing fully-greedy strategy (i.e., evaluating the log-likelihood of all possible neighbors), and to the

maximum-likelihood score obtained by running RaxML-NG (i.e., the final likelihood only), when all three searches were initiated from the same random tree. Three different examples are presented, where: (a) the RL agent did not reach the best-known tree; (b) the RL agent discovered the optimal tree, while taking fewer moves than the fully-greedy procedure (five compared to six moves) by taking suboptimal moves; (c) the RL agent converged to a better tree than the greedy search.

Accuracy for datasets of relatively small size

We evaluated the performance of the trained RL model on unseen test data. First, we examined data composed of seven sequences. For this challenge of searching in a space of size 10^3, both the heuristic software (i.e., RaxML-NG) and our RL model converged to the optimal tree with an average accuracy of 0.99999 (95% confidence interval of 0.999 - 1). We next evaluated the performance on a much larger search space, such as that defined by MSAs containing 12 sequences (i.e., search space of size 654,729,075 topologies). The average accuracy score of this trained model was 0.969 (95% confidence interval of 0.945 - 0.993). This indicates that the trained RL agent successfully learns a search strategy that can be well generalized for empirical datasets of various sources.

The above results were obtained with 10 datasets for generating the training observations and 2,000 training episodes. These values were selected by analyzing the dependence between the prediction accuracy on the validation set and the number of MSAs used to generate the training data, focusing on datasets with 12 sequences. To this end, we increased the number of different empirical datasets based on which we generated the training observations from 1 to 10, 20, and 30 (but keeping the total number of episodes and transitions constant) and compared the performances (Supplementary Fig. 1). This analysis indicated that using only a single dataset for learning is significantly inferior to all other sizes (P-value < 0.03 for one-way ANOVA test for the means), but using more than 10 datasets does not significantly improve the performance (P-value > 0.64 for one-way ANOVA test for the means when comparing 10 to 20 and 30 datasets). Consequently, 10 different empirical datasets were used to collect the training data. To further investigate the main factors affecting the performance of the RL agent, we sought to investigate the impact of the number of episodes in the training phase on the validation accuracy. The accuracy increased as a function of the number of episodes (P-value < 0.004; Pearson correlation coefficient for testing non-correlation of the means), reaching a plateau at around 2,000 episodes. Though the increase in accuracy was statistically non-significant when increasing the number of episodes in the range between 1,500 to 5,000 (Supplementary Fig. 2; P-value > 0.37 for one-way ANOVA test for the means), the best accuracy was obtained when 2,000 episodes were used during test. To balance runtime and accuracy, the results across the entire analyses are presented using 2,000 episodes and 10 distinct empirical MSAs to generate the training data.

RL for large search space

Search spaces of datasets containing 15 and 20 sequences are of size 7.9×10^{12} and 2.2×10^{20} topologies, respectively. For these datasets, features extraction of all possible neighbors of a given tree, either at the learning stage or when searching for the best tree, is computationally demanding. Thus, we limited the number of considered neighbors of a given state by applying a restriction on the SPR moves, considering only local changes in the tree topology as commonly performed in various tree search heuristics[7,26,27] (see Methods). When applying this procedure both in training and in testing, the average performance of the trained models for 15 sequences was 0.999 (95% confidence interval of 0.998 - 1.001). This suggests that narrowing the range of possible neighbors should be considered as a technique for training RL agents and inferring the phylogenies for datasets with large phylogenetic search spaces. When datasets with 20 sequences were considered, the test

accuracy was lower, i.e., the average test performance was 0.89. We speculate that this performance could be improved using alternative, more exhaustive, data collection methodologies (see Discussion).

Employing pre-trained agents across data sizes

Our learning so far concentrated on RL training on datasets of specified size. To assess the potential of using agents that were trained on a specific dataset size to solve the phylogenetic search problem for varied number of sequences, we sought to apply zero-shot testing[28]. Specifically, we investigated the predictive power when testing pre-trained agents of up to 20 sequences on datasets with fewer sequences and found comparable performance (Table 1). For example, the performance of a zero-shot agent trained on datasets containing 15 sequences on unseen environments of datasets containing 12 sequences obtained an averaged accuracy score of 0.973, which is slightly better than that obtained by an agent that was trained and tested on an environment of 12 sequences (average accuracy of 0.969). Overall, this analysis indicates that a transfer between environments of different sizes does exist and that this approach could potentially assist in solving varied phylogenetic-search space environments.

Running times

In this study, we focused on developing the conceptual aspects of RL-phylogenetics, and as part of this we developed a prototype implementation. This prototype did not undergo cycles of optimization, e.g., a large portion of the computational runtime is devoted to feature extraction, which in the current version is implemented inefficiently in Python. For 15 sequences, for example, the training of an RL agent took 600 CPU hours. However, once the agent is trained, the time required to predict the optimal tree takes a few seconds. Specifically, we compared the runtime required to reconstruct the optimal tree to that of RaxML-NG. For datasets with 15 sequences running the trained agent took 8.7 s on average (ranged between 8.4 and 9.3 seconds), of which, 7 s for extracting the features, and 1.7 s for all other computational tasks, e.g., estimating the Q function. Noticeably, the running time does not depend on MSA length (Fig. 4). For the same datasets, the likelihood computation of RaxML-NG took 8.7 s on average, but these varied widely from less than half a second for short MSAs (up to 800 base pairs) to 18 s for very long ones (more than 16,500 base pairs). The same trend was observed when datasets with 20 sequences were considered (Fig. 4).

Discussion

For many biological domains, spanning diverse fields such as ecology, genomics, systematics, and epidemiology research, an accurate inference of the underlying phylogeny is indispensable. As such, the development of more accurate phylogeny-reconstruction techniques is an ongoing effort that continuously progressed with the type and size of data analyzed, the computational resources available, and algorithmic developments. Numerous computational techniques were imported from the fields of statistics and computer science to improve phylogenetic tree reconstruction. These include treating character evolution as a Markov process[29], Branch-and-Bound[30], Markov chain Monte-Carlo[31], genetic algorithms[5], simulated annealing[4], and more recently, machine learning[7,11,12]. Despite these improvements, commonly used algorithms still lack the ability to provide an optimal solution. In this study, we propose an out-of-the-box AI approach for phylogenetic reconstruction, namely, reinforcement learning.

The idea of introducing RL algorithms to the task of finding the optimal phylogenetic tree is based on the concept of optimizing a strategy for the tree search, rather than incrementally optimizing the likelihood gain within a series of steps. RL includes several aspects that together could prove particularly beneficial to phylogeny inference. First, similar to simulated annealing, it allows taking suboptimal steps as part of the search strategy. Our results above demonstrate that this often enables more efficient convergence to optimal trees. Second, and unlike any other existing approach, our algorithm directly optimizes a policy based on empirical training data, without the need of predetermined heuristics. This means that an agent can decide to be greedy or to take suboptimal moves according to the specific characteristics of the data and the specific position in the tree space. Third, our agent moves without optimizing the likelihood directly, potentially reducing running time, especially for long sequences.

When a phylogenetic tree is provided as input to machine learning algorithms, it must be represented as a vector. In a recent study we represented a tree and its SPR neighbors as a vector of 19 features, and showed that we could predict optimal SPR moves without computing the likelihood function[7]. In this work, we exploited such tree representation technique for training an RL agent, which can successfully traverse previously unseen phylogenetic spaces of empirical datasets. This study could thus serve as a benchmark for different representations of a phylogenetic search space, which reportedly until current days, was missing. We expect that additional improvement in representing trees and alignments would further improve RL-based tree search.

Two recent studies employed RL to phylogeny[32,33]. In these studies RL was used in the context of distance-based methods, which are known to be faster, albeit less accurate than likelihood-based methods[34,35]. These studies and ours showed the potential of using RL for the complicated tree search problem and emphasized the challenge of training an agent on topological spaces of more than 20 sequences. While our algorithm and representation are not theoretically limited by the number of sequences in the data, at present, accuracy was not satisfactory for datasets with more than 20 sequences. We believe that promising future directions towards the application of RL to large datasets should concentrate on the following: (1) Improving the training data. This includes the size of the training observations, as well as its quality. Increasing the number of training examples necessitates training the agent longer, which depends on the availability of computational resources. In this regard, transfer learning should enable repeatedly using previous models trained on small datasets as the starting point for learning larger ones[36]. As for the data quality, it requires developing means to collect training observations of those cases that would maximize the learning of the agent. For example, by collecting more observations from regions of high likelihood, which could provide valuable information for traversing these important parts of the likelihood surface; (2) Improving code efficiency. Although the running time of the proposed methodology is hardly affected by the input sequences lengths and thus is suitable for large-scale data, a more efficient implementation with regard to feature extraction could enable better usage of the computing resources available, particularly during training; (3) Using an alternative, automatic, representation of the tree search space. For example, it has been recently proposed to represent tree topologies with embedded node features based on graph neural networks[37]. This direction of extracting learnable topological features can potentially better capture the complexity of empirical phylogenetic environments, without requiring to hand-craft additional features.

Another possible direction for improving the effectiveness of RL for phylogenetics could be considering alternative immediate reward functions, e.g., directly calculating the likelihood-function as the reward during inference instead of estimating the likelihood change. Additionally, while in this work we considered SPR actions only, the combination of complementary neighborhood definitions for local-search phylogenetic-reconstruction algorithms, such as nearest-neighbor interchange (NNI)[38] and tree bisection and regrafting (TBR)[39], could be considered when modeling the tree search dynamics. Expanding the range of possible actions could thus help the agent fine tune the search strategy when it is in low or high likelihood regions. Lastly, there are a large number of variants of RL algorithms. As part of the development of the current implementation, we have examined the applicability of alternative RL-based schemes, e.g., policy-based algorithms. This procedure demands more computation resources and was attempted for relatively small data sizes of up to 12 sequences. However, other existing RL frameworks could prove beneficial for the task of phylogenetic reconstruction.

The main conceptual novelty of our approach is to view phylogenetic tree reconstruction as a dynamic game, in which the rules are specified, but the winning strategy is unknown and difficult to optimize. In such a case, better inference is obtained following numerous games generated in-silico. We expect that, with time, RL will be introduced for additional evolutionary genomics optimization problems, including multiple sequence alignment, synteny inference, and elucidating complex patterns of population dynamics.

Methods

A reinforcement-learning algorithm for predicting the maximum-likelihood phylogeny

The environment

We defined the environment using $(S,\ A,\ R)$, where S is the state space, i.e., all possible trees given a set of aligned sequences, and A is the set of possible actions, i.e., all possible SPR moves given a tree topology. R is the immediate reward function following a transition from state s to s'. In our setting, (s, a) deterministically determines the next state s'. $R(s, s')$ is defined as the log-likelihood difference, scaled by LL_{NJ} (the log-likelihood of the reconstructed BioNJ[40] tree as implemented in PhyML 3.0[41]) so that the reward function would have the same magnitude across different datasets: $R(s, s') = \frac{LL_{s'} - LL_s}{LL_{NJ}}$.

The Features

Each state-action pair (s, a) is represented by a set of 27 phylogenetically informative features from an input tree (Table 2). The feature vector, $\phi(s, a)$, captures properties of the current state (the topology and its branch lengths) and the action (one possible SPR move). Of these, 19 were previously developed in the context of predicting the optimal neighbor as part of a tree search[7] and capture, for example, features related to the topological differences between the starting and resulting trees and properties related to their branch lengths. Eight additional features were implemented in this work and are based on non-parametric bootstrap computations[7].

The algorithm

The developed RL algorithm is based on a Deep Q-network (DQN)[19], a model-free and off-policy RL algorithm. In the DQN setting, the agent learns a value function, named the quality function $Q(s, a)$, which represents the estimated benefit of a specified action in gaining some future reward, given a

certain state. More specifically, we implemented a neural-network Q_θ (with weight parameters θ) that predicts the quality function of a state-action pair, given the feature vector $\phi(s,a)$. This predicted value is termed $Q(s,a)$ and is explained in more details below. Starting from state s we estimated $Q(s,a)$ for all possible SPR actions and chose the action with maximal $Q(s,a)$, which defines the next state (Algorithm 1). The number of unique state-action pairs to be computed when conducting a move is $2(n-3)(2n-7) = O(n^2)$ where n is the number of sequences in the input MSA[39]. The starting state for each episode, s_0, was randomly sampled (using RaxML-NG[25] random tree generator), such that the agent could start the trajectory from anywhere in the tree space.

The model

The main strength of Q-learning lies in its ability to construct a policy that maximizes the commulative reward[17]. In deep Q-learning, neural networks are trained to estimate the value of the Q function for unseen states and thus it combines Q-learning with a deep artificial neural network (ANN). The recursive form of the optimal return function, known as the Bellman equation[18] is:

$$Q^*\big(\phi(s_t,a_t)\big) = r_t + \gamma \max_a Q^*\big(\phi(s_{t+1},a)\big) \quad (1)$$

Where Q^* is the optimal state-action value function, and r_t is the immediate reward obtained at time-step t. The γ hyperparameter is the discount-rate, a constant $0 \leq \gamma \leq 1$, which weights rewards from the uncertain far future less than the ones in the fairly confident near future. That is, a reward received *k* time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. Of note, there exists a $\gamma = 1-\epsilon,\ \epsilon > 0$, for which the optimal policy corresponds to the shortest path from the starting topology to the topology with the highest likelihood (up to H SPR moves away).

We would like the agent to learn a policy that does not involve taking an unlimited number of actions, i.e., we would like to balance the runtime with the improvement in likelihood. This balance is controlled by the horizon hyperparameter, denoted as H, which specifies the number of actions taken from the starting tree (Algorithm 2). Importantly, the specific γ value inspires a different optimal policy (see Supplementary note 1). Of note, each search stops after a predefined number of H steps. This is termed an episode. As stated above, the total number of episodes during learning is also a hyperparameter. Following each episode, the network weights are updated. Specifically, as in standard DQNs, we used the experience replay method to hold the agent's training trajectories, i.e., a buffer of a pre-determined size containing transition observations (state-action pairs together with their rewards and next state-actions). At the end of each episode, H new memories are added to the buffer (and the H oldest memories are discarded), and the ANN is trained based on a batch of trajectories sampled (with replacement) from the memory buffer 'time-to-learn' times (Algorithm 2). The sizes of the memory buffer and the batch, as well as the 'time-to-learn', are hyperparameters of the algorithm.

Additional hyperparameters are related to the deep network architecture and the learning dynamics. These include the number of fully-connected hidden layers, the number of neurons in each layer, the activation function, the loss function, the optimization algorithm, and the learning rate (Supplementary Table 1).

To control the exploration-exploitation tradeoff during training we allowed the agent to take suboptimal moves with respect to the Q function. We used the SoftMax exploration strategy that selects an action a based on the following probability: $\frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')}}$. This allows greater value actions to be selected with greater chance, yet permitting some randomness. T is a hyperparameter that controls the level of exploration.

We implemented the ANN in Python using PyTorch[42]. The above hyperparameters were optimized via Optuna framework[43], which is an automatic hyperparameter optimization package, particularly designed for machine learning (summary of the model hyperparameters values and further details are described in Supplementary Table 1).

The performance metric

To measure the performance at the end of an episode, we computed the following metric: let LL_{gain} be the log-likelihood difference between the final tree and the starting tree. Let LL_{rax} be the log-likelihood difference between the maximum-likelihood tree and the starting tree, as obtained by executing RAxML-NG from the same starting tree. Both resulting topologies were subject to branch-lengths optimization. The ratio between these two terms (LL_{gain}/LL_{rax}) is a number that reflects the improvement in likelihood score achieved by an agent, and can be used to compare the performance between different datasets.

Empirical data preparation

We selected all empirical datasets with 7, 12, 15, and 20 sequences from the training data collected in Azouri et al.[7]. These represent nucleotide coding alignments[44], user-submitted phylogenies from TreeBase[45], plant phylogenies reconstructed using the OneTwoTree pipeline[46], and genomic sequences that were aligned according to the tertiary structure of its encoded proteins[47]. For each dataset size (i.e., the number of sequences in the alignment), 30 MSAs were randomly fixed as validation data, 10 MSAs were randomly fixed as test data, and from the rest, 10 datasets (unless otherwise specified) were randomly sampled to generate the training samples.

Transition data collection

To apply the memory buffer method, the transition observations (state-action pairs together with their corresponding likelihood estimates, and the corresponding next state-action pair) need to be collected through many training episodes. To this end, each episode was initiated from a random tree and the obtained reward was calculated (using RaxML-NG) and stored for each transition taken. Precisely, a model that is trained for 2,500 episodes of 20 SPR moves each is essentially trained over $25{,}000 \times 20 = 50{,}000$ training observations. The substitution rate parameters of a GTR+I+G model[21] were optimized once for each dataset, based on a reconstructed BioNJ[40] tree as implemented in PhyML 3.0[41] and were then fixed for the following likelihood calculations of the respective dataset.

When datasets of size 15 and 20 sequences were considered, the computational resource required to compute the features for all neighbors were beyond the scope of the study conducted here. Therefore, we limited the space of possible neighbors by applying a restriction on the range of SPR moves, allowing each pruned subtree to be regrafted up to a pre-defined radius. This radius

defines the number of branches in the path between the pruned and regrafted branches, not including these branches (Supplementary Figure 3). Setting a radius of four narrowed the neighborhood space to a feasible task. Additionally, when datasets with 20 sequences were considered, we applied an alternative approach to collect experiences, which proved superior to the one used for smaller data sizes (see Supplementary note 2).

Data collection for the two-step pre-analysis

We collected from the training data all MSAs containing 7 and 12 sequences (i.e., 51 and 81 datasets, respectively). Next, 100 random starting trees were reconstructed for each dataset using RAxML-NG. We then obtained all their respective 32,640,000 and 1,049,760,000 possible two-step trajectories. Precisely, we: (1) obtained all single-step SPR neighbors for each starting tree and recorded all likelihoods; (2) recorded all likelihood scores of the single-step SPR neighbors of each of the latter trees. This allowed us to identify the best two-step neighbor of each starting tree, as well as the best neighbor reached by applying the single-greedy step twice sequentially. The likelihoods throughout this analysis were computed using RAxML-NG, allowing for branch-lengths optimization. The substitution rate parameters were optimized once for each dataset, based on a reconstructed BioNJ[40] tree as implemented in PhyML 3.0[41] and were then fixed for the following likelihood calculations of the respective dataset, assuming the GTR+I+G model[21].

Continuation Work – A New Phylogenetic Tree Representation

Why Do We Need a New Phylogenetic Tree Representation?

As stated in the main discussion section, we expect that additional improvement in representing trees and alignments would further improve RL-based tree search, the continuation work described in this section tries to create such a new representation of trees and alignments. Until now, our work has relied on a specific set of 27 features that are informative for phylogenetic tree reconstruction. These features were created using domain knowledge and rely on topological differences between the starting and resulting tree of the SPR move. Our aim here is to create a new set of features that will allow a model to extract information in a more unsupervised way, as in other successful approaches in domains like vision and NLP. One of our hopes is to transfer to a representation of the phylogenetic tree alone ($\varphi(s)$) instead of the current representation that requires both a phylogenetic tree and an SPR move ($\varphi(s, a)$). This will allow us to work with a state-only representation, which is more efficient than a state-action representation that requires looping over all available actions on each step. Our goal is to utilize a useful representation of the state, which is the phylogenetic tree given a current MSA file, to later enable usage of other existing RL algorithms.

Tree To Vector – Calculating the New Representation

A phylogenetic tree's likelihood is calculated using the tree's topology and a multiple sequence alignment (MSA) holding the sequences of the tree's leaves. When trying to construct new features for the tree-reconstruction problem, which is the problem of finding the tree with the maximal likelihood, we sought to encode exactly that information. Hopefully, this will enable any model to at least accurately predict the likelihood of a given tree. We chose to encode the topology in the most straight forward way – through an adjacency matrix, representing the graph structure of the tree, where a value of the matrix in cell i, j is 1 if there exists an edge between node i and node j, and 0 if there exists no such edge. For the matrix to remain consistent over all phylogenetic trees,

we define a constant order over the nodes. There are many viable possibilities for encoding the sequences of the tree leaves, we tried to encode the required information in the simplest way possible. We assumed the simplification that the relationship between the topology and the sequences is one where a closer pair of sequences should correspondingly be a closer pair of leaves in the tree. With that intuition we encoded each pair of sequences of the MSA using some simple distance metric for matching sequences. We experimented with the hamming distance, the alignment score calculated using the Needleman-Wunsch global alignment algorithm implemented by the BioPython library, and a similarity ratio measuring the number of matching elements (characters) between the two input sequences, considering deletions, insertions, and substitutions, and then dividing it by the total number of elements in the longest string (the higher the ratio, the more similar the two strings are). We concatenated the encoding of the topology, together with the 3 different encodings of the sequence pairs to create a new feature vector representing a given phylogenetic tree.

Validation Of the New Representation

Before deploying our new feature set to the RL task, we performed several sanity tests, using simpler tasks. 19 of our 27 topological handcrafted features described in previous chapters were originally used to predict the likelihood of a given phylogenetic tree. The logic here states that we should be able to solve less complex phylogeny tasks (such as likelihood prediction) with our new representation as well.

The Regression Task – Predicting Log-Likelihood

Here we used the same model used for the DQN, described in Supplementary Table 1, as a regression model, where the goal is to predict the log-likelihood of a phylogenetic tree. We compared the original representation on this task, together with a combination of the tree-matrix representation with each of the sequence matching scores separately. We evaluated on datasets containing 12 and 7 species. For each number of species, we used 10 different datasets, The data was collected during 5000 episodes of 20 steps each, using random walks, meaning an overall of 100,000 examples of them we divided randomly 80,000 as train and 20,000 as test (train data leakage might occur). Our results on the test data can be seen at figure 5. as one can clearly see from our results – the model achieves low test loss and quickly converges, which shows potential for using this representation in more complex tasks.

The Classification Task – Classifying Actions

To compare our new representation directly with the previous representation used on the full RL task, we needed to create a state action representation for our new features as well. As described above – our new features represent a pure state, for the purpose of comparing to our old features – we added an action representation to the state representation. We created 4 one hot vectors, where each vector represents a node in the tree. One SPR move is defined by choosing 2 edges in the tree, and each edge can be defined by choosing 2 nodes, a SPR move can be defined by choosing 4 nodes at the tree. Notice not every 4-node tuple defines a legal SPR move, in fact, a vast majority of tuples don't correlate with any SPR moves for a given tree. This opens a new Binary classification task – being able to predict whether a new state-action representation defines a legal action. Here we used the same model used for the DQN, described in Supplementary Table 1, as a binary classification model to classify state-action representations into legal and illegal categories. We used 10 different datasets, each comprised of 7 species. The

data was collected during 50 episodes of 20 steps each, taking each action of the 81 possible actions at every step, meaning an overall of 81,000 examples of legal actions. We generated another 81,000 examples of illegal actions which differ only at the action representation, thus creating a balanced dataset. Of the overall examples we divided randomly 80% as train and 20% as test. Training for 100 epochs, we were able to achieve an accuracy of 96.67%, an F1-Score of 0.9677 and an area under the ROC curve of: 0.9790. Our ROC curve can be seen at Figure 6.

Results

Finally, we compare the new and old state-action representations for the RL algorithm described in this work. Our results, show convergence for both the new and old representations on 10 datasets of 7 species (can be seen in Figure 7.), and a positive trend for the new representation on 1 dataset of 20 species (can be seen in Figure 8.). However, one can clearly see the old representation presents a much faster convergence rate. These results, which persisted when we experimented with datasets of 12 species, show that while it is possible for an agent to learn an interesting policy from our new representation, it is in no way easier or more efficient than the old representation. The promise of this new representation remains in the ability to represent a pure state – and hence introduce new algorithms and methods to this problem.

Discussion – The Problem Of Transferring From $\phi(s,a)$ To $\phi(s)$

One of the key motives for creating the new representation is the ability to represent a pure state in the phylogeny reconstruction game. This should enable experimentation with many off the shelf RL algorithms. However, one major unsolved issue is the number of possible SPR moves. If we define an SPR move as a 4-node tuple, for a 7 species dataset, naively we have 12^4 actions to choose from, but only 81 of them are valid SPR moves. This creates a beyond challenging framework for learning. This problem isn't prevalent in the state-action representation setting, where we iterate over all state actions and choose whatever action we predict will be best. Meaning, for the state-action representation setting, we iterate only over legal actions. This iteration over all legal actions is heavily time consuming, but to skip it we need to find a consistent mapping of the legal 81 actions for the model to choose from, for any possible tree. Notice, if one would define a setting with exactly 81 actions, map the model's selection of the n^{th} action to whichever legal action appeared in the n^{th} location when enumerating the legal actions, the model would have no chance to learn a policy, as the first action in the current tree doesn't correspond with the first action of the next tree after an SPR move.

Data availability

The datasets contained within the empirical set have been deposited in GitHub with the identifier https://github.com/michaelalb/ThePhylogeneticGame[48].

Code availability

The code that supports the findings of this study was written in python version 3.9.7 has been deposited in GitHub with the identifier https://github.com/michaelalb/ThePhylogeneticGame[48]. Computation of likelihoods were executed using the following application versions: PhyML 3.0[41], RAxML-NG 0.9.0[25]. The ANN was implemented in PyTorch[42] version 1.13.1.

References

1. Yang, Z. PAML 4: Phylogenetic analysis by maximum likelihood. *Mol. Biol. Evol.* **24**, 1586–1591 (2007).
2. Chor, B. & Tuller, T. Maximum likelihood of evolutionary trees: Hardness and approximation. *Bioinformatics* **21**, i97-106 (2005).
3. Whelan, S. New approaches to phylogenetic tree search and their application to large numbers of protein alignments. *Syst. Biol.* **5**, 727–740 (2007).
4. Stamatakis, A. An efficient program for phylogenetic inference using simulated annealing. in *Proceedings - 19th IEEE International Parallel and Distributed Processing Symposium* (2005).
5. Lewis, P. O. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Mol. Biol. Evol.* **3**, 277–283 (1998).
6. Tarca, A. L., Carey, V. J., Chen, X., Romero, R. & Drăghici, S. Machine Learning and Its Applications to Biology. *PLoS Comput. Biol.* **3**, e116 (2007).
7. Azouri, D., Abadi, S., Mansour, Y., Mayrose, I. & Pupko, T. Harnessing machine learning to guide phylogenetic-tree search algorithms. *Nat. Commun.* **12**, 1983 (2021).
8. Ecker, N., Azouri, D., Bettisworth, B., Stamatakis, A., Mansour, Y., Mayrose, I., Pupko, T. A LASSO-based approach to sample sites for phylogenetic tree search. *Bioinformatics* **38**, I118–I124 (2022).
9. Loewenthal, G. *et al.* A probabilistic model for Indel evolution: Differentiating insertions from deletions. *Mol. Biol. Evol.* **38**, 5769–5781 (2021).
10. Abadi, S., Avram, O., Rosset, S., Pupko, T. & Mayrose, I. ModelTeller: Model selection for optimal phylogenetic reconstruction using machine learning. *Mol. Biol. Evol.* **37**, 3338–3352 (2020).
11. Suvorov, A., Hochuli, J. & Schrider, D. R. Accurate inference of tree topologies from multiple sequence alignments using deep learning. *Syst. Biol.* **69**, 221–233 (2020).
12. Zou, Z., Zhang, H., Guan, Y., Zhang, J. & Liu, L. Deep residual neural networks resolve quartet molecular phylogenies. *Mol. Biol. Evol.* **37**, 1495–1507 (2020).
13. Schrider, D. R. & Kern, A. D. Supervised Machine Learning for Population Genetics: A New Paradigm. *Trends Genet.* **34**, 301–312 (2018).
14. Haag, J., Höhler, D., Bettisworth, B. & Stamatakis, A. From easy to hopeless-predicting the difficulty of phylogenetic analyses. *Mol. Biol. Evol.* **39**, msac254 (2022).
15. Zaharias, P., Grosshauser, M. & Warnow, T. Re-evaluating deep neural networks for phylogeny estimation: The issue of taxon sampling. *J. Comput. Biol.* **29**, 74–89 (2022).
16. Szepesvári, C. Algorithms for Reinforcement Learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **4**, 1–103 (2010).
17. Sutton, R. S. & Barto, A. G. *Reinforcement learning : an introduction*. (MIT Press, 1998).
18. Puterman, M. L. *Markov decision processes : discrete stochastic dynamic programming*. (Wiley, 1994).
19. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
20. Wooding, S. Inferring phylogenies. *Am. J. Hum. Genet.* **74**, 1074 (2004).
21. Abadi, S., Azouri, D., Pupko, T. & Mayrose, I. Model selection may not be a mandatory step for phylogeny reconstruction. *Nat. Commun.* **10**, 934 (2019).
22. Abdo, Z., Minin, V. N., Joyce, P. & Sullivan, J. Accounting for uncertainty in the tree topology has little effect on the decision-theoretic approach to model selection in phylogeny

estimation. *Mol. Biol. Evol.* **22**, 691–703 (2005).
23. Huelsenbeck, J. P. Performance of phylogenetic methods in simulation. *Syst. Biol.* **44**, 17–48 (1995).
24. Edwards, A. W. F. Assessing molecular phylogenies. *Science.* **267**, 253–253 (1995).
25. Kozlov, A. M., Darriba, D., Flouri, T., Morel, B. & Stamatakis, A. RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics* **35**, 4453–4455 (2019).
26. Stamatakis, A., Ludwig, T. & Meier, H. RAxML-III: A fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics* **21**, 456–463 (2005).
27. Stewart, C. A. *et al.* Parallel implementation and performance of fastDNAml. in *Proceedings of the ACM/IEEE conference on Supercomputing* 20 (ACM, 2001).
28. Higgins, I. *et al.* DARLA: Improving zero-shot transfer in reinforcement learning. in *34th International Conference on Machine Learning, ICML* vol. 3 2335–2350 (2017).
29. Felsenstein, J. Evolutionary trees from gene frequencies and quantitative characters: finding maximum likelihood estimates. *Evolution (N. Y).* **35**, 1229 (1981).
30. Hendy, M. D. & Penny, D. Branch and bound algorithms to determine minimal evolutionary trees. *Math. Biosci.* **59**, 277–290 (1982).
31. Yang, Z. & Rannala, B. Bayesian phylogenetic inference using DNA sequences: A Markov Chain Monte Carlo method. *Mol. Biol. Evol.* **14**, 717–724 (1997).
32. Liptak, P. & Kiss, A. Constructing unrooted phylogenetic trees with reinforcement learning. *Stud. Univ. Babeș-Bolyai Inform.* **66**, 37 (2021).
33. Zhu, T. & Cai, Y. Applying neural network to reconstruction of phylogenetic tree. in *13th International Conference on Machine Learning and Computing* 146–152 (ACM, 2021).
34. Saitou, N. & Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* **4**, 406–25 (1987).
35. Ogden, T. H. & Rosenberg, M. S. Multiple Sequence Alignment Accuracy and Phylogenetic Inference. *Syst. Biol.* **55**, 314–328 (2006).
36. Karimpanal, T. G. & Bouffanais, R. Self-organizing maps as a storage and transfer mechanism in reinforcement learning. *ALA - Adapt. Learn. Agents - Work. Fed. AI Meet.* (2018).
37. Cheng, Z. Learnable topological features for phylogenetic inference via graph neural networks. in *ICLR* (2023).
38. Robinson, D. F. Comparison of labeled trees with valency three. *J. Comb. Theory, Ser. B* **11**, 105–119 (1971).
39. Allen, B. L. & Steel, M. Subtree transfer operations and their induced metrics on evolutionary trees. *Ann. Comb.* **5**, 1–15 (2001).
40. Gascuel, O. BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Mol. Biol. Evol.* **14**, 685–695 (1997).
41. Guindon, S. *et al.* New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. *Syst. Biol.* **59**, 307–321 (2010).
42. Paszke, A. *et al.* PyTorch: An imperative style, high-performance deep learning library. in *Advances in Neural Information Processing Systems* vol. 32 (2019).
43. Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* 2623–2631 (2019).

44. Moretti, S. *et al.* Selectome update: Quality control and computational improvements to a database of positive selection. *Nucleic Acids Res.* **42**, D917–D921 (2014).
45. Piel, W. H. *et al.* TreeBASE v. 2: A Database of Phylogenetic Knowledge. in *e-BioSphere* (2009).
46. Drori, M. *et al.* OneTwoTree: An online tool for phylogeny reconstruction. *Mol. Ecol. Resour.* **18**, 1492–1499 (2018).
47. Carroll, H. *et al.* DNA reference alignment benchmarks based on tertiary structure of encoded proteins. *Bioinformatics* **23**, 2648–2649 (2007).
48. Azouri, D. et al. Code & Data - The tree reconstruction game: phylogenetic reconstruction using reinforcement learning. https://github.com/michaelalb/ThePhylogeneticGame (2023).
49. Cock, P. J. A. *et al.* Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* **25**, 1422–1423 (2009).
50. Sokal, R. R., Michener, C. D. *A statistical method for evaluating systematic relationships*. *Univ Kans Sci Bull* (1958).

Figure legends

Figure 1. Modeling phylogenetic-tree search as RL framework. A schematic flowchart of the RL framework applied in this study. Given an empirical sequence dataset, the environment represents all phylogenetic trees (states), their possible single-step SPR moves (actions), and the estimated likelihood score. We first extracted feature vectors that represent a state with its actions. These were then fed into the agent's neural network, which outputs a prediction for the best action to be taken in the agent's state, taking into account both immediate and future rewards. The environment was then updated with the reward (ΔLL; the estimated likelihood change) obtained following the action conducted.

Figure 2. The ranking percentiles of beneficial suboptimal first moves. The distribution of the percentiles of first moves that led to better trees than two-step greedy moves. The box inside each violin shows the quartiles of the dataset with the white dot being the median, while the whiskers extend to show the 1.5 × IQR past the low and high quartiles.

Figure 3. Typical examples of the log-likelihood improvement as the search progresses. An example of the likelihood gain of a trained agent (blue), a hill-climbing-fully-greedy strategy trajectory (orange), and a maximum-likelihood tree obtained by RaxML-NG (green). Different panels represent different tests, on datasets containing 12 sequences (a,b) and 15 sequences (c).

Figure 4. Running time. The average inference running time in seconds (y axis) relative to the length of the sequences analyzed (x axis; 100 data points binned to 17 groups). In blue and orange are the average running times of inferring the optimal tree for datasets with 15 sequences using the RL trained agent and RaxML-NG (with the same single-random-starting point), respectively. Similarly, in green and red are the running times for datasets containing 20 sequences, of the RL agent and RaxML-NG, respectively.

Tables

Table 1. Accuracy scores of zero-shot experiments

	7	12	15	20
7	*0.999*	-	-	-
12	0.999	*0.969*	-	-
15	0.998	0.973	*0.993*	-
20	0.999	0.93	0.993	*0.892*

The table details the performance of each pre-trained agent of a certain dataset size (row) to each other smaller dataset size (column). Each cell shows the accuracy score of the trained model, averaged over the test datasets.

Table 2. The features used to represent a state-action pair

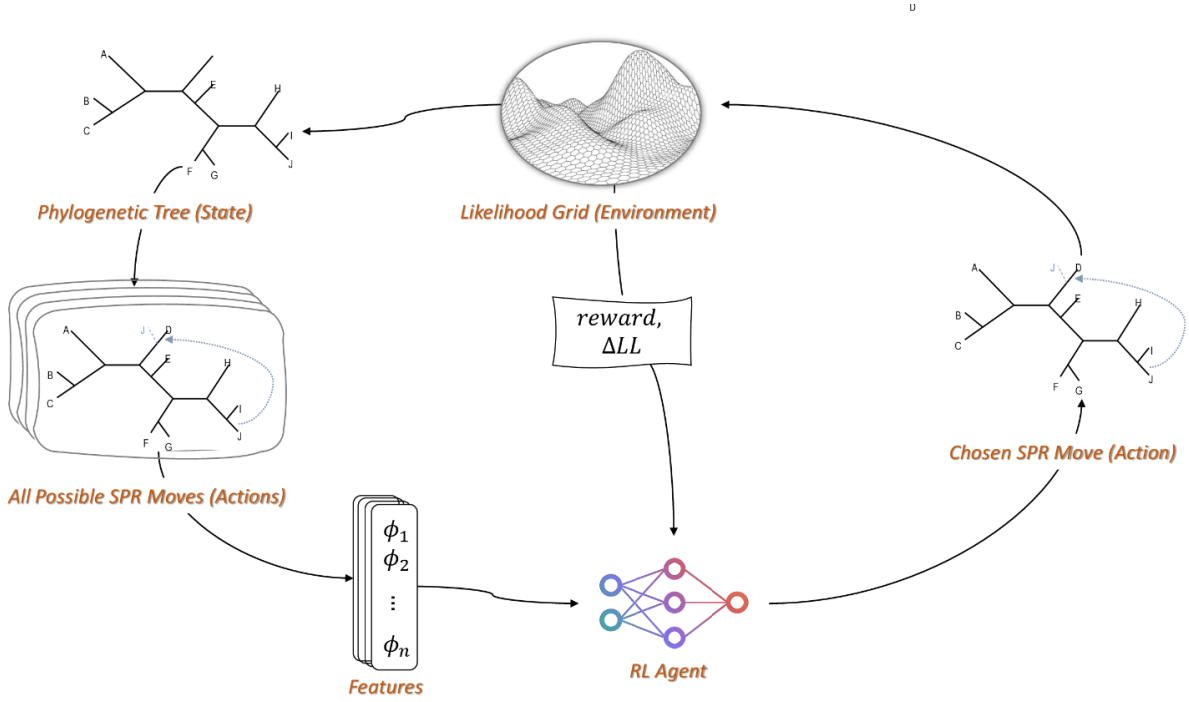
# Feature	Feature name	Details	Represented action	Tree considered
1-19	Detailed in Table 1 in Azouri et al.[7]			
20	Bootstrap – UPGMA	The approximated bootstrap support of the internal branch (that defined a split) that was being pruned or regrafted. More specifically, 10,000 bootstrapped trees were generated based on both UPGMA and NJ distance matrices, as implemented in Biopython package[49] version 1.79. Splits that do not exist in those 10,000 trees received a value of zero, while splits that lead to an external node received a value of 100.	Pruning	Current tree (s)
21			Regrafting	
22			Pruning	Next tree (s'), namely the resulting tree following an SPR move
23			Regrafting	
24	Bootstrap – NJ		Pruning	Current tree (s)
25			Regrafting	
26			Pruning	Next tree (s'), namely the resulting tree following an SPR move
27			Regrafting	

The table lists the 27 features on which the RL state-action representation is based. Features 20-23 were extracted base on the UPGMA algorithm[50], while features 24-27 were based on the NJ algorithm[40].

Figures

Figure 1.

Figure 1. Modeling phylogenetic-tree search as an RL framework. A schematic flowchart of the RL



framework applied in this study. Given an empirical sequence dataset, the environment represents all phylogenetic trees (states), their possible single-step SPR moves (actions), and the estimated likelihood score. We first extracted feature vectors that represent a state with its actions. These were then fed into the agent's neural network, which outputs a prediction for the best action to be taken in the agent's state, taking into account both immediate and future rewards. The environment was then updated with the reward (ΔLL; the estimated likelihood change) obtained following the action conducted.

Figure 2.

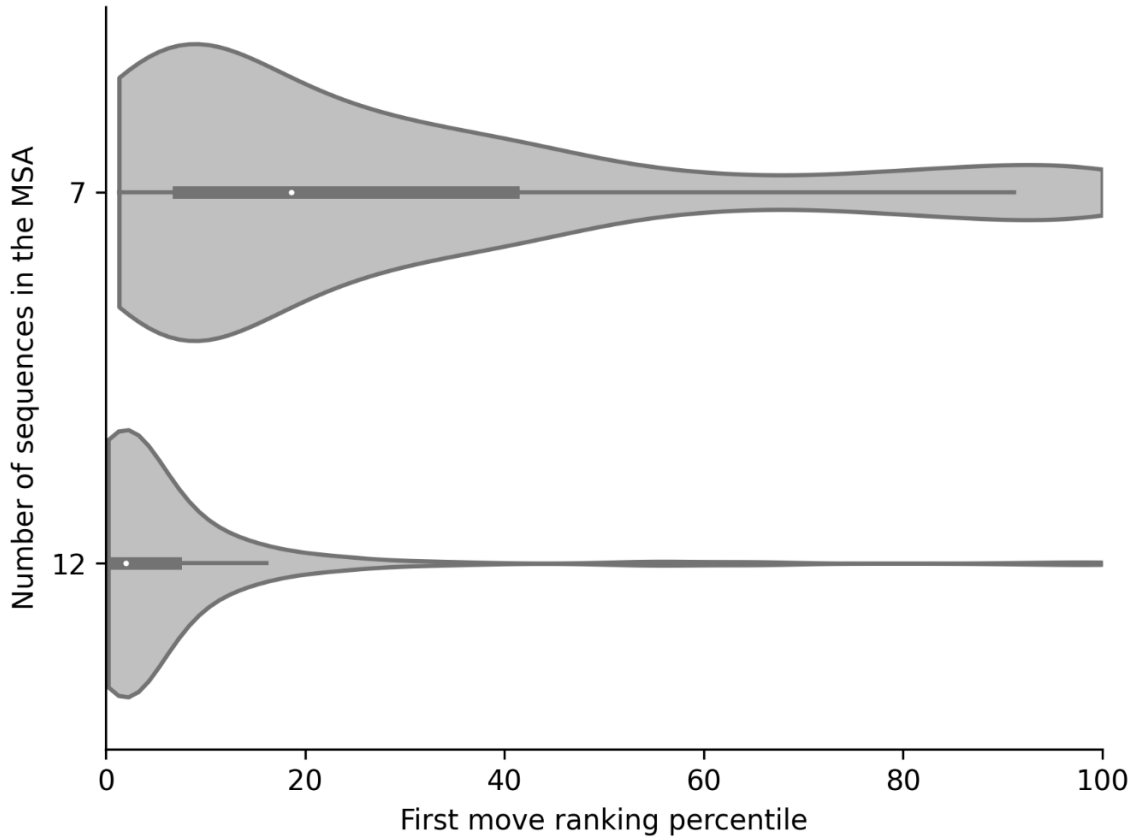


Figure 2. The ranking percentiles of beneficial suboptimal first moves. The distribution of the percentiles of first moves that led to better trees than two-step greedy moves. The box inside each violin shows the quartiles of the dataset with the white dot being the median, while the whiskers extend to show the 1.5 × IQR past the low and high quartiles.

Figure 3.

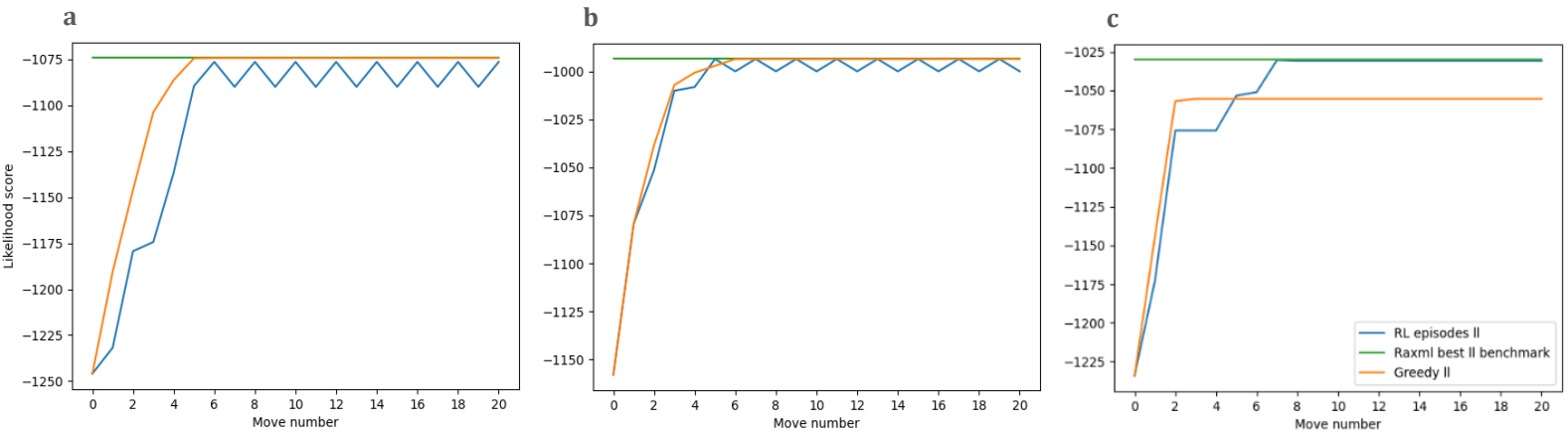


Figure 3. Typical examples of the log-likelihood improvement as the search progresses. An example of the likelihood gain of a trained agent (blue), a hill-climbing-fully-greedy strategy trajectory (orange), and a maximum-likelihood tree obtained by RaxML-NG (green). Different panels represent different tests, on datasets containing 12 sequences (a,b) and 15 sequences (c).

Figure 4.

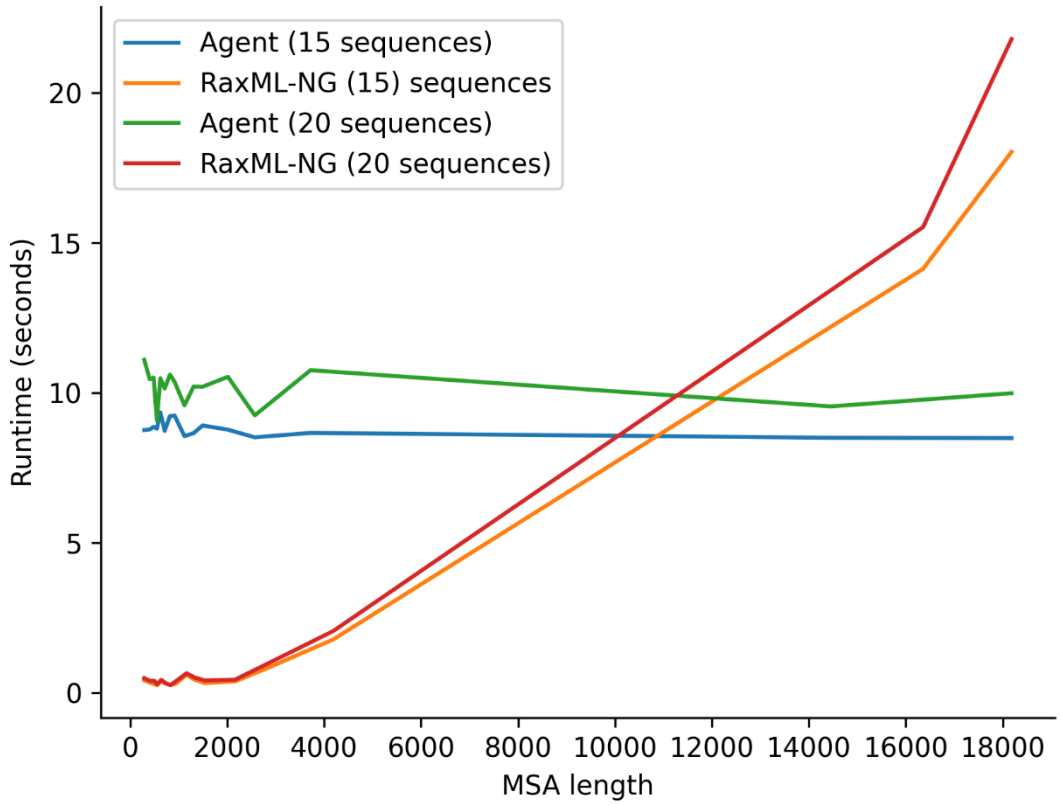


Figure 4. Running time. The average inference running time in seconds (y axis) relative to the length of the sequences analyzed (x axis; 100 data points binned to 17 groups). In blue and orange are the average running times of inferring the optimal tree for datasets with 15 sequences using the RL trained agent and RaxML-NG (with the same single-random-starting point), respectively. Similarly, in green and red are the running times for datasets containing 20 sequences, of the RL agent and RaxML-NG, respectively.

Figure 5.

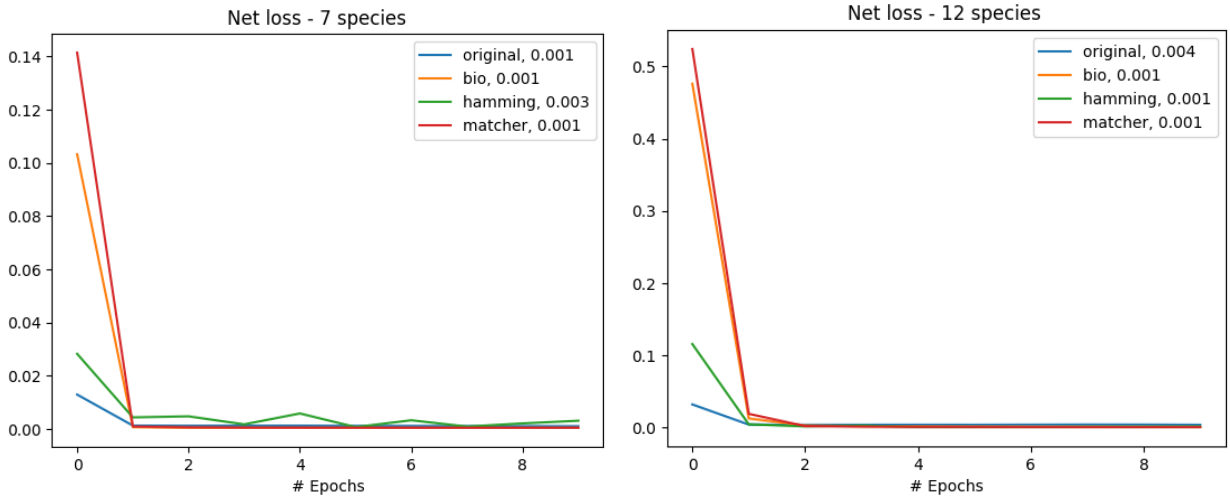


Figure 5. Regression test loss. The loss on test data (y axis) relative to the number of epochs on the train data (x axis). The data was collected during 5000 episodes of 20 steps each, using random walks, meaning an overall of 100,000 examples, of them we divided randomly 80,000 as train and 20,000 as test (train data leakage might occur here). The left figure is of datasets containing 12 species and the right figure is of datasets containing 7 species.

Figure 6.

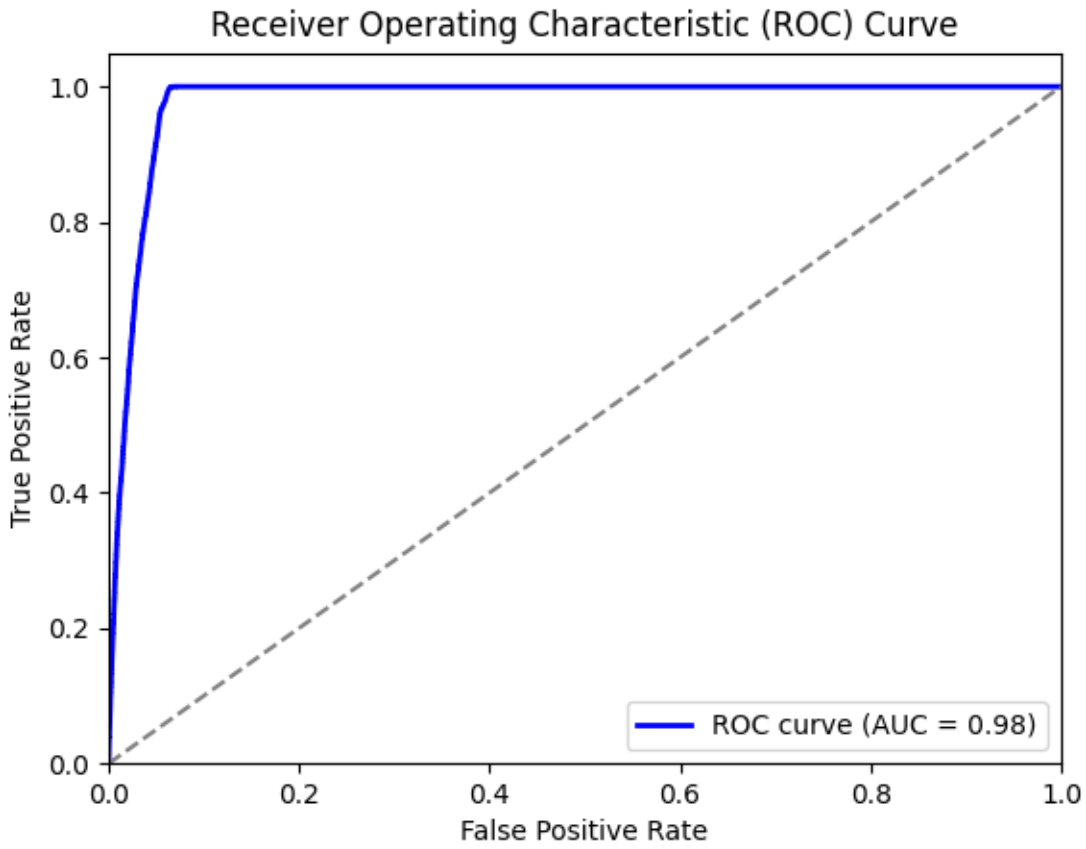


Figure 6. ROC Curve. The performance of the classification model at all classification thresholds, on test data. The data was collected during 50 episodes of 20 steps each, taking each possible action at every step, meaning an overall of 81,000 examples of legal actions. We generated another 81,000 examples which differ only at the action representation, of illegal actions. Of the overall examples we divided randomly 80% as train and 20% as test.

Figure 7.

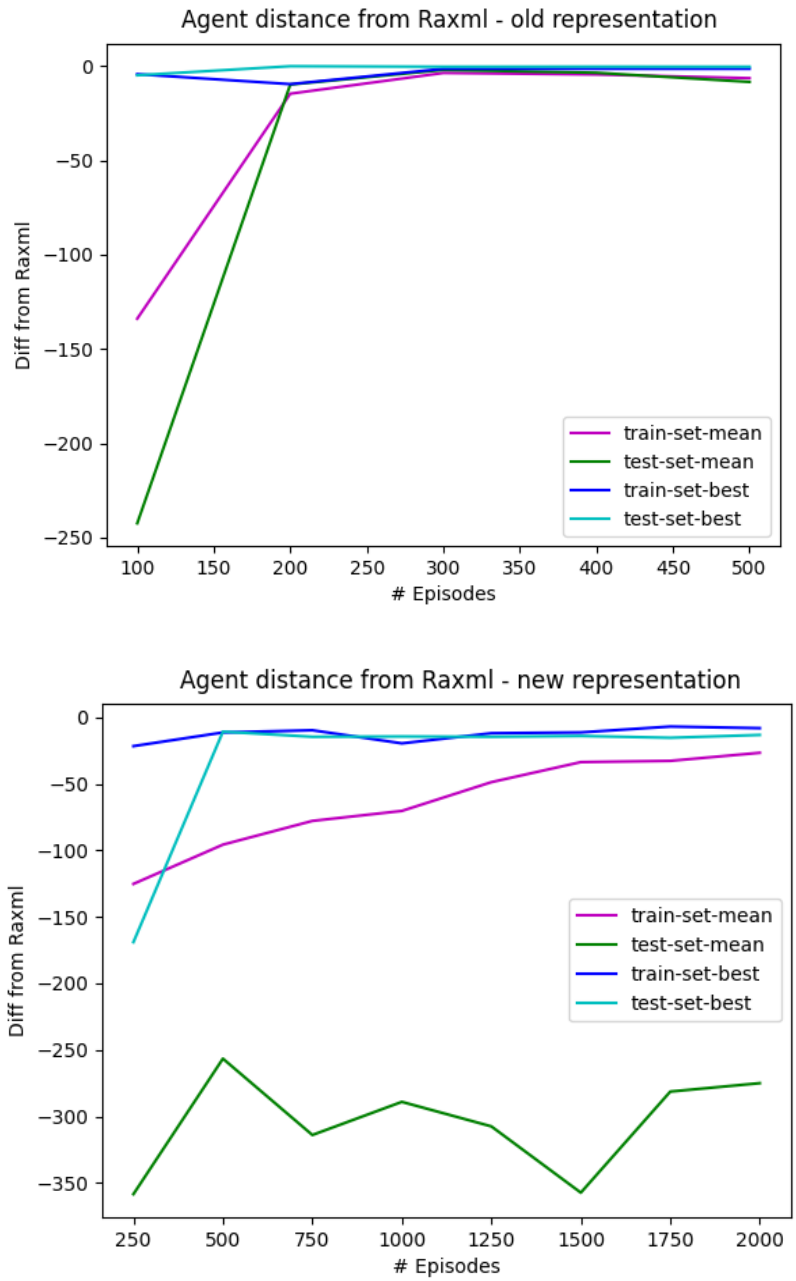


Figure 7. Comparison of agents with the original and modified representations on 10 datasets containing 7 sequences. An example of the likelihood gains of an agent during training, as compared to Raxml-ng's performance. The upper panel shows the performance with the original representation. The lower panel shows the performance with the new representation suggested at the continuation work chapter.

Figure 8.

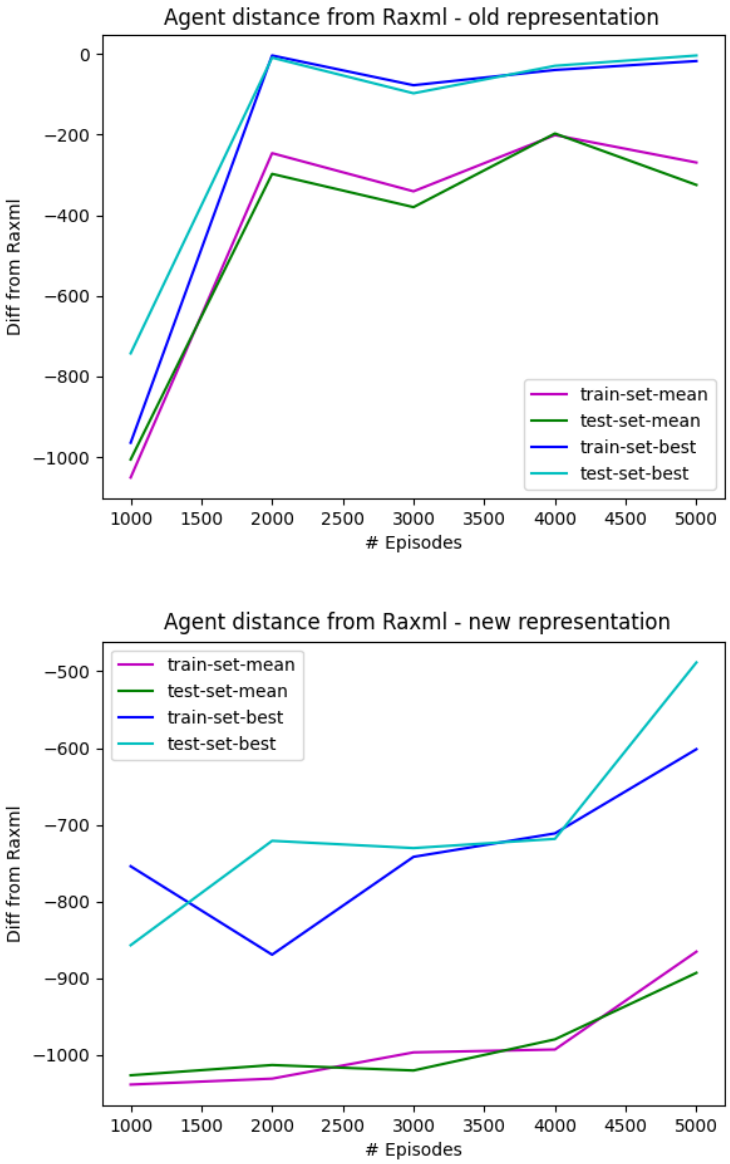


Figure 8. Comparison of agents with the original and modified representations on 1 dataset containing 20 sequences. An example of the likelihood gains of an agent during training, as compared to Raxml-ng's performance. The upper panel shows the performance with the original representation. The lower panel shows the performance with the new representation suggested at the continuation work chapter.

Algorithm 1 - inference

Algorithm 1 Deep Q-Learning of Tree Reconstruction - Inference

Given a Feature-Extractor function ϕ and an action-value function Q_θ
Sample starting tree s_0

for $h = 0...H$ **do**
  for all possible SPR-moves a in s_h **do**
    Evaluate $Q_\theta(\phi(s_h, a))$
  end for

  select $a_h = \arg\max_a Q_\theta(\phi(s_h, a))$
  Execute SPR-move a_h and observe next tree s_{h+1}

end for

return s_h

Algorithm 2 - training

Algorithm 2 Deep Q-Learning of Tree Reconstruction - Training

Given a Feature-Extractor function ϕ
Initialize action-value function Q_θ with a random set of weights θ
Initialize replay buffer $\mathcal{D}$ to capacity N

for $episode = 1...M$ **do**

Sample starting tree s_0
for $h = 0...H$ **do**

for all possible SPR-moves a in s_h **do**
Evaluate $Q_\theta(\phi(s_h, a))$
end for

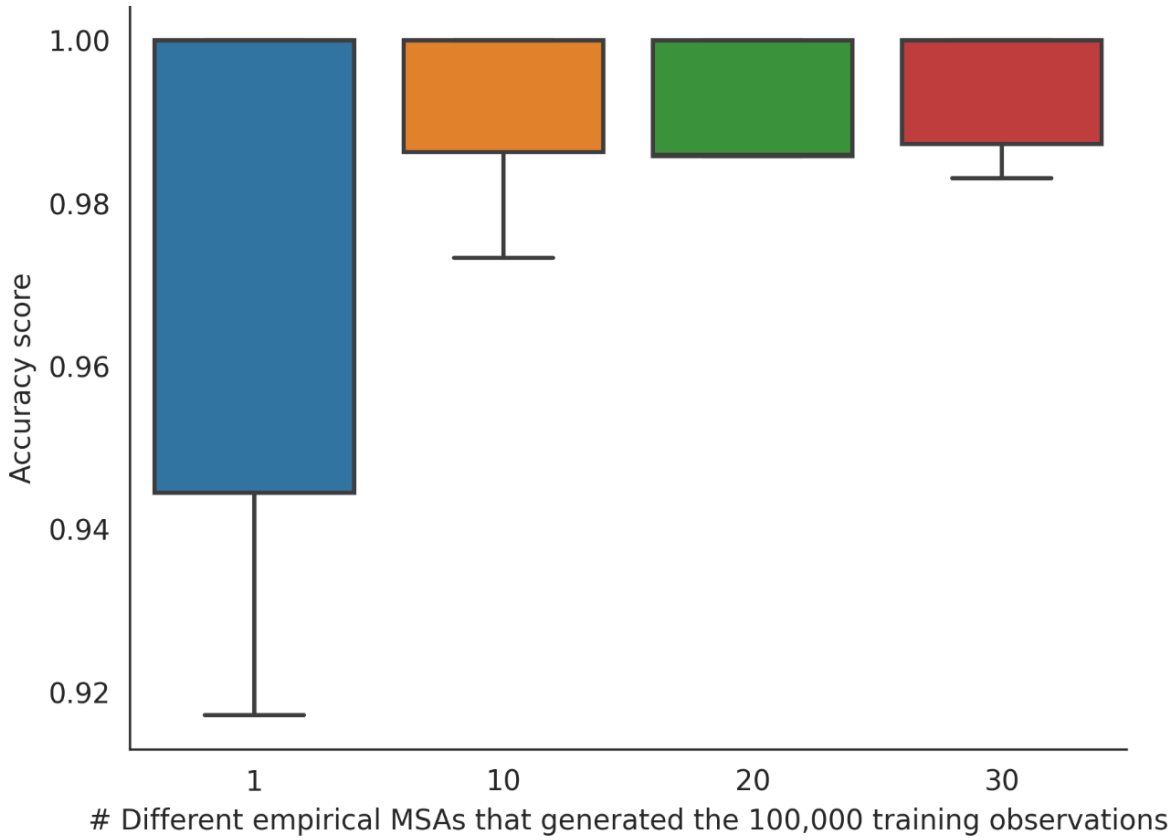
if $h > 0$ **then**
$a' = \arg\max_a Q_\theta(\phi(s_h, a))$
Store the transition $(\phi(s_{h-1}, a_{h-1}), r_{h-1}, \phi(s_h, a'))$ in $\mathcal{D}$
end if

Sample from exploration policy π: $a_h = \pi(s_h)$
Execute SPR-move a_h, collect reward r_h and observe next tree s_{h+1}
end for

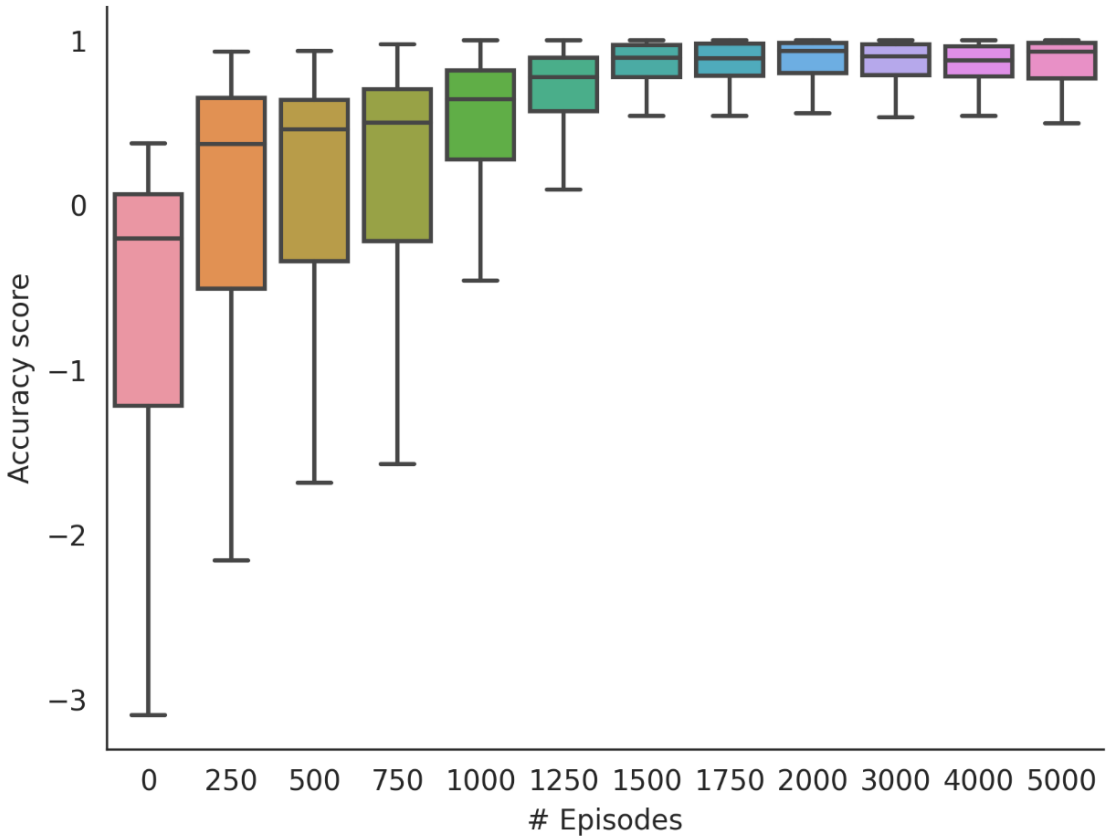
for $t = 0...times - to - learn$ **do**

Sample mini-batch of transitions $(\phi(s_j, a_j), r_j, \phi(s_{j+1}, a'_j))$ from $\mathcal{D}$
Perform a gradient descent step using
$Q_\theta(\phi(s_j, a_j)) = r_j + \gamma Q_\theta(\phi(s_{j+1}, a'_j))$
with respect to the online parameters θ
end for

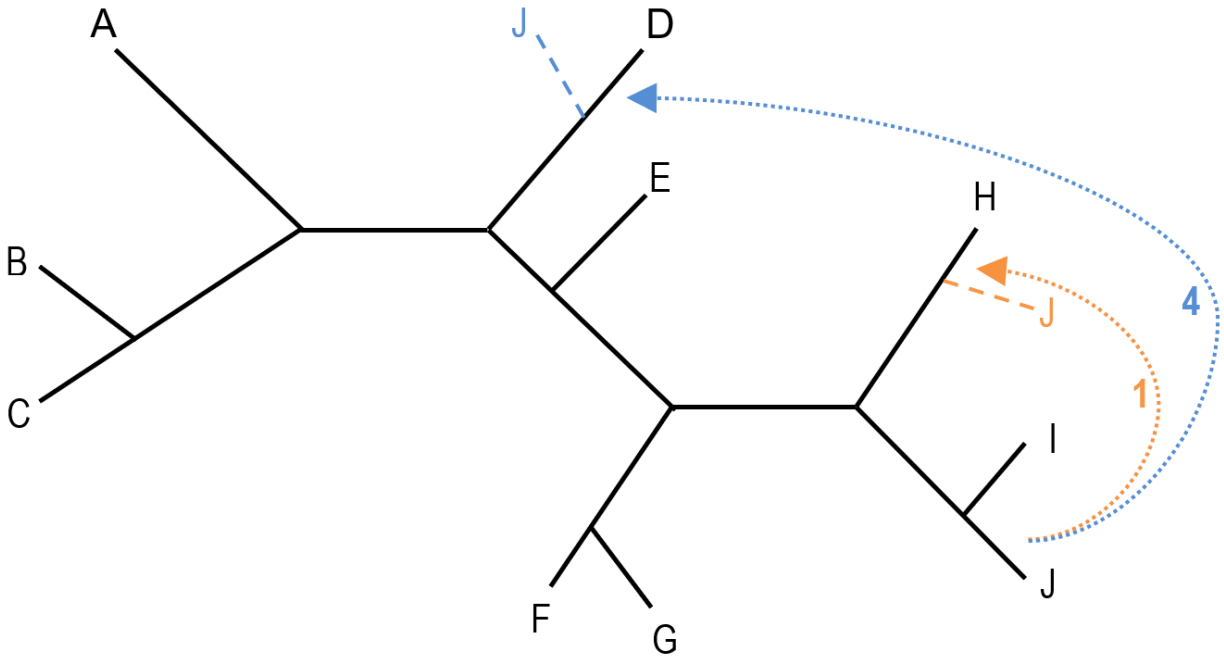
end for



Supplementary Figure 1. The impact of the number of datasets used to generate the training data on the prediction accuracy. The RL model performance on the validation set containing 12 sequences (y axis) when using an increasing number of distinct datasets to generate the transition data. The box shows the quartiles of the dataset while the whiskers extend to show the 1.5 × IQR past the low and high quartiles.



Supplementary Figure 2. The impact of the number of training episodes on the validation accuracy. The accuracy performance on 10 distinct empirical validation datasets (y axis) when using an increasing number of episodes in the training phase (x axis). The box shows the quartiles of the dataset while the whiskers extend to show the 1.5 × IQR past the low and high quartiles.



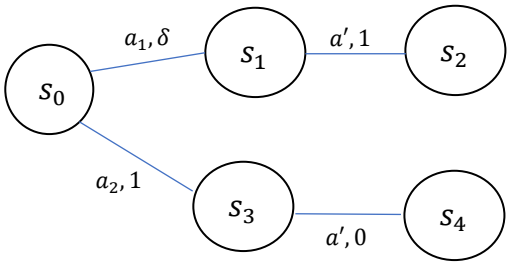
Supplementary Figure 3. The radius restriction on SPR moves for relatively large datasets. An illustrated example of an SPR move of radius one (“1”; in orange) and of radius four (“4”; in blue). In the developed RL framework, this restriction was applied for datasets containing 15 and 20 sequences.

Supplementary Table 1. **The table details the hyperparameters values and further details of the RL configuration.**

Parameter name	Value in the trained model	Additional details
NN architecture	Five fully connected hidden layers, in addition to the input layer (containing 27 neurons) and the output layer (containing a single node)	Number of neurons within each layer: {1: 4096, 2: 4096, 3: 2048, 4: 128, 5: 32}
Loss function	Mean Square Error	
Activation function	Leaky ReLU	
Optimizer	Adam	
Discount factor (γ)	0.9	
Replay buffer size	10,000	The maximal size of transitions collected during training
Times-to-learn	50	The number of times we sampled a batch to train the ANN
Horizon *H*	20 (for data of 7, 12, and 15 sequences), 30 (for 20 sequences)	The number of SPR moves in each episode. This hyperparameter was reoptimized when we considered different number of sequences in the analysis
Batch size	128	
Learning rate	10^{-5}	
Exploratory policy	SoftMax	With T parameter = 1
Episodes	2,000	Number of episodes in training

Supplementary Note 1: Optimizing the $\boldsymbol{Q}$ function

In reinforcement learning, the Q-value function is a measure of the expected return (i.e., the immediate reward and future discounted rewards) of a particular action in a particular state. It is denoted as $Q(s, a)$, where s is the state and a is the action. When an agent is in a state s, it will choose the action with the highest Q-value, i.e., the action that is expected to yield the highest return. The Q-value function is typically estimated through trial and error, using an algorithm such as Q-learning. As the agent interacts with the environment, it updates its estimates of the Q-values based on the rewards it receives. Over time, the Q-value function converges to the optimal value, enabling the agent to make the best decisions possible in any given state. The optimal Q-value function, denoted as Q^*, represents the expected optimal return from any state-action pair. It can be used to determine the optimal policy for an agent, which is the set of rules that maximizes the agent's cumulative reward in an environment. In our case, $Q(s, a)$ corresponds to the maximal $\mathbb{E}_{s_0=s,\, a_0=a}\left[\sum_{t=0}^{H-1} \gamma^t r_t(s_t, a_t)\right]$, which we assume to be close to $Q^*(s, a)$. Notice that for any given dataset, there exists a $\gamma = 1 - \epsilon,\ \epsilon > 0$, which induces a Q^* such that the optimal policy leads to the optimal topology from any starting topology. As an example, suppose we are in an environment with five states, $\{s_0, s_1, \ldots s_4\}$ where s_2, s_4 are terminal states, and the reward for transitioning from s_0 to s_1 is δ, from s_0 to s_3 is 1, from s_1 to s_2 is 1, and from s_3 to s_4 is 0:



$Q^*(s_0, a_1) = \delta + \gamma \cdot 1$
$Q^*(s_0, a_2) = 1 + \gamma \cdot 0 = 1$

Here, s_2 is the optimal terminal state, but under a certain δ value, the Q value is greater for s_4. Specifically, under $\gamma < 1 - \delta$ the optimal policy in s_0 is a_2, which leads us away from the global optimal topology s_2.

Supplementary Note 2: Data collection for 20 sequences

The data collection strategy described throughout the paper was applied when training agents for 7, 12, 15, and 20 sequences. For datasets containing 20 sequences, we tested an additional strategy, in which we trained several agents, each on a single MSA, such that each constructed a different memory buffer. Next, all the memory buffers were combined to a single, larger, buffer of training observations. A new agent was then trained on this combined buffer. The purpose of training several agents on each single MSA was to simplify the RL environment, enabling our agents to collect observations with high-likelihood. Empirically, this agent achieved an average test accuracy of 0.89, while a regular agent (trained as described in the main text for smaller datasets) achieved an average test accuracy of 0.84 on MSAs containing 20 sequences.

תקציר

החיפוש החישובי אחר העץ הפילוגנטי בעל הסבירות המקסימלית הוא בעיה NP קשה. כבזה, אלגוריתמים נוכחיים של חיפוש עצים עשויים להוביל לעץ שהוא אופטימלי מקומית, לא גלובלית. כאן אנו מציגים שינוי פרדיגמה לחיזוי עץ הסבירות המקסימלית, על ידי קירוב רווחי סבירות ארוכי טווח במקום מקסום רווח סבירות בכל שלב של החיפוש. הגישה שלנו רותמת את הכוח של למידת חיזוק ללימוד אסטרטגיית חיפוש אופטימלית, המכוונת לאופטימום הגלובלי של מרחב החיפוש. אנו מראים כי השיטה המבוססת על למידה חיזוקית שלנו משיגה ציוני סבירות דומים לאלו המתקבלים בטכניקות מתקדמות בעת ניתוח נתונים אמפיריים המכילים עשרות רצפים. יש לציין כי ביצועים אלו מושגים ללא צורך בביצוע אופטימיזציות יקרות של סבירות מלבד תהליך האימון, ובכך עשוי לאפשר שיפור אקספוננציאלי בזמן הריצה. הממצאים שלנו מדגישים את הכיוון המבטיח שמספקת למידת חיזוק בהתמודדות עם האתגרים של שחזור עצים פילוגנטיים.

The Raymond and Beverly Sackler
Faculty of Exact Sciences
Tel Aviv University

הפקולטה למדעים מדויקים
ע"ש ריימונד ובברלי סאקלר
אוניברסיטת תל אביב

אוניברסיטת תל אביב
הפקולטה למדעים מדויקים ע"ש ריימונד ובברלי סאקלר
בית הספר למדעי המחשב

משחק שחזור העצים: בניית עצים פילוגנטיים בעזרת למידה מחיזוקים

לקראת התואר "מוסמך אוניברסיטה" במדעי המחשב
על ידי
עוז גרניט

עבודה זו נכתבה בבית הספר למדעי המחשב בהנחיית **פרופ' ישי מנצור**

מחקר זה נוהל במשותף על ידי, מיכאל אלבורקרקה (אוניברסיטת תל אביב) ודנה אזורי (אוניברסיטת תל אביב). כתב זה נכתב על ידי ונבדק ונערך ע"י כל התורמים. פרופ' טל פופקו, פרופ' איתי מירוז ופרופ' ישי מנצור הנחו עבודה זאת באופן משותף.

אוקטובר 2023

●●● קריית האוניברסיטה, ת"ד 39040, רמת אביב, תל אביב 6139001 | טל': 03-6408237, 03-6408268 | פקס: 03-6409264
Tel Aviv University, P.O.box 39040, Ramat Aviv, Tel Aviv 6139001, Israel | Tel: 972-3-6408237, 972-3-6408268 | Fax: 972-3-6409264