The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

Generative NNI Transformation Strategies in Binary Trees Using Reinforcement Learning

Shirin Shirvani, Manfred Huber
Department of Computer Science and Engineering
University of Texas at Arlington, Arlington, TX 76019-0015
shirin.shirvani@mavs.uta.edu, huber@cse.uta.edu

Abstract

Learning strategies to address problems on graph and tree structures with no a-priori size limitations in cases where no known solution exists (and thus supervised data is hard to obtain), is a difficult problem with potential applications in a wide range of domains ranging from biological networks to protein folding and social network search. The main challenges here arise from the variable size representation that needs to be resolved in the context of Reinforcement Learning (RL) to address the problem. In this paper we consider a common, specific tree problem and show that it can be addressed using a combination of feature engineering and carefully designed learning processes. In particular, we consider the classical *Nearest Neighbor Interchange (NNI)* distance between unrooted labeled trees, which is defined as the minimum-cost sequence of operations that transform one tree into another. We introduce a representation and a reinforcement learning method that learns the transition dynamics and iteratively changes an arbitrary initial labeled tree into a goal configuration reachable through NNI. The differential tree representation and NNI actions permit the system to learn a strategy that is applicable to arbitrary sized trees. To train the system, we introduce a training process that uses randomly sampled trajectories to incrementally train more and more complex problems to overcome the difficulty of the overall strategy space. Experiments performed show that the system can successfully learn a strategy for effective NNI on complex trees.

Introduction

Graph or tree structures are a very commonly used in many fields to represent problems or information, to encode processes, or to encode information sharing constraints. Comparing two given trees to determine their similarity or distance is a problem that is highly important in a variety of contexts, including in applications in natural language processing, document similarity evaluation, medical image processing, comparison of RNA secondary structures, quantifying neuronal morphology, discovering and comparing shape classes, character recognition, similarity joining and querying of XML documents, and information extraction. Dissimilarity between combinatorial trees has been computed in the past literature largely by recourse to one of two approaches: either comparing edges or counting edit distances. In structure prediction problems, such as the NNI problem, where the possible structures are unlimited, the complete underlying representation for the problem instances is usually exponential in the size of the structure and thus unlimited for the complete problem domain. This frequently make exact methods intractable for large size trees and generally make it impossible if no upper limit on the tree size exists. We thus have to rely on approximations in terms of the effectively used representation of the tree and the specification of the problem to make the problem tractable. An example is the conversion of the problem into an approximate Markov Decision Process. In this paper we take this route to address the general NNI problem by mapping it first to a sequential decision problem with a fixed action set and then developing a finite representation that captures the approximate differences between the current tree and the target tree, independent of size. Using this representation and process model, we use Reinforcement Learning to learn a strategy that converts the tree into the goal tree with the fewest possible interchange operations. The resulting strategy can then be used to either show the transfer or, by analyzing the number of interchange operations applied, to determine the approximate NNI distance between the initial and target tree.

To permit training with arbitrary size trees, we develop an automatic, sampling-based training approach that incrementally trains the system with increasingly more complex instances, resulting in a strategy that can address trees independent of their size. Evaluation shows that the system is successful at learning a generative strategy that is effective at approximately addressing most instances of the NNI problem.

Related Work

The NNI distance between two trees is the minimum number of NNI moves required to transform one binary tree into another. Computing the NNI distance is an NP-complete problem (Li, Tromp, and Zhang 1996), (Robinson 1971), (Krivanek 1986) and has been actively studied in recent years. An important property of NNI is that the number of trees in the one step neighborhood increases linearly with the num-

ber of leaves of the tree, making NNI practical for very large trees. For n leave nodes, there are $(n-3)$ internal edges in an unrooted tree; we can split on each of these edges resulting in two new trees. By splitting on each internal edge we generate a NNI neighborhood of $(2n-6)$ trees. The relatively small neighborhood size of NNI leads to small increases in the search space with each iteration and in a tendency toward less local optima (Whelan 2007). Dasgupta et al. (DasGupta et al. 1997) have proven that given two trees τ_1 and τ_2 with unique leaf labels, computation of the NNI distance, $\delta_{nni}(\tau_1, \tau_2)$, is NP-complete. Therefore, there is no computationally tractable discrete algorithm for computing the NNI distance between two trees and the only way to compute the exact answer is to enumerate all possible answers. In an attempt to solve the problem, Waterman et al. (Waterman and Smith 1978) have introduced the "closest partition metric", d_{cp}, and conjectured that d_{cp} is equal to the NNI distance. The CP algorithm is based on the partitions induced by the interior branches of a binary tree (Jarvis, Luedeman, and Shier 1983). In their method, the closest partition distance, $CP(T_1, T_2)$, for trees sharing a partition is found recursively as the sum of the two distances between the related induced trees resulting from clustering each tree into two. For trees T_1 and T_2 that do not have a shared partition, $k-$step NNI operations are made to achieve a shared partition between T_1 and T_2, $d_{cp} = k + C(T^{'}, T_2)$, where k is the minimum number of NNI operations required to transform a tree T_1 into tree $T^{'}$ that shares a partition with tree T_2. CP is a weak measure, however, because it leads to multiple choices as results for $T^{'}$. Li et al. (Li, Tromp, and Zhang 1996) and Jarvis et al. (Jarvis, Luedeman, and Shier 1983) presented counter examples against Waterman et al.'s theorems and proved that there exist some trees T_1 and T_2 that share a partition that is not shared by any intermediate tree on a shortest path from T_1 to T_2. This lemma shows that the shape of the shortest path between two trees can significantly depend on whether two subtrees (partitions) are within a certain linear distance from each other, and gives the problem a sense of discontinuity. This phenomenon can possibly be exploited to prove an NP-completeness result (Li, Tromp, and Zhang 1996).

A number of approximation algorithms have been presented, attempting to design an efficient solution for computing NNI distances. In all these approximation algorithms, finding non-shared edges is a key step, and typically is the most time-consuming part. More precisely, all these approximation algorithms run in $O(n \log n + \beta)$ time where β is the time complexity to find the non-shared edges between each pair of trees T_1 and T_2 (Hon et al. 2004),(Hon and Lam 1999). For unweighted degree-3 trees, an $O(n \log n)$-time algorithm has previously been presented (Li, Tromp, and Zhang 1996). For trees of varying degree, existing approximation algorithms take $O(n^2)$ time for both un-weighted and weighted cases (DasGupta et al. 1997),(Hon et al. 2004).

Background and Notation

For our problem, we consider an unrooted, undirected full binary tree $G_\tau = (V_\tau, E_\tau)$. In other words, we consider an acyclic connected graph, with nodes V_τ, consisting of n labeled leaves (nodes with a degree of one) and $(n-2)$ internal, unlabeled nodes, all having a degree of three, and edges E_τ consisting of $n-3$ edges between internal nodes, and n edges connecting leaves to internal nodes. The measure we consider is derived from *Nearest Neighbor Interchange (NNI)*, a simple tree transforming operation (swap operation) over given internal edges e_τ.

Let $\Phi(\tau)$ denote the set of unrooted non-degenerate binary trees with n labeled leaves which is generated by *NNI* operations from tree τ. For each tree τ_i in $\Phi(\tau)$ we generate the *NNI* neighbors. This is done by performing an *NNI* operation on each internal edge ie_k in τ_i. The *NNI* operation swaps the subtrees attached to each internal edge. As shown in Fig. 1, there are 2 new trees created by applying an NNI operation.

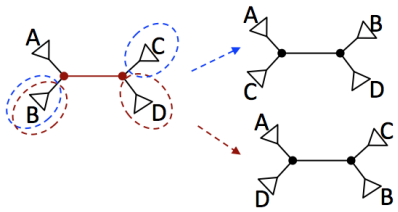


Figure 1: An NNI operation swaps two subtree that are separated by an internal edge.

The set of new trees created in this manner is the 1-*nni* neighborhood of τ_i, where the k-nni neighborhood would contain all trees that can be obtained by performing at most k successive *NNI* operations on the given tree. By applying the *NNI* operation on all possible internal edges $\{ie_i\}_{i \leq (n-3)}$ of a given tree τ_i, its 1-*nni* neighborhood, $\varphi^{<1-nni>}$, would be:

$$\varphi^{<1-nni>}(\tau_i) = \{\tau_j'\}_{j \leq (2n-6)} \quad (1)$$

NNI Distance δ_{nni}

The *Nearest Neighbor Interchange (NNI) Distance* is a tree rearrangement technique that has been widely used to calculate the distance between trees. In general the NNI distance, δ_{nni}, between two trees τ_i and τ_j is defined as follow:

$\delta_{nni}(\tau_i, \tau_j)$ = Min{ $\rho(A)$ | set A is a sequence of swap actions taking τ_i to τ_j }.

Before we formalize our framework for transferring the underlying problem into a sequential decision problem on an MDP space, consider a motivating example. Suppose given inputs of τ_1 and τ_2 with the same labeled leaves but different topology, and the output $\delta_{nni}(\tau_1, \tau_2)$, which is the NNI distance between these two trees. In general, there is a non-unique sequence of NNI operations that, if performed, would transfer τ_1 to τ_2. The tree space along all possible NNI sequences between two trees, $\Phi(\tau)$ grows exponential as the number of leaves increases. Thus, the brute-force search becomes exponential in terms of time and space in this context. We thus propose to transform the problem and use reinforcement learning to find an approximate solution in the form of a generated, compact sequence ρ_i.

Reinforcement Learning and MDP

The concept of reinforcement learning (Sutton and Barto 2017) provides a way in which agents can optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents must derive efficient representations of the environment from high dimensional inputs, and use these to generalize past experiences to new situations. Usually the control process in these problems is modeled as a Partially Observable (POMDP) or fully-observable Markov Decision Process (MDP).

An MDP is a tuple $< S, A, T, R >$ where S is the state space, representing the relevant aspects of the environment, A is the available action set for the agent, and $R(s, a, s')$ is the reward for taking action a in state s and ending in state s'. $P(s'|s, a) = T(s, a, s')$ is the probability of transitioning to state s' given a prior state s and action a with $\gamma \in [0, 1]$ as a discount factor. Standard approaches for solving MDPs include value and policy iteration. The optimal policy provides a mapping of states to actions such that the long-term expected reward of the policy is maximized.

To treat the NNI Task described here as a generative decision problem in this framework, the current and target tree have to be converted into observations, and ultimately a state representation, and the NNI operations have to be converted into a finite action space. During neighboring tree exploration in the NNI task, the complete state can be observed since the complete tree configurations are known. Because we can directly observe the state at all times, we can use this tree navigation problem as a Markov Decision Process (MDP). The main problem arising here is that if the size of the trees is unlimited, a complete representation of the current and target trees would require an infinite number of features and thus a more tractable, potentially approximate representation has to be derived.

Reinforcement Learning of a Tree NNI Transformer

To apply Reinforcement Learning to the NNI problem, a tractable representation for the tree pairs (current and target) as well as a finite representation for the discrete action set have to be derived, both of which should be independent of the tree size. Given such a transformation to a tractable (PO)MDP representation, a policy can be learned that sequentially converts the current tree into the target tree.

Reduction to an RL Problem

Consider S to be the space of states representing pairs of trees, and A be a set $a_i, \ldots, a_k$ of actions. The goal of RL is to find a policy $\pi : S \rightarrow A$ that maximizes the expected cumulative rewards $E[V_{\rho^*}]$, where $V_\rho = \sum_{t=0}^{T-1} R(s_t, a_t, s_{t+1}, a_{t+1})$. To facilitate this for the NNI problem, a state, action, and reward representation has to be derived.

Action Set In the traditional NNI problem, the set of available operations consists of two swap operations for each internal edge of the tree. For a tree, τ_i, with n leaves it therefore contains $2(n-3)$ operations, making the original set of operations specific to the tree size. To address this issue, the action set for the (PO)MDP formulation is defined here locally with respect to a current active edge, ei_i, which forms the center of the swap operations and which can be changed using local walk operations which allow the active edge to move over the tree.

Given an active edge ei_i as shown in Fig. 2, there are thus two types of possible discrete actions: (i) two swap operations that exchange subtrees attached to the current active edge as shown in Fig. 1, and (ii) four move operations which move the active edge from its current location to one of the four neighboring edges. For each active edge the four possible move directions, *LL: Left-Left, LR: Left-Right, RL: Right-Left, RR: Right-Right* are defined as $\{a_1, a_2, a_3, a_4\}$ and the two possible swap (NNI) operations are $\{a_5, a_6\}$.

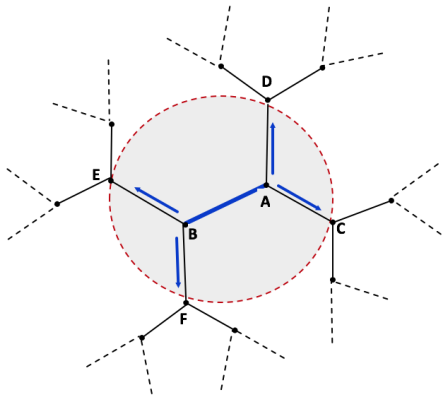


Figure 2: Active edge $ie_i = (A, B)$ with partial observation zone.

State Space To make the generative NNI problem addressable, we also need to transfer the current and target tree configuration into a fixed length feature representation that permits observations of the current problem instance. To obtain a representation that captures the information efficiently, we represent each pair of current and goal tree, (τ_c, τ_g), in terms of differential features aimed at expressing the differences between the trees. To be able to do this in the context of the proposed action set, this has to be done relative to the current active edge, ie_c, in the current tree which represents a virtual root for the tree, leading to a rooted tree, $\bar{\tau}_c$. Using this, the pair of trees, $(\bar{\tau}_c, \tau_g)$, is represented by 4 normalized feature categories $\bar{F} = (\bar{f}_1, \bar{f}_2, \bar{f}_3, \bar{f}_4)$, where the number of features is independent of the size of the tree and captures differences to the target tree.

The performance of our model is directly related to the existence of good and reliable meta-features for the action-state evaluation. We here design generic normalized meta-features that have strong predictive power across the dataset. The features are a combination of statistical, matching distance, geometric distance, and similarity measures. One of the principles considered is that the meta-features should be calculated fast. In order to satisfy the need to capture

relevant information and to be easily computable, we apply decomposition on the current tree to generate 6 induced subtrees as shown in Fig. 3. Then, we calculate normalized meta-features on each of the subtrees that capture their differences with respect to the corresponding subtrees in the target tree. To achieve a unique differential feature representation, we first identify the best virtual root for the target tree by evaluating which edge yields the most consistent subtree composition relative to the rooted current tree. Using this, the subtrees of the current and target tree are aligned and the meta-feature of the current state are derived as the combination of all extracted features for all partitions.

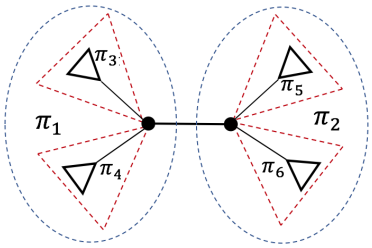


Figure 3: Dynamic decomposition strategy recursively decomposes the current state around the active edge, into a set of 6 induced subtrees $B^{\tau} = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\}$. The features are calculated based on each induced piece.

To derive the differential representation we defined the following meta-feature for trees $\bar{\tau}_i$ and $\bar{\tau}_j$

- *Matching Distance:* Given an active edge, $D_m(\bar{\tau}_i, \bar{\tau}_j)$ between rooted trees $\bar{\tau}_i$ and $\bar{\tau}_j$ is the weight of the minimum-weight perfect matching of the trees with size of $[1 \times 6]$.
- *Maximum agreement subtree:* This is the maximum normalized cordiality (largest) isomorphic subset of the current and goal trees.
- *Statistical features:* These are normalized statistics of the tree nodes, such as the ratio of the shared population of nodes, and the standard deviations of node distances
- *Geometric distance:* These are geometric measures capturing aspects of the structure such as minimum relative eccentricity, and depth, and diameters.

Each of the features is computed for each of the 6 pairs of induced subtrees, resulting in a feature vector, $\bar{F}(\bar{\tau}_c, \bar{\tau}_g)$, with total $|\bar{F}| = 1 \times 32$ normalized features. Important to note here is that this normalized, differential feature representation is designed to be independent of the size of the tree and the specifics of the target tree, and thus permits learned strategies to transfer to different sized trees.

Using this representation we can formulate the generative NNI problem. In particular, we want to learn a strategy such that, given an instance pair of (τ_c, τ_g), it generates a trajectory output, ρ_y such that $\rho_y = \{\bar{\tau}_1, ..., \bar{\tau}_k\}$, where $\bar{\tau}_1 = \bar{\tau}_c$, $\bar{\tau}_k = \bar{\tau}_g$, and transition $\bar{\tau}_i \Rightarrow \bar{\tau}_{i+1}$ for $1 \leq i \leq k-1$ is generated by an action $a_i \in A$. To solve the NNI problem, let C be a Cost function that assigns costs to the operations (for the standard NNI problem this would be a cost of 1 for a swap and a cost of 0 for a move operation), $C(\rho) = \sum_{i=1}^{i=|\rho|-1} C(a_i)$. The best solution is here a trajectory with the lowest cost.

To transform our problem into the *RL* framework, let each virtually rooted tree pair be a state, $s_t = F(\bar{\tau}_t, \bar{\tau}_g)$. Moreover, let the reward of a transition from s_t to s_{t+1} using action a_t be defined as:

$$R(s_t, a_t, s_{t+1}) = -C(a_t) \tag{2}$$

with an additional reward, R_s, for successfully generating a trajectory (i.e. reaching the goal tree).

The objective is then to learn a policy that maximizes the expected cumulative rewards.

Approximate MDP The generated meta features provide a finite dimensional observation space for unlimited trees by building a differential representation that captures the approximated differences between the local information of the current tree with respect to the goal tree. While each observation, F_t, is local since it represents information in the local neighborhood of the active edge more precisely than for parts of the tree that are further removed, treating the resulting system as a POMDP is computationally very expensive and not generally tractable. However, since the differential features also include features of the subtrees, we can interpret the feature representation also as an approximate state estimate, and thus treat the problem as an MDP and try to solve for an approximate solution. The rationale here is that since the meta features represent the differential statistic not just of the current neighbors but also, at a coarser resolution, for the remainder of the tree, they contain global information that permit treatment as an approximated MDP. The key components of the approximated MDP model are:

- The continuous state space: $S = \{F(\bar{\tau}_i, \bar{\tau}_j) : \tau_i, \tau_j \in \tau\}$
- The discrete action set $A = \{a_1 ... a_6\}$
- Transition Probabilities: $P_t(s_{t+1}|s_t, a_t)$
- A reward function: $R(s_t, a_t, s_{t+1}) = C(a_t), R(s_g = F(\bar{\tau}_g, \bar{\tau}_g)) = R_s$

To approximately solve the generative NNI problem, we then use Reinforcement Learning to find a strategy that achieves maximum expected cumulative rewards.

Function Approximation

Since the state space formed by our differential features for the NNI problem is continuous, a function approximator has to be used to represent the value function. We use the engineered meta-features as parameters of the function approximation for a Q-function, $\hat{q}(s, a, w) \in \mathbb{R}$, where w is a parameter vector for the function approximator.

$$\hat{q}(s, a, w) \approx q_{\pi}(s, a) \tag{3}$$

Since the state representation is already in the form of a real-valued feature vector and the action set is discrete, the most direct function approximation scheme would natively use the features $f_i(s)$ as the basis for either a linear or non-linear function approximation. For linear function, this would yield:

$$\hat{q}(s, a, w) = f(s)^T w(a) = \sum_{i=1}^{k} f_i(s) w_i(a) + b \tag{4}$$

Tile Coding as Function Approximation Based on generated normalized meta-features, the state is represented by multi-dimensional continuous spaces. Considering the character of the features used here, which represent normalized differences over (sub)tree statistics, a linear function approximator would not be sufficient to capture the relation between the features and the value function. To achieve efficiency, we use tile coding as a non-linear approximation of the function. In tile coding, the approximation is represented by a set of overlapping partitions of the feature space, called tilings. We use tilings generated by diagonal, vertical, and horizontal stripes in 2-dimensional sub-spaces. Each element of a tiling, called a tile, is a binary feature activated if and only if the a given state falls in the region delineated by that tile. The approximated function that the tile coding represents is determined by a set of weights, one for each tile in each tiling, such that

$$\hat{q}(f(s), a, w) = \sum_{i=1}^{n} b_i(f(s))w_i(a) \tag{5}$$

where b_i is a binary vector for tiling i with a single 1 for the tile within the tiling that state s falls in.

Trajectory Sampling

Generating a random training set is a big challenge because the varying complexity of trees in terms of size and topologies. In order to be able to learn efficiently in the context of dramatically different complexities, it is useful to train the system systematically starting from simple problem instances towards more complex ones over time. To do this, we developed a random backward sampling approach that allows to generate problem instances with particular complexity bounds. In this approach, random action sequences are sampled backward from the goal tree to the current tree, allowing them to be grouped into approximate complexity sets. These samples are then used as part of the training process to provide a bias towards increasingly complex problem instances as the system learns to address the simpler ones.

Reinforcement Learning Algorithm

Given a current tree $\tau_c \in \phi$ and a goal tree $\tau_g \in \phi(\tau_c)$, our algorithm learns an action-value function $Q(s_t, a_j)$ that predicts the value of using a_j in state s_j and a corresponding generative NNI policy using the SARSA algorithm with tile coding function approximation (Sutton and Barto 2017), as indicated in Algorithm 1. As a termination, each episode terminates when either the target tree is generated or if learning exceeds the upper-bound of time steps.

Experimental Results

To evaluate the proposed approach, we analyze the performance of the RL method on trees with different hierarchical structure and size.

Simulated Synthetic Data

We have used artificial tree collections to see how the algorithm scales across a wide range of tree sizes and NNI

Algorithm 1 SARSA on-policy Algorithm for Estimating $\delta(\tau_i, \tau_j)$

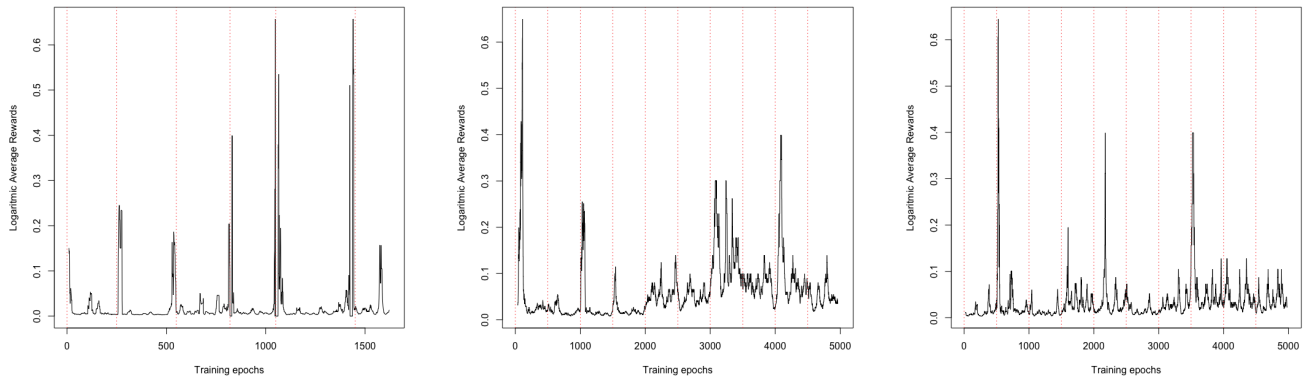
procedure (:)
  INPUT: Initial $\bar{\tau}_c \in \phi$ and desired step complexity R
  OUTPUT: predicted ρ_y
  Generate $\bar{\tau}_g$ using R step random walk in A from $\bar{\tau}_c$
  Initialize $s = F(\bar{\tau}_c, \bar{\tau}_g)$
  Choose a from s using ϵ-greedy exploration on Q
  while no termination **do**
    Generate s' by applying a on tree $\bar{\tau}_c$ in s
    Choose a' from s' ϵ-greedy exploration on Q
    Update tile coding parameters $w = w - \alpha \sum_i \frac{dQ_w(s,a)(}{dw}(Q_w(s,a) - [r(s,a) + \gamma Q_w(s', a'))$
  return trajectory ρ_i

complexities. To generate our dataset, we first generated a set of random binary leaf-labeled rooted target trees τ_{g_i} with n nodes, where $n \in \{15, 25, 50, 100, 200\}$. These trees were generated randomly using the algorithm in (Arnold and Sleep 1980) that assigns equal probability to all members of the family of trees with n nodes. To generate the source trees for our tests, we took the target tree and performed a number of actions $a \in A$ to generate tree pairs with a particular approximate path complexity. In our experiments, given each generated target tree, we applied k rounds of NNI operations to our target to generate the source trees. This put an upper bound of k steps between our trees. This method of generating the source trees from our target trees was chosen since it offers a clear upper bound on the maximum number of NNI operations to get from the source tree to the target tree. If we had selected both the source and target trees completely randomly, we would not have an upper bound on the minimum number of NNI operations separating the trees, thus being unable to control the complexity of the training examples.

Results

To evaluate the ability of the system to learn NNI policies for variable size trees, we train the system with increasing complexity trees. In particular, we increase the trajectory distance for the training trees every 250 or 500 episodes during training, starting with tree pairs that are solvable in a single step, increasing it to more and more complex tree pairs. Fig. 4 (a,b,c) show the resulting learning curve in terms of the lognormal of the average reward.

This figures show that the system successfully learns to solve the tree problems. Every time the complexity is increased, a spike in the value indicates a drop in performance with a subsequent improvement back to the solution value. The spike is due in part to the system seeing new, more complex problems but mainly to the fact that exploration rates are increased to permit the system to more efficiently adapt to the new tree pairs. Fig. 5 shows the number of steps the learned policy requires to solve a tree pair for each of the complexity levels. This figure shows the expected behavior where the policy requires an increasing number of steps for more complex problems. It can be observed that for the



(a) (b) (c)

Figure 4: Learning curves showing log-normal of the average reward. The complexity of the training tree pairs is increased every 250 episodes in (a) and every 500 episodes in (b) and (c), and the exploration rate is set back to an initial value from which it decays exponentially.

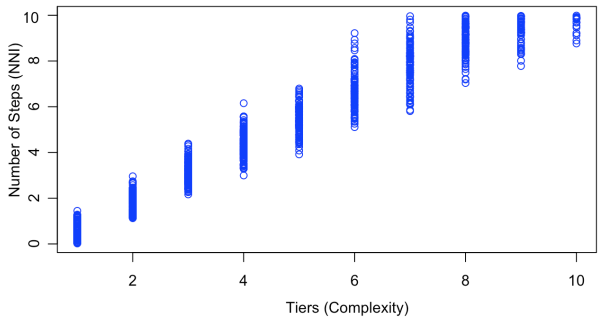


Figure 5: The performance of the learned policy on a test set for different trajectory length. The y-axes shows the agent's steps while the x-axis represents the generation complexity.

range of problem complexities used here, the length of the learned trajectories increases close to linearly with the complexity of the problems.

Conclusions

Learning strategies for graph and tree problems is challenging due to the variable size of the underlying representation. In this paper we introduce an approach that uses Reinforcement Learning to approximately solve the NNI problem on general size trees. To facilitate this, a representation of the state and action space is developed as well as a means of biasing the training set to increasingly more complex problem instances. Results obtained show that the system can learn a strategy that can generate NNI trajectories for variable size trees for a significant percentage of the tested problems. While the designed representation shows success in the context of the tile coding approach used here, we will in future work investigate the use of deep learning methods to not only learn a strategy but also automatically derive appropriate state features to represent graph problems.

References

Arnold, D. B., and Sleep, M. R. 1980. Uniform random generation of balanced parenthesis strings. *ACM Trans. Program. Lang. Syst.* 2(1):122–128.

DasGupta, B.; He, X.; T. Jiang, M. L.; Tromp, J.; and Zhang, L. 1997. On distances between phylogenetic trees. In *Proc. of the 8th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA '97)*, 427–436.

Hon, W., and Lam, T. W. 1999. Approximating the nearest neighbor interchange distance for evolutionary trees with non-uniform degrees. *Computing and Combinatorics* 1627(1999):61–70.

Hon, W.; Kao, M.; Lam, T.; Sung, W. K.; and Yiu, S. 2004. Non-shared edges and nearest neighbor interchanges revisited. *Information Processing Letters* 91(3):29–134.

Jarvis, J. P.; Luedeman, J. K.; and Shier, D. R. 1983. Counterexamples in measuring the distance between binary trees. *Mathematical Social Sciences* 4(3):271–274.

Krivanek, M. 1986. Computing the nearest neighbor interchange metric for unlabeled binary trees is np-complete. *Journal of Classification* 3(1):55–60.

Li, M.; Tromp, J.; and Zhang, L. 1996. On the nearest neighbor interchange distance between evolutionary trees. *Journal of Theoretical Biology* 182(4):463–467.

Robinson, D. F. 1971. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory* 11(2):105–119.

Sutton, R., and Barto, A. 2017. *Reinforcment Learning: An Introduction*. MIT press.

Waterman, M. S., and Smith, T. F. 1978. On the similarity of dendrograms. *Journal of Theoretical Biology* 73(4):783–800.

Whelan, S. 2007. New approaches to phylogenetic tree search and their application to large numbers of protein alignments. *Systems Biology* 56(5):727–740.