Representation and Strategy Learning for Variable-Size Tree Transformation Using Reinforcement Learning

by

SHIRIN H. SHIRVANI

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2021

To my little cheerful Nousheen.

ACKNOWLEDGEMENTS

First, I thank my Professor Manfred Huber for the continuous support during my PhD studies at UTA. His trust in my success and the provided scientific freedom and guidance were decisive to the result of this thesis. Next, I would like to express my deepest appreciation to my PhD committee for being my thesis committee members; Professor Farhad Kamangar, Professor David Levine, and Professor Vassilis Athitsos. The completion of my thesis would not have been possible without their unparalleled support and ingenious suggestions.

July 30, 2021

ABSTRACT

Representation and Strategy Learning for Variable-Size Tree Transformation Using Reinforcement Learning

SHIRIN H. SHIRVANI, Ph.D.

The University of Texas at Arlington, 2021

Supervising Professor: Manfred Huber

Trees as acyclic graphs are ubiquitous in representing different context where they encode connectivity patterns at all scales of organization, from biological systems to social networks. Trees are powerful resources which have been used many times for the exploration and discovery of interactions and properties in different context. Tree data structure representation approaches have led to remarkable discoveries in different real-world applications.

In the last decades, extensive research and algorithms have been developed on tree or acyclic graph data structures with deep theoretical properties. The cost of solving these various problems ranges from simple linear time algorithms, to more complex ones that solve NP-hard problems. However, trees as acyclic graph datasets are generally not easily amenable to data-driven techniques, and unlike other big data will not easily benefit from the growing scale of available data with sparse nature. Recently deep learning has shown significant progress for images, texts and signals, typically with little domain knowledge. However, the combinatorial and discrete nature of tree space or acyclic graph data makes it non-trivial to apply deep learning

in this domain. This thesis investigates several aspects of how to build a connection between deep reinforcement learning and the classical algorithms for tree space.

This dissertation introduces the use of reinforcement learning for closing the gap between these two complementary approaches. Commonly greedy, heuristic and supervised clustering methods provide automated tools to discover hidden built-in structures in generally complex-shaped and high-dimensional configuration spaces of trees. We show some potential applications of such tree transformation tools and sampling strategies to a common problem related to tree distance and manipulation with applications to problems in genomics, planning and control.

The first part of the dissertation presents the use of reinforcement learning for relaxed, deterministic coordination and control of an agent to learn a tree edit distance task. We reinterpret this classical method task for unsupervised learning as an abstract formalism for identifying and representing tree transformations by relating the continuous space of configurations to the combinatorial space of trees.

The second part of the dissertation introduces a generalized approach with automated representation exploration in an edit neighborhood representation, learning to identify a neighborhood of a tree that captures the local geometric structure of a configuration space around the tree's instantaneous configuration. Based on this edit neighborhood representation, we use reinforcement learning to learn a NNI distance strategy to find the minimum-cost sequence of operations that transform one tree into another.

The third part of the dissertation presents a generic framework for learning representation and behavior on a tree dataset in hyperbolic space on a finite action space, integrating policy learning using reinforcement learning. In this work, we study the use of value function approximation in hyperbolic space and reinforcement

learning problems with high dimensional state and a finite action space based on a generalized representation and policy iteration.

The obtained results strongly suggest that reinforcement learning is, indeed, an effective approach for automatically extracting inherent structures in configuration spaces relevant to the solution of tree edit distance, and that it might play a key role in the design of computationally efficient planners in complex, high-dimensional configuration spaces in different application domains.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

ABSTRACT v

LIST OF ILLUSTRATIONS xii

LIST OF TABLES xvi

Chapter Page

1. Introduction 1

    1.1 Background 3

    1.2 Organization of the Thesis 4

2. Background and Related Works 6

    2.1 Tree Representation Learning with Variable Size 6

    2.2 Tree Transformation Representation 8

    2.3 Sampling-Based Methods 9

    2.4 Tree Generative Model 10

    2.5 Preliminaries and Notation 11

        2.5.1 Tree Embedding For Relational Reasoning 12

    2.6 Reinforcement Learning for Tree Transformation 13

        2.6.1 Action Space 14

        2.6.2 Reward 14

        2.6.3 Reduction to RL 15

3. Generative NNI Transformation Strategies in Binary Trees Using Reinforcement Learning 17

    3.1 Introduction 18

3.2 Related Work 19
3.3 Background and Notation 21
3.3.1 NNI Distance δ_{nni} 22
3.3.2 Reinforcement Learning and MDP 22
3.4 Reinforcement Learning of a Tree NNI Transformer 23
3.4.1 Reduction to an RL Problem 24
3.4.2 Function Approximation 29
3.5 Trajectory Sampling 30
3.5.1 Reinforcement Learning Algorithm 30
3.6 Experimental Results 31
3.6.1 Simulated Synthetic Data 31
3.6.2 Results 32
3.7 Conclusions 34
4. A Deep Reinforcement Learning Approach to Learning Tree Edit Policies on Binary Trees 35
4.1 Introduction 36
4.2 Background 38
4.2.1 Tree Space 40
4.3 Supervised Tree Representation Model 42
4.3.1 Direct Encoding of Pairs of Trees 42
4.3.2 NNI Neighboring Aggregate Strategies 44
4.3.3 Supervised Transfer Learning 45
4.4 Transfer Representation Policy Learning 46
4.4.1 DQN Learning Policy 48
4.4.2 Learning Q-Values For NNI Distance 49
4.4.3 Deep Q-Learning 50

4.5 Rollout Approach 51
4.6 Simulated Synthetic Data 53
4.6.1 Distance Learning Evaluation 54
4.7 Evaluation of Policy Learning 55
4.8 Conclusion 57
5. Local Tree Neighborhood - Learning Hyperbolic Representations of Tree Pair Differences 58
5.1 Introduction 58
5.2 Background 59
5.3 Related Works 60
5.4 Preliminary and Notation 61
5.5 Hyperbolic Transfer Representation 62
5.5.1 Geometric Embedding of Tree Pairs 64
5.5.2 The Optimization Problem 65
5.6 Measure of Tree Pair Embedding 66
5.6.1 Hyperbolic Tree Pair Embeddings for NNI Distance Prediction 69
5.7 Conclusions 72
6. Deep Reinforcement Learning for Tree Transformation using Hyperbolic Representations of Tree Pairs 74
6.1 Introduction 74
6.2 Background and Related Works 75
6.3 Preliminary 76
6.4 Feature Based Learning 76
6.5 Action Based Features 77
6.6 Space Exploring Policy 77
6.7 State Representation: Tree Pairs in Local Neighborhood 78

6.8 Improving Deep-Q RL with Rollout using Hyperbolic Geometry Approximation 79
6.9 Results 81
6.10 Conclusion 84
7. Conclusions and Future Work 85
7.1 Conclusions 85
7.2 Future Work 86
REFERENCES 87

LIST OF ILLUSTRATIONS

Figure Page

3.1 An NNI operation swaps two subtree that are separated by an internal edge. 21

3.2 Active edge $ie_i = (A, B)$ with partial observation zone. 25

3.3 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. 26

3.4 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. 33

3.5 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. 34

4.1 An NNI operation swaps two subtrees that are separated by an internal edge. 39

4.2 A snapshot of generated network of tree space as a simple indirect graph G by NNI neighbors in $3 - nni$ steps. 41

4.3 Direct encoding of a pair of trees (τ_i, τ_j) using traversal sequences with multi-layer LSTMs. We inject the tree in different fashion either in-order, pre-order, or post-order. 43

4.4 $3 - nni$ neighboring aggregation: the training set $\Phi = \{\phi_i\}_{i=1}^{3}$ is generated by sampling random walks with NNI steps starting from tree τ_i. 44
4.5 DeepQ - NNI Policy Learning Framework. 51
4.6 Rollout Cycle 52
4.7 This framework supports batching with different size, which can be enabled by setting the batch size. 53
4.8 a. Mean squared error (MSE) losses for NNI distance prediction during training and validation on a tree space with 12-nni distance radius and vaying tree sizes. Validation losses were calculated using batches with variable size validation examples after every training batche. b. (MSE) losses of NNI distance prediction during training and validation with trees of a fixed size 55
4.9 a. The agent's learning convergence for nni transfer. b. Success rate of reaching the target for different size of tree with maximum 5-nni distance. 56
4.10 MSE of distance loss with maximum 12-nni distance (actual steps) and with max budgets of 32 steps. Rollout effectively can approximate the target with between [4-8] NNI step. 57
5.1 Tree in the hyperbolic disk. Each edge has to be embedded to a minimum length depending on the cone in which it is embedded. The distance of nodes increases exponentially (relative to their Euclidean distance) the closer they are to the boundary. 64
5.2 Embedding pairwise nni distance (τ_i, τ_j) in the hyperbolic disk. 64

5.3 Sample of embedding pairwise nni distance (τ_i, τ_j) in the hyperbolic disk. Positions of node vectors in 2-D space from an actual Poincaré embedding with leaf nodes labeled with the leaf labels and intermediate node labels prefaced with i. Nodes for tree 1 (green) are labeled above the mark while nodes for tree 2 (red) are labeled below. Overlapping nodes are green and indicated with a bold outline. Note that as we move toward the boundary, we begin to see nodes lower and lower in the hierarchy. Also, nodes similar to each other are close or overlap each other. 67

5.4 Supervised architecture model 70

5.5 Training and test performance for NNI distance prediction in terms of loss in the number of NNI steps (left) and estimated squared error (right) 72

6.1 Overall architecture and training pipeline. The hyperbolic embeddings of tree pairs are constructed in the first stage and, as a tree traversal sequence of current/target node pairs fed into the LSTM embedding network. The output of this network, having initially been pre-trained using NNI tree distances, serves as the feature representation for the deep-Q Reinforcement Learning component for the extrapolated tree transformation policy learning task. 82

6.2 Performance of agent during training on variable sized tree pairs with NNI distances between 1 and 5. The system successfully learns to transform the trees in an average of 8 steps. Reward ={ Goal=1000, edit and move=$-1, 0$} 83

6.3 Performance of rollout during training on variable sized tree pairs with NNI distances between 1 and 5 with variable size tree without normalization. Average loss in nni distance across 10 runs for variable size tree without normalization 83

LIST OF TABLES

Table Page

CHAPTER 1

Introduction

Tree data structures as acyclic graphs are ubiquitous in many real world applications. With an increase in exploratory approaches and use of tree and tree-based tasks in different applications, from biology to planning, it has become even more crucial to efficiently explore related data spaces to accomplish given tasks (such as, for example, edit distance or finding structural patterns). In bioinformatics, the molecules can be represented by acyclic graphs or trees with atoms as nodes and bonds as edges; in knowledge graphs, entities and corresponding relationships form a graph; in social networks, users and their interactions can also be characterized as graphs; in correct efficient planners it is essential to model and understand the typologies of configuration spaces of exploratory systems. Despite it being such an apparently easy representation, the discrete, combinatorial, and sparse nature of trees and acyclic graphs also lead to many difficulties when integrating them into various machine learning problems. Difficulties in learning within the data space of tree data structures are manyfold and have led to them not frequently being used or being significantly simplified and constrained within the learning domain. Here we briefly mention three fundamental types of difficulties in the current state-of-the-art of integrating tree structures into machine learning applications.

- Representation Learning: learning to represent the acyclic graphs or trees in more traditional, fixed dimensional vector representations has challenges that

have to be addressed to enable the scalability and efficiency of existing learning algorithms.

- Generative modeling: Generating and reconstructing the tree is also a hard task. Despite significant progress in domains with continuous noise tolerant data such as images, the discrete nature of graphs typically forces the data to be accurate. For example, in a molecule data set, the node can have at most a fixed number of degrees, making this data very sensitive to noise and making it difficult to apply many machine learning techniques. Also, the combinatorial and discrete nature of graphs makes it hard to apply gradient based adjustments during the generation procedure.
- Combinatorial Optimization problems with NP-complete nature such as genome sequencing tasks, tree edit distance, nni-distance, protein interaction, etc. pose significant challenges in the context of both pattern and action depend tasks.

The design of efficient and, if possible, provably correct models in tree space inevitably requires to model and understand the topologies of configuration spaces of exploratory systems. Two commonly encountered approaches to tackle the efficient exploration problem are configuration space planning and sampling-based planning [140, 55]. Reinforcement Learning (RL) techniques have also been extensively used to solve many machine learning problems. Recent progress in deep learning techniques allow RL to be extended to various new application domains. However, tree spaces generally have complex shapes and are difficult, if not impossible, to explicitly describe in terms of standard similarity, geometric and topological models. Moreover, the complexity of tree spaces is known to grow exponentially as the size of the trees and

the configuration space grows in dimension [46]. These limitations therefore often restrict the applicability of exploring network space to low dimensional settings.

1.1 Background

This thesis investigates methods and case studies for analyzing tree networks such as biological networks and extracting actionable insights. As a result, it provides techniques for next-generation algorithms for tree and acyclic data structure problems for generic tree modeling, network modeling for systems with acyclic nature, as well as structure analysis and optimization:

- How do we represent tree structures as acyclic graph data and what techniques can be used to unify data coming in different formats and topologies in the form of a time series, and how can we deal with such data coming potentially from different experimental technologies? How do we represent different types of entity objects such as nodes and edges, (representing entities such as genes, diseases and drugs), and how do we represent different relation types using networks and embed these components?
- How can we embed larger tree structures and acyclic graphs, such as sub-trees, and entire acyclic graphs, such as molecular graphs, into a low-dimensional vector space? How to handle multi-relational and multi-size networks?
- The basic principle in tree networks is that the same configurations (i.e. sub-tree structures) tend to interact with each other in similar ways (for example in the context of genes or proteins). How do we mathematically encode such principles into a machine learning model and extract information about interactions between components?

- Tree structures associated with the same function or topology tend to cluster in the same network neighborhood. How do we use this notion to predict new typologies or functions?
- How can representation learning methods identify the components which need to be edited in order to yield the target structure?

While these are fundamental questions for the use of acyclic graph structures within advanced learning and reasoning systems, this thesis tries to make progress in this direction by focusing on the problem of efficiently embedding such structures into deep learning architectures without the need for a priori limitations on the size and scope of the graphs. Moreover, it develops techniques for the integration of Reinforcement Learning techniques on these structures to allow addressing problems on variable size graph sets. Throughout the development, the work presented here focuses on the common problem of Nearest-Neighbor Interchange tree transformation and tree distance calculation. However, many of the techniques developed here are generic and should apply equally to other problems on tree structure representations.

1.2 Organization of the Thesis

The thesis is organized in the following way: in Chapter 2, we present the related works in the underlying areas, such as representation learning, generative learning, and combinatorial optimization on acyclic graphs. Then in Chapter 3 we introduce an approach that combines human designed structure embeddings of current/target tree pairs in the context of a deep learning framework to learn effective tree transformation strategies. This illustrates how human designed algorithms and deep learning can benefit from each other and presents a deep reinforcement learning approach to learning tree edit policies. In Chapter 4 we then investigate the use of

deep learning techniques to automatically acquire an embedding for variable sized trees without the need of hand coding or applying bounds on the size of the tree representation. Combining this with pre-training and deep Reinforcement Learning is again used to demonstrate its potential at learning tree transformation policies. To address limitations in the efficiency of the learned embeddings, Chapter 5 studies a more effective way to encode local tree neighborhoods by learning hyperbolic representations of tree pairs. Based on this, Chapter 6 then proposes an improved deep reinforcement learning approach that combines learning of an efficient tree difference embedding from the hyperbolic geometry representation of variable size tree pairs with a deep reinforcement learning architecture using rollouts. Using this new embedding approach, this chapter investigates the differences in performance across representations. Finally in Chapter 7, we conclude this dissertation and discusses potential future research directions.

CHAPTER 2

Background and Related Works

In this chapter, we review related works in tree similarity, distance editing, and tree transformation with variable size learning problems by highlighting the different objectives, points of interest, and methods.

2.1 Tree Representation Learning with Variable Size

Trees and acyclic graphs are invaluable approaches to data structures to represent real-world relationships, e.g., in natural language processing, molecular structure in biological systems, path planning, social and interaction networks, etc.. Study and analysis based on editing tree structured data, including prediction over nodes, edges, and the whole tree, becomes more and more attractive and has been applied to many problems.

There are different mathematical abstract formalisms for the configuration and similarity of a tree data structure in tree space. Once an explicit representation of the tree space is generated, a number of configuration space representations can be used to satisfy the given task specifications. However, tree configuration spaces generally have complex shapes and are difficult, if not impossible, to explicitly describe in terms of standard geometric and topological models. Moreover, the complexity of possible topology is known to grow exponentially as the configuration space grows by number of nodes. These limitations therefore often restrict the applicability of tree data structures to low dimensional settings.

Most of these tasks depend on embedding the hierarchy and tree information into vectors, which is crucial to the success of these applications [1, 2]. This approach is generally referred to as tree embedding, i.e., the mapping of the representation of trees and tree similarities into a vector space. A common solution for the representation of nodes and edges in trees depends on engineered domain-specific features. Regardless of the hard effort and requirement for expert knowledge for engineering features, such engineered features may not be sufficiently powerful to capture the properties of the tree structures for the specific target tasks.

Recently, deep learning techniques have opened a new promising avenue for some of these tasks, depending on the quality and applicability of the tree embedding. One promising direction is preserving some desired characteristics of the tree structure through approximating the corresponding statistics with embeddings. For example, DeepWalk [3] extends the word2vec [4] model in NLP to graph based structure embeddings, which tries to preserve the connection between nodes in the local structure obtained by random walks. Different heuristic methods, such as the biased random walk model [5] have been proposed.

Another existing approach is built on the convolution operator over graph structured data, which has been showing success in covering the long-range information in graphs in real-world applications. By using convolution in a spatial local structure embedding domain, graph neural networks (GNNs) [6] proposed a general framework of performing neural network operators over structured data. However, the scope of these models is restricted since

- the variable size of the graphs and long range information within the graph might not be easily integrated in a computationally or statistically efficient way (either via longer random walks in traversal methods or high-order proximity concepts in convolutional settings);

- these approaches are featureless and trained unsupervised, i.e., independent of the down-streaming tasks; as a result, the generated embeddings may be weak in covering the relevant information with respect to the labels.

2.2 Tree Transformation Representation

In the literature, various methods have been proposed in the scope of tree transformation as a combinatorial problem [7, 8, 9]. Since these algorithms rely on measures of similarity between two trees in the used combinatorial pattern matching approaches with approximation and heuristic search techniques, most studies use edit distance to measure the dissimilarity between trees. In general, similarity computation is the dual problem of distance computation. Usually, both their memory and computational cost become prohibitive for large-scale tree spaces, which grow exponentially with respect to the tree size.

Common approaches are based on tree rearrangement moves and edits such as nearest neighbor interchange (NNI), subtree pruning and re-grafting (SPR), and tree bisection and re-connection (TBR) to estimate partition functions of trees under some implied probability distribution on tree topology, branch lengths, or ancestral sequences [10]. The tree edit distance (TED) measure in tree transformations is a reference to measuring similarity of tree structured data. The tree edit distance is defined as the minimum-cost sequence of node edit operations that transform one tree into another.

Every interactive system requires a mechanism to encode and present the agent the data it handles. Most times the relations between the data and its presentation is complex. Further, most times, it is mediated by a representation technique, allowing the agent to describe how the data should be presented.

In this dissertation, we propose a deep reinforcement learning model for the embedding and representation of tree pairs and for the construction of tree transformation strategies.

2.3 Sampling-Based Methods

In general, the tree space, i.e. the network of trees generated by any tree transformation, is a graph network environment. Tree sampling is a technique to pick a set of trees from the original tree space. In some scenarios, the whole space of the environment is known and the purpose of sampling is to obtain a sequence of sample trees. In other scenarios, the space is unknown and sampling is regarded as a way to explore and partially construct the environment. Commonly used techniques are Node Sampling, Edge Sampling and Traversal Based Sampling [11]. In principle, navigation in the network of trees is a trivial matter, because the number of trees over a finite set of leaves is finite. However, in practice, the cardinality of trees grows super exponentially based on the number of leaves or edits. Since in many real-world networks their size is very large, and they continuously evolve due to the dynamic nature of the problem and the applicable transformations, the network representing the tree space is often sampled in order to facilitate study. Thus, similarity and sampling of tree navigation in the network of trees become fundamental to the analysis. For these reasons, a more thorough and complete understanding of network sampling is critical to support the field of tree transformation. Similarity search for similar trees has been extensively studied. The relations between these approaches have been studied and the theoretical analysis and practical implementation has been presented in previous work [12].

Sampling-based methods, such as probabilistic, rapidly-exploring random trees, and their variants, address and partially resolve some of the limitations by pro-

ducing randomly sampled configurations and simple connectivity criteria; Although they require no explicit construction of configuration spaces, sampling-based methods strongly rely on efficient nearest neighbor and graph search algorithms, effective sampling strategies (especially around narrow passages) and informative metric selection. Sampling from the likelihood distribution of labelled tree topologies is one the common approaches to sample, also known as the ancestral likelihood [13].

However, the theoretical understanding of sampling approaches on trees remains considerably less developed than that for optimization approaches with regard to the number of sampling steps needed to produce accurate samples of tree partition functions [12]. Despite the many advantages in principle of being able to sample trees from sophisticated probabilistic models, there is little theoretical basis for concluding that the best performing sampling approaches do in fact yield accurate samples from those models within realistic numbers of steps [14].

2.4 Tree Generative Model

Recent work on similarity in hierarchical structures, trees, or acyclic graphs are focused on deep learning by embedding the structure with neural networks. Generating models for the natural construction of the graph structure have been a hot topic in combinatorial graph theory and bioinformatics and machine learning, especially since the invention of generative models which can be represented with tensors in discrete space. When the discrete structures have a language representation with explicit syntax and semantics, it can be typically reduced to a sequence generation problem. This has been well studied, for example in the seq2seq [15] framework that models the generation of sequences as a series of token choices parameterized by recurrent neural networks (RNNs).

Additionally, several developments have been proposed within the structure generation domain, including an extra validator model [16], data augmentation [17], active learning in discrete sequences [18] and the use of reinforcement learning [19]. However, with the lack of formalization of the syntax and semantics serving as the restriction of the particular structured data, underfitted general-purpose string generative models will often lead to invalid outputs.

2.5 Preliminaries and Notation

We begin by defining the concepts with basic terminology and notation. We represent a tree τ as a graph $G_\tau = (V_\tau, E_\tau)$ with the vertex set V_τ and edge set E_τ. Let τ be an unrooted, undirected full binary tree. The dummy root of τ is denoted by root(τ) which is used for navigation on tree. The size of τ is defined by $|V_\tau|$. Each tree τ on N leaves has $N-2$ internal nodes with degree of three. The length of the connection between two nodes $v_1, v_2 \in V_\tau$, $len(v_1, v_2)$, is the number of edges on the path from $v1$ to $v2$ or vice-versa. The in-degree of a node v, $deg(v)$, is the number of children of v. A leaf is a node with no children and every other node is an internal node. The number of leaves of τ is denoted by $leaves(\tau)$. We denote the parent of node v by $parent(v)$. Two nodes are siblings if they have the same parent. For two trees τ_1 and τ_2, we will frequently refer to $leaves(\tau_i)$ and $deg(\tau_i)$ by L_i and D_i, $i = 1, 2$. Let $\tau(v)$ denote the subtree of τ rooted at a node $v \in \tau(v)$. Obviously, removing an edge in any tree disconnects the tree into two subtrees, each of which has a non-empty intersection with the set of leaves. The topology of a subtree $\tau(v)$ is defined as the set of all splits obtained by removing an edge of v.

If $u \in V(\tau(v))$ then v is an ancestor of u, and if $u \in V(\tau(v)) \setminus \{v\}$, then v is a proper ancestor of u. If v is a (proper) ancestor of u then u is a (proper) descendant of v. A tree τ is ordered if a left-to-right order among the siblings is given. For processing

trees, all nodes need to be visited through one of a set of different traversal strategies. There are different ways to traverse a tree structure depending on the order in which children nodes are visited. Through this study, the three most common strategies: preorder, inorder and postorder, are implemented. For tree τ with dummy root v and children $u_1, ..., u_i$, the preorder traversal of $\tau(v)$ is obtained by visiting v and then recursively visiting $\tau(u_k), 1 \leq k \leq i$, in order. Similarly, the postorder traversal is obtained by first visiting $\tau(u_k), 1 \leq k \leq i$, and then v.

The tree space is a network defined as $\Phi(\tau)$ containing the set of all possible trees generated by moves and edits. A sample tree τ' in $\Phi(\tau)$ is defined with a set of leaves v and internal nodes with degree three or more. A common approach to solving the tree similarity inference problem is to search through the space of all topologically distinct $\Phi(\tau)$.

2.5.1 Tree Embedding For Relational Reasoning

Learning relations in tree structures by using recurrent neural networks to model the interactions or relations has shown promising results in visual question answering [20] and reasoning over graphs [21]. Graph neural networks [21] integrate message passing as part of the architecture in order to capture the inherent relations between nodes. GCNs use convolutions to efficiently learn a continuous-space representation for a graph of interest. Many of these relational reasoning models can be realized in terms of an attentive read operation [22]. More formally, tree embeddings can be characterized in terms of the maintenance of a tree metric. We define the tree metric as follow:

Definition. Tree Metric: A pair (V, d) for trees $\tau = (V', E')$ where,

1. τ is a sample tree on V' that has only non-negative edge lengths
2. $V' \subseteq V$

is a tree metric if for each pair of nodes, u and v in V' the distance between nodes in τ equals the distance in the metric, $d_{u,v} = d(u, v)$. For simplicity it is common to simply refer to the accurate tree metric as τ instead of the pair (V, d). The distance between any two vertices $u, v \in V$ is denoted then as τ_{uv}. And because τ is a tree, the path from u to v is unique, which implies τ_{uv} is the distance of the shortest u-v path in τ.

Distortion: A distortion is a measure for an approximate tree metric (V, d) represented by a value α such that $d_{uv} \leq \tau_{uv} \leq \alpha\ d_{uv} \forall u, v \in V'$.

Tree Embedding: An embedding can then be defined as an approximate tree metric (V, d) that approximates the accurate metric (V, τ) with distortion α. Note that it is common to say that (V', τ) embeds into $(V,' d)$ with distortion α.

Basically, this defines a tree embedding as a representation in which node distances in the embedded tree are approximately preserved to within the distortion factor α.

Tree Pairs Embedding: An embedding for a pair of trees can then be defined as an approximate tree pair metric (V, d) that approximates the accurate metric $(V, (\tau_i, \tau_j))$ with distortion α.

2.6 Reinforcement Learning for Tree Transformation

When trying to solve for a tree transformation, the goal is usually to determine the sequence of transformation steps that converts the initial tree to the target configuration while optimizing an objective function (usually the number of transformation steps). Due to the discrete nature of the transformation tree space, this is generally a combinatorial optimization problem and thus computationally intractable. In this dissertation we take the approach where we use reinforcement learning to learn a strategy that performs an approximately optimal transformation on arbitrary start/target

tree pairs. For this, the learning framework explores and learns over trajectory samples, ρ_i, in the transformation tree space. In the learning process we optimize the objective, but searching over all generated trajectory samples ρ_i and evaluating the expectation of reward are both difficult in general, providing the main incentive for us to use reinforcement learning to find an approximate solution.

To utilize reinforcement learning we need to define a set of elements, namely actions, states, and rewards. For our problem, the state is simply the embedding of the pair of trees.

2.6.1 Action Space

While tree transformation problems usually define their set of operations in terms of edges in the tree, and thus have an action space that scales with the size of the tree, this action space can be reduced to a constant sized space by providing tree traversal (move) actions that allow the system to incrementally select an edge of interest by traversing the tree, and by only including the transformation actions for the selected edge. Given the set of actions $\mathcal{A}$ which represent the resulting traversal or modification operations, we are interested to learn a policy and thus the state representation features that related to elements of the action space. As indicated, the action space here is a constant set with the combination of two types of actions, moves or edit, which are independent of the size and labeling of the underlying trees.

2.6.2 Reward

The reward is the main driver for learning a successful sampling strategy to pick the necessary tree pairs and ignore the rest while solving the task. The reward function is based on the task on the graph and indicates the incremental progress towards the task objective in each step. If minimizing the number of transformations is the

objective, for example, a negative reward of -1 in each step can represent the incremental benefit with a large reward for reaching a state in which the transformation objective is achieved.

2.6.3 Reduction to RL

Consider S to be the space of states and A be a sequence $a_i, \ldots, a_k$ of actions. The goal of RL is to find a policy $\pi : S \rightarrow A$ to maximize the expected cumulative rewards $E[R_{\rho^*}]$, where $R_\rho = \sum_{t=0}^{T-1} R(s_t, a_t, s_{t+1}, a_{t+1})$. We now formalize the intuition from reinforcement learning.

Given an instance pair of (τ_i, τ_j), we would like to infer a single trajectory output, ρ_y using the RL algorithm such that $\rho_y = \{\tau_1, ..., \tau_n\}$ where the initial tree is the start tree and the final tree is the target tree.

An action derivation from τ_i to τ_j is a sample trajectory of trees $\rho = \{\tau_0, ..., \tau_k\}$ with length k such that $\tau_i = \tau_0$, $\tau_j = \tau_k$, and $\tau_{i-1} \Rightarrow \tau_i$ for $1 \leq i \leq k$ is generated by a sequence of random walk (traversal) actions and swapping (edit) actions over active edges reached by the traversal actions. Let C be a Cost function that assigns a cost to the sequence of operations. $C(\rho) = \sum_{i=1}^{i=|s|} C(\rho_i)$

To transform our problem into the *RL* framework, let each tree be a state $S_t = \tau_t$ and $\{\tau[0]..., \tau[t], j[0], ..., j[t-1]\}$ be the entire history of a sample trajectory ρ. Also, let the action $a_t = A_{j[t]}$ refer to the transition that generated $\tau[t+1]$ from $\tau[t]$. We let the cumulative rewards be the improvement:

$$R(s_t, a_t, s_{t+1}) = \tau[t+1] \sim A_{j[t]}(.|\tau[t]) \tag{2.1}$$

The objective is to maximize the expected cumulative rewards for the sample trajectory ρ as output in following equation (2.2).

$$\begin{aligned} \max_{\rho} \quad & E[R_{\rho^*}] \\ \text{s.t.} \quad & B_\rho \leq B^* \\ & LB_{exp} \leq B^* \leq UB_{Exp} \end{aligned} \tag{2.2}$$

Where B^* is the computation budget for length of all possible trajectories ρ. We define $B^* = n\log(n)$ as diameter ΔG (the maximum distance between any two trees with N nodes). The computation budget B^* is defined based on approximation algorithms that have been presented in [23, 24, 25] attempting to design an efficient solution for computing NNI distances. 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 tree pair (τ, τ_j). For unweighted degree-3 trees, an $O(n\log n)$ time algorithm has previously been presented.

CHAPTER 3

Generative NNI Transformation Strategies in Binary Trees Using Reinforcement Learning

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 chapter 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.

3.1 Introduction

Graph or tree structures are 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 makes exact methods intractable for large size trees and generally makes 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 chapter 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.

3.2 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 [7], [8], [9] 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 number 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 [26]. Dasgupta et al. [27] 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. [28] 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 [29]. 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. [7] and Jarvis et al. [29] 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 [7].

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 [30],[31]. For unweighted degree-3 trees, an $O(n \log n)$-time algorithm has previously been presented [7]. For trees of varying degree, existing approximation algorithms take $O(n^2)$ time for both un-weighted and weighted cases [27],[30].

3.3 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. 3.1, there are 2 new trees created by applying an NNI operation.

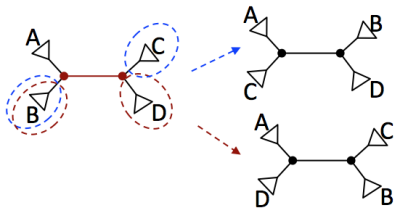


Figure 3.1: An NNI operation swaps two subtree that are separated by an internal edge.

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:

$$\varphi^{<1-nni>}(\tau_i) = \{\tau'_j\}_{j\leq(2n-6)} \tag{3.1}$$

3.3.1 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:

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

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 exponentially 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.

3.3.2 Reinforcement Learning and MDP

The concept of reinforcement learning [32] 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.

3.4 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.

3.4.1 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.

3.4.1.1 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. 3.2, 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. 3.1, 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 $\{a_5, a_6\}$.

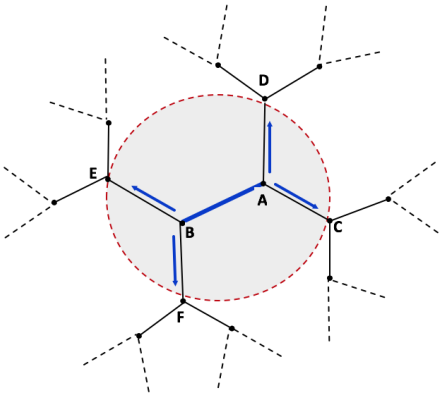


Figure 3.2: Active edge $ie_i = (A, B)$ with partial observation zone.

3.4.1.2 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. 3.3. 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-features of the current state are derived as the combination of all extracted features for all partitions.

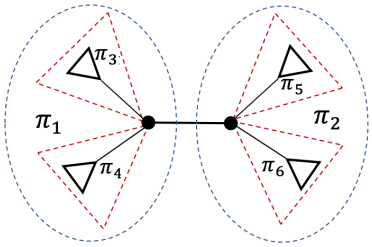


Figure 3.3: Dynamic decomposition strategy recursively decomposes the current state around the active edge, into a set of 6 induced subtrees $B^{\tau} = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\}$. The features are calculated based on each induced piece.

To derive the differential representation we defined the following meta-features 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 cardinality (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:

$$R(s_t, a_t, s_{t+1}) = -C(a_t) \tag{3.2}$$

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.

3.4.1.3 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.

3.4.2 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.

$$\hat{q}(s, a, w) \approx q_\pi(s, a) \tag{3.3}$$

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:

$$\hat{q}(s, a, w) = f(s)^T w(a) = \sum_{i=1}^{k} f_i(s) w_i(a) + b \tag{3.4}$$

3.4.2.1 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

$$\hat{q}(f(s), a, w) = \sum_{i=1}^{n} b_i(f(s))w_i(a) \tag{3.5}$$

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.

3.5 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.

3.5.1 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 [32], 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.

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

3.6 Experimental Results

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

3.6.1 Simulated Synthetic Data

We have used artificial tree collections to see how the algorithm scales across a wide range of tree sizes and NNI 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 [33] 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.

3.6.2 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. 3.4 (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. 3.5 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 range of problem complexities used here,

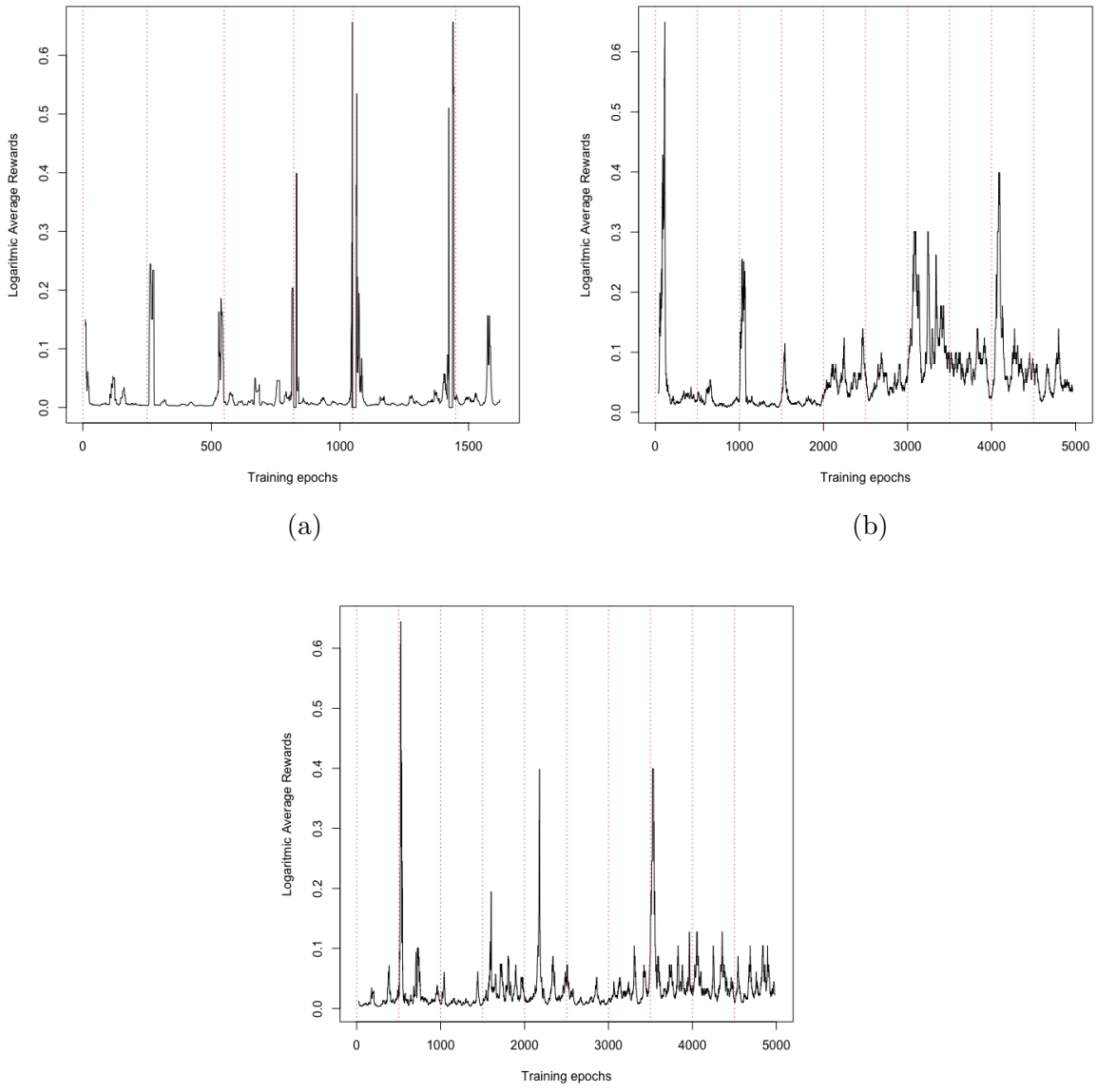


Figure 3.4: Learning curves showing log-normal of the average reward. The complexity of the training tree pairs is increased every 250 episodes in (a) and every 500 episodes in (b) and (c), and the exploration rate is set back to an initial value from which it decays exponentially.

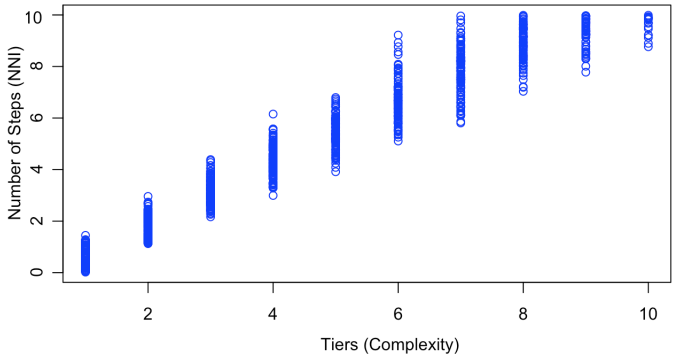


Figure 3.5: The performance of the learned policy on a test set for different trajectory length. The y-axes shows the agent's steps while the x-axis represents the generation complexity.

the length of the learned trajectories increases close to linearly with the complexity of the problems.

3.7 Conclusions

Learning strategies for graph and tree problems is challenging due to the variable size of the underlying representation. In this chapter 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 the following chapters of this dissertation investigate the use of deep learning methods to not only learn a strategy but also automatically derive appropriate state features to represent graph problems.

CHAPTER 4

A Deep Reinforcement Learning Approach to Learning Tree Edit Policies on Binary Trees

Modeling sequential transformations between two trees is a fundamental task in domains such as bioinformatics. Traditional methods to address this usually rely on hand-coded heuristics and algorithms which are often hard to derive. The large advances in deep learning technologies and computational power have recently opened up new potential avenues. In particular, representation and policy learning provides an interesting opportunity to model the dynamic evolution of two trees, where each tree can be embedded in a Euclidean space and its evolution can be modeled by an embedding trajectory in this space.

In this chapter of the dissertation we expand on the work in Chapter 3 where hand coded features for tree differences were used and tile coding was employed to perform reinforcement learning to acquire generative policies for the NNI tree transformation problem. In particular, to avoid the need for hand-coding we here propose a representation and policy learning framework that learns a representation for arbitrary sized binary tree pairs using recurrent LSTM networks and a policy to transfer one tree into the corresponding target tree using Reinforcement Learning. Here, the representation is pre-trained on tree transfer similarity to transform pairs of tree-structured data into an approximate numerical multidimensional vector which encodes the original structure information. This model, used with a deep reinforcement learning approach, yields a constructive method for generating basis functions for

approximating value functions and permits to learn an efficient, general tree transfer policy that incrementally transforms the source tree into the target tree.

4.1 Introduction

Trees have been used extensively to model a wide range of problems. An important aspect in using trees as a data representation is the concept of tree similarity to be able to compare and transform trees. A common similarity measure for trees is the edit distance. Defining such a tree distance similarity is a key for many interesting problems and applications such as speech recognition, machine translation, and imitation. Search for edit distance of pairs of trees is still an open problem due to the high complexity of computing the tree edit distance [34],[35],[36],[23],[37].

The main challenge in machine learning on tree edit is identifying a way to integrate information about the structure of the difference of trees into the machine learning model. For example, edit distance learning needs to encode pairwise properties between trees, such as topological similarities or the number of common nodes and subtrees. Likewise, for tree classification, information about the global position of a tree in the tree space or the structure of the tree's local edit neighborhood is required.

To extract structural information from trees, traditional machine learning approaches often rely on summary statistics information (e.g., node degrees or membership within a clustering of sub-trees), pre-designed kernel functions, or carefully engineered features to measure local neighborhood structures [38] [3][39]. However, these approaches are limited due to inflexibility of such hand-engineered features.

More recently, extensive work has been done to learn representations that encode structural information for trees or graphs. The idea is to learn a mapping that embeds the basic elements (e.g. nodes, wordarc, label, subtree, or attribute) as well as

the connecting structure in the data, as points in a vector space, R^d [3][39]. The goal is to optimize the mapping so that geometric relationships in this learned space reflect the structure of the original dataset. After optimizing the embedding space, the learned embeddings can be used as feature inputs for downstream machine learning tasks. The key distinction between these recent representation learning approaches and previous work is how they treat the problem of capturing structural information about the tree. Previous works treated this problem as a pre-processing step, whereas representation learning approaches treat it as a machine learning task itself, using a data-driven approach to learn embeddings that encode graph structure.

Learning tree edit policies that perform tree transformations from a source to a target tree requires an embedding approach for the representation of the structural similarity measure between trees. Although several methods for tree similarity measures have been proposed, high computational complexity and accuracy remains a concern.

In this chapter we present an approach that learns both a representation for arbitrary sized tree pairs and a policy to transfer one tree into another. First we train a recurrent neural network on a challenging task of tree Nearest Neighbor Interchange (NNI) distance similarity and next use deep Q-learning to learn a policy to perform NNI transformation tasks. The goal here is to use the first, supervised task to pre-train a representation for tree pairs in the context of NNI distance, and then use Reinforcement Learning to learn a policy that can perform the actual transformation and generalize it to different size trees. Since the pre-training requires supervised data, it relies here on the generation of a set of training examples with known NNI distances, making this generation computationally complex and thus limiting the maximum size and scale of trees that can be used for pre-training. The Reinforcement Learning step only requires tree pairs and a reward function which is

significantly easier to obtain and is thus not only used to learn the policy but also to further refine the representation with a larger range of tree pairs. To allow for variable and arbitrary size trees, the representation approach here treats tree pairs as sequences of tree traversals and uses this sequence information as the input to an LSTM-based representation learning network. To pre-train, a fully connected regression network is added that predicts the NNI distance, while for the policy learning task, a deep-Q network is added to the representation network to predict the utility of the transformation actions. For training and experimentation we use a synthetic dataset derived using a dataset generator, which is a popular benchmark for measuring the quality of these models, whilst being small and relatively fast to train. We present NNI tree similarity prediction and a training procedure for optimizing transformation policy models based on NNI distance for pairs of trees.[3]

4.2 Background

We consider an unrooted, undirected full binary tree $G_\tau = (V_\tau, E_\tau)$. V_τ is a finite set of nodes and E_τ is a binary relation on N where each pair $(u, v) \in E$ represents the parent-child relationship between two nodes $u, v \in N$. Node u is the parent of node v and v is one of the child nodes of u. By adding a dummy root, every arbitrary labeled tree of this form can be represented as a binary rooted tree. There exists only one dummy root node, denoted as $Root(\tau) \in N$, which has no parent. Every other node of the tree has exactly one parent and it can be reached through a path of edges from the root. The nodes which have a common parent u (i.e., all the children of u) are siblings. Each node $n \in N$ has a label $l(n)$ that specifies node information. Topology of a tree is the set of all splits induced by the edges of that tree. A binary tree, in which all interior nodes have degree 3 contains $2n - 1$ splits, whereas unresolved (or degenerate or non-binary) trees contain fewer than $2n - 1$

splits. The measure we consider is derived from *Nearest Neighbor Interchange (NNI)*, a simple tree transforming operation (swap operation) over given internal edges e_τ. The *NNI* operations swap the subtrees attached to each internal edge. As shown in Fig. 4.1, there are 2 new trees created by applying an NNI operation.

Let $\Phi(\tau)$ denote a tree space, i.e. the set of rooted non-degenerate trees with n leaves which is generated by *NNI* operations. Note that the number of hierarchies in $\Phi(\tau)$ grows exponential as the number of leaves increases. Thus, the brute-force search for a transformation sequence between elements of this set is exponential in terms of time and space in this context.

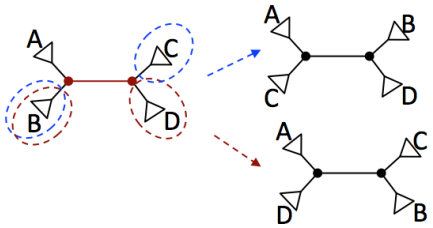


Figure 4.1: An NNI operation swaps two subtrees that are separated by an internal edge.

The set of all new trees created in the manner shown in Figure 4.1 is the 1-*NNI* neighborhood of τ_i, whereas the *k-nni* neighborhood contains all trees that can be obtained by performing at most k successive *NNI* operations on a given tree. By applying *NNI* operations on all possible internal edges $\{ie_i\}_{i\leq(n-3)}$ of a given tree τ_i, its 1-*NNI* neighborhood, $\varphi^{<1-nni>}$, can be defined as:

$$\varphi^{<1-NNI>}(\tau_i) = \{\tau'_j\}_{j\leq(2n-6)} \tag{4.1}$$

While in its standard notation the set of actions is defined in terms of swaps around individual edges and thus grows with the number of edges in the tree, we

can redefine this action set into a set comprised of two modification operations that swap subtrees around an edge of interest (the other transformations produce identical trees when ignoring the child order) and four traversal operations that move the edge of interest to one of the neighboring edges. This reformulation of the action space provides us with a finite action set that is independent of the size of the tree and that therefore can be used more easily for subsequent policy learning. Given this set of actions, $\mathcal{A}$, we are interested to learn to represent the relevant aspects of tree pairs with respect to this action space. While the goal of pure feature learning is often to encode all information about the tree pair and its relation in the tree space, this is often not necessary and actually detrimental in terms of the efficiency of the resulting representation in the context of a specific task. There can be different goals in the tree space which we like to explore. We have two types of features, *generic features* which are learned when pre-training using distance learning, $\mathcal{F}_G$, and *action-based features* resulting from refinement during policy learning, $\mathcal{F}_G(a)$. Here, we focus on both aspects. The combined approach allows the agent to learn representations that imitate both the past experience and the inherent large-scale environment, as well as learning policies by using customized basis functions to approximate utility functions for particular tasks.

4.2.1 Tree Space

To accelerate the NNI distance similarity searching in the space of possible trees, we do not explicitly calculate the difference between all possible pairs of trees with arbitrary size. Instead, we compress each search space to a network of trees created by fixed-size $k-$steps of hybrid random walk and NNI actions. The generated network represents the space of rooted binary trees with NNI distance connectivity. Combined, coordinated action and swapping can boost efficiency, robustness, and

flexibility in achieving complex tasks such as search and area exploration, as well as edit and modifications.

The random walk based method samples a large number of fixed-length random walks starting from each tree, τ_i, and gradually increases NNI distance. Each (τ_i, τ_j) pair of trees is proportional to the probability of visiting τ_j on a fixed-length random walk starting from τ_i. $P^k(\tau_i, \tau_j)$ is the likelihood of visiting τ_j on a length-k random walk starting at τ_i with k *nni* steps. The advantage of using random walks is to prevent a computationally expensive search for the entire neighborhood. Fig. 4.2 shows how the space of possible trees looks like for a tree with 15 leaves created by random walk in 3 *nni* steps.

Figure 4.2: A snapshot of generated network of tree space as a simple indirect graph G by NNI neighbors in $3 - nni$ steps.

By generating the tree space using random walks, we reduce the problem of finding similar trees to the problem of finding points in NNI distance neighborhood space. It is intuitive that this is a valid similarity measure; the more NNI steps between the pair of trees (τ_i, τ_j), the more distant the pair of trees are in the network.

4.3 Supervised Tree Representation Model

To permit unlimited tree sizes in a Neural Network architecture, input trees are here presented as sequences in the form of unique tree traversals. Using this, the tree representation is then fed into a Recurrent Neural Network(RRNs) which is trained to output a corresponding fixed sized feature vector.

In NNI distance, the embedding has to maintain structural relations between two trees that are relevant to the sequence of NNI edits. LSTMs are principally designed to avoid the long-term dependency problem and be able to learn and remember relations and dependencies between distant parts of the input. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn [40]. As a result, long term dependencies are the significant features of LSTM-based RNNs which makes them extremely useful in learning persistent representations. In the NNI distance similarity task, LSTMs are able to connect aspects of the current and target trees even though the relevant parts are at different points in the traversal sequences for the two trees.

4.3.1 Direct Encoding of Pairs of Trees

The core of our model contains a standard LSTM network to encode features that represent the differences in trees in terms of NNI distance. In the encoding phase, the aggregation method builds up the representation by converting batches of pairs of trees recursively into parallel node sequences derived by the traversal of the two trees in each pair, and then feeding these into the LSTM network to yield a hidden state representation at the output of the LSTM network that encodes the information about the tree pair. To be able to train it, this combined embedding is fed through a dense neural network layer to predict a known property of the tree pair. In this case, *nni* distance from the previously introduced tree space will be used for this as

it should retain important aspects needed for our learning tasks. For computational reasons, we process data in mini-batches (i.e batch size = 512, 1028). It is important to note that each batch of pairs of trees has arbitrary size and topology. Every tree in a batch should correspond to a time series with a node being present at time t.

As Fig. 4.3 shows, a recurrent neural network learns a set of features in R^d for a batch size of one pair. The result is a feature representation as the tree embedding which is generalized across different trees.

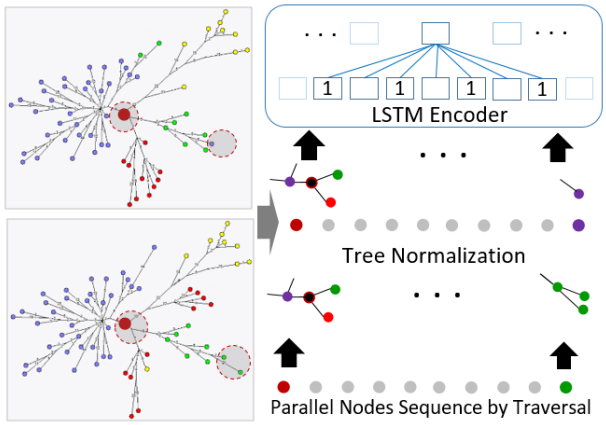


Figure 4.3: Direct encoding of a pair of trees (τ_i, τ_j) using traversal sequences with multi-layer LSTMs. We inject the tree in different fashion either in-order, pre-order, or post-order.

In the network space of trees with NNI distance connectivity, a tree pair corresponding to an edge with the two trees corresponding to connected nodes in the network is the smallest element which is considered as *direct encoding*. The goal is to encode tree pairs with different size as fixed low-dimensional vectors that summarize their tree position and the structure of their local tree neighborhood. These low-dimensional embeddings can be viewed as encoding, or projecting, trees into a latent space, where geometric relations in this latent space correspond to interactions

(e.g., subtree) in the tree space. Consider function $\hat{F}$ which maps a tree pair (τ_i, τ_j) to a vector embedding $z_t \in R^d$.

$$\hat{F} : (\tau_i, \tau_j) \rightarrow z_t \in R^d \tag{4.2}$$

4.3.2 NNI Neighboring Aggregate Strategies

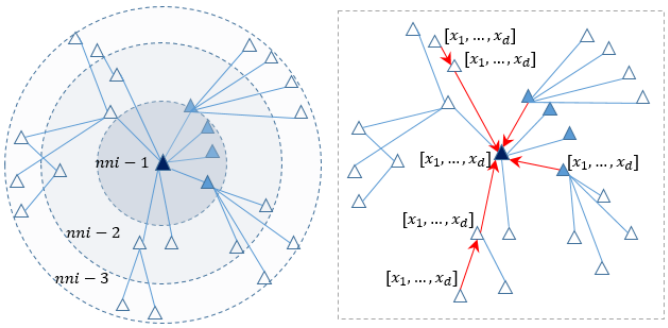


Figure 4.4: $3-nni$ neighboring aggregation: the training set $\Phi = \{\phi_i\}_{i=1}^{3}$ is generated by sampling random walks with NNI steps starting from tree τ_i.

Adopting from direct encoding and its sequence to vector view, the intuition of this model is that it generates embeddings for a tree pair by aggregating information incrementally from its local neighborhood in $nni - 1$ steps, but not necessarily the entire tree space with unlimited size. This aggregation method relies on traversal tree features. This method represents a tree pair as a function of its surrounding potential neighborhood. As Fig. 4.4 shows, the neighboring aggregation method generates the representation for tree pairs recursively.

Algorithm 2 NNI neighboring algorithm: Transfer Supervised Learning

INPUT: Tree Space Dataset Φ; depth K

OUTPUT: Vector representation $z_{(\tau_i,\tau_j)}$ for all $(\tau_i, \tau_j) \in \Phi$

for $k = 1...K$ **do**

for $(\tau_i, \tau_j) \in \phi_k$ **do**

$h^k_{N(\phi_k)} = nni_k(\{h^{k-1}_u, \forall(\tau_i, \tau_j) \in N(\phi_k)\})$

$h^k_{(\tau_i,\tau_j)} = \sigma(\omega^k.(h^{k-1}_{(\tau_i,\tau_j)}, h^k_{N(\phi_k)})$

$h^k_{(\tau_i,\tau_j)} \rightarrow Norm(h^k_{(\tau_i,\tau_j)})$

$z_{(\tau_i,\tau_j)} = h^k_{(\tau_i,\tau_j)}, \forall(\tau_i, \tau_j) \in \Phi$

Algorithm.2 generates the differentiable representation for the pair of trees recursively. After this aggregation, every tree pair is assigned a new embedding, equal to its aggregated neighborhood vector combined with its previous embedding from the last iteration. As the process iterates, the tree pair embeddings contain information aggregated from further and further reaches of the tree space.

4.3.3 Supervised Transfer Learning

In this section we focus on the learning of *generic features* and transfer learning. The key idea is optimizing the tree pair embeddings so that trees have similar embeddings if they tend to co-occur on short random walks over the tree space. Thus, instead of using a deterministic measure of tree similarity proximity, the random walk method employs a flexible, stochastic measure of tree pair proximity through neighboring aggregation, which has led to superior performance in a number of settings. We define the NNI edit transfer learning task as minimizing a loss function on a set of pairs of trees as a pre-training procedure. This function measures the NNI edit

distance similarity between trees and generates the pair of trees' embedding. A loss function, $\mathbb{L}(\delta, \bar{\delta})$, is used which determines how the quality of the pairwise reconstructions is evaluated in order to train the model, i.e., how $\hat{\delta}_{nni}(\tau_i, \tau_j)$ is compared to the true $\delta_{nni}(\tau_i, \tau_j)$ values.

We assume that the primary input to our representation learning algorithm is a dataset Φ of input and output pairs such that $\Phi \equiv \{\phi_k = ((\tau_i, \tau_j), Y(\delta_{nni}))_m\}_{k=1}^{N}$ generated by random walks using NNI operations. We are interested in learning a mapping from an input pair of trees to a NNI edit distance $\delta_{nni} \in \mathbb{Y}$ as target output for different level of complexity tasks (e.g. $nni-2$ distance is harder than $nni-1$).

$$\mathbb{L} = \sum_{(\tau_i, \tau_j)} \parallel \delta(nni)_{(\tau_i, \tau_j)} - \hat{\delta}(nni)_{(\tau_i, \tau_j)} \parallel_2^2 \tag{4.3}$$

The goal of the pre-training is not necessarily to get the optimal distance but to force it to encode features that represent the differences in trees in terms of NNI edit distance. Therefore, the distances learning task does not have to be perfect because the main goal is not to predict the NNI distance. We are using it to pre-seed the representation that we use for policy learning.

4.4 Transfer Representation Policy Learning

The dynamic embedding representation phase generates the embedding of the NNI edit tree transfer space given actions by the agent and does not explicitly model the future trajectory of the agent in the embedding space of the environment. Therefore, it does not explicitly model the optimal policy for generating the trajectory of the tree transfer in the embedding space. Here we propose a deep Q-learning model that learns the embedding trajectories between two trees. The proposed method employs

deep recurrent neural networks as function approximators to update the embedding of a pair of trees at every action.

The concept of reinforcement learning provides a way in which agents can optimize their control of an environment. However, to use reinforcement learning successfully in situations approaching real-world complexity, 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.

We started with a tree NNI edit distance exploration task in tree space. The goal is to visit as many of the trees in the space as possible within a fixed number of steps. This is motivated by applications such as mapping in a geometric environment, where an agent explores an environment and builds a map. In this phase, the agent only observes a batch of trees in a small neighborhood around its current location and does not know the whole tree space it has visited. At each time step, it has a batch of 6 possible actions corresponding to the location. We use the following configuration to set up each element in the batch for this task:

- **Observation:** the observed pair of trees contains the active edge and the connectivity for the part of the tree space the agent has traversed up to time t.

- **Reward:** a positive reward is only received if the target is visited. Otherwise a negative reward with respect to estimated distance is obtained.
- **Termination:** when the agent has reached the target, or has used up the exploration budget T.

We use an episodic setting, where each episode starts with a state independently sampled from S_0, and ends in finite $T \leq T_{max}$ steps. Our performance evaluation metric is the episode reward, which is the non-discounted sum of all rewards collected in a full episode. We train on random binary tree space of different size trees with up to 200 leave nodes, and test on the same space from the same distribution. The starting location is chosen randomly. We allow the agent to traverse the space and perform NNI edits within the given budget, and report the average distance. The budget is defined based on the NNI edit upper bound on the tree space.

4.4.1 DQN Learning Policy

The Transfer Policy learning technique learns an inductive bias that accelerates the learning of a new task by training on a large of number of easier tasks. Training on tasks from set $\Phi = \{\phi^k\}$ involves learning a policy.

$$\hat{\theta} = \arg\min_{\theta} Loss \tag{4.4}$$

To adapt the policy, the DQN agent samples experiments from the training replay buffer that are similar. This boosts the amount of available data for adaptation. However, this process is difficult to train due to the large potential bias.

Algorithm 3

Generative Representation Policy Iteration on Tree with DQN($\Phi, \omega, k, \epsilon, \pi_0$)

INPUT: D; Source of samples (s, a, r, s');ω, ϵ, ρ_o; depth K

OUTPUT: policy ρ_*

1. **Random Walk: Sample Collection**
2. **Representation Learning**
3. **Learning Representation Policy**

for $k = 1...K$ **do**

Initialize policy $\pi^{'}$

repeat

$\pi_t = \pi^{'}$

$\pi^{'} = DQN(D_0, \omega, k, \epsilon, \pi_t)$

$t = t + 1$

until $\pi_t \sim \pi^{'}$

We consider a distinct output unit for each possible action in the action set and use the state representation as input to the Deep neural network. The outputs correspond to the predicted $Q-$values of the individual actions for the input state. The key advantage of this type of approach is the ability to compute $Q-$values for all possible actions in a given state with only a single forward pass through the network.

4.4.2 Learning Q-Values For NNI Distance

We propose to use a Reinforcement Learning algorithm to calculate optimal Q-values exactly and efficiently for the value metrics of negative NNI distance

$$V(\hat{\tau}_{nni}, \tau_{nni}) = -\delta_{nni} \tag{4.5}$$

which can be easily embedded with a reward function that assings -1 to an nni (swap) operation and 0 to a move action.

Actions are chosen using a greedy method, gerneating the next step sample from the tree space environment. We record the results in the replay buffer and also run an optimization step in every iteration. Optimization picks a random batch from the replay buffer to do training of the new policy. A target network is used in the optimization to compute the expected Q values. It is updated occasionally to keep it current.

4.4.3 Deep Q-Learning

The main idea behind this deep Q-learning model is to utilize an RNN trained on data, as the state representation. The model uses a standard DQN implementation, complete with an experience buffer and Target Q-network. A trained RNN is used to supply the initial values of the states in the Q-network and Target Q-network. In this deep Q-learning, as Figure 4.5 shows, we use LSTM-FC as the distance estimator. The pre-trained LSTM-FC takes a batch of pair of trees as input, converting it to an embedding state s at its output.The corresponding output will then be used as input to the Q-network as the policy network which is used to approximate $Q(s, a)$ values for each action in state s.

The loss function here is mean squared error of the predicted Q-value and the target Q-value, Q*, which is derived from the Q-value update equation derived from the Bellman equation, $Q(S_t, A_t) = Q(S_t, A_t) + (\alpha\gamma \max_{a'} Q(S_t, a)) - Q(S_t, A_t)$.

$$Loss = \| \ r + \gamma max_{a'} Q(s', a'; \theta') - Q(S_t, a; \theta) \ \|_2$$

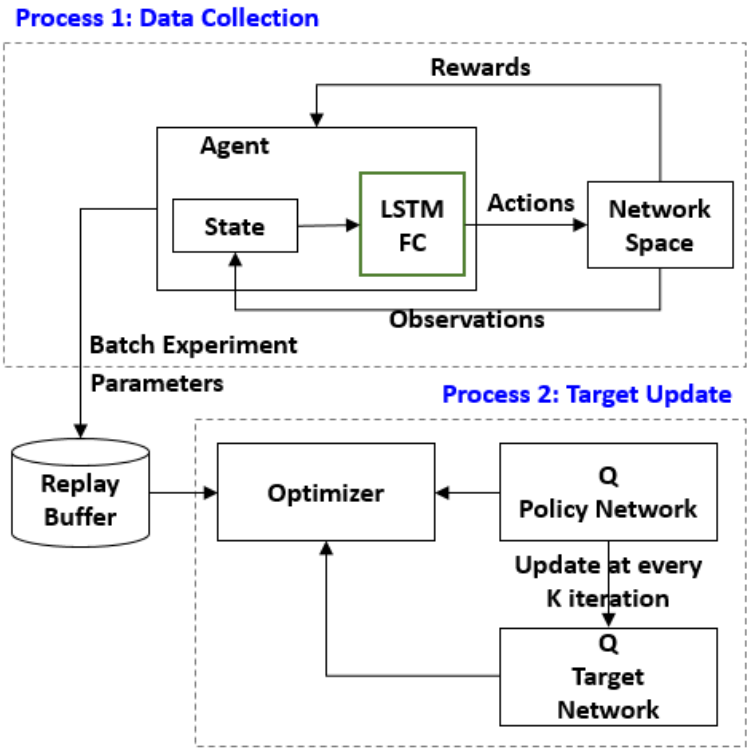


Figure 4.5: DeepQ - NNI Policy Learning Framework.

4.5 Rollout Approach

Especially in early stages of training in RL systems, basing value updates on a single time step can be inefficient and performing longer look-aheads can improve learning performance. A common way to include such look-aheads is in using rollouts. In this section we show how we apply the rollout algorithm to train the agent to master the nni distance in the tree space environment. The rollout algorithm is a decision-time planning algorithms based on Monte-Carlo tree search to simulated trajectories that all begin at the current environment state. It estimates action values for a given policy by averaging the returns of many simulated trajectories that start with each possible action and then follow the given policy. When the action-value estimates are

considered to be accurate enough, the action having the highest estimated value is executed [41].

In a training step, the system executes a single episode of the task using Monte-Carlo tree search with the given agent network. It returns the experience tuples collected during the search. During Tree search it performs multiple simulations in the tree (following trajectories) until a given number of nodes have been encountered and it returns the leaf nodes which were encountered. Then it expands and evaluates these leaf nodes. Given the number of leaf states which the agent network can evaluate at once, the tree search limits the number of simulations to stay efficient. Figure 4.6 shows the cycle of four phases such as Selection, Expansion, Rollout/Simulation, and Back-propagation in Monte Carlo tree search.

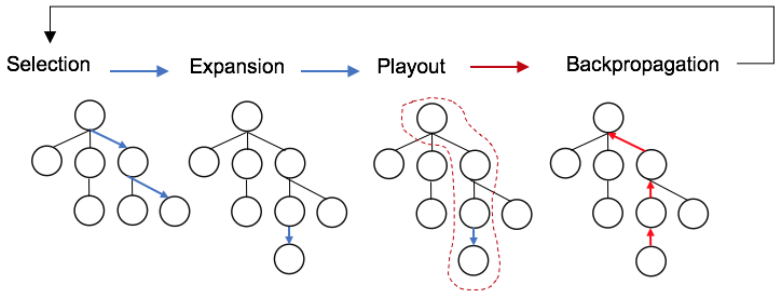


Figure 4.6: Rollout Cycle

This algorithm builds on the idea of iteratively improving a deep policy network and a tree search by looking at the successors. The policy improves its estimates by planning ahead via the tree search. Conversely, the tree search makes use of the progressively more accurate policy to estimate the best branches to explore during the search. It changes the way to estimate the action or the successor state. The basic idea is that by looking deeper in the future, the value which is backed up is closer to the real value than the value obtained when only looking at the direct successors.

The value function for any tree pair that is closer to the goal should always be a more accurate estimate than for a pair that requires a large number of nni operations to be resolved.

Figure 4.7 shows the architecture of the proposed approach for deep reinforcement learning that uses the rollout algorithm for optimization of the deep neural network. Tree search performs multiple simulations in the tree (following trajectories) until a given amount of leaves to expand have been encountered. Then it expands and evaluates these leaf nodes. If it reaches the goal state, it does not need the agent network to bootstrap. It can backup the value right away. Otherwise, it enqueues the leaf for evaluation by the agent network. Finally it evaluates the leaf-states all at once and backs up the value estimates.

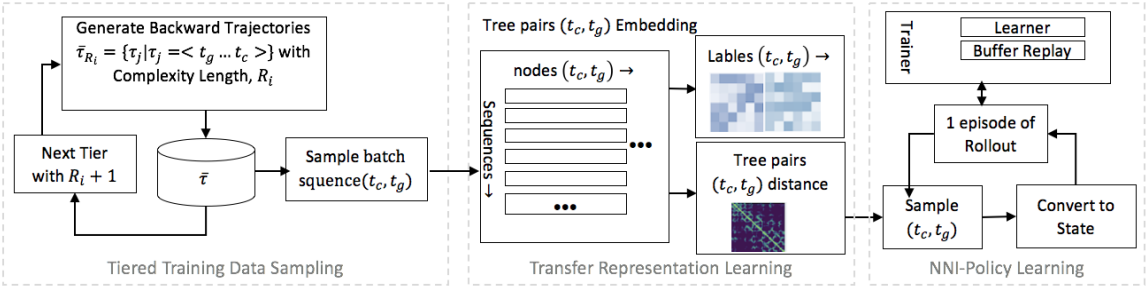


Figure 4.7: This framework supports batching with different size, which can be enabled by setting the batch size.

4.6 Simulated Synthetic Data

We have used artificial tree collections to predict how the performance of the algorithms scales across a wide range of dataset sizes. To generate artificial datasets, we first generate an initial random binary leaf-labeled rooted tree τ_i with n nodes, where $n \in \{15, 25, 50, 100, 200\}$. These trees were generated randomly by using the

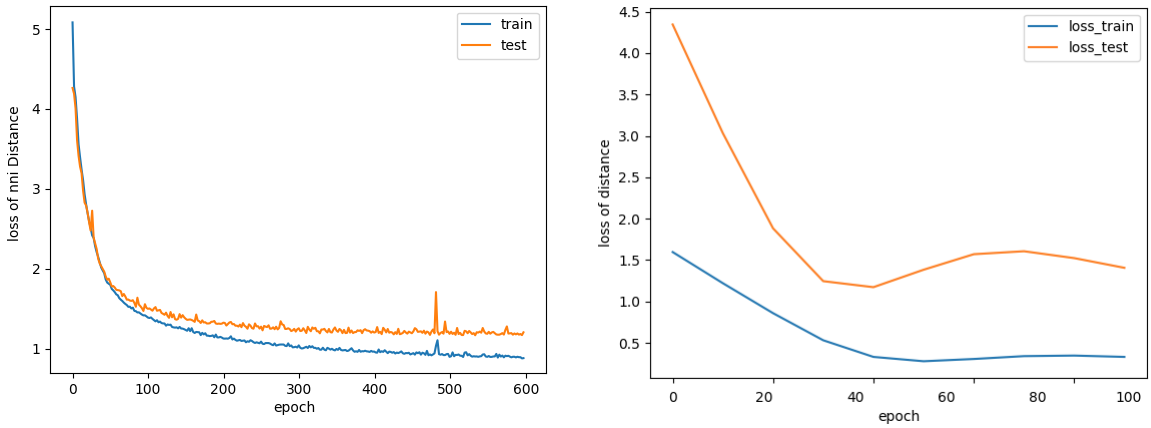
algorithm introduced by Arnold and Sleep [14] that assigns equal probability to all members of the family of trees with n nodes. This method produces a uniform distribution of random binary trees [15]. These are the target trees for our algorithm.

To generate the source trees for our tests, we take the target tree and perform a number of actions A to generate exhaustive neighborhood trees a certain number of steps away. At each tier, we choose a uniform distribution of random neighbors to perform all possible numbers of actions A on all internal edges of the tree to generate all distinct new neighbors. We then select a tree from the resulting neighborhood. This process is repeatedly performed to generate trees with a set number of NNI operations away from our source trees. In our experiments, given each generated target tree, we applied k rounds of NNI operations to our target to generate the source tree. This puts an upper bound of $k - nni$ steps between our trees. For these experiments, we used $1 - nni$, ..., $k - nni$.

4.6.1 Distance Learning Evaluation

In this section, we study the effectiveness of transferable exploration in the domain of predicting NNI distance. In this phase, our model proposes inputs (test cases) to the program being tested, with the goal of finding the target point. Figure 4.8 shows convergence of the error in predicting the number of NNI steps in 600 epocs. Mean squared error (MSE) loss was minimized with stochastic gradient descent and plotted here against training epoch count. The validation error tracks the training error without upward divergence, demonstrating a stable training with NNI bias-variance trade-off.

The train and test set from the tree space is represented in two dimensions based on the configuration and size of the trees. To fully evaluate the performance in learning general representations, we investigate three aspects in multiple dimensions



a. b.

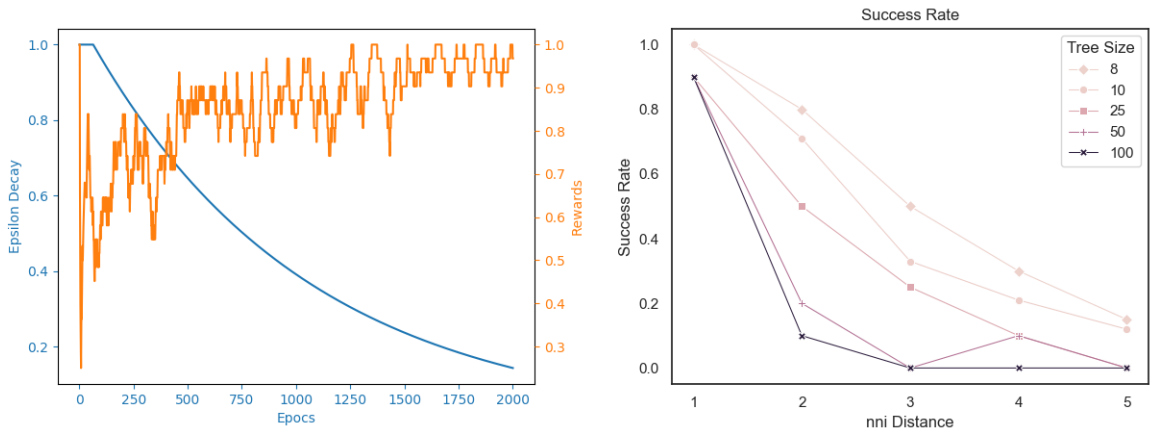
Figure 4.8: a. Mean squared error (MSE) losses for NNI distance prediction during training and validation on a tree space with 12-*nni* distance radius and vaying tree sizes. Validation losses were calculated using batches with variable size validation examples after every training batche. b. (MSE) losses of NNI distance prediction during training and validation with trees of a fixed size

based on configuration and size of the data set. First, we show that the system can predict the configuration of a tree which has not been seen previously. The next investigation is whether the size is outside the envelope. Last, it is tested whether overfitting is happening in the hidden representation. We evaluate the model with different ranges of tree sizes and *nni* distances by generalizing the range n between $N_1 < N_2$ such that $N \in \mathbb{N}$.

4.7 Evaluation of Policy Learning

In this section, we study the effectiveness of transferable exploration in the domain of NNI. Our model proposes inputs (test cases) to the program being tested, with the goal of learning a policy to generate as many actions as necessary to reach the target. The information needed to be able to built a policy might still be finite given the infinite space with its finite representation. The training set is represented

as a finite set of elements of the infinite tree space. By generating a test data set, we showed that the proposed system can learned a successful tree transformation policy based on the tree structure in a form of a representation that has not been trained before. Fig 4.9 shows the success rate to reach the goal without the use of rollouts for different tree sizes.



a. b.

Figure 4.9: a. The agent's learning convergence for nni transfer. b. Success rate of reaching the target for different size of tree with maximum 5-*nni* distance.

This data shows that while the system is able to learn policies for smaller NNI distances, the failure rate becomes high for more complex problems. To address this, the addition of rollouts through Monte-Carlo Tree search is used to bolster performance by allowing for more competent backups. Fig 4.10 shows the performance of Monte Carlo Tree Search combined with a policy that is used to narrow the search tree pairs in the complex tree space.

This data shows a significant improvement in the policy with much higher success rates of the policy even with complex tree transformation problems.

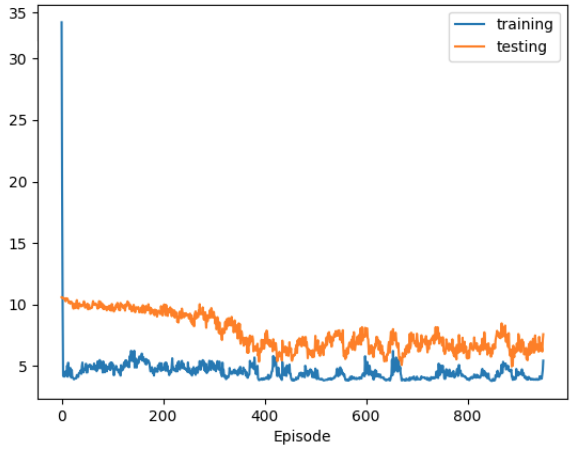


Figure 4.10: MSE of distance loss with maximum 12-*nni* distance (actual steps) and with max budgets of 32 steps. Rollout effectively can approximate the target with between [4-8] NNI step.

4.8 Conclusion

In this chapter we proposed a representation and policy learning framework that learns a representation for arbitrary sized binary tree pairs using recurrent LSTM networks and a policy to transfer one tree into the corresponding target tree using Reinforcement Learning. Here, the representation is pre-trained using tree transfer similarity to transform pairs of trees into an approximate numerical multidimensional vector which encodes the original structure information. This model, used with a deep reinforcement learning approach, yields a constructive method for generating basis functions for approximating value functions and permits to learn an efficient, general tree transfer policy that incrementally transforms the source tree into the target tree. We have shown that in the context of the tree transformation with arbitrary sized problem, an agent version of the rollout algorithm has greatly reduced computational requirements while still maintaining the fundamental cost improvement property of the standard rollout algorithm.

CHAPTER 5

Local Tree Neighborhood - Learning Hyperbolic Representations of Tree Pair Differences

5.1 Introduction

Learning tree pair edit distance representations via low dimensional embeddings that preserve relevant tree properties is an important class of problems in machine learning and bioinformatics. Particularly, we are interested in geometrically modeling tree pair and tree transfer structures via low dimensional embeddings. Examples of applications include predicting maximum likelihood, tree edit distance, and reconstruction of phylogenetic trees from a given set of genetic data [42] which are widely used for tasks ranging from tree transformation to sentiment analysis. Thus, representation learning has become an invaluable approach for learning dynamic behavior in tree space such as tree pair edit distance. However, state-of-the-art embedding methods typically do not account for latent hierarchical structures which are characteristic of many complex symbolic datasets as well as the related action sequence for tree pair edit distance representations.

In this chapter, we present a novel method for embedding tree pairs in the hyperbolic space for use in edit distance tasks. This approach improves the representational capacity of previous embeddings on the tree pair edit distance prediction task. Recent experiments show that hyperbolic spaces provably model tree-like structures better than Euclidean geometry. Also, hierarchical relations can be presented as partial orders. The key components are to induce a partial order relation in the

embedding space and have an appealing closed form expression derived in a principled manner.

5.2 Background

Different schemes and algorithms were introduced over the past decades based on interesting characteristics of hyperbolic embeddings [43]. Here, we introduce a new approach for learning hierarchical representations for tree pair edit distance by embedding tree pairs into hyperbolic space or more precisely into an n-dimensional Poincaré ball before utilizing this representation to derive a fixed size embedding vector for edit distance prediction using a recursive LSTM network. Due to the inherent implicit hyperbolic geometry property, this allows us to learn parsimonious representations of tree pairs by simultaneously capturing hierarchy and similarity. We present an efficient method to learn the embeddings based on Riemannian optimization and show experimentally that Poincaré embeddings can outperform Euclidean embeddings significantly on data with latent hierarchies, both in terms of representation capacity and in terms of generalization ability.

In general, tree structured data sets are very high dimensional. In order to maintain connectivity between local patterns, we need to know structure and topology better. Relation with neighbors and the dimension depends on the taken action to reach the corresponding neighborhood. As a result, the pattern should equally likely exist for every orientation in the space of trees, except if the edge or node has a particular property that assigns it to a particular dimension. They should be variable but all in the same direction. For example, given a change of orientation of one, all dependent elements (i.e. nodes and edges) should be rotated in the same way. So one nice property of tree networks is that they are orientation-free. Otherwise the local connectivity will be distorted. Therefore, the structure of a pattern will be preserved

and orientation is taken into account by embedding the direction. The orientation in a graph is defined by the selection of the edge to traverse. Changing orientation corresponds to remapping the dimensions in the underlined space.

We consider an unknown dynamic environment, where the agent observes a network of trees through random walks, with each tree corresponding to a visited unique environment state, and each edge of the traversal tree corresponding to an experienced transition. In this configuration, the network grows in size during an episode, and the agent controls the speed of this growth. The agent constructs the map for the unknown dynamic environment while keeping track of its local zone. The more of the network one can visit, the better the trajectory map reconstruction could be. With limited time or simulation trials, having a good exploration strategy is important. The key innovation is a representation of value functions for variable size trees, using the "task based features" of local neighborhood.

5.3 Related Works

Generating high quality feature representations and corresponding re-mapping of data such as text or images is a central point of interest in artificial intelligence. A large area of research focuses on embedding discrete data such as graphs [5, 44] or natural language processing and linguistic instances [45, 46] into continuous spaces that present desirable geometric feature properties.

Existing different embedding approaches for tree and acyclic graph structures are based on Neural Networks such as Graph Convolution Neural Networks (GCNs) [47] or GraphSAGE [48] which are state-of-the-art models for representation learning in graph data sets, where nodes or edges in graphs or trees are mapped into points in Euclidean space [49]. However, many real-world graphs, such as protein interaction networks and social networks are scale-free or have a tree structure for their data

representation [50]. Therefore, existing GCNs or GraphSAGE embeddings generate a highly biased and deformed Euclidean embedding of the graph structures. In general, the volume or size of a tree, defined as the number of leaves within some radius from a given center node as a root of the tree, grows exponentially as a function of the radius [51, 52]. However, the volume of balls in Euclidean space only grows polynomially with respect to the radius, which leads to high distortion embeddings while volume grows exponentially [53, 54]. Moreover, the embedding techniques with GCNs can only handle networks with particular size and can not handle variable size data structures. Once it becomes bigger, as a result the network has to grow and extend.

Alternatively, [55] proposed embeddings in hyperbolic space. Hyperbolic geometry enables embeddings with smaller distortion when embedding scale-free graphs [55]. However, current hyperbolic embedding techniques only account for the graph structure and do not leverage rich node features. On the other hand, in contrast to Euclidean space where each dimension is flat, in hyperbolic space, the space is curved. Therefore naturally points in this space move faster apart from each other as they move away from the center, so this means the available space through moving, grows faster compared with Euclidean space. As a result, it changes the density of leaves, making them easier to separate.

5.4 Preliminary and Notation

Given a graph $G(V, E)$ representing a network in tree space, i.e. each node in this graph be an undirected or directed tree τ. A tree τ^v in this space as a source tree and a set of transformed candidate trees τ^u could generate a partial local observation with given edges connecting the trees to tree τ^v through NNI and move operations. We label those edges of τ^v that generate new sample states in the future as target

nodes $N^D = \{\tau_1^d, ..., \tau_k^d\}$, and other belief nodes which τ^s does not generate edges to (no-linked nodes) $L = \{l_1, ..., l_n\}$. We label candidate nodes as a partial local observation with a set $O = \{o_i\} = N^D \cup L$. We assume N^D to be a set of positive nodes and nodes in L as negative training samples.

Given u and v (i.e. tree pair (τ^u,τ^v)), We assume that each edge (u, v) has a corresponding feature vector $\hat{\chi_{uv}}$ that describes the nodes u and v (i.e., pair of trees). For each edge (u, v) in G, we calculate the distance as the score of connectivity level $a_{uv} = F_w(\hat{\chi_{uv}})$. Function F_w parameterized by w takes the edge feature vector $\hat{\chi_{uv}}$ as an input and estimates the corresponding edge score a_{uv} that models the tree transformation. It is exactly the function $F_w(\hat{\chi_{uv}})$ that we learn in the training phase of the learning representation algorithm. Now our task is to learn the parameters w of function $F_w(\hat{\chi_{uv}})$ that assigns each edge.

5.5 Hyperbolic Transfer Representation

In this step, we are interested in geometric modeling of tree pair relations in the context of NNI distance and similarity via low dimensional embeddings of the tree pair as a vector $x \in \mathbb{R}^n$ which represents a more general concept than any embedding in Euclidean space.

Our goal is to produce a hyperbolic embedding that preserves all distances between nodes in the tree but also encodes similarity between the two trees. Starting from the same motivation [56], the hyperbolic embeddings method explicitly models hierarchical data structures. [57] shows the connection between hyperbolic space and tree distances and provides the relationship between reconstruction and learning for hierarchical data structure embeddings through partially ordered structure.

Partially ordered sets: The concept of hierarchy forms a partially ordered set (poset), which is a set $\mathbb{X}$ equipped with reflexive, transitive, and anti-symmetric binary relations $\preceq$. We extend this idea for application to a tree pair. Considering the reachability from one node to another in hierarchy order, we obtain a partial ordering. The partial ordering is equivalent to the inclusive relation between certain subsets called the lower cone (or lower closure), $C_x = y \in \mathbb{X} | y \preceq x$ [43].

Different version of hyperbolic models include *Poincaré disk*, *Poincaré ball*, *Poincaré half-plane*, and *hyperbolic cones*. For example, as Figure 5.1 shows, hyperbolic space with Poincaré disk D^2 is a two-dimensional model of hyperbolic geometry with points located in the interior of the unit disk . The hyperbolic space provides several useful properties, with a key property of this space being the relation between the mapping and the corresponding interpretation in Euclidean space. In particular, angles are preserved between hyperbolic and Euclidean space.

The hyperbolic distance: in a Poincaré disk D^2, the distances between any two points x_1 and x_2 are given by curves minimizing the distance between these two points and are called geodesics of the hyperbolic plane. To calculate the distance of a geodesic between two points x_1 and x_2 and thus obtain their hyperbolic distance D^H, we use the Poincaré metrics which is an isometric invariant.

$$D^H(x_1, x_2) = acosh(1 + \frac{2(|x_1 - x_2|^2))}{(1 - |x_1|^2)(1 - |x_2|^2)}) \tag{5.1}$$

The hyperbolic distance $D^H(x_1, x_2)$ is additive along geodesics and is a Riemannian metric [58].

A NNI-distance tree hyperbolic embedding is a mapping $F : (\tau_i, \tau_j) \rightarrow V$ for a pair of trees τ_i, τ_j and an embedding v with distances $d(\tau_i, \tau_j)$ and D_H.

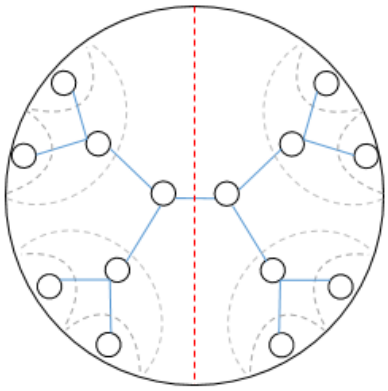


Figure 5.1: Tree in the hyperbolic disk. Each edge has to be embedded to a minimum length depending on the cone in which it is embedded. The distance of nodes increases exponentially (relative to their Euclidean distance) the closer they are to the boundary.

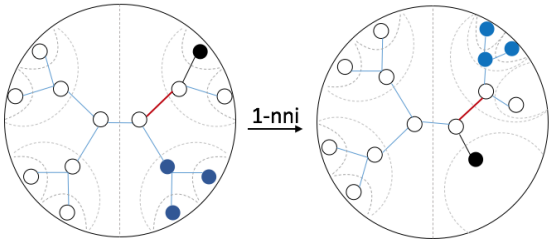


Figure 5.2: Embedding pairwise nni distance (τ_i, τ_j) in the hyperbolic disk.

5.5.1 Geometric Embedding of Tree Pairs

The nature of hyperbolic space lends itself towards perfect tree embeddings. Therefore, it is possible to embed trees into the Poincaré disk H^2 with arbitrarily low distortion.

We use a two-step process for embedding a tree pair into hyperbolic space:

- Embed two trees (τ_i, τ_j) into one tree T,
- Embed T into the Poincaré ball H^d.

The idea is to embed the root of two trees at the origin and recursively embed the children of each node in each tree in the pair by spacing them around a sphere centered at the parent. As Figure 5.2 shows, the radius of the sphere can be set to

precisely control the distortion. There are two factors to good embeddings: local and global. Locally, the children of each node should be spread out on the sphere around the parent. Globally, subtrees should be separated from each other; in hyperbolic space, the separation is determined by the sphere radius.

The goal of embedding a pair of tree (τ_i, τ_j) into a space V is to preserve the local structure of the tree within the embedding while maintaining information related to the nni distance (the shortest path with nni swap between a pair of tree) in the space V. If (τ_i, τ_j) are two trees and their nni distance is $d_{nni}(\tau_i, \tau_j)$, we would like the embedding to have $d^v_{nni}(\tau_i, \tau_j)$ close to $d_{nni}(\tau_i, \tau_j)$. Now consider two children (τ_i, τ_j) as subtrees of parent z in a tree. We place z at the origin O. The graph distance is $d(\tau_i, \tau_j) = d(\tau_i, O) + d(O, \tau_j)$, so normalizing we get $\frac{d(\tau_i,\tau_j)}{d(\tau_i,O)+d(O,\tau_j)} = 1$. Starting from the root, z, we embed each node by mapping it to coordinates x and y in the two-dimensional hyperbolic space. We start with x and y at the origin, and with equal speed move them toward the edge of the unit disk as nodes move away from the root.

5.5.2 The Optimization Problem

The training data contains information that source node τ^s will generate edges to target nodes $d \in D$ and not to nodes $l \in L$. So, we define the objective to set the parameters w of function $f_w(\hat{\chi_{uv}})$ so that it will assign edge weights a_{uv} in such a way that a walk will be more likely to visit nodes in D than L, i.e., $p_l < p_d$, for each $d \in D$ and $l \in L$. To compute Poincaré embeddings for a set of tree pair distance $S = \{(\tau_i, \tau_j)_k\}_n^{k=1}$, we are then interested in finding embeddings $\theta = \{\theta_i\}_{i=1}^n$ which map each node in the trees to a location in the hyperbolic space such that it maintains node distances in the trees where $\theta_i \in H^d$ can be used in predicting the distance of the tree pair. We assume loss function $L(\theta)$ which encourages semantically similar

tree pairs to be close in the embedding space according to their Poincaré distance. To estimate θ, we then solve the optimization problem θ

$$\begin{aligned} &\arg\min_{w} \quad \theta = L(\theta) \\ &\text{s.t.} \\ &\qquad \forall \theta_i \in \theta, ||\theta_i|| < 1 \end{aligned} \tag{5.2}$$

where in particular, we minimize the following loss function:

$$L(\theta) = \sum_{(\tau_i,\tau_j)} \sum_{(u,v)\in D} \log \frac{e^{-d(u,v)}}{\sum_{v'\in N(u)} e^{-d(u,v)}} + \sum \omega(u).d(u,v) \tag{5.3}$$

In loss function 5.3, the first term is aimed at preserving the structure information in each tree by attempting to maintain the node distances in each tree within their hyperbolic embedding. The second term, by contrast, is aimed at encoding the difference between the two trees (related to their nni distance) by regulating the distance between identical leaf nodes in the two trees. Through this we rank loss where related nodes should be closer than nodes for which their relationship has not been observed. This loss function is motivated by the fact that we don't want to push nodes belonging to distinct subtrees arbitrarily far apart as their subtrees might still be close. Instead we want them to be farther apart than nodes with an observed relation. We evaluate the performance of the embeddings similar to graph embeddings. For each observed relationship (u, v), we rank its distance d(u,v) among the ground-truth negative examples.

5.6 Measure of Tree Pair Embedding

Given a tree pair with 1-NNI distance, it can be presented in the Poincaré disk, H. For a given tree with n leaves $\{V_0, .., V_n\}$, the embedding coordinate v_i, denoted $V(v_i)$ is the set of points whose distance to v_i is not larger than the distance to v_j for

any $j \neq i$. Figure 5.3 shows a projection of the embedding into Cartesian space for a pair of trees with NNI distance of 1 and 8 leave nodes.

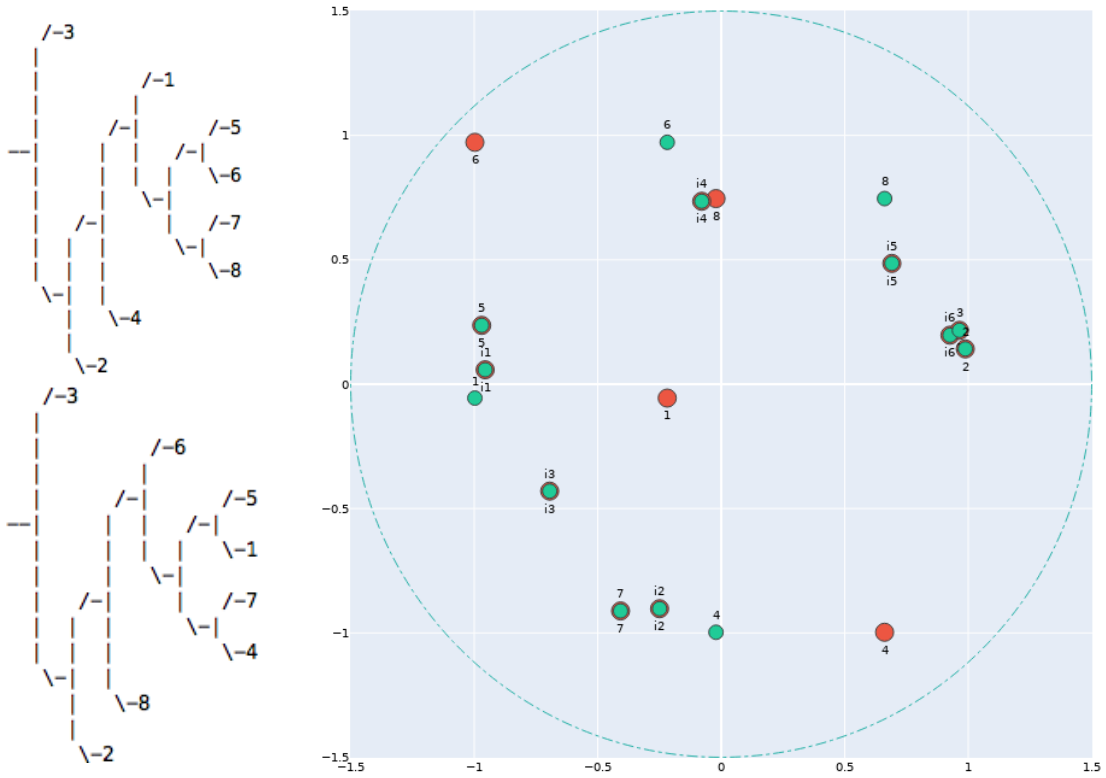


Figure 5.3: Sample of embedding pairwise nni distance (τ_i, τ_j) in the hyperbolic disk. Positions of node vectors in 2-D space from an actual Poincaré embedding with leaf nodes labeled with the leaf labels and intermediate node labels prefaced with i. Nodes for tree 1 (green) are labeled above the mark while nodes for tree 2 (red) are labeled below. Overlapping nodes are green and indicated with a bold outline. Note that as we move toward the boundary, we begin to see nodes lower and lower in the hierarchy. Also, nodes similar to each other are close or overlap each other.

This figure shows the embedded tree nodes labeled with the node ID. Nodes of the first tree are green while nodes for the second tree are shown in red with bold circled nodes in the embedding representing situations where the nodes for the two trees embed to approximately the same coordinate. This figure demonstrates how

the proposed hyperbolic embedding for tree pairs can maintain information about structural similarity within the hierarchical tree structure between the two trees.

In the first set of experiments, we evaluate the ability of the proposed Poincaré embeddings to embed tree pairs with NNI distance to present the relevant information in a clear latent hierarchical structure. For this purpose, we conduct experiments on tree pairs with variable size. In tree space, we need to measure the quality of the embedding based on a local or global view. Based on a local view, we might want a local measure that checks whether neighbors remain closest to each other without considering the exact distances between them. We also want a global measure that keeps track of these explicit values. We use mean average precision (MAP), proposed by [43] for our local measure for a tree pair; MAP captures how well each node's neighborhoods are preserved. It exactly measures the quality of an embedding.

We measure the quality of embeddings with MAP based on local and global points. For a dynamic environment of a tree space $G^\tau = (V^\tau, E^\tau)$, let a tree $\tau_i \in V^\tau$ have a neighborhood tree set $N_\tau = \{\tau_1, \tau_2, ..., \tau_{2(n-3)}\}$, where $2(n-3)$ denotes the number possible neighbors for τ_i with n leaves. In the embedding F, consider the points closest to $F(\delta_{nni}(\tau_i, \tau_j))$, and define ϕ_{τ_i,τ_i} to be the smallest set of such points that contains (τ_i, τ_j) (that is, ϕ_{τ_i,τ_j} is the minimum set of nearest points required to retrieve the $1 - nni$ neighbor of τ_i in F). The MAP is defined by Equation (5.4).

$$\begin{aligned} MAP(F) &= \frac{1}{|V^\tau|} \sum_{\tau \in} \frac{1}{|N_\tau|} \sum Precision(\phi_{\tau,\tau_i}) \\ &= \frac{1}{|V^\tau|} \sum_{\tau \in} \frac{1}{2(n-3)} \sum \frac{|N_\tau \cap_\phi \tau, \tau_i|}{|\phi_{\tau,\tau_i}|} \end{aligned} \tag{5.4}$$

In the ideal case, $MAP(f) \leq 1$ where equality is the best case. MAP is not affected by the underlying distances, but only by the ranks between the $1 - nni$ distances of immediate neighbors with respect to the target.

The standard metric for the quality of graph embeddings is distortion D. For an n point embedding as a global measure, we use distortion.

$$distortion = \sum_{(\tau_i,\tau_j)\in G,\tau_i\neq\tau_j} \frac{|d(\tau_i,\tau_j) - d_H(\tau_i,\tau_j)|}{d(\tau_i,\tau_j)} \tag{5.5}$$

5.6.1 Hyperbolic Tree Pair Embeddings for NNI Distance Prediction

The goal of the proposed hyperbolic embedding of tree pairs and tree pair differences in this dissertation is to utilize it to obtain a better representation for NNI tree transformation learning. The rationale here is that by embedding the trees in a way that maintains tree structure and tree difference information we provide the LSTM network used to address the variable size of the used trees with a more informative and easier to process representation which explicitly focuses on structure and tree differences while allowing to eliminate the need to explicitly encode tree labels in the input representation. The tree pair embedding model achieves this by defining a loss function that penalizes low distances between unconnected nodes and high distances between connected nodes, and optimizing it by training over the list of connected nodes, along with randomly sampled negative examples, using gradient descent. This a leads to significant improvements in terms of representation and generalization ability for tree pairs. For instance, Figure 5.3 shows how efficient a tree pair can be embedded in hyperbolic space.

Following the approach for NNI distance prediction and tree transformation policy learning for variable and unlimited sized trees, we again utilize a recurrent LSTM network to further embed the tree representations into a fixed length vector. The input to the recurrent neural network is a tree pair that represents a state. To allow the hyperbolic embedding of the tree pair to be fed to the LSTM network, a tree traversal on the current tree starting with the active edge is performed and the

input to the network is the corresponding sequence of node pairs from the current and target tree. Each set of coordinates represents the position of the node in the current tree with respect to its location in the target and is used at the input to the recurrent network at a time-step $t - T + 1, ..., t$. To further normalize the input, the tree space is oriented relative to the perspective of the current tree. Figure 5.4 shows the model architecture used for the hyperbolic embedding and its use for constant length tree embedding and NNI distance prediction.

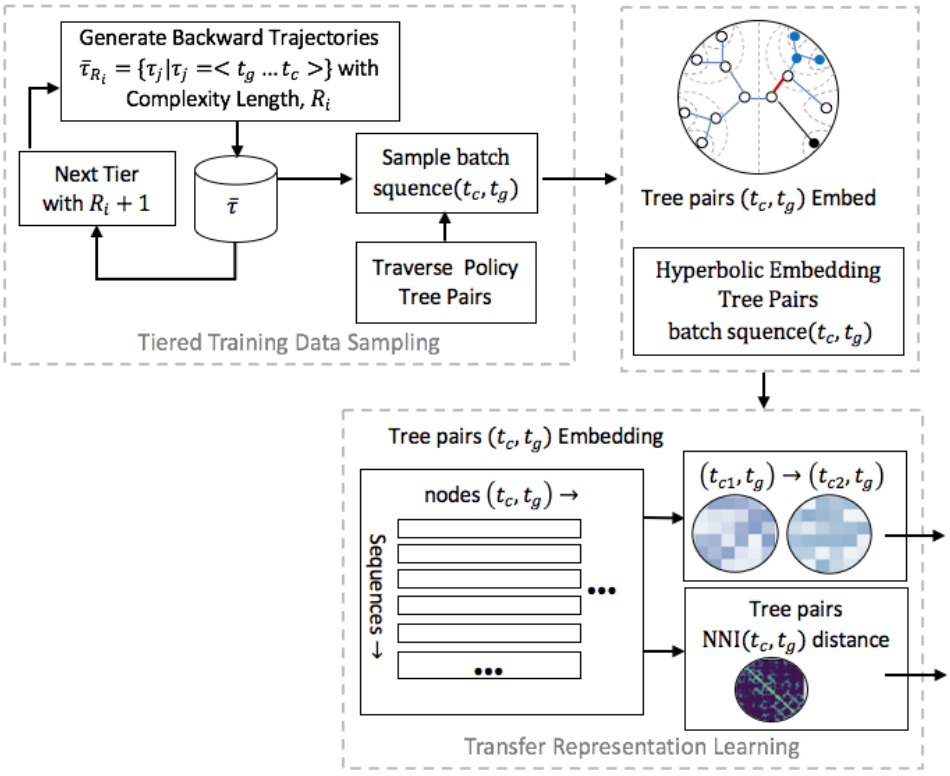


Figure 5.4: Supervised architecture model

As in Chapter 4, we use the recurrent network architecture to address variable sized tree pairs, utilizing that LSTMs can effectively preserve the characteristics of historical information in sequences, and extract local features of the tree pairs using

the structure of NNs. We again proposes a hybrid model of LSTM and fully connected NN, where we construct a fully connected NN model for NNI distance prediction on the top of an LSTM which is intended to learn a fixed size embedding of the relevant features of the tree pairs. In this way, the tree pair feature vector output from LSTM serves as the state representation and is driven by the task performed by the subsequent NN structure and learning task.

As opposed to the more direct application to the structure learning, the use of the hyperbolic embedding of the tree pair in the input to the LSTM should here reduce the burden on the LSTM. In particular, the LSTM does here not have to deal with local connectivity in tree space in the same way as the hyperbolic embedding deals with low level connectivity. Moreover, as leaf nodes of the two trees are now fed as pairs, the need for leaf labels to be included in the input disappears, further easing the burden on the LSTM network, allowing it to more directly concentrate on handling the arbitrary size of the trees and on identifying features relevant to the NNI prediction task.

The resulting combined approach should better enable agents to learn representations that reflect both their past experience and the inherent large-scale information of the environment, as well as learn policies by using these customized basis functions to approximate value functions.

To verify the benefits of the proposed hyperbolic embedding, we evaluated the performance of embeddings for predicting distance of tree pairs in a complex tree space generated by NNI edits. Since each tree in such networks can often be explained via latent hierarchies over their nodes, we are interested in the benefits of Poincaré embeddings in terms of representation of similarity and generalization performance for predicting distance metrics. For training, we randomly sample tree pairs and collect experimental results on the NNI transformation distance in different size trees with

in 5-NNI distance up to 5. Figure 5.5 shows the result of training and evolution for 500 steps. Experimental results demonstrate that our approach is extremely effective for distance prediction.

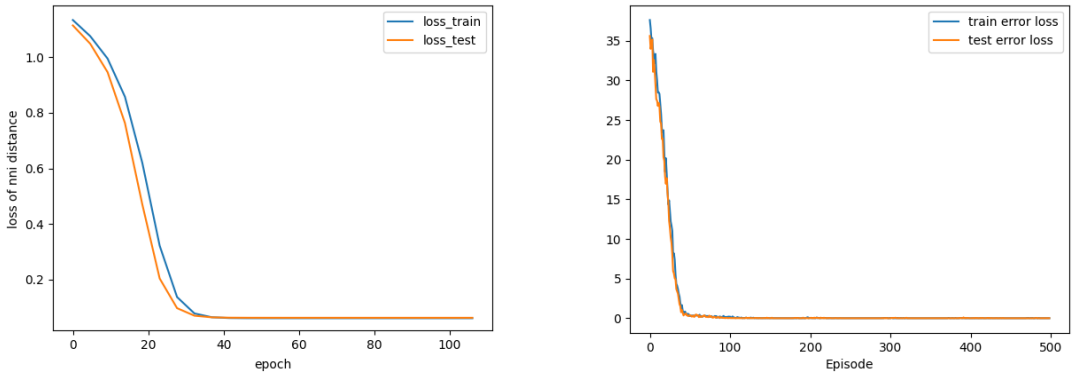


Figure 5.5: Training and test performance for NNI distance prediction in terms of loss in the number of NNI steps (left) and estimated squared error (right)

5.7 Conclusions

In this chapter, we introduced Poincaré embeddings for learning representations of tree pairs and showed how they can simultaneously learn the similarity and the hierarchy of two trees to predict the edit distance. In Addition, we proposed an efficient algorithm to compute the embeddings and showed experimentally, that Poincaré embeddings provide important advantages over Euclidean embeddings on tree edit distance. First, Poincaré embeddings enable parsimonious representations that allow us to learn high quality embeddings of variable size trees. Second, excellent distance prediction results indicate that hyperbolic geometry can introduce an important structural bias for the embedding of complex data such as genome. In the next chapter, we study Poincaré embeddings for value function approximation in

reinforcement learning problems with high dimensional state and a finite action space based on a generalized representation policy iteration.

CHAPTER 6

Deep Reinforcement Learning for Tree Transformation using Hyperbolic Representations of Tree Pairs

6.1 Introduction

In this chapter, we consider the classical NNI distance problem and efficient exploration of unseen environments in a network of trees with variable size by adapting a novel approach for learning tree pair representations utilizing embedding entities into hyperbolic space. We propose a learning exploration strategy where we learn a sampling policy from an unseen environment generated by NNI operations. The policy aims to generalize the sampling strategy over NNI and move steps to visit the minimum number of unique sample states in a limited number of NNI and move steps. We study a fundamental aspect of representation learning on tree pairs, in particular, the impact of the underlying geometry of embedding tree pairs into hyperbolic space.

We particularly focus on environments with tree-structured state-spaces that are encountered in many important real-world applications like bioinformatics and formulate this task as a reinforcement learning problem where the exploration agent is rewarded for transitioning to previously unseen environment states and employs a graph-structured memory to encode the agents past trajectory. Our proposed model uses the original AlphaZero algorithm [59], a general purpose Monte-Carlo tree search (MCTS) algorithm.

We present a generic framework for learning representation and behavior of tree datasets in hyperbolic space with a finite action space, integrating policy learning using MCTS with automated hyperbolic representation exploration. We evaluate the

value function approximation in reinforcement learning problems with high dimensional state and a finite action space based on a generalized representation policy iteration in hyperbolic space. We consider the limitations at accurately approximating the value function in low dimensions and highlight the importance of features learning for an improved low-dimensional value function approximation. We answer two questions:

- How can we build a representation of the local tree neighborhood which gives enough information for a tree pair to decide the next node?
- How should the agent move to gain the highest accumulated rewards through tree space?

6.2 Background and Related Works

The study of tree transformation is a fundamental topic both in computer science and bioinformatics. Tree edit distance subsequently became the grand challenge task for a generation of researchers since these tasks are highly tuned to their domain, and cannot be generalised to other problems without significant human effort.

We consider an unknown dynamic environment, where the agent observes a tree at each step, with each tree corresponding to a visited unique environment state, and each edge between trees corresponding to an experienced transition. In this configuration, the network of trees grows in size during an episode, and the agent maximizes the speed of this growth. The agent constructs the map for the unknown dynamic environment while keeping track of its local zone. The more trees one can visit, the better the trajectory map reconstruction could be. With limited time or simulation trials, having a good exploration strategy is important.

The key innovation is a representation of value functions for variable size trees, using the "task based feature" as a smooth function with partial observation based on

the local neighborhood. This approach allows applying powerful mathematical tools for basis function generation by exploiting information obtained during training. This representation may not transfer information as much as generic features but is able to focus on features relevant to a specific task. There might be a lot of information which may be irrelevant with respect to the set of goals which we are trying to solve, but not irrelevant across different actions in the local neighborhood.

6.3 Preliminary

Given in a graph $G(V, E)$ that represents a network in tree space (i.e. each node in this graph is a tree), a source tree τ^s as a source node state and a set of candidates of tree τ^c could generate a partial local observation with given edges connecting them to tree τ^s through NNI and move operations. We label those edges from τ^s that generate new sample node states in the future as target nodes $N^D = \{\tau_1^d, ..., \tau_k^d\}$.

We assume the availability of a tree space $G(V, E)$ with a given set of actions A and a derived path $P = \{p_i\}_{i \in k}$ with known target, where p_i is a graph query (generated induced graph). The goal of RL is to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to maximize the expected rewards with finding correct trajectories.

Without having access to all the environment, we need to define a navigation strategy that can traverse the space with the aim to find the target. In an episodic manner, the upcoming observations rely on the current query. The policy is how the agent can find the given particular pattern based on the current set of observations.

6.4 Feature Based Learning

Intuitively, we generalize from seen nodes to unseen nodes in the network environment by feature learning. Thus, the challenge is that the current node is very

highly correlated with respect to the next node that will be visited. The nodes' data that is obtained by following a trajectory is thus highly correlated and learning that relies on iid techniques is not effective. Therefore we require a training method that is suitable for non-stationary and non-iid data. We estimate the value of the state corresponding to a pair across the nodes and edge space by considering differentiable function approximation $Q(s,a) \approx Q^h_\phi(s,a)$ in the hyperbolic geometry. In the first step, the learning algorithm (feature navigator) is to derive a set of base tree features using the tree pair topology and some available attributes on each sampled nodes and edges. These features represent generic aspects of the tree pairs.

6.5 Action Based Features

Action based features are conceptually easier and may be better at representing the important aspects of the state than general state features in the context of Reinforcement Learning. But they may not transfer information as much as generic features. The reason for this is that there might be a lot of information which may be irrelevant with respect to the set of goals which we are trying to solve.

6.6 Space Exploring Policy

We first define the building blocks that we use to define our search space in the given environment. We consider we have a set of seed nodes as initial states:

Action-based features should carry all information from the past versions of G_s, of which we only get to observe the final in the form of a static snap-shot. Note that we also do not know the ages of nodes and edges. Let G^s_{t+1} represent graph G^s_t at some point in time $t+1$, when it had exactly N^R_{t+1} nodes. Now, we want to find a new sample G_s with N_R nodes that is most similar to graph $G^{N_R}_{t+i}$, i.e. an expansion

of the current snapshot when graph G_s was of the same size as S. The hard part here is that we want to match patterns describing the temporal evolution together with the patterns defined on a single snapshot of a graph, which also changes over time. If one would have node ages, then the best possible approach would be to simply roll-back the evolution (addition/deletion of nodes and edges over time). Note that our sampling algorithms do not know the age of individual nodes and edges. So the question here is whether we can roll-back the time without having any temporal information (age of nodes/edges).

6.7 State Representation: Tree Pairs in Local Neighborhood

Let the state be the representation of observed tree pairs in tree space with the given action set. In the work presented here, we utilize recurrent neural networks to learn the representation and the input to the recurrent neural network is a tree pair that represents the state. Each set of feature values/coordinates represents the position of a tree with respect to a target at a time-step t. The tree space is oriented to the perspective of the current tree.

In this section we describe the state representation of the tree pair inputs, and the representation of the action outputs, used by the recurrent neural network block, a re-trained model of the LSTM network introduced in the previous chapter. The pre-trained LSTM model was trained to predict the number of steps to be taken to transfer one state into another one. LSTMs were chosen as they can effectively preserve the characteristics of historical information in sequences, and extract local features. We proposes a hybrid model of LSTM and fully connected layers. At first, an unsupervised NN constructs the hyperbolic embedding of the tree pair as described in Chapter 5 which, in the form of a sequential tree traversal of current/target tree node pairs serves as the input to the LSTM network. The LSTM learns to compress

this traversal sequence into a relevant, fixed size feature vector that addresses the variable tree size and captures the task-relevant aspects of the state as driven by the learning problem applied on one of the follow-up networks, either a fully connected network for NNI distance training during the pre-training stage, or a Q-Value network for the policy learning stage. The output of the LSTM represents the hidden state representation used for both the prediction and the policy learning problem. We use the last hidden layer before the fully connected one from the pre-trained model as the initial state representation for the value function.

The resulting combined approach enables agents to learn representations that reflect both their past experience and the inherent large-scale information of the environment, as well as learn policies by using these customized basis functions to approximate value functions.

6.8 Improving Deep-Q RL with Rollout using Hyperbolic Geometry Approximation

The algorithm builds on the idea of iteratively improving a deep policy network and a tree search in tandem. The policy improves its estimates by planning ahead via the tree search. Conversely, the tree search makes use of the progressively more accurate policy to estimate the best branches to explore during the search. The basic mechanism of the tree search-based rollout used during pre-training and policy learning is shown in Algorthm 4.

Our implementation was adapted from the original AlphaZero [59] algorithm implementation, which is limited to the game of Go (two-player game). We modified the algorithm to play single-player games and based it on a previous version from a repository that provides an implementation of the MCTS algorithm that is independent of any deep learning framework. Each search consists of a series of simulated NNI transformations done by the agent that traverse a sub-tree of the tree space from

Algorithm 4 General Rollout with Pre-Train

procedure LEARN_NNI_DISTANCE
Input: Starting transition $\tau_0 = (X_s^1, a_1, r_1, X_s^0)$.
    Π: Policy to be evaluated
    Returns(X_s): an empty list, $\forall X_s \in S$
    V_h: Value Function
    Dynamic model f with error estimate
    Reward model r.
Output: Transitions Trajectory $\tau_1, \tau_2, .., \tau_k$
Initialize Weights from pre-trained block:
Repeat
    Generate an episode using Π
    For each state X_s observed in the episode:
        Return that path P occurrence from state X_s
        Append P to Returns(X_s)
        $V(X_s)$ Average(Returns(X_s))

root (corresponding to the current tree pair towards the leafs (corresponding to tree pairs where the current tree is equivalent to the target tree or tree pairs that are at the fringe of the search horizon). As Algorithm 4 explains, each MCTS simulation proceeds by choosing in each state X_s an action with low visit count, high move probability and high value (averaged over the leaf states of simulations that selected action a from X_s) according to the output estimation of the current recurrent neural network $F(\theta)$. The search returns a vector Π representing a probability distribution over actions, either proportionally or greedily with respect to the visit counts at the root state. The parameters θ of the deep recurrent neural network are trained by

reinforcement learning, starting from pre-trained parameters θ. At the end of run, the terminal position X^T is scored according to the rules of the reward function to compute the outcome $Z : -1$ for a swap, 0 for a move, and $+1000$ for reaching the target state. The neural network parameters θ are updated so as to minimise the error between the predicted distance v_t and the NNI distance outcome Z, and to maximise the similarity of the policy vector pt to the search probabilities Π_t. Specifically, the parameters θ are adjusted by gradient descent on a loss function that sums over mean-squared error.

This mechanism is integrated into an overall learning architecture that utilizes multiple neural network structures to achieve hyperbolic tree pair embedding to a traversal sequence of pairs of tree coordinates for arbitrary sized trees, fixed size embedding learning into a feature vector representing important tree structure differences, prediction of NNI distances, and, finally, learning of tree transformation strategies and corresponding value functions using deep Q-learning. As shown in Figure 6.1, which presents an overview of the complete RL architecture used, we provide a pipeline in which the agent learns to find the best strategy to perform NNI tree transformations and thus the best path through the tree space environment by using tree pair similarity.

6.9 Results

We have tested the fixed steps rollout with numbers of steps of $\{1, 2, 3, 4, 5, 8, 32\}$ on the NNI distance problem. We have used 2-layer recurrent neural networks to learn the fixed-length state approximation f and a 2-layer fully connected network for the the value function learning V, with hidden layer dimensionality of 64 for f and 256 for V. f and V both use a dropout rate of 0.05. All neural networks use tanh as their hidden activation and no output activation. For optimizer, we used the Adam

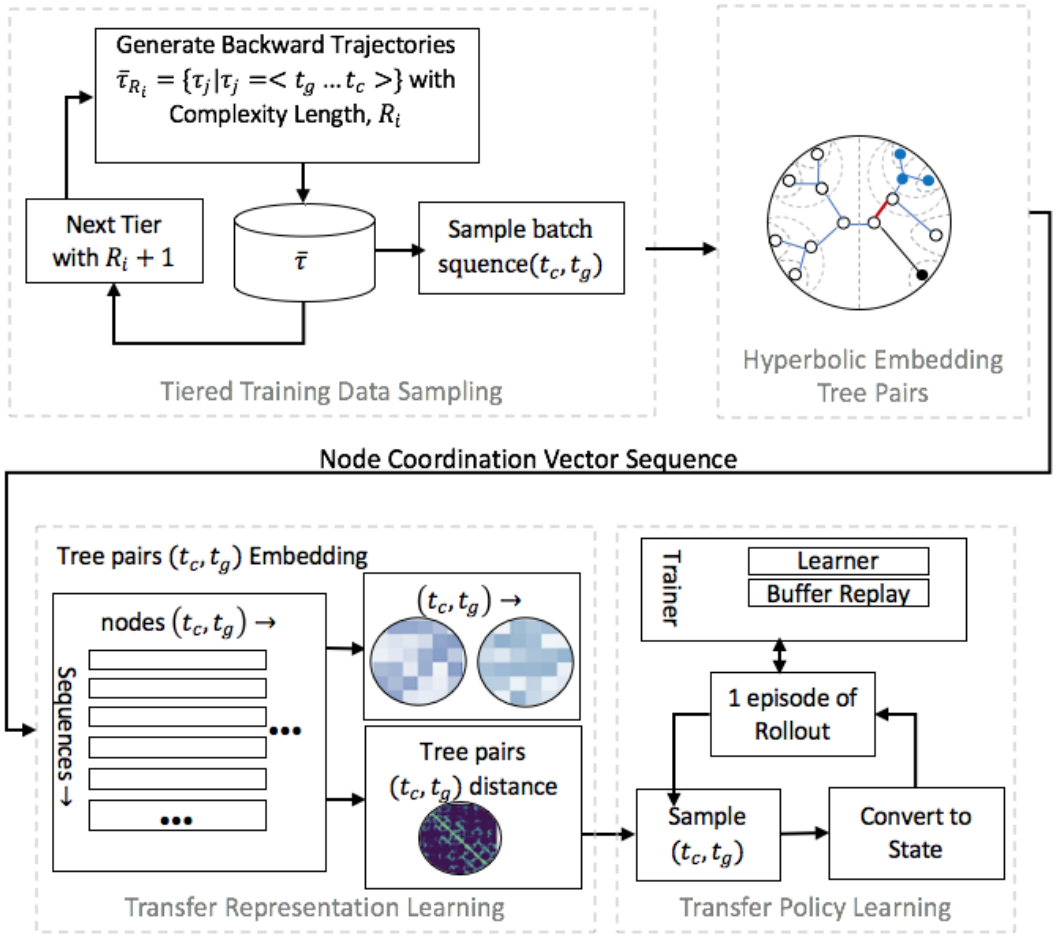


Figure 6.1: Overall architecture and training pipeline. The hyperbolic embeddings of tree pairs are constructed in the first stage and, as a tree traversal sequence of current/target node pairs fed into the LSTM embedding network. The output of this network, having initially been pre-trained using NNI tree distances, serves as the feature representation for the deep-Q Reinforcement Learning component for the extrapolated tree transformation policy learning task.

optimizer [60] for training with a learning rate of 0.001. Figure 6.2 shows the average episode rewards for 800 Epocs averaged over 10 runs. Running 10 runs for variable size tree $\{8, 10, 25\}$, Figure 6.3 shows the performance of rollout during training on variable sized tree pairs with NNI distances between 1 and 5 with variable size tree without normalization.

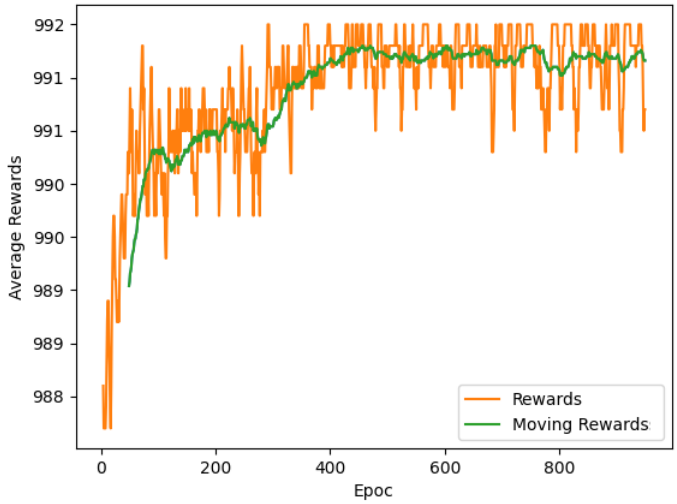


Figure 6.2: Performance of agent during training on variable sized tree pairs with NNI distances between 1 and 5. The system successfully learns to transform the trees in an average of 8 steps. Reward ={ Goal=1000, edit and move=$-1, 0$} .

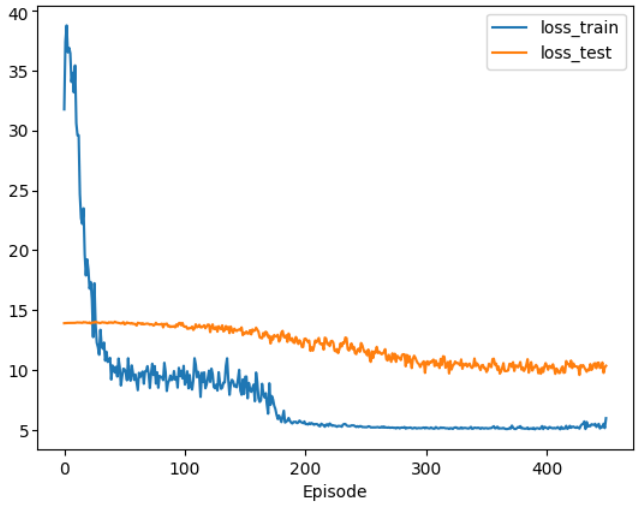


Figure 6.3: Performance of rollout during training on variable sized tree pairs with NNI distances between 1 and 5 with variable size tree without normalization. Average loss in nni distance across 10 runs for variable size tree without normalization

6.10 Conclusion

We present a generic framework for learning representation and behavior on tree datasets in hyperbolic space on a finite action space, integrating policy learning using Markov decision processes with automated representation exploration. In this work, we study value function approximation in reinforcement learning problems with high dimensional state and a finite action space based on a generalized representation policy iteration. We consider the limitations at accurately approximating the value function in low dimensions and we highlight the importance of features learning for an improved low-dimensional value function approximation.

CHAPTER 7

Conclusions and Future Work

7.1 Conclusions

This dissertation introduced a representation and reinforcement learning approach to solve tree transformation problems without the need to impose limitations on tree size. We showed potential applications of these tree transformation tools and learning-based sampling strategies to a common problem related to tree distance and tree manipulation with applications to problems in genomics, planning and control.

The first part of the dissertation presented the use of reinforcement learning for relaxed, deterministic coordination and control of an agent to learn a tree edit distance task. We reinterpreted this classical method task for unsupervised learning as an abstract formalism for identifying and representing tree transformations by relating the continuous space of configurations to the combinatorial space of trees.

The second part of the dissertation introduced a generalized approach with automated representation exploration in an edit neighborhood representation, learning to identify a neighborhood of a tree that captures the local geometric structure of a configuration space around the tree's instantaneous configuration. Based on this edit neighborhood representation, we used reinforcement learning to learn a NNI distance strategy to find the minimum-cost sequence of operations that transforms one tree into another.

The third part of the dissertation presents a generic framework for learning representation and behavior on a tree dataset in hyperbolic space with a finite action space, integrating novel hyperbolic embeddings of tree pairs with recurrent networks

to obtain effective representations of tree distances in arbitrary sized trees, and policy learning using reinforcement learning. In this work, we study the use of value function approximation in hyperbolic space and reinforcement learning problems with high dimensional state and a finite action space based on a generalized representation and policy iteration.

The obtained results strongly suggest that reinforcement learning is, indeed, an effective approach for automatically extracting inherent structures in configuration spaces relevant to the solution of tree edit distance, and that it might play a key role in the design of computationally efficient planners in complex, high-dimensional configuration spaces in different application domains.

7.2 Future Work

Applying a recurrent architecture for the encoder networks might allow discovering such structures in uncertain environments such as Partially Observable Markov Decision Problems. Additionally, the equivalence properties of predicted value function bounds in the proposed approach needs to be verified. Furthermore, it would be interesting to map states and action spaces of multiple environments to the same latent space. Given such a representation, any policy learned for the latent transition model, can be projected back to the real world environments.

REFERENCES

[1] L. G. Shapiro and R. M. Haralick, "Structural descriptions and inexact matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-3, no. 5, pp. 504–519, 1981.

[2] A. Wong, M. You, and S. Chan, "Algorithm for graph optimal monomorphism," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 20, pp. 628 – 638, 06 1990.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," *CoRR*, vol. abs/1403.6652, 2014. [Online]. Available: http://arxiv.org/abs/1403.6652

[4] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[5] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *CoRR*, vol. abs/1607.00653, 2016. [Online]. Available: http://arxiv.org/abs/1607.00653

[6] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[7] M. Li, J. Tromp, and L. Zhang, "On the nearest neighbor interchange distance between evolutionary trees," *Journal of Theoretical Biology*, vol. 182, no. 4, pp. 463–467, 1996.

[8] D. F. Robinson, "Comparison of labeled trees with valency three," *Journal of Combinatorial Theory*, vol. 11, no. 2, pp. 105–119, 1971.

[9] M. Krivanek, "Computing the nearest neighbor interchange metric for unlabeled binary trees is np-complete," *Journal of Classification*, vol. 3, no. 1, pp. 55–60, 1986.

[10] K. Takahashi and M. Nei, "Efficiencies of fast algorithms of phylogenetic inference under the criteria of maximum parsimony, minimum evolution, and maximum likelihood when a large number of sequences are used." *Molecular biology and evolution*, vol. 17 8, pp. 1251–8, 2000.

[11] N. K. Ahmed, J. Neville, and R. Kompella, "Network sampling: From static to streaming graphs," *ACM Transactions on Knowledge DIscovery from Data*, vol. 8, no. 2, p. 7, 2014. [Online]. Available: http://arxiv.org/abs/1211.3412%5Cnhttp://www.arxiv.org/pdf/1211.3412.pdf

[12] N. Misra, G. Blelloch, R. Ravi, and R. Schwartz, "An optimization-based sampling scheme for phylogenetic tree," *Journal of computational biology : a journal of computational molecular cell biology*, vol. 18, no. 11, pp. 1599–1609, 2011.

[13] J. P. Huelsenbeck and F. Ronquist, "Mrbayes: Bayesian inference of phylogeny," *Bioinformatics*, vol. 17, no. 8, p. 754755, 2001.

[14] P. Diaconis and S. Holmes, "Random walks on trees and matchings," *Electronic Journal of Probability*, vol. 7, no. 6, pp. 17–28, 2002.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215

[16] D. Janz, J. van der Westhuizen, and J. M. Hernndez-Lobato, "Actively learning what makes a discrete sequence valid," 2017.

[17] E. J. Bjerrum, "Smiles enumeration as data augmentation for neural network modeling of molecules," 2017.

[18] D. Janz, J. van der Westhuizen, B. Paige, M. J. Kusner, and J. M. Hernndez-Lobato, "Learning a generative model for validity in complex discrete structures," 2018.

[19] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik, "Objective-reinforced generative adversarial networks (organ) for sequence generation models," 2018.

[20] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," 2017.

[21] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," 2017.

[22] C. Gulcehre, M. Denil, M. Malinowski, A. Razavi, R. Pascanu, K. M. Hermann, P. Battaglia, V. Bapst, D. Raposo, A. Santoro, and N. de Freitas, "Hyperbolic attention networks," 2018.

[23] M. Li, J. Tromp, and L. Zhang, "On the nearest neighbor interchange distance between evolutionary trees," *Journal of Theoretical Biology*, vol. 182, no. 4, pp. 463–467, 1996.

[24] W. Hon and T. W. Lam, "Approximating the nearest neighbor interchange distance for evolutionary trees with non-uniform degrees," *Computing and Combinatorics*, vol. 1627, no. 1999, pp. 61–70, 1999.

[25] W. Hon, M. Kao, T. Lam, W. K. Sung, and S. Yiu, "Non-shared edges and nearest neighbor interchanges revisited," *Information Processing Letters*, vol. 91, no. 3, pp. 29–134, 2004.

[26] S. Whelan, "New approaches to phylogenetic tree search and their application to large numbers of protein alignments," *Systems Biology*, vol. 56, no. 5, pp. 727–740, 2007.

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

[28] M. S. Waterman and T. F. Smith, "On the similarity of dendrograms," *Journal of Theoretical Biology*, vol. 73, no. 4, pp. 783–800, 1978.

[29] J. P. Jarvis, J. K. Luedeman, and D. R. Shier, "Counterexamples in measuring the distance between binary trees," *Mathematical Social Sciences*, vol. 4, no. 3, pp. 271–274, 1983.

[30] W. Hon, M. Kao, T. Lam, W. K. Sung, and S. Yiu, "Non-shared edges and nearest neighbor interchanges revisited," *Information Processing Letters*, vol. 91, no. 3, pp. 29–134, 2004.

[31] W. Hon and T. W. Lam, "Approximating the nearest neighbor interchange distance for evolutionary trees with non-uniform degrees," *Computing and Combinatorics*, vol. 1627, no. 1999, pp. 61–70, 1999.

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

[33] D. B. Arnold and M. R. Sleep, "Uniform random generation of balanced parenthesis strings," *ACM Trans. Program. Lang. Syst.*, vol. 2, no. 1, pp. 122–128, 1980.

[34] B. Allen and M. Steel, "Subtree transfer operations and their induced metrics on evolutionary trees," *Annals of Combinatorics*, vol. 5, no. 1, pp. 1–15, 2001.

[35] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *CoRR*, vol. abs/1709.05584, 2017. [Online]. Available: http://arxiv.org/abs/1709.05584

[36] M. Krivanek, "Computing the nearest neighbor interchange metric for unlabeled binary trees is np-complete," *Journal of Classification*, vol. 3, no. 1, pp. 55–60, 1986.

[37] D. F. Robinson, "Comparison of labeled trees with valency three," *Journal of Combinatorial Theory*, vol. 11, no. 2, pp. 105–119, 1971.

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

[39] B. Perozzi, V. Kulkarni, and S. Skiena, "Walklets: Multiscale graph embeddings for interpretable network classification," *CoRR*, vol. abs/1605.02115, 2016. [Online]. Available: http://arxiv.org/abs/1605.02115

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] R. S. Sutton and A. G. Barto, "Reinforcement learning," *Learning*, vol. 3, no. 9, p. 322, 2012.

[42] M. Gast and M. Hauptmann, "Efficient Parallel Computation of Nearest Neighbor Interchange Distances," *Computing Research Repository (CoRR) preprint arXiv:1205.3402 [cs.DS]*, pp. 1–17, 2012. [Online]. Available: http://arxiv.org/abs/1205.3402

[43] M. Nickel and D. Kiela, "Poincaré embeddings for learning hierarchical representations," *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 6339–6348, 2017.

[44] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, p. 7894, Jul 2018. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2018.03.022

[45] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[46] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://aclanthology.org/D14-1162

[47] E. Ravasz and A. L. Barabási, "Hierarchical organization in complex networks," *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, vol. 67, no. 2, p. 7, 2003.

[48] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[49] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1024–1034. [Online]. Available: http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

[50] S. Mahadevan, "Representation policy iteration," *CoRR*, vol. abs/1207.1408, 2012. [Online]. Available: http://arxiv.org/abs/1207.1408

[51] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Bogu, "Hyperbolic Geometry of Complex Networks," *Physical Review E*, vol. 82, no. 036106, Oct 2010.

[52] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1024–1034. [Online]. Available: http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

[53] R. Sarkar, "Low distortion Delaunay embedding of trees in hyperbolic plane," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7034 LNCS, pp. 355–366, 2012.

[54] F. Sala, C. De Sa, A. Gu, and C. Ré, "Representation tradeoffs for hyperbolic embeddings," *35th International Conference on Machine Learning, ICML 2018*, vol. 10, no. 1, pp. 7071–7096, 2018.

[55] I. Chami, Z. Ying, C. Ré, and J. Leskovec, "Hyperbolic graph convolutional neural networks," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 4868–4879. [Online]. Available: http://papers.nips.cc/paper/8733-hyperbolic-graph-convolutional-neural-networks.pdf

[56] C. D. Sa, A. Gu, C. Ré, and F. Sala, "Representation tradeoffs for hyperbolic embeddings," *CoRR*, vol. abs/1804.03329, 2018. [Online]. Available: http://arxiv.org/abs/1804.03329

[57] R. Sarkar, "Low distortion delaunay embedding of trees in hyperbolic plane," in *Graph Drawing*, M. van Kreveld and B. Speckmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 355–366.

[58] A. F. Beardon and D. Minda, "The hyperbolic metric and geometric function theory," *Quasiconformal mappings and their applications*, no. November 2004, pp. 9–56, 2007.

[59] D. Silver, "Lecture 4 : Model-Free Prediction," *UCL,Computer Science Department, Reinforcement Learning Lectures*, 2015.

[60] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.