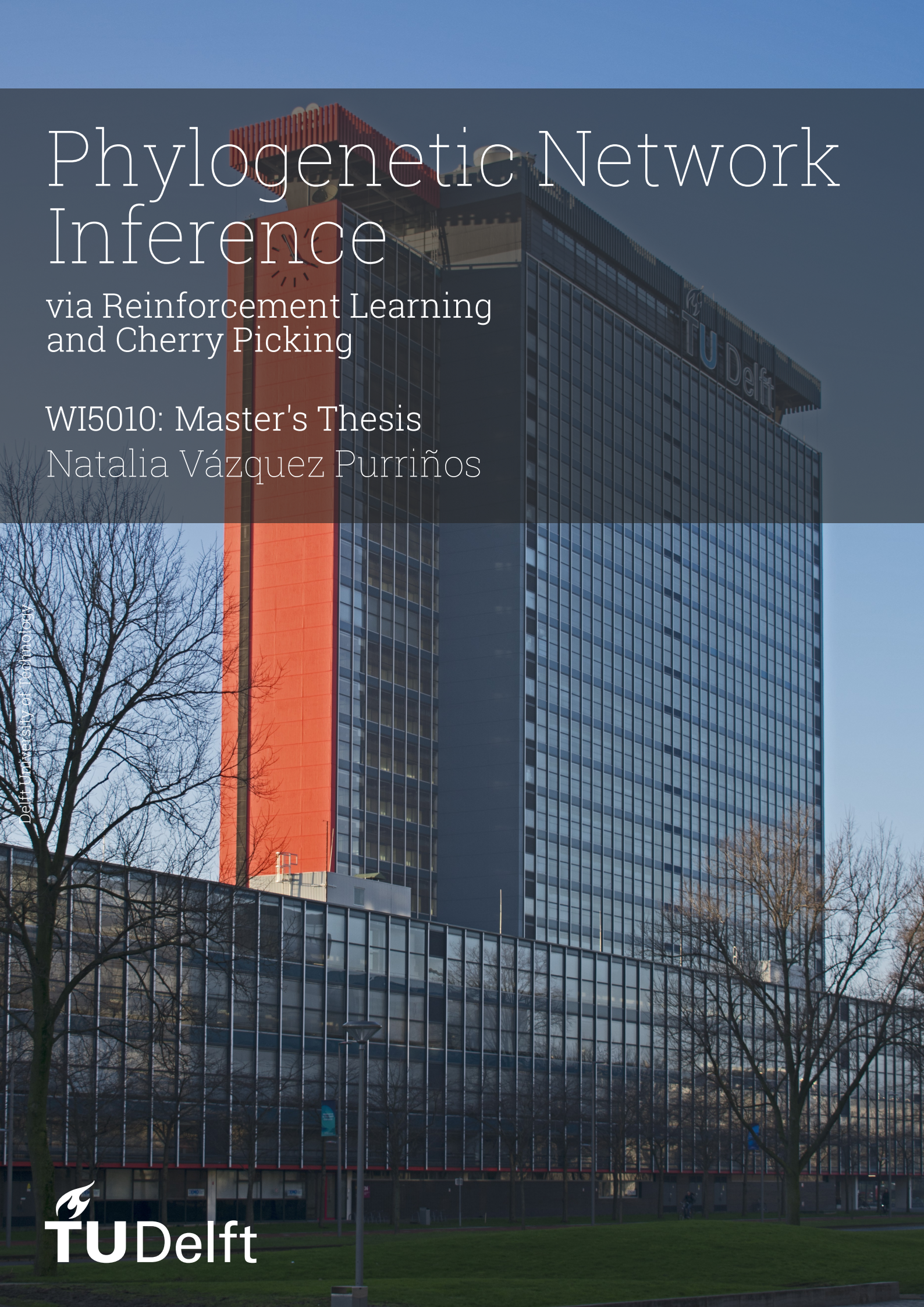
Phylogenetic Network Inference

via Reinforcement Learning and Cherry Picking

WI5010: Master's Thesis

Natalia Vázquez Purriños



Delft University of Technology

TU Delft

Phylogenetic Network Inference

via Reinforcement Learning and Cherry Picking

by

Natalia Vázquez Purriños

to obtain the degree of Master of Science in Applied Mathematics
at Delft University of Technology,
to be defended publicly on Tuesday May 26, 2026 at 13:00 PM.

Student number:	5917778	
Project duration:	September 15, 2025 – May 26, 2026	
Thesis committee:	Dr. ir. L.J.J. van Iersel,	TU Delft, supervisor
	Prof. Dr. ir. G. Jongbloed,	TU Delft

Cover:	By Ymblanter - Own work, CC BY-SA 4.0, Wikipedia
Style:	TU Delft Report Style, with modifications by Natalia Vázquez Purriños

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements

This thesis was developed within the Discrete Mathematics and Optimization section at DIAM, at TU Delft. Between September, 2025 and May 2026, I conducted this research focused on the optimization of phylogenetic networks through computational heuristics and reinforcement learning.

I was initially drawn to this research as it represents the convergence of my three core passions: mathematics, genetics, and artificial intelligence. The opportunity to bridge these disciplines, applying neural-guided heuristics to solve complex problems in evolutionary genetics through a rigorous mathematical lens, has been an immensely rewarding journey. This project has not only allowed me to explore the interplay between these fields but has also provided a profound learning experience for which I am truly grateful.

I would like to acknowledge the use of Google's Gemini for assistance in improving the clarity of the English prose.

I express my deepest gratitude to my supervisor, Prof. Leo van Iersel, for his invaluable guidance throughout this project. His consistent feedback and insights were essential during the process of writing this thesis. Finally, I would like to thank Prof. Geurt Jongbloed, as a member of my graduation committee, for his time and evaluation.

As this journey as a student at TU Delft comes to an end, I look back with immense gratitude on the learning and experiences that defined this period. I am grateful to my friends for providing balance and support during the most demanding stages of this research, and to my family for their unconditional support.

Natalia Vázquez Purriños
Delft, May 2026

Summary

Understanding evolutionary history is a cornerstone of biological research. While phylogenetic trees represent vertical descent, complex phenomena such as hybridization and horizontal gene transfer require more versatile structures known as phylogenetic networks. Reconstructing these networks under the principle of maximum parsimony leads to the Hybridization Problem, a combinatorial optimization task that is NP-hard and poses significant scalability challenges for existing exact and heuristic methods.

This thesis investigates whether learning-guided search can provide a scalable approximation framework for the Hybridization Problem for temporal phylogenetic networks. We reformulate the problem as a Markov Decision Process (MDP) based on the theory of cherry-picking sequences, which establishes a fundamental equivalence between sequence weight and the network's reticulation number.

Inspired by the AlphaZero paradigm, we propose a framework that integrates Monte Carlo Tree Search (MCTS) with deep learning. Our approach employs a modular architecture featuring a Value Network designed to provide direct state evaluations, replacing computationally expensive heuristic rollouts. To handle the graph-theoretical nature of the problem, we implement a hierarchical feature engineering pipeline that summarizes local leaf-level and global structural properties into a compact representation for neural guidance.

We evaluate our framework by comparing our method against a randomized heuristic and exact solvers. This work contributes a novel, learning-enhanced approach to phylogenetic reconstruction, offering a promising path toward handling larger and more complex evolutionary datasets.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
CPS	Cherry Picking Sequence
FPT	Fixed-Parameter Tractable / Tractability
GNN	Graph Neural Network
IDE	Integrated Development Environment
ILS	Incomplete Lineage Sorting
LCA	Lowest Common Ancestor
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
PUCT	Predictor + Upper Confidence Bound applied to Trees
RF	Robinson-Foulds (metric)
RL	Reinforcement Learning
TFRH	Trivial-First Randomized Heuristic
UCB	Upper Confidence Bound
UCT	Upper Confidence Bound applied to Trees
URS	Unconstrained Random Search
V-MCTS	Value-guided Monte Carlo Tree Search

Symbols

Symbol	Definition
Phylogenetic Structures	
$\mathcal{T}$	A set (forest) of phylogenetic trees
X	Finite set of taxa (leaf labels)
n	Number of leaves (taxa) in an instance
m	Number of trees in a phylogenetic forest
k	Best-known minimum reticulation number
Markov Decision Processes	
S	Finite set of states in an MDP
A	Finite set of actions available in an MDP
R	Reward signal or function in an MDP
γ	Discount factor for future rewards
σ	A partial or complete cherry-picking sequence
h	Cumulative hybridization cost
Search and Statistics	
c_{puct}	Exploration constant in the PUCT algorithm
ρ	Spearman rank correlation coefficient
R^2	Coefficient of determination
β	Estimated coefficient in logistic regression

1

Introduction

Ever since the publication of Charles Darwin's pioneering studies, understanding evolution and the origin of species has remained a central objective of biological research. In 1837, Darwin sketched what is now known as his first evolutionary tree (see Figure 1.1). Even if this concept existed long before Darwin, with the first branching trees dating back to 1766 in the work of Peter Simon Pallas, Darwin's contribution further popularized this already existing metaphor that continues to shape evolutionary biology today [1, 2]. These structures, known as phylogenetic trees [3], represent ancestral relationships among present-day species [4] and are equally fundamental in modeling gene and molecular sequence evolution [5]. According to [3], a phylogenetic tree is a diagram that illustrates the evolutionary relationships among a group of species based on shared ancestry, where species are represented at the ends of branches and the distance between groups reflects their degree of relatedness. A formal mathematical definition will be provided later.

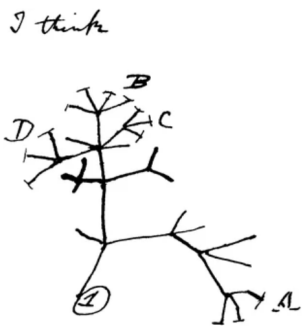


Figure 1.1: An evolutionary tree designed by Darwin, found in one of his notebooks [1].

The tree model, however, assumes that evolutionary history follows a purely branching, acyclic pattern of vertical descent. Biological reality is often more complex. Processes such as horizontal gene transfer and hybridization generate reticulate evolutionary patterns that cannot be captured within a tree structure [6]. In hybridization events, for instance, distinct lineages merge to form a new species, producing a reticulation in the evolutionary history [7]. In parallel, horizontal gene transfer involves the non-vertical exchange of genetic material between distinct lineages, resulting in reticulate patterns of inheritance that bypass direct ancestral descent.

To account for such phenomena, phylogenetic trees are generalized to *phylogenetic networks*, which allow for reticulate nodes and provide a more expressive representation of evolutionary relationships (see Figure 1.2 for an example). From a mathematical perspective, a phylogenetic network is a directed acyclic graph containing nodes with indegree-2 or higher that represent the previously mentioned reticulation events.

Phylogenetic networks also serve to represent multiple phylogenetic trees within a single unified struc-

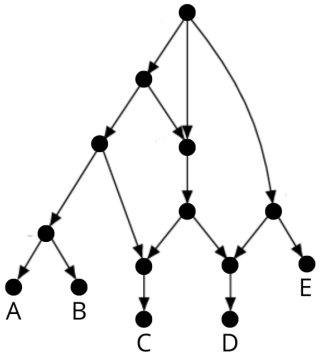


Figure 1.2: A rooted directed phylogenetic network on $\mathcal{X} = \{A, B, C, D, E\}$. Image adapted from [8].

ture, a problem commonly referred to as network reconstruction. A key practical application is the reconciliation of gene trees, each representing the evolutionary history of an individual gene, into a comprehensive species network. Since network reconstruction is generally non-unique, selecting a biologically plausible model requires the application of additional criteria. Among these, the parsimony principle is particularly prominent.

The parsimony principle favors the simplest explanation consistent with the observed data [6]. In the present context, this translates into minimizing the number of reticulation events while ensuring that all input trees are displayed by the network. Accordingly, throughout this thesis, we adopt the *maximum parsimony* criterion and regard a network as optimal if it contains the smallest number of reticulate events necessary to represent the entire collection of trees.

1.1. Problem Context and Motivation

Network reconstruction is not solely a biological question—it is equally a mathematical and computational challenge. Although motivated by evolutionary phenomena, the task of minimizing the number of reticulation events required to represent a given collection of trees gives rise to a highly nontrivial combinatorial optimization problem, commonly known as the *Hybridization Problem* [9].

From a complexity-theoretic perspective, this problem is NP-hard even in highly restricted settings, such as the case of two binary input trees [10]. This hardness result highlights the intrinsic combinatorial nature of the reconstruction task and explains why exact computation quickly becomes intractable as instance size increases.

In practice, existing exact and heuristic methods for estimating phylogenetic networks often exhibit poor scalability. Their computational cost grows rapidly with the number of taxa and with the topological complexity of the output network, making large datasets particularly challenging [7]. Consequently, developing algorithmic approaches that combine theoretical soundness with practical efficiency remains a central objective in this field.

In parallel to these computational challenges, recent years have witnessed an unprecedented expansion in Artificial Intelligence (AI), and in particular in Machine Learning (ML), both in theoretical development and practical impact across scientific domains [11, 12]. Learning-based methods have demonstrated remarkable capacity to approximate solutions to highly complex combinatorial and decision-making problems, often outperforming handcrafted heuristics.

Phylogenetics has not remained untouched by this development. In particular, machine learning techniques have recently been explored for the reconstruction of phylogenetic networks. A notable example is the work of Bernardini et al. [10], where supervised learning methods are combined with cherry-picking strategies to guide the construction of phylogenetic networks. Their approach illustrates how learned models can assist in navigating the combinatorial search space underlying the Hybridization Problem.

However, supervised learning approaches rely heavily on the availability of large, high-quality labeled

datasets [13]. In the context of phylogenetic network reconstruction, generating optimal or near-optimal solutions at scale is itself computationally demanding due to the NP-hardness of the underlying problem [10]. Consequently, obtaining sufficiently rich training data becomes a significant bottleneck.

This limitation motivates the exploration of alternative learning paradigms. In particular, Reinforcement Learning (RL) offers a framework in which an agent learns to make sequential decisions through interaction with the environment, without requiring explicit supervision in the form of optimal solutions [14]. Inspired by the remarkable success of the integration of RL and Monte Carlo Tree Search (MCTS)[1] in combinatorial domains such as board games [12], we investigate whether similar ideas can be adapted to the reconstruction of temporal phylogenetic networks. Our focus on this specific class of networks is grounded in the requirement of chronological plausibility. A temporal (or time-consistent) network models evolutionary histories in which reticulation events occur only between contemporaneous lineages—meaning that hybridization is restricted to species that coexist. A formal mathematical definition will be provided later in Section 2.1.1.

Within this temporal framework, we explore whether RL can be integrated with MCTS to navigate the space of Cherry-Picking Sequences (CPS) more effectively. This choice is motivated by the fact that CPSs provide a constructive pathway for network assembly, transforming the reconstruction problem into a sequence of discrete decisions. In our proposed framework, a learned value function is employed as a guiding component within the search procedure. Rather than relying on precomputed optimal solutions, the learning process would be driven by simulated rollouts and self-play, in the spirit of the AlphaZero paradigm [12]. AlphaZero is a RL algorithm presented by Silver et al. in 2018 that combines deep neural networks with MCTS to achieve superhuman performance in games like Go, Chess, and Shogi. The key strength of AlphaZero lies in its ability to learn from scratch, using self-play to improve its policy over time without requiring prior human knowledge or labeled datasets [12]. This makes it a particularly powerful candidate for adaptation to phylogenetic reconstruction, where the search space is vast and optimal solutions are often unknown.

1.2. Research Question

This thesis investigates whether learning-guided search can provide a scalable approximation framework for the Hybridization Problem in temporal phylogenetic networks.

Formally, we address the following research question:

> Can a RL framework, inspired by the AlphaZero paradigm, learn a value function that effectively guides MCTS in constructing phylogenetic networks with close to optimal number of reticulations?

This central question gives rise to three complementary investigations:

- To what extent does MCTS outperform uninformed sampling in minimizing the number of reticulate events, both in terms of solution quality and computational efficiency?
- Can a learned value function effectively reduce the expected remaining reticulate events and improve search efficiency?
- Can the combination of RL and tree search offer a scalable approximation framework for phylogenetic reconstruction, compared to classical exact and heuristic methods?

Ultimately, this work seeks to evaluate whether learning-guided search can improve the practical tractability of the Hybridization Problem, enabling better approximations within realistic computational budgets.

1.3. Contributions

The primary contribution of this thesis is the design, implementation, and evaluation of a novel algorithmic framework that treats phylogenetic network reconstruction as a sequential decision-making problem. Specifically, this work provides the following contributions to the field of computational phylogenetics:

[1] Technically, MCTS is a sampling-based planning algorithm. However, when applied to an MDP to iteratively improve a policy through simulated experience—as we do in this framework—it functions as a core component of the Reinforcement Learning architecture.

- **Formalization of CPS as an MDP:** We provide, to the best of our knowledge, the first formalization of the CPS reduction problem as a Markov Decision Process (MDP). This includes the definition of a discrete state-action space where tree reductions are modeled as transitions, enabling the application of modern RL paradigms.
- **The CPS-MCTS-NN Framework:** We developed a complete hybrid architecture inspired by the AlphaZero paradigm. This framework integrates MCTS with a deep neural network designed to predict the remaining hybridization weight of a forest state, effectively replacing blind rollouts with a predictive value function.
- **Phylogenetic Feature Engineering:** We designed a specialized feature extraction pipeline capable of translating complex forest topologies into numerical tensors suitable for neural network inference. This includes the encoding of leaf-neighbor relationships and forest-wide consistency metrics, which were based on the framework of Bernardini et al [10].
- **Systematic Empirical Evaluation:** We conducted a large-scale benchmark involving 460 independent simulations across a heterogeneous dataset of binary trees. This study provides a rigorous comparison between guided search (V-MCTS), standard MCTS, and stochastic baselines (URS and TFRH), establishing a success rate of 91.74% for the MDP-based approach.
- **Open Research Baseline:** The implementation, developed entirely in Python, serves as an extensible, well documented open-source baseline for future researchers. It demonstrates that RL-based heuristics can achieve high-precision results within realistic computational budgets (ranging from milliseconds to minutes), even before low-level language optimization.

2

Literature Review and Theoretical Framework

2.1. Theoretical Framework

We now formalize the combinatorial objects underlying our approach. The terminology and basic definitions follow standard conventions in phylogenetic network theory, in particular the framework presented in [15, 16].

2.1.1. Basic Structures.

Throughout the following sections, let $\mathcal{X}$ be a finite set of taxa. While these represent biological entities, they are mathematically represented as the *leaves* of the network—specifically, nodes with in-degree 1 and out-degree 0. A *tree vertex* is defined as a node with in-degree 1 and out-degree at least 2. Together with *hybridization vertices* (in-degree ≥ 2, out-degree 1) and the *root* (in-degree 0), these categories constitute the exhaustive set of node types in a phylogenetic network (see Definition 2).

Definition 1 *A **(rooted) phylogenetic tree** on $\mathcal{X}$ (see Figure 2.1) is a rooted tree with no vertices of indegree 1 and outdegree 1, whose leaves are bijectively labelled by the elements of $\mathcal{X}$. A leaf is a node with in-degree 1 and out-degree 0.*

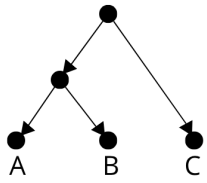


Figure 2.1: Example of a (rooted) phylogenetic tree with three leaves.

Definition 2 *A **(rooted) phylogenetic network** on $\mathcal{X}$ (see Figure 2.2) is a rooted directed acyclic graph with no vertices of indegree 1 and outdegree 1, whose leaves are bijectively labelled by the elements of $\mathcal{X}$. A network is **tree-child** if every tree vertex has at least one outgoing tree arc.*

A relationship between trees and networks is captured by the following notion.

Definition 3 *A phylogenetic network $\mathcal{N}$ **displays** a phylogenetic tree $\mathcal{T}$ (see Figure 2.3) if $\mathcal{N}$ contains a subdivision of $\mathcal{T}$ as a subgraph.*

In a biological context, this definition is crucial for interpreting gene evolution. If the evolutionary history of a particular gene is strictly tree-like and is not confounded by stochastic events such as *Incomplete Lineage Sorting (ILS)*, then the resulting gene tree T is necessarily a *tree displayed in* the species

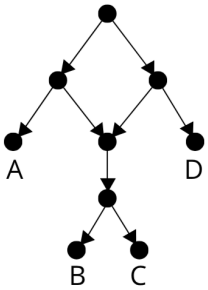


Figure 2.2: Example of a (rooted) phylogenetic network with four leaves.

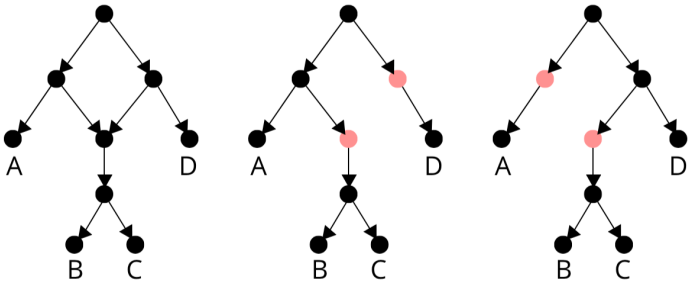


Figure 2.3: Example of a network (on the left) displaying two trees (on the right). The pink nodes correspond to vertices with in-degree 1 and out-degree 1, which will be suppressed.

network N. ILS is a phenomenon in which ancestral polymorphisms are randomly partitioned into diverging lineages, potentially resulting in gene trees that conflict with the species history.

Under this structural representation, the network acts as a container for all possible tree-like histories. When ILS is absent, any discordance between different gene trees is attributed solely to reticulation events (such as hybridization or lateral gene transfer) rather than population-level variance. Each displayed tree, therefore, represents one possible resolution of the reticulate nodes, effectively 'picking' a single ancestral path at each point of the genetic exchange.

As the number of independent reticulate events increases, so does the discrepancy between the simple tree-like paths and the overall network topology. To measure the structural complexity of a network, we use the reticulation number.

Definition 4 *A **reticulation** (see Figure 2.4) is a vertex with indegree at least* 2*. The **reticulation number** of a phylogenetic network is defined as*

$$|E| - |V| + 1, \tag{2.1}$$

that is, the number of edges that need to be deleted to transform the network into a tree.

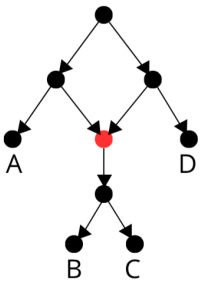


Figure 2.4: Example of a reticulated network, showing the reticulation node (in red) that represents complex evolutionary processes.

While phylogenetic networks allow for general reticulate structures, not all such networks are biologically feasible. In this thesis, we focus on temporal (or time-consistent) networks, which impose a constraint on the timing of evolutionary events to ensure they are historically plausible. Before providing a formal definition, it is necessary to distinguish between tree arcs and reticulation arcs, as these form the basis of the temporal consistency condition.

Definition 5 *Let $\mathcal{N} = (V, E)$ be a rooted binary phylogenetic network. According to the degrees of the vertices, we distinguish two types of edges in E [17]:*

- ***Reticulation arcs:*** *All arcs directed into a reticulation vertex (see Figure 2.5).*
- ***Tree arcs:*** *All other arcs in the network, typically representing vertical inheritance.*

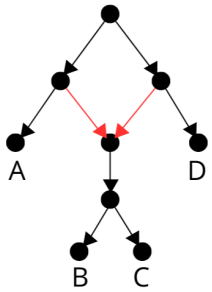


Figure 2.5: Example of a reticulated network, showing reticulation arcs in red and tree arcs in black.

Definition 6 *A tree-child phylogenetic network $\mathcal{N} = (V, E)$ is called **temporal** [17] (or **time-consistent**) if there exists a temporal labeling function $t : V \to \mathbb{N}_{\geq 0}$ such that, for each arc $(u, v) \in E$, the following properties hold:*

- $t(u) = t(v)$ *if (u, v) is a reticulation arc;*
- $t(u) < t(v)$ *if (u, v) is a tree arc.*

2.1.2. Cherry-Picking Sequences.

We now introduce the combinatorial machinery that allows us to characterize the reticulation number through sequential leaf deletions. This concept, originally introduced by Humphries et al. [18], will play a central role throughout this work.

Definition 7 *A pair of leaves forms a **cherry** if they have a common parent (see Figure 2.6).*

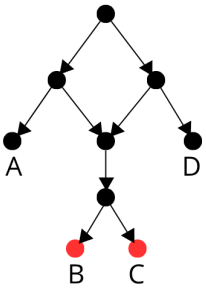


Figure 2.6: Example of a cherry (in red) on a phylogenetic network.

While a cherry is defined within a single tree, when working with a set of trees it is crucial to distinguish between cherries that are consistently present across the set (trivial) and those that arise only in some trees (non-trivial).

Definition 8 (Trivial and Non-Trivial Cherries) *Let $\mathcal{T}$ be a set of phylogenetic trees on leaf set X. An unordered pair of leaves $\{x, y\} \subseteq X$ is called a cherry of $\mathcal{T}$ if it forms a cherry in at least one tree in $\mathcal{T}$.*

*Moreover, $\{x, y\}$ is a **trivial** cherry of $\mathcal{T}$ if, for every tree $T \in \mathcal{T}$ that contains both x and y, the pair $\{x, y\}$ is a cherry in T (see the red leaf and its pair in Figure 2.7). Otherwise, $\{x, y\}$ is called a **non-trivial** cherry of $\mathcal{T}$ (see the blue leaf and its pair in Figure 2.7).*

These distinctions are crucial when constructing a CPS.

Definition 9 *A **cherry-picking sequence** of a set of trees $\mathcal{T}$ (see Figure 2.7) is a permutation $(s_1, \ldots, s_n)$ of the leaves such that each s_i is contained in a cherry in every tree after deleting $s_1, \ldots, s_{i-1}$, for all $i < n$ and suppressing any resulting in-degree 1 out-degree 1 vertices.*

In a CPS, each leaf is selected based on its presence as a cherry in all current trees, but the nature of the cherry (whether trivial or non-trivial) affects the complexity of the sequence. A trivial cherry leads to a straightforward reduction that is valid in all trees, whereas a non-trivial cherry may involve more complex decisions depending on how the trees differ in their structure. In fact, Bernardini et al. [10] demonstrate, in a slightly different context, that prioritizing trivial cherries serves as an effective heuristic for minimizing the total reticulation number of the reconstructed network.

To quantify the contribution of each leaf to the overall complexity, we define the following weight function.

Definition 10 *The **weight** $w(x)$ of a leaf x is the number of distinct cherries it belongs to minus one. The **weight** $w(s)$ of a CPS s is the sum of the weights of the leaves at the moment they are picked (see Figure 2.7).*

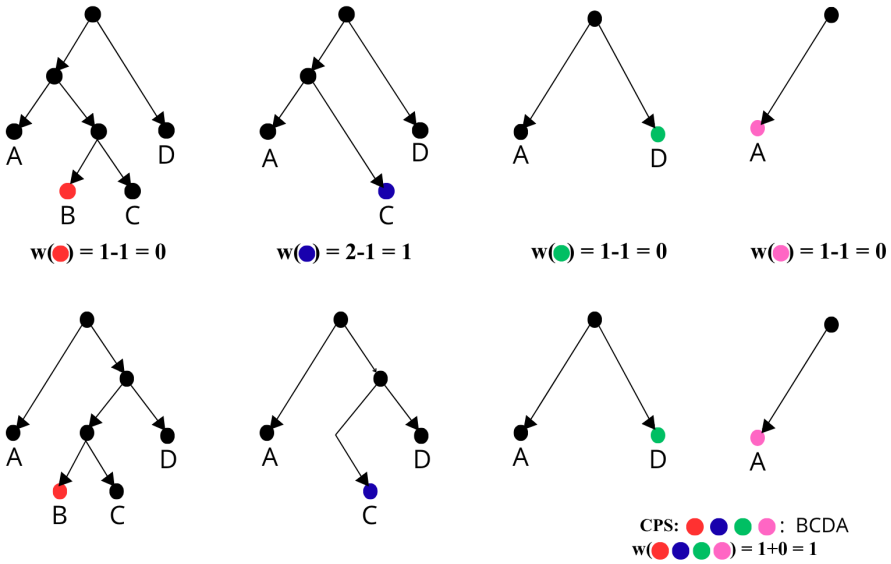


Figure 2.7: Illustration of a CPS. The image demonstrates the process of sequentially selecting leaves and recording the weight of each picked leaf. At the end, the total weight of the CPS is shown, reflecting the cumulative weight of the selected leaves.

The following theorem establishes the fundamental equivalence between temporal networks and CPSs.

Theorem 1 (Humphries, Linz and Semple [18]) *There exists a temporal phylogenetic network displaying $\mathcal{T}$ with reticulation number k if and only if there exists a CPS of $\mathcal{T}$ with weight k.*

This equivalence forms the theoretical foundation of our methodology. Rather than constructing temporal networks directly, we determine the reticulation number by searching for minimum-weight CPSs, whose weight can be computed incrementally from the local contributions of each leaf at the moment it is selected. By seeking to minimize this reticulation number, we are essentially addressing the previously mentioned Hybridization Problem. While the problem is known to be NP-hard even for two binary input trees, its formal definition provides the objective for our search process [10]:

Definition 11 (Hybridization Problem) *The Hybridization Problem is defined as follows [10]:*

- *Input: A set of phylogenetic trees $\mathcal{T}$ on a set of taxa X.*

- *Output: A phylogenetic network displaying $\mathcal{T}$ with the minimum possible reticulation number.*

While the definition above covers the general hybridization task, this study narrows its scope to temporal networks. Consequently, we specifically address the constrained version of this task: the Temporal Hybridization Problem. In this variant, the resulting network must not only display the input trees $\mathcal{T}$ with a minimal reticulation number but also satisfy a strict temporal labeling where every hybridization event occurs between coexisting lineages.

2.2. Search in Combinatorial Optimization

The minimization of a CPS's weight can be naturally formulated as a combinatorial search problem [19]. Each partial sequence defines a state in a search tree whose branches correspond to feasible leaf selections. The size of this search space can increase factorially with the number of taxa, making exhaustive enumeration infeasible except for very small instances.

Classical approaches to combinatorial search include exhaustive search, branch-and-bound methods, greedy heuristics, local search, parametrized algorithms and randomized sampling [20]. Exhaustive methods guarantee optimality but suffer from exponential time complexity. Branch-and-bound techniques can prune portions of the search space but rely heavily on effective bounding strategies, which are difficult to design for highly structured problems such as CPSs, as the dependencies between selected leaves and the complexity of reticulation events make it challenging to define tight, easily computable bounds. Greedy strategies, while computationally efficient, often suffer from myopic decision-making and may quickly converge to suboptimal solutions. Local search methods facilitate iterative refinement by exploring neighboring solutions, although they are highly susceptible to becoming trapped in local optima and face substantial challenges in defining valid neighborhood moves that preserve the temporal feasibility of a sequence. In parallel, parametrized algorithms attempt to achieve optimality by focusing on the exponential complexity on a specific parameter, such as the hybridization number k—however, their running time scales poorly as this parameter increases, eventually becoming computationally prohibitive for dense phylogenetic networks. Randomized sampling provides a simple baseline by exploring the search space without structural bias. However, uninformed sampling becomes increasingly inefficient as instance size grows, since the probability of encountering high-quality solutions tends to decrease rapidly [20].

These limitations motivate the exploration of guided search strategies that balance structured exploration with adaptive decision-making. In this context, MCTS offers a well-founded framework for navigating large combinatorial search spaces while dynamically allocating computational effort toward promising regions, as it combines the precision of tree search with the generality of random sampling [21]. Building on this foundation, learning-guided search, inspired by the principles of AlphaZero, would enhance the search process by using neural networks to guide decision-making and focus on a more effective exploration.

2.3. Learning-Guided Search

In combinatorial optimization, guiding the search process with learned models has proven to be an effective strategy, specially in NP-hard problems where traditional heuristics may struggle [22]. The AlphaZero algorithm, originally designed for game playing, uses a deep neural network to evaluate and guide the search during MCTS. Specifically, AlphaZero employs a neural network with two outputs: a policy head that provides a probability distribution over actions, and a value head that estimates the long-term reward of a state. This approach allows the search to focus on promising regions of the search space while avoiding random exploration [12].

2.3.1. The Role of Neural Networks

AlphaZero uses a neural network to guide the search process. The network has three main components: the trunk, the policy head, and the value head [23, 24].

The Trunk (Main Network): The trunk is the main part of the network. It processes the input and extracts useful features that will be used later in the heads. Specifically, it consists of the following components:

- *Convolutional layers:* A series of convolutional filters[1] that scan the input to detect important patterns. These filters enable the network to understand the spatial structure of the game state.
- *Residual blocks:* The trunk contains residual blocks, each consisting of two convolutional layers with BatchNorm [2] (normalization) and ReLU[3] (activation) functions. These residual blocks facilitate gradient propagation through the use of shortcut connections, which mitigates the vanishing gradient problem and enables the effective training of deeper architectures [24].

Policy Head: The policy head provides us with the action prior, which tells us 'where it is worth exploring'. It is responsible for predicting the probability distribution over actions that guides the search process during MCTS. The policy head consists of:

- *Logits:* The raw outputs from the policy head, denoted by $z_\theta(s, a)$, where s is the state and a is the action.
- *Softmax:* These logits are passed through a softmax function, which converts them into a probability distribution over actions for a given state, $p_\theta(a \mid s)$. This distribution guides the exploration in MCTS.
- *Masking:* Legal actions for a given state are assigned proper logits, while illegal actions are masked by assigning them a very low logit value (typically $-\infty$), ensuring that their probability is 0 after applying softmax.

The policy head is crucial for guiding the MCTS to explore the most promising parts of the game tree [23].

Value Head: The value head gives a scalar evaluation of the state, $v(s) \in [-1, 1]$ in the original algorithm, representing how favorable that state is, i.e., the outcome of the game or objective. The value head is structured as follows:

- *Convolutional layer:* The output from the trunk is processed by a 1x1 convolutional layer to extract relevant features for the value estimation.
- *Multilayer Perceptron (MLP):* The processed features are passed through an MLP, a neural network with ReLU activation composed of multiple fully connected layers[4], where each layer learns higher-level representations of features, allowing the modeling of complex non-linear relationships.
- *Tanh activation:* The final output is passed through a tanh activation function, which squashes the value into the range $[-1, 1]$, where 1 represents a winning state, and -1 represents a losing state.

The value head estimates how good the current state is and is essential for the MCTS algorithm to evaluate states efficiently [23].

Dirichlet Noise for Exploration: In the *root node* of the MCTS, Dirichlet noise is added to the policy prior to encourage exploration. The policy at the root node is modified by mixing the original policy with Dirichlet noise:

$$P_{\text{root}}(s, a) = (1 - \varepsilon)p_\theta(a \mid s) + \varepsilon\eta_a, \quad \eta \sim \text{Dir}(\alpha) \quad a \in \mathcal{A}(s) \tag{2.2}$$

where η_a is a sample from the Dirichlet distribution with parameter α and ε controls the level of exploration.

[1] A convolutional filter (or kernel) is a small matrix used to scan over an input, such as an image or game state, performing element-wise multiplication followed by summation to detect specific features, such as edges or patterns. It is a core component of convolutional layers in neural networks, enabling the network to learn spatial hierarchies in the data [24, 25].

[2] BatchNorm (Batch Normalization) normalizes activations by standardizing the mean and variance, helping stabilize training and speeding up convergence [24].

[3] ReLU (Rectified Linear Unit) is an activation function that sets negative values to zero and leaves positive values unchanged, helping mitigate the vanishing gradient problem, a situation where gradients become very small during training, making it difficult for the model to learn in deeper layers [24].

[4] Fully connected layers are layers in a neural network where each neuron is connected to every neuron in the previous layer, allowing each output to influence all the inputs of the next layer [24].

For *nodes other than the root*, the prior $P(s, a)$ is simply the policy predicted by the neural network, without Dirichlet noise:

$$P(s, a) = p_\theta(a \mid s) \tag{2.3}$$

The neural network outputs from the policy and value heads are used in conjunction with MCTS to guide the selection of actions and the evaluation of states, significantly improving the search efficiency and enabling AlphaZero to perform at an extremely high level in games such as chess, Go, and Shogi [12].

2.3.2. Monte Carlo Tree Search

Monte Carlo Tree Search is a sampling-based planning algorithm designed to approximate optimal decisions in large combinatorial search spaces. It is commonly formulated in the context of MDPs, where a problem is defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function, and a reward signal. In this framework, MCTS incrementally builds a partial search tree rooted at the current state, using Monte Carlo simulations to estimate the long-term value of actions [21].

One iteration of the standard MCTS procedure consists of four phases:

1. **Selection.** Starting from the root node (current state s), the algorithm recursively selects actions according to a tree policy until a leaf node is reached. In the classical *UCT formulation* [26], which stands for Upper Confidence Bounds applied to Trees, action selection at a node s is based on the *Upper Confidence Bound (UCB) principle*. This principle balances exploration and exploitation, addressing the fundamental tension between gathering information about uncertain actions and selecting actions believed to yield high rewards [27]. By effectively managing this trade-off, the search can be guided towards promising regions of the search space while avoiding the inefficiencies associated with purely random exploration. The UCB gives rise to the following selection rule:

$$a^* = \arg\max_{a \in \mathcal{A}(s)} \left[Q(s, a) + c\sqrt{\frac{\ln N(s)}{N(s, a)}} \right], \tag{2.4}$$

   where:

   - $N(s)$ denotes the total number of visits to state s,
   - $N(s, a)$ is the number of times action a has been selected from s,
   - $Q(s, a)$ is the empirical mean value of action a at state s,
   - $c > 0$ is an exploration constant controlling the exploration–exploitation trade-off.

   This rule is derived from the UCB1 algorithm [28][5] for multi-armed bandits and provides asymptotic guarantees of convergence toward the optimal action provided that rewards are stationary and bounded.

   In AlphaZero-style MCTS, this rule is replaced by the *PUCT* algorithm, which stands for Predictor and Upper Confidence bounds applied to Trees [12]. This formulation is a modification of UCT where the exploration term is weighted by a prior policy predictor [12]. Specifically, it incorporates prior probabilities provided by a neural network, allowing the search to prioritize promising regions of the state space based on learned experience rather than relying on uniform exploration. PUCT's selection rule is defined as follows:

$$a^* = \arg\max_{a \in \mathcal{A}(s)} \Big[Q(s, a) + U(s, a) \Big], \tag{2.5}$$

[5] UCB1 selects at round t the arm maximizing $\hat{\mu}_i + \sqrt{\frac{2 \ln t}{n_i}}$, where $\hat{\mu}_i$ is the empirical mean reward and n_i the number of times arm i has been played. The logarithmic exploration term decreases with n_i, ensuring a principled trade-off between exploration and exploitation [28].

$$U(s,a) = c_{\text{puct}}\, P(s,a)\, \frac{\sqrt{\sum_{b \in \mathcal{A}(s)} N(s,b)}}{1 + N(s,a)}, \qquad Q(s,a) = \frac{W(s,a)}{\max\{1, N(s,a)\}}. \tag{2.6}$$

Here, $P(s,a)$ is the prior probability returned by the policy network and c_{puct} controls the degree of exploration.

2. **Expansion.** When an unvisited state is encountered, the node is expanded by generating its legal successor states.
3. **Simulation (Rollout) and Backup.** In classical MCTS, once a newly expanded node is reached, a simulation (or rollout) is performed until a terminal state is obtained. The value of the simulation is then propagated backwards along the visited path.

   In its simplest form, rollouts are executed using random action selection. However, in structured combinatorial optimization problems, purely random simulations often yield poor value estimates. In the present work, we therefore employ a greedy heuristic—prioritizing trivial cherries—during the rollout phase to obtain a more informative approximation of the remaining objective value.

   The resulting value v is propagated backwards, updating:

$$\begin{aligned} N(s,a) &\leftarrow N(s,a) + 1, \\ W(s,a) &\leftarrow W(s,a) + v, \\ Q(s,a) &= \frac{W(s,a)}{N(s,a)}. \end{aligned} \tag{2.7}$$

   In AlphaZero-style MCTS, the rollout phase is replaced by a neural network evaluation that directly provides a value estimate $v(s)$ and action priors, eliminating the need for simulation.
4. **Action Selection.** After S simulations, the action to execute from the root is chosen according to the visit counts. Typically, a distribution proportional to $N(s,a)^{1/\tau}$ is used, where the temperature parameter τ regulates exploration.

After executing the selected action, the corresponding child node becomes the new root, and the previously constructed subtree is reused. A graphic representation of the main steps of a single iteration can be seen in Figure 2.8.

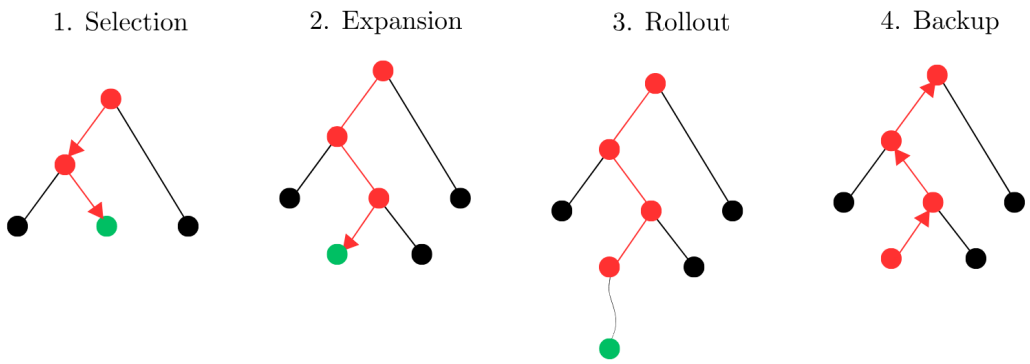


Figure 2.8: The four main steps of each iteration during the MCTS algorithm while guided by a value neural network: selection, expansion, simulation (rollout) and backpropagation. A robust framework for decision-making in complex search spaces.

In the context of this thesis, the construction of CPSs is modeled as an MDP, where each feasible leaf selection defines an action and each state is defined by the pair $\bar{s}_t = (\mathcal{T}_t, X_t)$, with $\mathcal{T}_t$ representing the current set of trees after leaf-pruning, and X_t being the set of remaining leaves. We first analyze the performance of vanilla MCTS using greedy-heuristic rollouts to estimate sequence weight. Subsequently, we investigate whether replacing the heuristic evaluation with a learned value function can improve guidance of the search process. Detailed descriptions of this learning framework are provided in Section 4.3.

3

Reformulating the Problem as a Markov Decision Process

In order to apply reinforcement learning to our problem, the first step is to formulate it as an MDP. This follows the common approach in reinforcement learning for combinatorial optimization, where the task is cast as a sequential decision-making problem. At each step, an *agent* interacts with an *environment* by selecting an *action* and the environment responds by producing a new *state* and a *reward*, progressively constructing a solution (see Figure 3.1) [27]. Formally, the agent is the learner and decision-maker, whereas the environment comprises everything external to the agent. MDPs provide a standard and well-established mathematical framework for representing such sequential interactions and their associated dynamics [29]. Throughout this work, the environment corresponds to the CPS instance and its current partially pruned configuration, while the agent is the decision-maker that iteratively selects actions to construct a CPS.

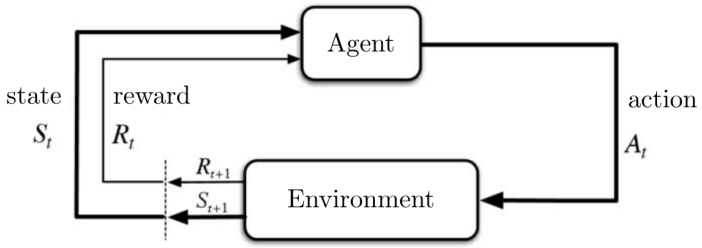


Figure 3.1: Reinforcement Learning framework used in Markov Decision Processes.

3.1. Preliminaries: Markov Property and MDP Components.

Consider how an environment might evolve at time $t+1$ as a result of the action performed at time t. In the most general scenario, this reaction could be influenced by all preceding events. In this case, the system dynamics can only be fully described by outlining the entire probability distribution [27]:

$$\Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_0 = s_0, A_0 = a_0, \ldots, S_t = s_t, A_t = a_t\}, \quad \forall r, r_i \in \mathcal{R}, s', s_i \in \mathcal{S}, a_i \in \mathcal{A}(\int_{\rangle}). \tag{3.1}$$

On the other hand, there are also scenarios in which the response at $t+1$ depends only on the state and action representations at t. In this case, we say that the state signal has the *Markov Property*.

Definition 12 *A problem is said to hold the **Markov property** when the environment's dynamics can*

be defined by specifying only the current state and chosen action:

$$p(r, s' \mid s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a\}, \quad \forall s, s' \in \mathcal{S}, a \in \mathcal{A}(\int), r \in \mathcal{R}. \tag{3.2}$$

In other words, a state has the Markov Property and is a Markov State if and only if (3.1) *is equal to* (3.2) *for all* $r \in \mathcal{R}, s' \in \mathcal{S}$ *and for all possible values of past events [27].*

When a problem satisfies the Markov Property and both the state and action sets are finite, the problem can be formulated as a *finite Markov Decision Process (MDP)*. To define a finite MDP, we must specify the following elements [27]:

- *State space*: A finite set of states, denoted $\mathcal{S}$, which encompasses all possible states that the agent can occupy.
- *Action space*: A finite set of actions, denoted $\mathcal{A}$, which represents all the actions available to the agent.
- *State transition probability function*: $p(s' \mid s, a) \doteq \Pr\{S_{t+1} = s' \mid S_t = s,\ A_t = a\}$. This function defines the dynamics of the system: the probability of transitioning from state s to state s' given action a is the sum of the probabilities across all possible rewards $r \in \mathcal{R}$:

$$p(s' \mid s, a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{3.3}$$

  where $p(s', r \mid s, a)$ is defined as in Equation (3.2)
- *Reward function*: The expected reward obtained from taking action a in state s, denoted $r(s, a) \doteq \mathbb{E}[R_{t+1} \mid S_t = s,\ A_t = a]$, which is computed as:

$$r(s, a) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{3.4}$$

- *Discount factor*: In some literature, the discount factor $\gamma \in [0, 1)$ is included as part of the MDP tuple. This factor is used to discount future rewards, capturing the trade-off between immediate and future gains.

With the finite MDP framework in place, the remainder of this section focuses on the CPS formulation. Specifically, we define the state representation and action space, characterize the state-transition and termination mechanisms, and conclude with a definition of the objective that will guide the reinforcement learning approach.

3.2. State Representation

At decision step t, the environment is fully observable. The *environment configuration* can be described by the pair

$$\bar{s}_t \doteq \big(\mathcal{T}_t,\ X_t\big), \tag{3.5}$$

where $\mathcal{T}_t$ is the collection of trees obtained after pruning the leaves selected in the first t steps (with $\mathcal{T}_0 = \mathcal{T}$), and X_t is the set of leaves that remain available for selection (with $X_0 = X$). This information is sufficient to determine the feasible actions at step t and the deterministic transition induced by applying an action.

In addition to $\bar{s}_t$, we store two auxiliary variables:

- σ_t, the partial CPS constructed so far (with $\sigma_0 = \emptyset$), which is required to reconstruct the final solution;
- h_t, the cumulative cost up to step t (with $h_0 = 0$), which accumulates the step-wise cost increments associated with each reduction.

While these quantities are not needed to characterize the dynamics of the reduced instance, they provide direct access to the current partial solution and objective value, and are therefore convenient for search and evaluation.

The terminal condition is reached when $X_t = \emptyset$, i.e., when all leaves have been selected.

The dynamics are deterministic: given a state and a feasible action (leaf selection), the next reduced instance is uniquely determined by pruning that leaf from all current trees and updating the remaining leaf set. The resulting state transition is described in detail in the following section.

Implementation note: the Python class `State` stores the reduced trees and remaining leaves via the attributes `trees` ($\mathcal{T}_t$) and `remaining` (X_t), and additionally keeps `sequence` (σ_t) and `hybrids` (h_t) for solution reconstruction and for tracking the cumulative cost. The complete code, including the definition of the class `State`, is available for completeness and further exploration by the reader on GitHub[1].

3.3. Action Space

In the CP literature, a move is often represented as an ordered pair (x, y), where $\{x, y\}$ forms a cherry in (at least) one of the current trees and the reduction removes x [10]. In our formulation, we use an equivalent but more compact representation and define an action as selecting a single leaf x to be removed. This should belong to a cherry in every tree.

At state $s_t = (\mathcal{T}_t, X_t)$, the feasible action set is defined as

$$\mathcal{A}(s_t) \doteq \begin{cases} X_t, & \text{if } |X_t| \leq 2, \\ \Big\{ x \in X_t \;\Big|\; \forall T \in \mathcal{T}_t,\ \exists\, y \in X_t \setminus \{x\} \text{ such that } \{x, y\} \text{ is a cherry in } T \Big\}, & \text{if } |X_t| \geq 3. \end{cases} \tag{3.6}$$

If $\mathcal{A}(s_t) = \emptyset$ while $|X_t| \geq 3$, the process reaches a dead end (the trees are not fully reduced but no further legal reduction is available). We treat this as a failed episode and penalize it so that the agent learns to avoid such states.

Implementation note: In the Python class `State`, the feasible actions are generated by `get_actions()`, which computes global-cherry candidates from the current reduced trees. For computational tractability, we apply a branching heuristic by prioritizing the selection of trivial cherries when they exist[10]. If no trivial cherries are available, a random subset of non-trivial cherries is selected, up to `MAX_BRANCH`. This parameter limits the number of candidate actions considered at each step. This is done to improve computational efficiency by avoiding the exploration of all possible actions, focusing only on a manageable, but still promising, subset. While this threshold was not reached during our current experiments, it was included to ensure scalability. Its implementation future-proofs the framework, allowing it to generalize to the larger datasets and more complex search spaces that typically necessitate such constraints. This sampling influences which actions are *considered* by the algorithm, but the underlying environment dynamics remain deterministic.

3.3.1. State Transition: $s_t \longrightarrow s_{t+1}$

Given a feasible action $a_t = x \in \mathcal{A}(s_t)$, the successor state s_{t+1} and auxiliary variables are deterministically obtained by applying the following steps:

- *Updating the partial sequence:* Append the selected leaf to the sequence, $\sigma_{t+1} = \sigma_t \,\|\, [x]$.
- *Pruning the selected leaf:* Remove x from all trees in $\mathcal{T}_t$, resulting in the new set of trees $\mathcal{T}_{t+1}$.
- *Updating the remaining leaves:* Remove x from the remaining leaves set, $X_{t+1} = X_t \setminus \{x\}$.
- *Updating the cumulative cost:* Increment the cost based on the selected leaf,

$$h_{t+1} = h_t + \Delta h(\mathcal{T}_t, X_t, x) \tag{3.7}$$

  Here, $\Delta h(\cdot)$ denotes the cost increment associated with selecting leaf x at the current step, computed according to the global-cherry weighting rule (see Definition 10).

The transition process continues until the terminal state is reached, where all trees are fully reduced. Once that state is reached, the final cumulative cost, incrementally updated throughout the process, corresponds to the reticulation number, which is the objective we aim to minimize. This will be formalized in the next section, where the terminal state and the objective are defined.

[1] `https://github.com/nataliavazpur/PhylogeneticNetworkInference`

Implementation note: The state transition is carried out by `apply_action()` in the `State` class. This method deterministically computes the next state by pruning the selected leaf in all trees and updating the set of remaining leaves, along with the partial CPS and the cumulative cost.

3.4. Terminal State and Objective

The goal of the problem is to minimize the reticulation number. To formalize this, we define the following:

- A state s is considered *terminal* if all trees in the set have been fully reduced, i.e., all leaves have been selected.
- If a terminal state is not reached but no more legal moves are available, the process is considered to have reached a *failed episode* or *dead state*, and it will be penalized. This situation occurs when no further legal reduction can be made, but the trees are not fully reduced.
- Our objective is to minimize the cost associated with a completed CPS. Since the MCTS algorithm we use maximizes the expected value, we define the episode reward $R(s)$ as follows:

$$R(s) = \begin{cases} -w(s) & \text{if all trees are fully reduced,} \\ -M & \text{if the episode is a failed episode.} \end{cases} \tag{3.8}$$

  Here, $w(s)$ is the total weight of the completed sequence and M is a large penalty value applied in the case of a failed episode.
- The value network will be trained to approximate the expected reward $v(s)$ given state s under the current policy π, i.e.,

$$v(s) = \mathbb{E}[R \mid s, \pi]. \tag{3.9}$$

Implementation note: The terminal state is checked using `is_terminal()` in the `State` class, which returns `True` when no remaining leaves are left to be selected. The strong penalty for failed episodes, where no cherries are available during simulation, is applied within the MCTS algorithm by backpropagating a large negative reward when no valid sequence can be generated.

3.5. Search Tree Definition

The search tree is constructed using the MCTS algorithm, as previously discussed. MCTS iteratively explores potential sequences of leaf deletions, expanding the search tree and simulating possible outcomes in order to estimate the expected reward. As outlined in the previous section, the objective is to minimize the total hybridization cost, which corresponds to the weight of the optimal CPS (see Theorem 1).

The inputs to the MCTS algorithm are as follows:

- *Root state:* The initial state from which the search begins, representing the current reduced trees and the remaining leaves.
- *Iterations:* The number of iterations to be performed by the MCTS algorithm, determining the depth of the search.
- *Exploration parameter c:* A parameter controlling the exploration-exploitation trade-off during the selection phase of MCTS, as defined in Equation 2.4.

The output of the MCTS process is the optimal CPS and the minimum reticulation number found, returned after a predefined number of iterations.

Implementation note: The MCTS algorithm is implemented in Python through the `Node` class, which encapsulates the structure of the search tree. Each node represents a state in the search space, and the class defines the following key methods:

- `is_fully_expanded()`: Checks whether all possible actions have been explored for a given node.
- `expand()`: Generates new child nodes representing possible successor states.
- `best_child()`: Selects the child node with the highest UCB1 value.
- `backpropagate()`: Propagates the reward back through the tree to update the value estimates.

4

Approach and Methodology

4.1. Differences and Approach in Our Algorithm

Although our approach builds on the general principles of AlphaZero, there are several important differences:

- **Two Independent Neural Networks (Future Perspective)** Unlike AlphaZero, which uses a single neural network with a shared trunk and two heads for both policy and value functions, our initial approach aimed to treat these two components separately. The rationale behind this was to address the two fronts (policy and value) independently, given that the inputs for each are different. The policy function, in particular, would have required a separate focus on action-based features, while the value function would focus on state-based features. While we originally envisioned building two independent networks for these functions, time constraints led us to prioritize the development of only the value network at this stage. However, we acknowledge the potential for future research to incorporate the development of a policy network, which would complement the value network and provide a more complete solution for the problem.
- **Simpler Network Architecture** While AlphaZero typically requires a deep neural network with multiple layers, the value network in our approach is a shallow MLP with just one hidden layer[1]. This is sufficient to encode the domain-specific properties of the CPS problem.
- **State-Dependent Action Space** In our CPS problem, the action space is dynamic and varies depending on the current state, reflecting the legal leaf removals at each step. This differs from AlphaZero, where the action space is fixed and identical across all states.
- **Single-Agent Problem and Tree Search** Unlike AlphaZero, which learns through self-play in a two-player setting, the CPS problem is a single-agent, purely combinatorial problem. Training data is generated using tree search (MCTS) rather than adversarial self-play.
- **Exact Algorithm as Oracle** Our approach intends to utilize an exact algorithm as an oracle to bootstrap the value targets during the initial training phase. This allows for supervised pre-training with optimal solutions, whereas AlphaZero is trained from scratch without access to exact target values.

In our algorithm, we use a learning-guided search strategy to improve MCTS performance for the CPS problem. The key differences with AlphaZero lie in the neural network architecture and the dynamic nature of the action space. By using two independent components instead of a single NN and by defining a flexible action space, our approach captures the distinct aspects of the CPS problem. The combination of MCTS with learning-guided search allows us to efficiently navigate the large combinatorial search space of CPSs, utilizing both exploration and guidance from the neural networks.

[1] A hidden layer is a layer in a neural network that lies between the input and output layers. It consists of neurons that perform transformations on the inputs, allowing the network to learn complex representations and features from the data [24].

The following sections will explore the feature engineering process and the architecture chosen to implement the neural network.

4.2. Feature Engineering

4.2.1. State Representation for Neural Guidance

Feature-Based State Definition

The effectiveness of neural guidance in combinatorial search depends on a compact yet expressive representation of the problem state. In the context of CPSs, the neural network must guide decisions based on the current configuration of the phylogenetic tree set. To achieve this, we adopt a *feature-based state representation* that summarizes local leaf-level information and global structural properties. This approach allows the neural network to prioritize dominant factors influencing action quality without the computational burden of processing the full tree topology directly.

Our strategy builds upon the framework established by Bernardini et al. [10], who demonstrated that even when the underlying problem is graph-theoretical, decision-relevant information can be effectively captured through carefully engineered features. Drawing from this insights, we transitioned from our initial focus on Graph Neural Networks (GNNs)[2]. While theoretically powerful, GNNs risk introducing overfitting and excessive complexity at this stage. Instead, we adapt the feature set proposed in [10] to guide the search toward globally optimal networks while maintaining efficient exploration.

Predictive Framework and Feature Selection

To identify the most informative descriptors, Bernardini et al. utilized a *random forest classifier*[3] to predict the *reducibility* of leaf pairs. Their study identified 19 features categorized by topological and branch-length distances.

Through *feature importance* analysis, they found that descriptors such as 'Trivial' (the ratio of trees where a pair is a cherry relative to those containing both leaves, where a value of 1.0 characterizes a strictly trivial cherry) and 'Leaf distance' were the most critical predictors for optimal reconstruction. This empirical evidence provides a rigorous foundation for our baseline, allowing us to exploit a validated subset of features that capture essential structural signals.

4.2.2. Implementation and Adaptation for MCTS

Refactoring the Feature Extraction Module

While we utilize the theoretical foundation of the Cherry-Picking Heuristics framework from [10], our implementation requires a significant departure from the original `Features.py` module. The original code is designed for custom tree objects within a specific ML pipeline, whereas our framework operates on *Biopython*[4] tree structures integrated with a MCTS engine.

To bridge this gap, we developed a specialized feature extraction module (`FeaturesMyThesis.py`). This adapted version computes features at three distinct hierarchical levels—*Cherry*, *Action (Leaf)*, and *State*—ensuring compatibility with our internal data structures while maintaining the core logic of the original descriptors.

Hierarchical Feature Architecture

The feature extraction process is designed as a hierarchical pipeline where each level is constructed by synthesizing and augmenting the information from the preceding one. Instead of merely transferring raw descriptors through the hierarchy, we employ multiple aggregation methods—including the mean, maximum, and standard deviation—at each transition. This approach ensures that the neural network receives a comprehensive statistical profile of the state, capturing both average trends and critical

[2] GNNs process graph-structured data by learning node representations through iterative message passing [30].

[3] A random forest is an ensemble learning method that outputs the class representing the *majority vote* of its individual decision trees, reducing the risk of overfitting [31].

[4] Biopython is a widely used open-source library for biological computation in Python—specifically, the `Bio.Phylo` module provides a standardized object-oriented framework for reading, representing, and manipulating phylogenetic trees in the Newick format [32].

outliers across different scales. The transformation from raw input trees in Newick format[5] to the final state representation follows this structured sequence:

$$\mathcal{T} \xrightarrow{\text{Compute Cherries}} \{(x,y)_1, \ldots, (x,y)_n\} \xrightarrow{\text{Extract Features}} \phi(x,y) \xrightarrow{\text{Aggregate}} \Phi(s) \tag{4.1}$$

To provide a deeper understanding of how these structural properties are processed, we describe the specific features and the logic implemented at each of the three functional levels below. However, we must first formally define the fundamental metric upon which the majority of our topological features are built: the node depth.

Definition 13 *For a rooted tree, the **depth of a node** x, denoted as depth(x), is defined as the distance from the root of the tree to that node. Formally, this metric is computed as the sum of the branch lengths along the unique path connecting the root to x (see Figure 4.1). In the specific context of this work, where unit branch lengths are assumed, depth(x) corresponds to the number of edges separating the node from the root.*

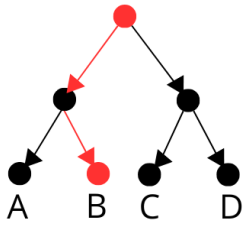


Figure 4.1: Visual representation of *node depth*. The distance from the root to the highlighted node (red circles) is 2, defined by the two unit-length edges (also in red) that form the unique path between them.

Cherry-Level Features At the base of the hierarchy, we compute a feature table for every reducible pair (x, y) identified in the current state. These features, represented by the vector $\phi(x, y)$, capture the immediate local context of the cherry across the tree set $\mathcal{T}$:

- *Count*: The absolute number of trees where (x, y) appears as a cherry.
- *Frequency (Tree Coverage)*: The ratio between the *Count* and the total number of trees in the set.
- *Mean Leaf Depth*: For each tree where (x, y) is a cherry, we calculate the average depth of both leaves from the root, defined as:

$$0.5 \cdot (\text{depth}(x) + \text{depth}(y)) \tag{4.2}$$

  . This value is subsequently averaged over all trees containing the cherry.
- *Mean Pair Distance*: The average distance between leaves across the tree set. For each tree, it is calculated as the length of the unique shortest path connecting x and y (see Figure 4.2), which is formally defined by the formula:

$$d(x,y) = \text{depth}(x) + \text{depth}(y) - 2 \cdot \text{depth}(LCA(x,y)) \tag{4.3}$$

  where LCA is the lowest common ancestor[6] of the two terminal nodes.
- *Mean Cherry Parent Depth*: The average distance from the root to the cherry's parent node, calculated across the subset of trees in which (x, y) is a cherry.

[5]The Newick format is a standard compact notation for representing the topology of rooted phylogenetic trees using nested parentheses and commas to group taxa by shared ancestry. For example, a pair of leaves $\{x, y\}$ forming a cherry is represented as `(x, y)`, where the parentheses enclose nodes that share a common parent [33].

[6]The Lowest Common Ancestor (LCA) of two nodes u and v in a rooted tree is the common ancestor that is farthest from the root. In a phylogenetic context, this node represents the most recent common ancestor from which both taxa u and v are descended.

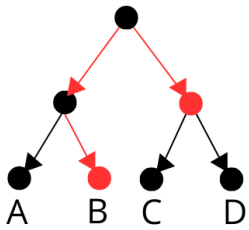


Figure 4.2: Visual representation of *pair distance*. The distance between the two red nodes is 3, which corresponds to the length of the unique shortest path connecting them (also in red). This value is obtained by summing the paths from each leaf to their Lowest Common Ancestor (LCA).

Note: In our implementation, missing branch lengths are defaulted to 1.0 to maintain a consistent distance metric. Consequently, under these unit length conditions, the *Mean Leaf Depth* is always exactly one unit greater than the *Mean Cherry Parent Depth*, making the two features mathematically redundant. Nevertheless, both are retained to ensure the framework's generalizability to trees with variable branch lengths, where they would capture distinct structural properties.

Leaf/Action-Level Features The second level of the hierarchy shifts the focus from specific pairs to individual leaves, which correspond to the available actions in our framework. We define S_x as the set of all reducible pairs where leaf x is involved, defined as:

$$S_x = \{(a, b) \in \textit{reducible\_pairs} \mid a = x \textit{ or } b = x\}. \tag{4.4}$$

This level constitutes the *action aggregation* stage, representing the structural profile of each potential action. In addition to the aggregated statistical summaries (mean, max, std) of the cherry-level features for S_x, we introduce two structural descriptors:

- *Degree*: The cardinality of S_x, representing the number of reducible pairs associated with leaf x.
- *Normalized Degree*: The ratio of the *Degree* to the maximum possible degree (the number of remaining leaves minus one).

State-Level Features The final level summarizes the global properties of the current CPS state in a single vector $\Phi(s)$, providing a comprehensive summary of the current problem configuration. This vector is generated by a global aggregation of the leaf-level features. For each descriptor, we compute its mean, maximum, and standard deviation across all active leaves to capture the overall complexity of the state. To situate these statistics in the global context of the problem, we incorporate two critical global scalars:

- *Number of Remaining Leaves*: A measure of the remaining size of the problem.
- *Hybridizations Found*: The cumulative count of reticulations identified so far in the sequence.

Input Mapping to Policy and Value Models

To prepare the extracted features for the neural networks, we define a specific mapping that organizes the data into input formats tailored for each model's objective. This structure ensures that each network receives information at the appropriate level of mathematical abstraction.

Policy Input Mapping The policy model f_θ^{pol} is conceived to evaluate actions in parallel. In this theoretical framework, its input is defined as a matrix $\mathbf{X}_{pol} \in \mathbb{R}^{n \times d}$, where n is the number of currently reducible leaves and d is the dimension of the leaf-level feature vector $\phi(x)$. The architecture is *intended* to process each row independently to compute a preference score, which would then be normalized across the n actions via a *softmax* layer to output the probability distribution $\pi(x \mid s)$.

Value Input Mapping The value model f_ω^{val} is designed to perform a global evaluation of the search state. It receives a single input vector $\Phi(s) \in \mathbb{R}^k$, where k integrates the aggregated statistics of all

leaves and the global state variables corresponding to the state-level features. This mapping compresses the topological complexity of the tree set into a fixed-size representation, allowing the model to estimate the scalar value $V(s)$ through a series of fully connected layers.

Detailed specifications regarding the internal layers and the implementation status of these two networks are provided in the following section.

4.3. Neural Network Architecture and Learning Strategy

While our framework is conceptually inspired by the AlphaZero paradigm, we implement a modular architecture where the policy and value functions are treated as independent entities. This separation is motivated by the distinct nature of their input representations and the specific computational challenges of the CPS problem.

4.3.1. Design Philosophy: Independent Models and Oracle Bootstrapping

A key difference from the standard AlphaZero approach is our intended use of the exact *Fixed-Parameter Tractable (FPT) algorithm*[7] developed by Borst et al. [15] as an oracle. In AlphaZero, networks are typically trained *tabula rasa* (from scratch) through self-play. In contrast, our framework is designed to *bootstrap* the learning process by using *optimal reference values* from this oracle as training targets. This supervised pre-training is intended to provide the network with high-quality signals from the start, bypassing the initial noise and inefficiency of random exploration.

This oracle relies on a $C++$ implementation that computes the exact hybridization number. While this solver is currently decoupled from our Python-based MCTS pipeline, requiring targets to be generated as a separate pre-processing step, it provides the optimal reference necessary for reliable state evaluation. Consequently, due to the computational overhead of this decoupling and the project's time constraints, heuristic approximations were employed as oracles in this instance to ensure the search remains computationally tractable.

Furthermore, we parameterize the policy and value functions as two independent MLPs rather than a dual-headed network. This separation allows for specialized optimization: the *value function* focuses on the global summary of the tree set, while the *policy function* is structured to evaluate discrete leaf-level actions.

4.3.2. The Value Network: Implementation and Training

The primary goal of the value network, $V_\omega(s)$, is to provide a direct estimate of the expected final cost from any non-terminal state s. By learning this mapping, we can replace the computationally expensive heuristic rollouts within the MCTS engine with a single forward pass of the network (see Figure 4.3). This allows the MCTS to evaluate states much faster, increasing the number of possible simulations within a fixed time budget.

Training Data and Trajectory Simulation

To generate a diverse dataset, necessary to train the network, we simulate multiple *trajectories*. A trajectory is defined as a complete sequence of leaf removals, starting from an initial tree set and proceeding until a terminal state is reached. We store each intermediate state s along these paths to create training samples $(\Phi(s), V^{target}(s))$. To maintain computational efficiency, we *limit* the trajectory length and the number of sampled states per tree set.

Target Definition and C++ Solver Decoupling

The exact algorithm used as our oracle is implemented in $C++$ [15]. Currently, this solver is decoupled from our Python-based MCTS framework, making it impractical to generate thousands of optimal targets in real-time. Consequently, we utilize *rollout-based targets* as an initial supervision source. For a given state s, a greedy rollout completes the CPS and returns an estimated number of additional hybridizations, $\widehat{h}_{\text{remaining}}(s)$. The learning target is defined as the negative of the total estimated cost to

[7] Fixed-Parameter Tractability (FPT) refers to a class of problems that can be solved efficiently if a specific parameter k is small, even if the general problem is NP-hard. In this case, the complexity is exponential in terms of the parameter k but polynomial in the input size n [34].

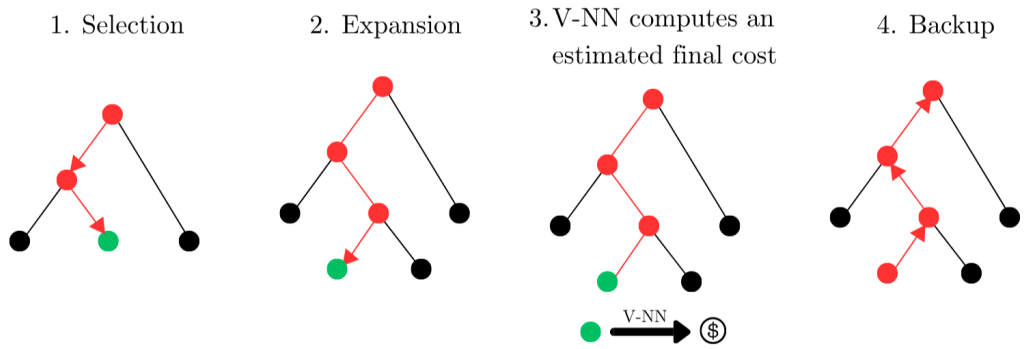


Figure 4.3: The four main steps of each iteration during the MCTS algorithm while guided by a value neural network: selection, expansion, estimation of the final cost and backpropagation.

ensure that higher values correspond to more desirable (lower-cost) states:

$$V^{target}(s) \approx -\left(h_{\text{so far}}(s) + \widehat{h}_{\text{remaining}}(s)\right) \tag{4.5}$$

Efficiency Justification: NN vs. Rollouts

Although rollout-based targets are readily available through simulation, evaluating the VNN is computationally much cheaper than running a full greedy completion. This speed-up is critical: by replacing heuristic rollouts with a fast neural network inference, the MCTS engine can evaluate states significantly faster. This allows us to increase the number of simulations within a fixed time budget, which is expected to translate into stronger search performance.

Model Implementation and Optimization

The VNN is implemented as an MLP regressor using *scikit-learn*. The architecture consists of fully connected layers with ReLU activations:

$$\mathbf{h} = \text{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b}), \quad \text{where ReLU}(z) = \max(0, z) \tag{4.6}$$

To ensure training stability, input features are normalized via standardization. The network is trained by minimizing the Mean Squared Error (MSE) between the predicted value $V_\omega(s)$ and the rollout-based target $V^{target}(s)$. This setup allows us to validate the full learning pipeline, from feature extraction to inference, before introducing more expensive exact supervision sources.

4.3.3. Theoretical Framework for the Policy Network

Although the implementation of a policy network has been deferred to future work, this section outlines the theoretical framework designed for its integration. This component is essential for optimizing search efficiency in problems characterized by high branching factors.

From Agnostic Exploration to Guided Search

In traditional MCTS, action selection is governed by the UCB rule, which initially treats all available actions as equally promising due to the lack of prior information regarding their quality. However, as discussed in Section 2.3.2, the introduction of a policy network f_θ^{pol} facilitates a transition to the *PUCT* selection rule (see Eqs. 2.5 and 2.6).

In this paradigm, the neural network does not act as a direct decision-maker. Instead, it provides 'suggestions' in the form of prior probabilities ($P(s, a)$) that reflect an initial intuition about which moves are most likely to be optimal. The MCTS engine subsequently validates and refines these suggestions through accumulated search statistics and simulations, effectively balancing the network's prior beliefs with the empirical evidence gathered during the search.

Challenges in Policy Training

The development of a high-quality policy model presents two primary challenges that motivated our decision to prioritize the value network in this stage:

- **The Tabula Rasa Challenge**: A common alternative, used in the AlphaZero algorithm [12], is to initialize the network with random parameters and train it from scratch through self-play. However, in the early stages of learning, such a network would provide noisy or near-uniform priors. Given that our current system is already informed by domain-specific heuristics, a randomly initialized policy would likely degrade search performance for a significant training period before becoming beneficial.
- **Computational Complexity of the Oracle**: Generating high-quality optimal reference targets for the policy would require solving multiple subproblems optimally for every state in a trajectory to identify the best possible move. Utilizing the exact $C++$ solver for this purpose is currently impractical without a *streamlined* integration of the oracle into our Python-based workflow [15].

Justification: Structural Constraints and Heuristic Priors
Despite these challenges, the absence of a policy network does not compromise the current efficacy of our solver due to the *structural constraints* already embedded in the pipeline. Our `get_actions()` function implements the domain-specific heuristics proposed by Bernardini et al. [10].

By filtering the action space to include only trivial cherries and a controlled subset of non-trivial ones, we are effectively applying a strong *implicit prior* that reduces the branching factor significantly. Consequently, the MCTS engine already operates within a highly refined search space. We therefore conclude that the most immediate performance gains are achieved by implementing the *value network*, which addresses the critical task of *predicting the final hybridization requirement* without the computational burden of heuristic rollouts.

4.4. Synthetic Dataset Generation Protocol

To effectively train the Value Network and benchmark the search algorithms, a controlled environment is required. Evaluating algorithms exclusively on empirical biological data is problematic because the true absolute minimum temporal hybridization number (H_{min}) is generally unknown. Therefore, a custom dataset generator was implemented in Python using the `NetworkX` library to synthesize phylogenetic forests with known constraints and a controllable level of reticulation complexity.

The generation of each dataset follows a systematic pipeline: base network construction, temporal hybridization, tree extraction, and binarization filtering.

4.4.1. Base Network Construction and Temporal Hybridization

The process begins by creating a random, balanced phylogenetic base tree for a chosen number of leaves. To ensure the final structure adheres to the strict chronological rules of our problem, each node is assigned a discrete time-stamp representing its depth.

Once the base tree is established, the algorithm transforms it into a temporal network by injecting a specific number of hybrid nodes (reticulations). To maintain temporal validity, the algorithm strictly samples pairs of edges that exist within valid chronological layers, ensuring no ‘time-traveling’ gene transfers or temporal cycles occur.

Instance Complexity: While the number of reticulations injected by the generator does not represent a strict mathematical bound for the optimal hybridization number, it serves as a proxy for the structural complexity of the instance. By controlling this parameter, we can generate datasets with varying levels of reticulation, providing a reference benchmark to evaluate how our heuristic algorithms scale and perform as the network’s complexity increases.

4.4.2. Extracting Displayed Trees (The `0/1 Switch')

To generate the input forest ($\mathcal{T}$) required by the algorithms, the generator extracts multiple *displayed trees* from the synthesized temporal network.

For each tree to be generated, the algorithm traverses the network. Whenever it encounters a hybrid node (a reticulation with multiple biological parents), it treats the incoming edges like a ‘0/1 switch’, stochastically choosing exactly one parental connection to keep while discarding the others. Following this extraction, any unreached nodes are pruned, and degree-two nodes (nodes left with only one

parent and one child) are suppressed, effectively cleaning the result back into a valid phylogenetic tree. To ensure uniformity, forests containing trees with missing leaves due to pruning are discarded, as a consistent leaf set is a prerequisite for the hybridization algorithms employed in this work. This process is repeated k times and saved in Newick format, where k is the desired number of trees.

4.4.3. Deterministic Binarization and Output Control

In early stages of the project, the generation algorithm did not strictly distinguish between binary and non-binary trees. To resolve this issue, a classification and binarization system was implemented, controlled by a user-defined `BINARY_FLAG`.

- **Binary Output Mode:** If strict binarization is requested, the algorithm applies a *deterministic binarization* routine. It sorts the child nodes of any polytomy (node with more than two children) alphabetically before expanding them into a sequence of binary splits (see Figure 4.4). This guarantees that structurally identical sub-trees extracted from the network always yield the exact same Newick string, preventing 'fake conflicts' that could confuse the neural network's loss function.
- **Non-Binary Output Mode:** If the flag is disabled, the algorithm utilizes a specific probabilistic edge-contraction function to force authentic polytomies, accurately simulating the evolutionary uncertainty often found in empirical biological datasets.

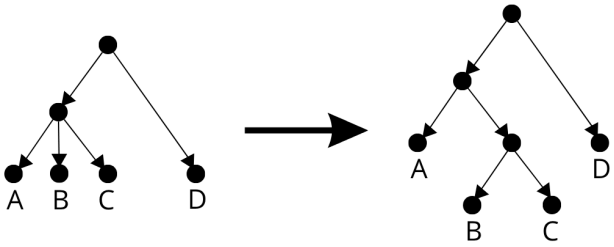


Figure 4.4: Transformation of a non-binary tree into a binary tree after deterministic binarization.

5

Algorithmic Engineering and Iterative Refinement

5.1. Foundations of Computational Complexity

Before presenting the empirical results, it is necessary to establish the theoretical complexity underlying the CPS problem. As established by Borst et al.[15], calculating the minimum temporal hybridization number for a phylogenetic forest is fundamentally NP-hard and defined by the interaction between the leaf count, the number of trees, and the hybridization weight. Consequently, the objective of the proposed algorithms is not to achieve an exact polynomial-time solution, which is computationally unfeasible[1], but rather to develop a reliable, highly scalable heuristic capable of providing robust upper bounds where exact solvers fail.

Currently, the most efficient FPT algorithm can solve this problem in $O(5^k \cdot n \cdot m)$ time. This complexity formula demonstrates that there exist algorithms for which both the number of leaves (n) and the number of trees (m) have a linear impact on the total running time. However, the hybridization number k would act as the exponential parameter (5^k), making it the primary computational bottleneck as the complexity of the network increases. While the 'linear load' of processing more taxa and trees remains manageable, the exponential growth driven by k quickly makes the search for the optimal CPS intractable for standard stochastic methods, justifying the development of value-guided heuristics to navigate the combinatorial explosion.

5.2. Evaluated Algorithms and Baselines

To rigorously evaluate the performance, scalability, and accuracy of our proposed learning-guided approach, we conducted a comparative analysis across five distinct algorithms. These algorithms range from unguided stochastic sampling to state-of-the-art exact solvers. Algorithms 1 through 4 were specifically designed and implemented in Python for this thesis, while Algorithm 5, corresponding to the FPT algorithm developed by Borst et al., serves as an external theoretical benchmark.

5.2.1. Unconstrained Random Search (URS)

The *Unconstrained Random Search* (implemented as the `Pure Random` baseline) represents the absolute lower bound of blind exploration. At any given state, URS identifies all legally reducible leaves—both trivial and non-trivial cherries—and selects one uniformly at random. It completely ignores the underlying topological structure and parsimony principles. To provide a fair comparison, URS operates within a time-budgeted multi-start loop, repeatedly generating random sequences and storing the one that yields the minimum hybridization cost. Its primary purpose is to demonstrate the vastness and complexity of the unguided search space.

[1] Unless P = NP.

5.2.2. Trivial-First Randomized Heuristic (TFRH)

The *Trivial-First Randomized Heuristic* serves as our informed stochastic baseline and acts as the 'expert' oracle for generating neural network training data. Unlike URS, TFRH applies a strict topological bias: it exclusively prioritizes the reduction of trivial cherries (leaves that do not induce hybridizations). Only when no trivial cherries are available does the algorithm randomly select a non-trivial cherry to force a structural conflict. Like URS, it runs in a continuous loop bounded by a time limit. While highly efficient on small, compatible networks, it lacks look-ahead capabilities and is prone to severe local minima in high-density forests.

5.2.3. Standard Monte Carlo Tree Search (MCTS)

The standard *MCTS* elevates the heuristic approach by introducing structured exploration. Relying on the traditional Upper Confidence Bound (UCB1) selection rule, MCTS systematically builds a search tree to evaluate the long-term consequences of pruning specific leaves. To estimate the value of unexplored nodes, MCTS relies on simulation rollouts utilizing the TFRH logic until a terminal state is reached. While it successfully escapes the short-sightedness of pure heuristics, its reliance on random rollouts creates a significant computational bottleneck as the factorial complexity of the problem increases.

5.2.4. Value-Guided Monte Carlo Tree Search (V-MCTS)

Our core methodological contribution, the *Value-Guided MCTS*, aims to resolve the scalability limitations of the classic MCTS. Inspired by the AlphaZero architecture, V-MCTS completely replaces the computationally expensive simulation phase with an inference from a trained Multilayer Perceptron (Value Network). This network evaluates the normalized topological features of the intermediate forest and directly predicts the future hybridization cost. By transitioning from rollout-based simulations to neural intuition, V-MCTS effectively bypasses the exponential depth of the search tree. A detailed breakdown of the evolutionary modifications made to V-MCTS—including step-wise root shifting and backtracking—is provided in subsequent sections.

5.2.5. Exact Solver (Fixed-Parameter Tractable Baseline in C++)

To establish a ground-truth benchmark for our heuristic and learning-based algorithms, we utilized the exact FPT algorithm developed by Borst et al. Implemented in C++, this external solver guarantees the discovery of the absolute minimum temporal hybridization number. To ensure environmental compatibility and avoid cross-platform toolchain issues, the code was compiled and executed within an Ubuntu environment using the Windows Subsystem for Linux (WSL), strictly following the authors' original build instructions.

While this exact solver provides an invaluable baseline, its applicability is severely constrained by the NP-hard nature of the problem, leading to rapid combinatorial explosion on massive network instances. Furthermore, although a nonbinary version of the algorithm is provided, empirical testing revealed that certain valid nonbinary instances trigger segmentation faults. This behavior aligns with the authors' explicit disclaimer that their C++ implementation is intended primarily as a proof of concept and may contain unresolved bugs.

Consequently, the exact solver is utilized strictly to validate the accuracy of our Python implementations on lower-dimensionality, binary datasets, where it performs with high reliability.

5.3. The Evolution of V-MCTS: An Iterative Engineering Framework

The development of the V-MCTS was not a single, direct implementation. Instead, the project was approached through an *iterative algorithm engineering* framework. In the context of NP-hard combinatorial optimization, it is rarely possible to design a flawless heuristic in a single pass. Rather, continuous empirical validation is required to systematically identify and overcome emerging computational bottlenecks.

This framework naturally dictates a three-step evolutionary cycle for each generation of the algorithm:

1. **Problem Identification:** Exposing the algorithm to increasingly complex phylogenetic networks to stress-test its limits and identify specific heuristic traps or computational inefficiencies.

2. **Algorithmic Solution:** Designing and integrating structural adaptations (such as step-wise root shifting or evaluation caches) to bypass the identified limitations.
3. **Empirical Result:** Re-evaluating the refined architecture to quantify internal performance gains and establish the new operational baseline.

This section details the evolution of the V-MCTS architecture across three major generations. By explicitly outlining the problems encountered, the solutions applied, and the resulting performance gains, we demonstrate how this continuous feedback loop allowed the algorithm to significantly refine its own efficiency and robustness, successfully overcoming the internal deadlocks that paralyzed its earlier versions.

5.3.1. V-MCTS v1.0: Global Search and the Inference Overhead

In its first iteration, V-MCTS maintained the global primary search structure of the baseline MCTS. The first innovation was replacing the random rollout phase with an MLP trained to predict the remaining hybridization cost, transitioning the algorithm from a 'blind' stochastic search to an 'intuition-guided' search.

Problem: Broad Exploration and the Iteration Bottleneck. Initial tests on a small baseline instance ($n = 10$ leaves, $m = 10$ trees) revealed a critical computational bottleneck. The standard UCB1 formula used in MCTS naturally promotes broad exploration across the search tree. When paired with the heavy per-node computational cost of extracting topological features and executing neural network inferences, this became a severe liability. Because the algorithm wasted resources evaluating thousands of shallow, sub-optimal branches instead of diving deep into the tree, it failed to reach a terminal state within the standard budget. To force the unoptimized V-MCTS to actually find a valid sequence, the iteration budget had to be massively inflated to 10,000 iterations. This forced expansion directly caused the algorithm to be significantly slower than the baseline MCTS (13.39s vs. 7.91s), entirely negating the benefits of the neural network.

Solution: State Caching and Hyperparameter Tuning. To mitigate this machine learning overhead, two structural changes were implemented. First, an *evaluation cache* (a programming technique also known as memoization) was introduced. By storing the neural network's output in a hash table using the immutable state parameters as a key, the inference time for previously visited or isomorphic states was reduced to an $\mathcal{O}(1)$ temporal cost. Second, the exploration parameter was aggressively lowered ($c = 0.1$). This hyperparameter tuning fundamentally shifted the algorithm's behavior: it forced the MCTS to stop exploring broadly and instead strictly exploit the neural network's predictions to reach the terminal leaves quickly.

New Result: Performance Crossover. By trusting the neural network's intuition, the algorithm required far fewer evaluations to converge on a valid path. As demonstrated in Table 5.1, the optimized V-MCTS required only 2,000 iterations to successfully identify the optimal solution. This reduction in the iteration budget, combined with the caching mechanism, allowed the V-MCTS to drop its execution time to 6.95 seconds, effectively outperforming the baseline MCTS.

Table 5.1: Performance comparison on an $n = 10, m = 10$ phylogenetic network instance.

Algorithm	Iterations	Time (s)	Result (Hybrids)
MCTS Baseline	1000	7.91	4
V-MCTS (Unoptimized)	10000	13.39	4
V-MCTS (Optimized)	2000	6.95	4

5.3.2. V-MCTS v2.0: Step-Wise Search and Intelligent Backtracking

Problem: The Greedy Bias and Search Starvation. While highly effective on $n = 12$, testing on higher-density networks ($n = 13, m = 35$) exposed a critical vulnerability. With a low exploration parameter ($c = 0.15$), V-MCTS followed the neural network's intuition too strictly, falling into a 'short-term gain' trap where immediate low-cost reductions forced multiple reticulations later (initially converging on 12 hybridizations instead of the optimal 6). Attempting to mitigate this by increasing exploration

($c = 2.0$) triggered *search starvation*: the algorithm distributed its computational budget too broadly across the massive global tree and failed to reach terminal states.

Solution: Step-Wise MCTS. To address this starvation, the architecture was transitioned to a *Step-Wise MCTS* (v2.0). Heavily inspired by AlphaZero's state progression, this version allocates a fixed simulation budget solely to determine the single best immediate action. Once an action is selected, the algorithm commits to that state, discards the previous tree, and roots a new MCTS.

Problem: Structural Myopia and Deadlocks. While the sequential model concentrated computational power effectively, it introduced a fatal flaw. If an early pruning sequence appeared optimal locally but logically locked the graph into a topological contradiction later, the forward-moving algorithm would crash into a deadlock.

Solution: Backtracking and Panic Resets. To guarantee sequence completion, an *intelligent backtracking* mechanism was engineered. V-MCTS v2.0 maintains a navigational pile of visited states. Upon detecting a deadlock, it reverts to the parent state, flags the toxic action as forbidden, and redirects the search. Furthermore, to prevent 'thrashing' between multiple dead-end branches, a *panic reset* protocol was introduced. If consecutive backtracks exceed a patience threshold, the algorithm aggressively resets to the absolute initial state, permanently banning the deceptive root action. The development of a more granular adaptation capable of banning only the specific failed sequence is left for future research. Due to time constraints, this approach was not explored, and we remain aware that while the current reset mechanism effectively prevents non-convergence, it may result in potentially sub-optimal solutions.

New Result: The Crossover Point. As summarized in Table 5.2, the step-wise V-MCTS successfully escaped local optima and matched the absolute global optimum. While MCTS was marginally faster on this specific instance, V-MCTS proved its capacity to generalize topological parsimony without starvation.

Table 5.2: Performance crossover on a high-density instance ($n = 13, m = 35$).

Algorithm	Leaves (n)	Trees (m)	Time (s)	Result (Hybrids)
MCTS Baseline	13	35	66.67	6
V-MCTS (Step-Wise)	13	35	71.14	6

5.3.3. V-MCTS v3.0: Scale-Invariant Features and Transfer Learning

Problem: Out-of-Distribution (OOD) Collapse. The final algorithmic hurdle emerged when attempting to scale the algorithm to massive phylogenetic networks ($n \geq 50$). Early neural network iterations utilized absolute topological magnitudes (e.g., the raw count of remaining leaves or absolute node depth). Because the network was trained on maximum leaf counts of 20, exposing it to the absolute magnitudes of a 50-leaf topology resulted in highly erratic predictions, misleading the MCTS into irrational pruning sequences.

Solution: Relative Metrics and Multi-Dataset Training. To enable robust transfer learning, the feature extraction pipeline was completely refactored. The absolute depth of a node is now divided by the maximum depth of its respective tree, and remaining leaf counts are calculated as percentages. Crucially, the system state dynamically inherits the `initial_leaf_count` from the root, anchoring these normalization metrics continuously throughout the sequence. Moving away from a single-instance oracle, the v3.0 network was also retrained on a highly diversified dataset comprising thousands of simulated trajectories from multiple independent phylogenetic forests, instead of simulated trajectories from a unique forest as happened in the previous versions of the algorithm.

New Result: High Predictive Accuracy (R^2). Following this rigorous, scale-invariant training regimen, the Multilayer Perceptron achieved a Coefficient of Determination (R^2)[2] of approximately 0.90. This ex-

[2]The Coefficient of Determination (R^2) is a statistical measure that represents the proportion of the variance in the dependent variable [35], in this case, the estimated hybridization cost, that is predictable from the independent topological features. Within this framework, a high R^2 value indicates that the neural network has successfully abstracted the underlying rules of evolutionary parsimony, distinguishing representative structural information from stochastic noise.

ceptionally strong metric proves the network successfully captured the underlying rules of evolutionary parsimony without overfitting ($R^2 = 1.0$). By utilizing aggressive regularization ($\alpha = 0.01$) and early stopping, the network abstracted a general resolution strategy, providing accurate heuristic guidance while leaving enough flexibility for the step-wise MCTS to explore and correct minor inaccuracies in massive state spaces.

6

Experimental Results and Performance Analysis

6.1. Implementation Details and Experimental Protocol

To ensure the reproducibility of the results, this section outlines the computational environment, the specific neural network hyperparameters, and the automated batch execution protocol used across these last experimental phases.

6.1.1. Computing Environment and Dual-Platform Workflow

The experimental phase of this project utilized a hybrid operating environment to balance ease of development with the strict requirements of the external exact solver. All simulations were conducted on an *LG Gram* laptop equipped with an *Intel Core i7* processor and *16 GB of RAM*.

The software stack was divided between two environments:

- **Native Windows Environment:** The core V-MCTS architecture, the dataset generation scripts, and the neural network training pipeline were implemented and executed using the *Spyder Integrated Development Environment (IDE)* within a native Windows environment. This allowed for efficient debugging and a streamlined workflow for the Python-based components. Additionally, the *RStudio IDE* was utilized within the same environment to perform the final data analysis and generate the comparative plots.
- **Linux Subsystem:** Due to the cross-platform toolchain requirements of the *exact FPT solver*, the C++ binary was compiled and hosted within the *Windows Subsystem for Linux (WSL2)* running *Ubuntu 22.04 LTS*.

6.1.2. Ground-Truth Validation Protocol

Due to the technical complexities involved in bridging the two environments, a fully automated end-to-end integration was not feasible. Consequently, the exact C++ solver was utilized as a selective *ground-truth benchmark* rather than as a continuous component of the batch process.

The validation protocol followed a two-tier strategy:

1. **Initial Calibration:** An initial extensive set of 70 phylogenetic network instances was processed manually through the C++ solver to establish a baseline of reliability for our Python implementations. Across the 70 validated instances, the outputs yielded by the exact C++ solver coincided perfectly with the minimum values identified by the suite of Python-based heuristics.
2. **Discrepancy-Triggered Validation:** For the remaining 460 experiments, the exact solver was invoked selectively. Verification was (manually) triggered only when at least one of the four heuristic algorithms (URS, TFRH, MCTS, and V-MCTS) provided different results, which happened on 131 occasions. Out of these cases, the exact solver revealed a discrepancy in only two instances,

where it found a lower reticulation count than the heuristics. This manual verification approach ensured that the absolute minimum hybridization number was available for critical data points and conflict resolution without incurring the prohibitive time cost of manual cross-environment execution for every individual simulated set.

In cases where all four heuristic strategies achieved consensus (i.e., returned the same hybridization number), that value was assumed to be the optimal result. This 'consensus-as-optimum' approach is justified by the computational scalability. The exact solver exhibits exponential growth in execution time relative to the number of reticulations. For instance, in a scenario with 17 found reticulations, the heuristic execution took 231 seconds, whereas the exact solver required 1126 seconds to terminate (a 4.8x increase).

Although this protocol introduces a minor heuristic element into the ground-truth definition, it ensures a feasible balance between mathematical rigor and the computational limits of manual cross-environment execution for large-scale datasets.

6.1.3. Software Stack and Core Libraries

The framework was developed in Python 3.10+. The following libraries were essential for the implementation:

- **NetworkX:** For the internal representation of phylogenetic forests as directed acyclic graphs.
- **Biopython (Bio.Phylo):** For the parsing, manipulation, and Newick serialization of tree clades.
- **Scikit-learn:** For the Value Network implementation, specifically utilizing the `MLPRegressor` and `StandardScaler` through an automated `Pipeline` to ensure consistent feature normalization.
- **Pandas & NumPy:** For the management of massive batch result datasets and numerical vector operations.

6.1.4. Value Network Training Strategy

The neural network architecture was refined through multiple iterations to balance predictive power and computational speed. The value network was trained using a *bootstrapping* strategy based on greedy trajectories.

The core learning parameters are detailed in Table 6.1. Notably, a *L2 regularization (alpha=0.01)* and *Early Stopping* were implemented to prevent the network from memorizing the expert's specific greedy biases, favoring a more generalized 'intuition' of the search space.

Table 6.1: Neural Network Architecture and Training Hyperparameters.

Component	Hyperparameter	Value
Architecture	Hidden Layer Sizes	(128, 64)
	Activation Function	ReLU
Optimization	Solver	Adam
	Initial Learning Rate	0.001
Regularization	Alpha (L2 penalty)	0.01
Training Logic	Validation Fraction	0.1
	Early Stopping	Enabled
	Maximum Iterations	300

6.1.5. Batch Execution and Stochastic Control

To mitigate the impact of stochastic variance, we implemented a fully automated batch testing script. This script facilitates the comparative analysis of our distinct approaches.

For each experimental configuration—defined by six combinations of leaf count n, tree count m, and target reticulations—the protocol executes 10 independent repetitions. Instead of using a single global seed, a unique random seed is generated and recorded for every batch. This approach balances statistical diversity with scientific reproducibility, ensuring that any specific outlier or successful trajectory

can be re-simulated by retrieving its original seed from the generated reports. The raw experimental outputs and CSV datasets are hosted in the project's GitHub repository, mentioned in Section 3.2.

6.2. Batch Performance Evaluation: A Comparative Study

The core of this research lies in evaluating diverse algorithmic paradigms to tackle the temporal hybridization problem. To provide a robust benchmark, we conducted a massive batch execution of 460 independent simulations across a heterogeneous dataset. These instances varied in leaf count (from $n = 6$ to $n = 30$), number of trees ($m = 2$ to $m = 35$) and best-known minimal reticulations ($k = 1$ to $k = 19$), ensuring that the results reflect generalizable performance rather than niche optimization.

The evaluation compares four distinct strategies: the proposed Value-Guided MCTS (V-MCTS), the MCTS implementation adapted for CPS reduction, the TFRH heuristic, and the Unconstrained Random Search (URS). The aggregate success rates, representing the percentage of instances where the algorithm identified the best-known minimum hybridization number, are summarized in Table 6.2 and visualized in Figure 6.1.

Table 6.2: Global success rates for the identification of the optimal temporal hybridization number across heterogeneous forest topologies.

Algorithm	Success Rate (%)
MCTS (Standard Rollouts)	**91.74%**
Unconstrained Random Search (URS)	89.13%
Trivial-First Randomized Heuristic (TFRH)	88.04%
V-MCTS (Value-Guided)	79.35%

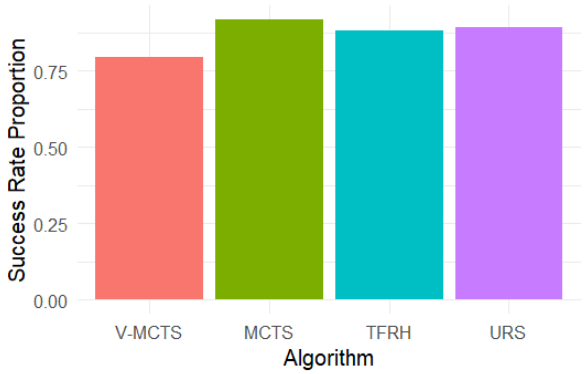


Figure 6.1: Global success rate comparison. Standard MCTS demonstrates the highest overall accuracy, while the neural-guided V-MCTS exhibits a comparatively lower success rate in this dataset.

To ensure a fair comparison between these diverse paradigms, the computational budget was balanced across two dimensions: search intensity and temporal parity. For the standard MCTS, a global budget of 1,000 iterations was allocated to explore the search tree. In contrast, the V-MCTS operates under a step-wise architecture, where a fixed simulation budget is allocated to determine each individual action. We assigned 500 simulations per step to the V-MCTS; given that a complete reduction sequence involves multiple steps, the total cumulative simulations performed by the V-MCTS often match or exceed the global iteration count of the standard MCTS, ensuring that the neural-guided search has sufficient local depth to validate its value-head predictions.

The specific choice of 1,000 iterations for the standard MCTS and 500 simulations per step for the V-MCTS was determined through preliminary empirical calibration. These values represent a optimization ceiling identified during initial testing, where increasing the search budget produced diminishing returns in success rates while significantly inflating execution time. Furthermore, a strictly numerical limit (number of iterations) was preferred over a temporal limit for the MCTS-based algorithms to ensure platform-independent reproducibility. By fixing the number of search operations rather than the execution time, the results remain consistent across different computing environments, effectively isolating

the algorithmic efficiency from the underlying hardware performance. This approach also provides a normalized baseline to evaluate the 'intelligence' of the neural guidance; by allowing both MCTS variants to explore a comparable number of states —in the sense that both budgets were calibrated to reach the previously mentioned performance plateau while taking the same order of time— we can directly observe whether the Value Network's intuition leads to superior paths compared to unbiased rollouts, regardless of the inference overhead introduced by the neural network.

Furthermore, a strict temporal constraint was imposed on the stochastic baselines (URS and TFRH). For each specific instance, the execution time for these heuristics was capped at the minimum time taken by either of the two MCTS variants. This 'minimum-time' policy ensures that the high success rates of the random searches are a result of their efficiency under restricted conditions, rather than an artifact of having an unlimited temporal budget. This approach was adopted after initial experiments showed that comparing algorithms by a fixed number of iterations was misleading, as the stochastic baselines required significantly more wall-clock time to complete the same number of cycles, leading to an inherently unequal comparison of computational effort. By capping the time instead, we highlight the comparative value of structured tree search over iterative random search.

6.2.1. Post-hoc Analysis and Statistical Significance

To determine if the differences in success rates observed in Table 6.2 were statistically significant across the entire algorithmic suite, we performed a *Cochran's Q test*. This non-parametric test is specifically designed to evaluate differences in proportions (success rates) among three or more matched sets of binary responses (success/failure on the same instances).

The test yielded a highly significant result, with a Cochran's Q test statistic of $Q(3) = 52.12$ (where 3 denotes the degrees of freedom) and a p-value of $p < 0.001$. This provides strong evidence to reject the null hypothesis of equal performance across all strategies and confirms that the choice of algorithmic architecture is a decisive factor in the likelihood of identifying the optimal hybridization number.

Given the global significance established by the Cochran's Q test, we proceeded with a *post-hoc pairwise analysis* to isolate the specific differences between algorithm pairs. To maintain statistical rigor and control the family-wise error rate, we conducted pairwise McNemar tests with a *Bonferroni correction*. This adjustment ensures that the significance threshold is tightened when performing multiple comparisons, preventing the inflation of Type I errors (false positives) [36].

McNemar's test is a statistical method used on paired nominal data. It is typically applied to 2×2 contingency tables with a dichotomous trait and matched pairs of subjects to determine 'marginal homogeneity': that is, whether the row and column marginal frequencies are equal. In the context of this research, a total of six pairwise comparisons were performed, each corresponding to a unique contingency table. For the sake of clarity and brevity, we present the contingency table for the primary comparison between V-MCTS and standard MCTS (Table 6.3) as a representative example of the discordant performance dynamics.

Table 6.3: Contingency table for V-MCTS vs. Standard MCTS success/failure counts.

		Standard MCTS	
		Success	Failure
V-MCTS	Success	345	20
	Failure	77	18

The contingency table illustrates the underlying dynamics that drive the statistical test. While both algorithms reached a consensus on the optimum in 345 cases, the *discordant pairs* provide the most significant insight: Standard MCTS succeeded in 77 instances where V-MCTS failed, while V-MCTS identified the optimum in 20 instances where the standard version did not. The fact that V-MCTS solved these 20 'unique' cases suggests that neural guidance can explore valid topological paths that are overlooked by unbiased rollouts, although it currently lacks the overall consistency of the standard MCTS.

The complete results for all six post-hoc pairwise comparisons, including the adjusted p-values after Bonferroni correction, are summarized in Table 6.4.The results reveal a clear performance gap between

the neural-guided approach and the other strategies. Specifically, V-MCTS was found to be significantly less effective than the standard MCTS, TFRH and URS (all with $p < 0.001$). This confirms that the current implementation of value-guided search introduces biases that may impede the identification of the optimum compared to unbiased stochastic methods.

Table 6.4: Post-hoc pairwise McNemar test results with Bonferroni correction.

Comparison	Raw p-value	Adj. p-value	Significance
V-MCTS vs. MCTS	4.59×10^{-9}	2.75×10^{-8}	***
V-MCTS vs. TFRH	4.50×10^{-5}	2.70×10^{-4}	***
V-MCTS vs. URS	2.52×10^{-6}	1.51×10^{-5}	***
MCTS vs. TFRH	0.0213	0.1280	n.s.
MCTS vs. URS	0.0884	0.5300	n.s.
TFRH vs. URS	0.5680	1.0000	n.s.

Significance levels: *** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$; n.s. (not significant)

Interestingly, the analysis showed *no statistically significant difference* between the standard MCTS and the two randomized heuristics, URS ($p = 0.530$) and TFRH ($p = 0.128$), despite the higher aggregate success rate of the former. In a standalone comparison, MCTS outperformed TFRH with a raw p-value of 0.0213, which would typically denote statistical significance. However, after applying the conservative Bonferroni correction to account for the multiple comparisons within our four-algorithm suite, the adjusted p-value rose to the previously mentioned 0.1280. While this exceeds the 0.05 threshold, *it highlights a strong performance trend that favors MCTS over traditional heuristics*, even if it cannot be strictly confirmed as a universal superiority under this rigorous framework.

Similarly, after applying the Bonferroni correction, the performance of TFRH and URS proved to be statistically indistinguishable ($p = 1.00$). In a strict statistical sense, a p-value of exactly 1.00 implies that the observed data aligns perfectly with the expectations under the null hypothesis (which postulates no underlying difference in performance between the two methods). It indicates a maximum probability of obtaining these exact results, or less extreme ones, assuming the null hypothesis holds true. While failing to reject the null hypothesis does not prove it to be true, this complete lack of statistical divergence, coupled with the marginally higher absolute success rate of URS, suggests that the heuristic strategy of prioritizing trivial reductions is largely ineffective for these specific instances. This counterintuitive finding, contrary to initial expectations, will be thoroughly analyzed and contextualized in Section 6.4.3.

6.2.2. Error Magnitude and Failure Robustness

Although the success rate provides a primary measure of performance, it does not quantify the magnitude of sub-optimality when an algorithm fails. The *error distance* addresses this by evaluating the gap between the algorithmic result and the best-known minimum (k). To effectively capture the central tendency, the dispersion of sub-optimal values, and the presence of extreme outliers, a modified boxplot visualization was employed (Figure 6.2). Because discrete error values inherently cause point overlapping (overplotting), we implemented a *dual visual encoding* strategy: both the size and the color gradient of the points represent the frequency of each specific outcome. This redundant encoding not only resolves the overplotting issue but also enhances visual accessibility, allowing for an immediate comparison of both the frequency and severity of algorithmic failures.

As observed in Figure 6.2, all algorithms exhibit a median error of zero, indicating that the heuristic framework is fundamentally robust and identifies the optimum in the majority of instances. However, the analysis of outliers reveals a significant distinction in the reliability of each approach. Notably, while the neural-guided V-MCTS has a lower overall success rate, it demonstrates a more 'conservative' failure behavior; its maximum error is strictly bounded at 8 reticulations. In contrast, the standard MCTS and the randomized heuristics (TFRH, URS) exhibit more severe 'catastrophic' failures, with error distances reaching up to 12 reticulations. This suggests that the neural component in V-MCTS might act as a regularizing guide that effectively constrains the search space, preventing the algorithm from diverging into highly sub-optimal regions.

To provide a precise numerical breakdown of these distributions, Table 6.5 details the exact frequency

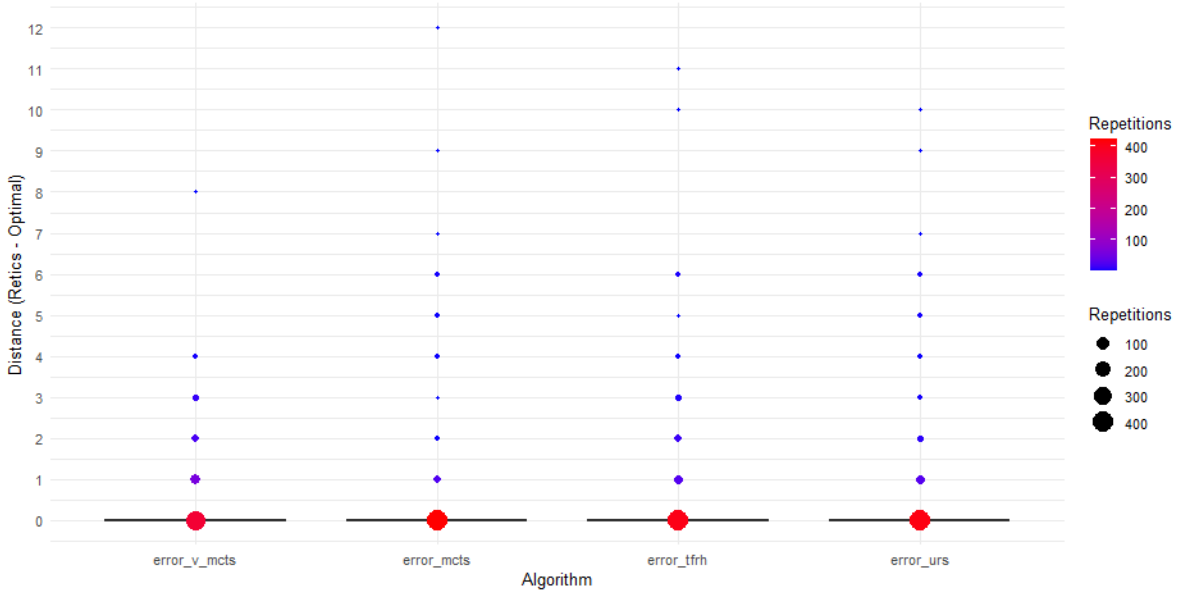


Figure 6.2: Distribution of the error distance to the optimum. To mitigate the overplotting of discrete values, a dual encoding approach is used: both point size and color gradient (from blue to red) indicate the number of repetitions. While all algorithms exhibit a median error of 0 (massive red dots at the baseline), the outliers highlight the maximum deviation. V-MCTS shows a more constrained error margin, avoiding the extreme outliers seen in the standard MCTS.

of each error margin. An in-depth review reveals a distinct difference in how errors are distributed between the neural-guided approach and the other algorithms. While the standard MCTS, TFRH, and URS achieve a higher absolute success rate (lower total errors), their failures are spread across a much broader spectrum of error distances, frequently extending into higher penalties (values between 5 and 12). Conversely, although V-MCTS accumulates the highest total number of errors (95, including 2 timeouts), these deviations are highly concentrated at the lowest end of the scale. Specifically, with the exception of a single isolated outlier at a distance of 8, every other sub-optimal execution of the V-MCTS missed the minimum by 4 reticulations or fewer, with the vast majority (58 cases) deviating by just a single reticulation. This dynamic highlights a clear trade-off: the value network in V-MCTS effectively acts as a safety net that localizes errors to tight, near-optimal neighborhoods, whereas the unguided exploration of the non-neural algorithms is more susceptible to becoming trapped in widely divergent, severe failure states.

6.3. Parametric Sensitivity and Complexity Drivers

To understand how the structural characteristics of the phylogenetic forests influence algorithmic performance, we conducted a sensitivity analysis. This allows us to move beyond aggregate success rates and isolate the specific *complexity drivers* that challenge each architecture.

6.3.1. Spearman Rank Correlation Analysis

We employed the *Spearman rank correlation coefficient* (ρ) to quantify the relationship between problem parameters (leaf count (n), number of trees (m), and best-known reticulation minimum (k)) and the binary success rate of each algorithm. Spearman was selected over Pearson's correlation because it does not assume normality and is capable of capturing *non-linear monotonic relationships*. This is essential given the discrete nature of our parameters and the binary outcome (success/failure) of the simulations.

The correlation values are derived from the association between the specific dimensions of each of the 460 instances and the corresponding algorithmic outcome. A positive coefficient indicates that increasing the parameter facilitates the identification of the optimum, while a negative value denotes an increase in problem difficulty.

The heatmap in Figure 6.3 reveals several critical insights:

Table 6.5: Frequency of error distances across algorithms. The error distance represents the difference between the number of reticulations found and the optimal solution. 'Total Number of Errors' indicates the frequency of non-optimal solutions, and 'Accumulated Error' aggregates the total deviation from the optimum. NA (Not Available) corresponds to runs that reached the maximum computational budget or iteration limit without converging to a valid sequence.

Error Distance	V-MCTS	MCTS	TFRH	URS
NA	2	0	0	0
0 (Optimal)	365	422	405	410
1	58	25	27	28
2	20	2	14	8
3	11	1	5	3
4	3	2	2	2
5	0	2	1	3
6	0	3	4	3
7	0	1	0	1
8	1	0	0	0
9	0	1	0	1
10	0	0	1	1
11	0	0	1	0
12	0	1	0	0
Total Number of Errors	**95**	**38**	**55**	**50**
Accumulated Error	**151**	**96**	**128**	**120**

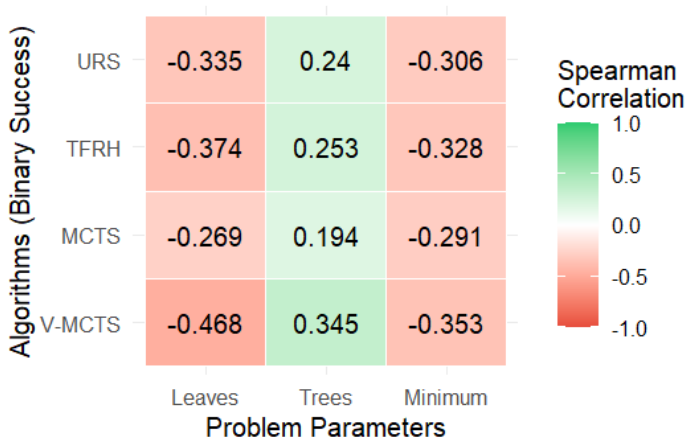
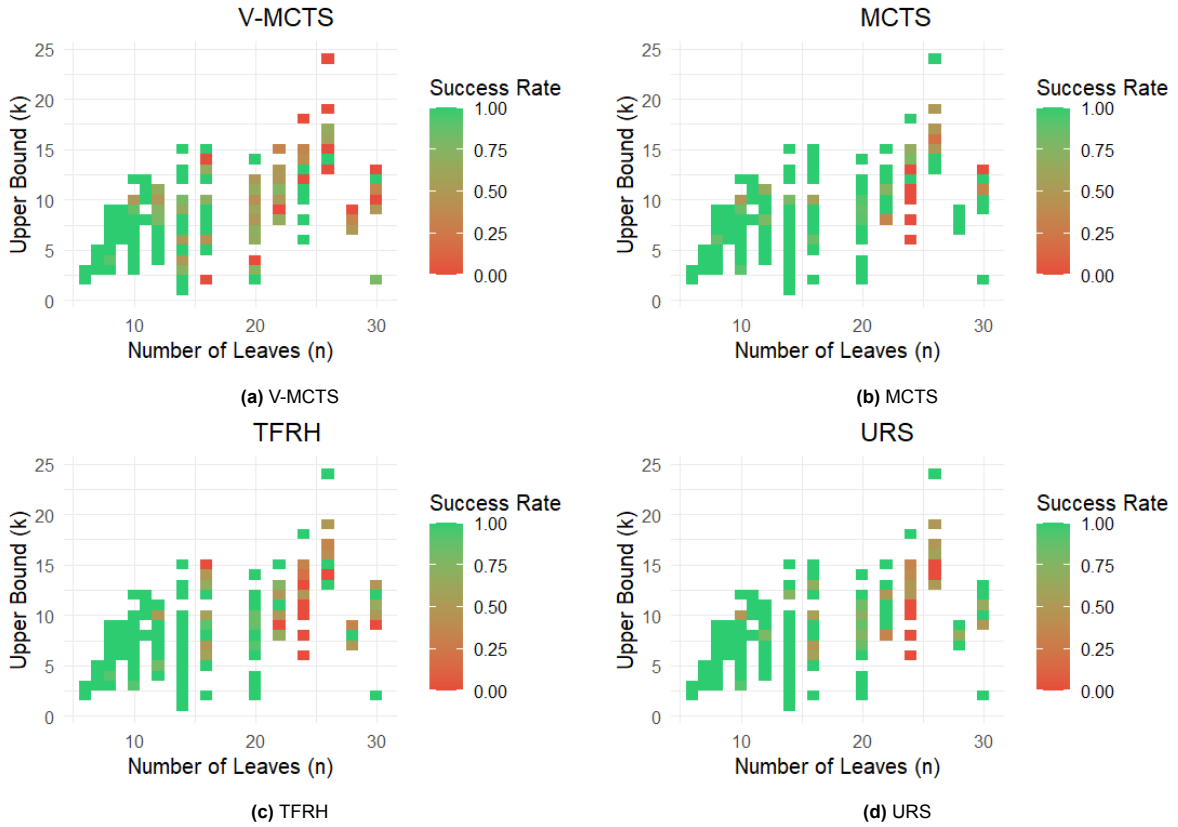


Figure 6.3: Heatmap of Spearman correlation coefficients between problem parameters and algorithm success. Negative values (red) indicate that increasing the parameter reduces success, while positive values (green) indicate improved performance.

- **The Impact of Leaves (n):** All algorithms show a negative correlation with leaf count, but *V-MCTS exhibits the highest sensitivity* ($\rho = -0.468$). This suggests that the current neural guidance model faces significant generalization challenges as the taxonomic scale increases.
- **The Reticulation Barrier (k):** The minimal reticulation count acts as a *universal complexity driver*. All algorithms show similar negative correlations (approx. $\rho = -0.30$ to $\rho = -0.35$), confirming that the search space depth is a fundamental constraint regardless of the search strategy.
- **The Facilitator Effect of Forest Size (m):** Notably, the number of trees exhibits a consistent *positive correlation* with performance. This suggests that a higher density of input trees provides a more coherent topological signal, effectively narrowing the search space and aiding the algorithms in identifying the optimal CPS. Nevertheless, this finding may be dependent on the tested range; since the upper limits of forest density were not exhaustively explored, it is possible that this facilitating effect could plateau or diminish in significantly larger instances where the increasing computational overhead might eventually outweigh the benefits of the additional topological signal.

6.3.2. Performance Stability across the Complexity Spectrum

To visualize the *stability region* of each strategy, we mapped the success rate across the two primary dimensions of complexity: the number of leaves (n) and the optimal reticulation number (k). Figure 6.4 illustrates how performance degrades as instances move toward the upper-right quadrant of the complexity space.



(a) V-MCTS **(b)** MCTS **(c)** TFRH **(d)** URS

Figure 6.4: Success rate stability across parametric complexity (n vs. k). While MCTS (b) maintains a high success profile across a wider range, V-MCTS (a) shows a more rapid transition to failure in high-leaf, high-reticulation scenarios.

The visual comparison reinforces the Spearman analysis. While MCTS and the randomized baselines (URS, TFRH) show a relatively stable performance for $n \leq 20$, the V-MCTS heatmap exhibits significant *thermal decay* (a shift from green to red) much earlier. This visualization confirms that the current neural-guided search is more susceptible to the *curse of dimensionality* inherent in larger phylogenetic reconstructions (larger sets of leaves).

6.3.3. Logistic Regression Analysis

To quantify the individual contribution of each parameter to the probability of algorithmic success, we fitted four *binary logistic regression models*, using the *glm* (Generalized Linear Models) function in R. Unlike correlation coefficients, these models allow us to estimate the *effect size* of each complexity factor while controlling the others. The estimated coefficients (β), standard errors, and significance levels are summarized in Table 6.6.

Table 6.6: Logistic regression coefficients for algorithmic success predictors.

Predictor	**V-MCTS** (β)	**MCTS** (β)	**TFRH** (β)	**URS** (β)
Intercept	3.907***	5.249***	5.145***	4.672***
Leaves (n)	-0.135***	-0.088**	-0.130***	-0.103***
Trees (m)	0.070**	0.030	0.048.	0.049.
Minimum (k)	-0.119**	-0.177***	-0.148***	-0.144***

Significance levels: *** $p < 0.001$; ** $p < 0.01$; * $p < 0.05$; . $p < 0.1$

The regression analysis provides a nuanced explanation for the performance gaps:

- **Scaling Sensitivity:** The negative impact of leaf count (n) is most pronounced for V-MCTS ($\beta = -0.135$), nearly 50% stronger than for MCTS ($\beta = -0.088$). This confirms that as the taxonomic scale increases, the probability of the value-guided search finding the optimum decays significantly faster than its stochastic counterparts.
- **The Signal-to-Noise Paradox:** V-MCTS is the only architecture where the number of trees (m) is a highly significant positive predictor ($p = 0.002$). This suggests that the *Value Network* effectively extracts topological information from larger forest sets to guide the search, whereas the performance of MCTS remains largely indifferent to forest density ($p = 0.270$).
- **Depth Constraints:** While MCTS is more robust to leaf count, it exhibits the highest sensitivity to the reticulation limit k ($\beta = -0.177$). This indicates that while MCTS handles 'wide' problems (high n) effectively, its performance is more sharply affected by the 'depth' of the hybridization network, likely due to the exponential expansion of the search tree. Nevertheless, it is important to note that the disparity between these coefficients across the four strategies is relatively narrow—this suggests that the reticulation limit remains a formidable bottleneck for every tested architecture, with only marginal variations in how each algorithm scales with k.

6.4. Discussion: The Landscape of Algorithmic Success

The results offer a revealing perspective on how different levels of stochasticity and heuristic guidance interact with phylogenetic complexity.

6.4.1. The Robustness of Monte Carlo Tree Search

The most significant finding is the superior performance of the standard MCTS (91.74%). The adaptation of MCTS to the Temporal Hybridization Problem proves to be a highly effective architectural choice. By combining systematic tree exploration with stochastic rollouts, the algorithm maintains a high degree of *exploratory plasticity*. Unlike purely greedy methods, the MCTS can 'reconsider' early reductions, which is crucial for navigating the NP-hard constraints established in Section 5.1. Its success suggests that for the tested complexity range, the combination of MCTS structure and unbiased stochastic sampling provides the most reliable path to the global optimum.

However, the *post-hoc pairwise analysis* reveals that this superiority should be interpreted as a robust *performance trend* rather than a statistically confirmed advantage; when applying the conservative *Bonferroni correction*, the differences between MCTS and the randomized heuristics did not reach the conventional significance threshold, despite the observed gap in aggregate success rates. Furthermore, as evidenced by the outlier analysis in Section 6.2.2, this exploratory plasticity comes with a risk: when the algorithm fails to converge, the lack of heuristic guidance can lead to larger error distances (up to 12 reticulations) compared to its value-guided counterpart.

6.4.2. Stochastic Density and the Random Baseline

The high success rate of the URS (89.13%) indicates that many instances in the heterogeneous set are 'stochastically dense', meaning they contain multiple valid paths to the optimal k. In these scenarios, the absence of heuristic bias is an advantage. While more complex algorithms may attempt to 'over-engineer' a path, the URS succeeds through the unconstrained diversity of its exploration, stumbling upon optimal sequences that fixed heuristics might overlook due to their rigid rules.

This points toward a form of algorithmic minimalism: for the tested complexity range ($n \leq 30$), optimal paths remain frequent enough to be identified by unconstrained search, suggesting that the implementation of complex, domain-specific heuristics may be redundant at this scale. This finding is significant as it defines a 'complexity threshold' below which the computational overhead of neural guidance is not yet justified, providing a clear benchmark for *when* a transition to more robust reinforcement learning architectures becomes strictly necessary.

6.4.3. The Stochastic Paradox: Evaluating the Efficiency of Iterative Random Loops

A particularly striking result from the batch evaluation is the performance hierarchy between the *URS* and the *TFRH*. Despite TFRH being built upon a recognized phylogenetic principle—prioritizing trivial cherries to minimize hybridization cost—it achieved a lower success rate 88.04% compared to the 89.13% achieved by URS. Even if the post-hoc analysis found no statistical evidence to reject the null hypothesis of equal algorithmic performance, the result remains counter-intuitive.

While this lack of significance does not definitively prove the null hypothesis, it challenges the initial expectation that a domain-specific heuristic—designed to prioritize biologically plausible reductions—would inherently outperform a purely unconstrained stochastic search. Although this heuristic was originally established in slightly different phylogenetic contexts, it was a highly plausible assumption that prioritizing such structural reductions would yield an accuracy advantage in the Temporal Hybridization Problem.

This outcome is remarkable when considering the computational and theoretical dynamics observed:

- **Throughput as an Accelerator:** TFRH exhibited a higher iteration frequency, completing an average of 10 additional iterations per session compared to URS within the same temporal budget. This confirms that the heuristic successfully acts as a computational *accelerator*, streamlining the decision-making process by resolving 'obvious' structural matches.
- **Heuristic Neutrality and Confluence:** The statistical parity seems to support the theory of *confluence* in trivial cherries. This theory suggests that, since these cherries are present in all trees, their reduction is mandatory and generally does not conflict with other operations. Therefore, whether these cherries are prioritized (as in TFRH) or selected at random (as in URS), the final cost k would remain largely invariant. In this light, the heuristic would serve as an engine for speed rather than a map for discovering superior paths.
- **The Paradox of Search Diversity:** However, the marginal lead of URS reveals a fundamental incongruity. If trivial reductions were perfectly commutative in every instance, the success rates would probably be identical. The fact that URS occasionally finds the optimum where TFRH fails hints at a possible *heuristic trap*. While we have previously mentioned that trivial reductions could be assumed to be confluent, the slight performance gap suggests that in highly complex instances, even these 'safe' moves might not be entirely neutral. In this way, it is conceivable that a rigid 'trivial-first' policy induces a form of *structural myopia*, where early reductions fix temporal or topological constraints that inadvertently block access to the global optimum—a path that the unconstrained, 'disordered' exploration of URS can still navigate. Although a formal counterexample to this confluency was not identified in this study, we leave it as an open possibility for future theoretical verification.

Furthermore, while heuristics similar to 'trivial-first' are frequently discussed in the literature, there is—to the best of the author's knowledge—a notable lack of formal studies investigating the efficacy of iterative, repetition-based random loops specifically for the Temporal Hybridization Number problem. The remarkable performance of these stochastic baselines, particularly the *>89% accuracy of URS*, suggests that for moderate-scale problem instances, the implementation of complex, domain-specific

heuristics may be redundant. This finding highlights a form of algorithm minimalism where simpler, unconstrained methods achieve results comparable to more structured approaches.

Ultimately, the observed results appear to be internally inconsistent with a single unified theory, revealing a counter-intuitive 'stochastic advantage' where 'less guidance' can occasionally yield more effective exploration. While these outcomes may be influenced by the specific characteristics of the synthetic datasets used in this study, they represent a significant insight into the behavior of stochastic solvers. We invite future research to investigate whether this efficiency persists in real-world biological datasets, where factors such as topological noise and incomplete lineage sorting may further complicate the parsimony landscape and test the limits of purely random exploration.

6.4.4. The V-MCTS Performance Gap: Bias and Expert Limits

The V-MCTS (79.35%) presents a lower aggregate success rate, which can be attributed to the 'heuristic trap' phenomenon. This sub-optimal behavior is likely underpinned by four critical architectural factors:

- **Compounding Error:** As the V-MCTS was trained using the TFRH (88.04%) as an expert, it may have inherited the structural biases of its teacher. In a sequential decision process, a single high-confidence but incorrect prediction from the Value Network can lead the search into a sub-optimal branch.
- **The Exploration Trade-off:** While the value network prunes the search space to handle exponential growth, it also reduces the 'lucky' stochastic diversity that allows URS and Standard MCTS to thrive in moderate-complexity instances.
- **Convergence vs. Optimality Trade-off:** The current backtracking implementation employs an aggressive reset mechanism to ensure convergence. When consecutive backtracks exceed a predefined patience threshold, the algorithm resets to the absolute initial state and permanently bans the root action (the first cherry picked in the failed sequence). While this effectively prevents the search from becoming trapped in non-productive cycles or infinite loops, it introduces a risk of discarding the global optimum. If the optimal solution indeed begins with the banned action but requires a specific sub-sequence that the heuristic failed to identify, the current logic may prematurely prune the optimal branch. Developing a more granular adaptation, capable of banning specific sub-sequences rather than entire root actions, is identified as a critical objective for future research to reconcile search stability with global optimality.
- **Feature Representation and Domain Mismatch**: A fourth, more fundamental factor may lie in the selection of input features. The current state representation is heavily informed by the work of Bernardini et al. [10], whose feature importance analysis identified the 'Trivial' descriptor as a primary predictor for optimal reconstruction in general CPSs (see Section 4.2.1). However, our observation of the 'stochastic paradox'—where the trivial-first heuristic failed to outperform purely random search—suggests that the structural signals governing the Temporal Hybridization Problem may differ significantly from those in non-temporal contexts. If the value network is predominantly guided by features that are less informative for temporal parsimony, it may be suffering from a form of 'signal noise', prioritizing suboptimal moves.

6.4.5. Operational Limits and Scalability Recommendations

To address the practical boundaries of the current framework, it is essential to delineate the parameter thresholds beyond which algorithmic performance and feasibility begin to decline. Although an exhaustive boundary analysis was not within the scope of this study, empirical observations suggest an operational ceiling for the current Python-based implementation. Specifically, the proposed suite is reliably recommended for problem instances of the order of 30 leaves (n), 20 trees (m), and 20 reticulations (k), as these configurations consistently yielded high-quality results within a manageable computational budget.Beyond these limits, we observed a significant 'scaling decay'. As instances approached the upper end of the parameter spectrum (e.g., $n > 30$ or $k > 17$), execution times for individual algorithms reached up to 20 minutes. In a comparative context, this translates to approximately one hour of processing time per single instance to complete the four-algorithm suite, making large-scale batch validation increasingly prohibitive. Furthermore, exceeding this threshold was associated with a sharp decline in success rates, as the search space expands factorially and the neural-guided

V-MCTS becomes more susceptible to the 'curse of dimensionality'. Consequently, while the current architecture is highly effective for moderate-scale datasets, these findings highlight the necessity of the low-level optimizations (such as a C++ transition) discussed in Section 7.3 to push the frontier of phylogenetic reconstruction toward larger, more complex evolutionary datasets.

6.4.6. The Efficacy of the Markov Decision Process Framework and Sequential Decision Success

One of the most significant contributions of this research is the formalization of the Temporal Hybridization Number problem as an *MDP*. To the best of the author's knowledge, this represents the first instance in the literature where the search for an optimal CPS has been modeled through the lens of sequential decision-making and RL architectures.

The results are highly compelling. Achieving a success rate of 91.74% with the baseline MCTS confirms that the MDP framework successfully captures the underlying topological logic of phylogenetic reductions. Beyond mere accuracy, the efficiency of this approach is remarkable:

- **Convergence Speed:** The majority of the algorithms converged within a range of 0.1 seconds to 5 minutes.
- **Prototyping and Optimization:** While instances at the upper end of the parameter spectrum reached execution times of 15 minutes on average, it is important to consider that the current implementation was developed entirely in *Python*. This suggests that a transition to a low-level language (such as C++) or further algorithmic optimization could reduce these times to near-instantaneous levels, even for complex instances.

In response to the foundational research question that initiated this study, whether RL can be effectively utilized to obtain fast, high-precision heuristics for phylogenetic networks, the evidence provides an affirmative and resounding answer. The ability of the MCTS to navigate the combinatorial landscape of the CPS demonstrates that RL-based architectures are not only viable but could potentially represent a superior path toward scalable and parsimonious evolutionary reconstruction.

7

Conclusions and Future Work

This research set out to investigate the viable use of RL and MDPs as a framework for solving the Temporal Hybridization Number problem. By transforming the search for an optimal CPS into a sequential decision-making task, we have bridged the gap between phylogenetic theory and modern heuristic optimization.

7.1. Summary of Findings

The experimental results provide a resoundingly affirmative answer to our initial research question. The primary conclusions of this work are as follows:

- **The Power of MDP Formalization:** We have successfully demonstrated that phylogenetic tree reduction can be modeled as an MDP. The 91.74% success rate achieved by the MCTS baseline proves that the state-action representation captures the essential topological constraints required to minimize hybridization events.
- **Efficiency and Scalability:** Despite the NP-hard nature of the problem, the proposed heuristics achieved convergence in time frames ranging from milliseconds to a few minutes. This confirms that RL-based architectures can bypass the exponential complexity $O(5^k \cdot n \cdot m)$ that paralyzes exact solvers, offering a scalable alternative for large-scale analysis.
- **The Heuristic-Stochastic Trade-off:** Our batch evaluation revealed a fascinating 'Specialization Paradox'. In moderate-scale instances, unbiased stochastic exploration (URS) outperformed guided heuristics (TFRH, V-MCTS). This suggests that while neural networks provide necessary intuition for massive search spaces, they can introduce a 'Heuristic Bias' that may lead to local optima in simpler scenarios—a finding that challenges traditional assumptions about greedy guidance.
- **Implementation Feasibility:** The use of Python for the entire pipeline proved sufficient for prototyping and research validation. The results indicate that the current 'linear load' of processing multiple trees is manageable, though it establishes a clear path for future low-level optimizations.

7.2. Limitations

While the results are highly satisfactory, this study encountered specific limitations that define the boundaries of the current architecture:

- **Reset vs. Granular Backtracking:** Due to time constraints, the algorithm employs an aggressive reset mechanism when encountering non-convergence. While effective for ensuring completion, this approach may discard valid sub-sequences, potentially leading to sub-optimal results in highly deceptive landscapes.
- **Expert Ceiling:** The Value Network was trained using a greedy heuristic (TFRH) as its teacher. Consequently, the network may have inherited specific topological blind spots, limiting its ability

to surpass the performance of its own training oracle.

- **Computational and Dimensional Scalability**: As analyzed in Section 6.4.5, the current framework is recommended for instances up to 30 leaves and 20 reticulations. Beyond this threshold, a sharp decline in success rates and an increase in execution times were observed, identifying a clear operational ceiling for the Python-based implementation.

7.3. Future Work

While this thesis provides a robust foundation for the Hybridization Number problem via RL, through the use of MDPs and MCTS, the results highlight several avenues where the framework can be significantly extended to reach its full potential.

Biological Realism and Topological Accuracy

Beyond the primary objective of minimising the reticulation number (k), future research must evaluate the *topological accuracy* of the reconstructed networks. While the current parsimony-based approach identifies the minimum number of reticulation events, multiple non-isomorphic networks may share the same optimal value of k. Testing the MCTS architecture on empirical DNA-derived trees will be crucial to assess its robustness against biological noise, horizontal gene transfers, and ILS.

To transition from pure mathematical optimisation to biological inference, the framework should incorporate metrics capable of comparing the inferred topology against a *ground truth*. We propose the integration of the following metrics:

- *Robinson-Foulds (RF) Metric:* Historically the standard for phylogenetic trees, the RF metric measures the distance between two trees by calculating the symmetric difference of their bipartition sets . Each internal arc in a tree defines a bipartition of the leaf set; the distance reflects how many of these evolutionary clades are present in one tree but absent in the other [37].
- *The μ-distance (Path-Multiplicity Vectors):* As a robust extension of the RF metric for networks, the μ-distance (or p-distance) provides a true distance measure for the class of *tree-child networks*. It represents each network as a multiset of p-vectors ($\mu(u)$), where each vector counts the number of distinct paths from a specific node to each leaf in the network . This metric is particularly powerful as it satisfies the separation axiom, ensuring that non-isomorphic tree-child networks are assigned a non-zero distance [37].
- *Tripartition Distance:* This metric generalises the RF concept to networks by associating a *tripartition* to each arc. For any given arc, the leaf set is divided into three disjoint groups: strict descendants, non-strict descendants, and non-descendants. Although tripartitions offer high resolution, they may fail to distinguish certain network pairs, necessitating their use in conjunction with more injective representations like μ-vectors [38].

Incorporating these distances into the MCTS reward function could shift the search priority from pure numerical minimisation toward *biological plausibility*. By penalising trajectories that deviate significantly from known reference topologies, the algorithm would ensure that identified hybridisation events reflect true ancestral relationships rather than just mathematical shortcuts for reduction.

High-Performance Algorithmic Optimization

A significant bottleneck identified in this study was the computational overhead of the Python-based reduction logic. Porting the core MDP environment and the cherry-picking logic to a low-level language like C++ would dramatically increase the sampling frequency. This would allow the MCTS to explore significantly deeper trajectories and a wider breadth of the search space within the same temporal budget, potentially overcoming the 'scaling decay' observed in larger leaf-count instances (Heuristic $n > 25$).

Refined Search and Tabu-Mechanisms

To mitigate the *heuristic trap* and the *greedy trap* discussed in Section 6.4, future iterations should move away from absolute state resets. Implementing a sequence-level *tabu search* adaptation would allow the algorithm to ban only the specific failed sub-sequences of reductions while keeping the rest of the search tree intact. It involves the implementation of a memory-based mechanism that systematically records and prohibits specific failed sequences of reductions, preventing the algorithm from

revisiting known sub-optimal trajectories while maintaining the exploration of the surrounding search space. This granular pruning would preserve the 'lucky' stochastic discoveries while systematically excluding proven sub-optimal paths.

Adaptive Exploration Constants

Our parametric analysis showed that MCTS and V-MCTS respond differently to increases in leaf count (n) and reticulation depth (k). A promising direction would be the implementation of *adaptive exploration constants* (e.g., c_{puct}), where the balance between exploration and exploitation is dynamically adjusted based on the estimated complexity or the remaining computational budget of the instance. This would allow the algorithm to autonomously increase the exploration pressure in high-complexity regions of the phylogenetic landscape, where finding the minimum number of recombination events is known to be NP-hard, while shifting toward exploitation as the computational budget nears exhaustion or the cherry-picking sequence approaches completion.

Domain-Specific Feature Engineering

As previously discussed in Section 6.4.4, the reliance on a feature set originally optimized for general CPSs may represent a significant structural limitation. While descriptors such as 'Triviality' were identified as primary predictors in non-temporal contexts, the observed parity between URS and TFRH suggests that this importance might decrease when strict chronological constraints are imposed. Future research should undertake a dedicated feature importance study—potentially utilizing a random forest classifier—to isolate the specific topological descriptors that drive optimality within the Temporal Hybridization Problem. By moving beyond a borrowed feature set and implementing a representation explicitly tailored to temporal parsimony, the value network could achieve more granular guidance. This refined implementation would allow the neural architecture to better distinguish between locally appealing trivial moves and the globally optimal trajectories that respect the complex chronological requirements of the phylogenetic landscape.

Transition to a Policy-Value Architecture:

A primary objective for future iterations is the integration of a dedicated *policy network* (f_θ^{pol}) to transition from agnostic exploration to a fully guided search. While the current solver relies on the UCB rule and hard-coded heuristic filters to manage the branching factor, a learned policy would provide prior probabilities ($P(s, a)$) to the PUCT selection rule. This would allow the MCTS to focus its computational budget on moves with a high 'neural intuition' of optimality, effectively replacing implicit heuristic priors with a more flexible, learned navigation of the search space. This architecture, inspired by the success of AlphaZero [12], is essential for scaling the solver to instances where the hybridization landscape becomes too vast for manual filtering.

Optimal Oracle Training and the Imitation Gap:

As noted in our analysis of current limitations, the V-MCTS is currently bounded by the performance of the TFRH used during supervised training. To eliminate this 'imitation gap', future research should focus on training the value network using an *optimal oracle*. By leveraging high-performance $C++$ exact solvers [15] to generate ground-truth hybridization numbers (k) for the training set, the network would learn the true underlying complexity of the Temporal Hybridization Problem rather than a heuristic approximation. This transition from 'heuristic' to 'optimal' supervision is expected to significantly improve the accuracy of state evaluations and the overall success rate in NP-hard scenarios. We propose this as a strategic intermediate step toward a potential *tabula rasa* implementation. This approach allows for the full validation of the neural guidance framework before committing the extensive computational resources required for autonomous self-play learning.

Transition to Self-Play and Reinforcement Learning

The original ambition of this project, developing a self-improving agent, remains a critical frontier. To move beyond the current supervised learning constraints regarding the training of the neural network, a *Self-Play* architecture similar to AlphaZero should be implemented. This would involve:

- **Tabula Rasa Initialization and Strategy Discovery:** A fundamental extension would be to initialize the policy and value networks with random parameters, moving away from the *imitation learning* phase. By starting from a *tabula rasa* state, the agent is not constrained by the structural

biases of the expert. This allows for the discovery of novel reduction strategies in the NP-hard hybridization landscape, potentially identifying sequences that prioritize non-trivial cherries in ways that current human-designed heuristics do not anticipate.

- **Continuous Iterative Learning Loop:** Instead of a static, one-time training process, future work should implement an *online reinforcement learning* cycle. In this paradigm, the MCTS acts as a policy improver: the search data generated in each session is used to continuously retrain the neural networks. This creates a self-reinforcing feedback loop where the networks provide better priors for the MCTS, and the MCTS, in turn, generates higher-quality labels for the next training iteration, ensuring the agent improves with every execution.
- **Curriculum Learning and Experience Replay:** To maintain stability during the transition to self-play, the framework could implement *curriculum learning*, where the agent initially masters moderate-scale instances before being exposed to higher reticulation depths (k). Coupled with a *Prioritized Experience Replay* buffer, the system would store and revisit 'critically informative' trajectories, those where the agent discovered a more parsimonious path than the current baseline, ensuring that rare but optimal discoveries are effectively integrated into the model's weights.
- **Multi-Objective Reward Shaping:** The self-play reward signal should be expanded beyond binary success. By incorporating the μ-distance [37] into the reinforcement signal, the agent can be rewarded for the *topological plausibility* of the resulting network. This would guide the self-play process toward solutions that are not only mathematically optimal in terms of k, but also structurally consistent with the ground-truth evolutionary history.

In conclusion, this thesis establishes a new precedent for the application of Reinforcement Learning in computational phylogenetics. The developed tools and the underlying MDP framework provide a solid foundation for future efforts to reconstruct the complex, reticulate history of life on Earth, bridging the gap between deep search algorithms and biological truth.

References

[1] T. Ryan Gregory. “Understanding Evolutionary Trees”. In: *Evolution: Education and Outreach* 1 (2008), pp. 121–137. DOI: 10.1007/s12052-008-0035-x.

[2] Mark A. Ragan. “Trees and networks before and after Darwin”. In: *Biology Direct* 4.43 (2009), pp. 1–38. DOI: 10.1186/1745-6150-4-43. URL: http://www.biology-direct.com/content/4/1/43.

[3] Aparna Gunjal and Sonali Shinde, eds. *Microbial Diversity in Hotspots*. Boca Raton: CRC Press, 2022.

[4] Magnus Bordewich and Charles Semple. “Computing the minimum number of hybridization events for a consistent evolutionary history”. In: *Discrete Applied Mathematics* 155.8 (2007), pp. 914–928. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2006.08.008. URL: https://www.sciencedirect.com/science/article/pii/S0166218X06003957.

[5] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2004.

[6] Lavanya Kannan and Ward C. Wheeler. “Maximum Parsimony on Phylogenetic Networks”. In: *Algorithms for Molecular Biology* 7.9 (2012). DOI: 10.1186/1748-7188-7-9.

[7] Sungsik Kong, David L. Swofford, and Laura S. Kubatko. “Inference of Phylogenetic Networks From Sequence Data Using Composite Likelihood”. In: *Systematic Biology* 71.2 (2022), pp. 279–297. DOI: 10.1093/sysbio/syab037.

[8] Niels Holtgrefe et al. “PaNDA: Efficient Optimization of Phylogenetic Diversity in Networks”. In: *bioRxiv* (2026). DOI: 10.1101/2025.11.14.688467. eprint: https://www.biorxiv.org/content/early/2026/02/25/2025.11.14.688467.full.pdf. URL: https://www.biorxiv.org/content/early/2026/02/25/2025.11.14.688467.

[9] Peter J. Humphries and Charles Semple. “Note on the hybridization number and subtree distance in phylogenetics”. In: *Journal of Mathematical Biology* 66.7 (2013), pp. 1507–1522. DOI: 10.1007/s00285-012-0549-9.

[10] Giulia Bernardini, Leo van Iersel, et al. “Constructing Phylogenetic Networks via Cherry Picking and Machine Learning”. In: *Algorithms for Molecular Biology* 18.1 (2023). DOI: 10.1186/s13015-023-00233-3.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[12] David Silver, Thomas Hubert, Julian Schrittwieser, et al. “A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play”. In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.

[13] C. Dewi et al. “Synthetic data generation using DCGAN for improved traffic sign recognition”. In: *Neural Computing and Applications* 34.24 (2022), pp. 21465–21480. DOI: 10.1007/s00521-022-07464-3.

[14] Shengbo Eben Li. *Reinforcement Learning for Sequential Decision and Optimal Control*. Singapore: Springer, 2023. DOI: 10.1007/978-981-19-7782-5.

[15] Ander Borst, Leo van Iersel, M. Jones, et al. “New FPT Algorithms for Finding the Temporal Hybridization Number for Sets of Phylogenetic Trees”. In: *Algorithmica* 84 (2022). Accessed: 2026-02-28, pp. 2050–2087. DOI: 10.1007/s00453-022-00946-8. URL: https://doi.org/10.1007/s00453-022-00946-8.

[16] Leo van Iersel and Steven Kelk. “When two trees go to war”. In: *Journal of Theoretical Biology* 269.1 (2011), pp. 245–262. DOI: 10.1016/j.jtbi.2010.10.024.

[17] Sungsik Kong et al. “Classes of Explicit Phylogenetic Networks and their Biological and Mathematical Significance”. In: *Systematic Biology* 71.6 (2022), pp. 1464–1482. DOI: 10.1093/sysbio/syac019.

[18] Peter J. Humphries, Simone Linz, and Charles Semple. “Cherry Picking: A Characterization of the Temporal Hybridization Number for a Set of Phylogenies”. In: *Bulletin of Mathematical Biology* 75.10 (2013), pp. 1879–1890. DOI: 10.1007/s11538-013-9878-8.

[19] Remie Janssen, Mark Jones, and Yukihiro Murakami. “Combining Networks using Cherry Picking Sequences”. In: *Algorithmica* 84.11 (2022), pp. 3207–3234. DOI: 10.1007/s00453-022-00958-3.

[20] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson, 2006.

[21] Cameron B. Browne, Edward Powley, Daniel Whitehouse, et al. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

[22] Maryam Karimi-Mamaghan et al. “Machine Learning at the service of Meta-heuristics for solving Combinatorial Optimization Problems: A state-of-the-art”. In: *European Journal of Operational Research* 296.2 (2022), pp. 393–422. DOI: 10.1016/j.ejor.2021.04.032.

[23] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (2017), pp. 354–359. DOI: 10.1038/nature24270.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[25] Advanced Deep Learning. “CNN Operation with 2 Kernels Resulting in 2 Feature Maps: Understanding the Convolutional Filter”. In: (2019). Accessed: 2026-02-27. URL: https://medium.com/advanced-deep-learning/cnn-operation-with-2-kernels-resulting-in-2-feature-mapsunderstanding-the-convolutional-filter-c4aad26cf32.

[26] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Proceedings of the 17th European Conference on Machine Learning (ECML)*. Springer, 2006, pp. 282–293. DOI: 10.1007/11871842_29.

[27] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. Cambridge, MA: MIT Press, 2018.

[28] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. “Finite-time Analysis of the Multiarmed Bandit Problem”. In: *Machine Learning* 47.2-3 (2002), pp. 235–256. DOI: 10.1023/A:1013689704352.

[29] Nina Mazyavkina et al. “Reinforcement learning for combinatorial optimization: A survey”. In: *Computers & Operations Research* 134 (Oct. 2021). Accessed: 2026-02-28, p. 105400. DOI: 10.1016/j.cor.2021.105400. URL: https://doi.org/10.1016/j.cor.2021.105400.

[30] William L. Hamilton. *Graph Representation Learning*. Vol. 14. Synthesis Lectures on Artificial Intelligence and Machine Learning 3. Pre-publication draft, released with permission. Morgan & Claypool Publishers, 2020, pp. 1–159.

[31] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. New York, NY: Springer Science & Business Media, 2009. DOI: 10.1007/978-0-387-84858-7.

[32] Peter J. A. Cock et al. “Biopython: freely available Python tools for computational biology and bioinformatics”. In: *Bioinformatics* 25.11 (2009), pp. 1422–1423. DOI: 10.1093/bioinformatics/btp163.

[33] Joseph Felsenstein. *Inferring Phylogenies*. Contains the formal specification of the Newick tree format. Sunderland, MA: Sinauer Associates, 2004.

[34] Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. London: Springer, 2013. DOI: 10.1007/978-1-4471-5559-1.

[35] Norman R. Draper and Harry Smith. *Applied Regression Analysis*. Vol. 326. New York: John Wiley & Sons, 1998.

[36] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. 6th. CRC Press, 2020.

[37] Gabriel Cardona, Francesc Rosselló, and Gabriel Valiente. “Comparison of Tree-Child Phylogenetic Networks”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 6.4 (2009), pp. 552–569. DOI: 10.1109/TCBB.2007.70270.

[38] Gabriel Cardona and Francesc Rosselló. “Tripartitions do not always discriminate phylogenetic networks”. In: *Mathematical Biosciences* 230.2 (2011), pp. 118–124. DOI: 10.1016/j.mbs.2011.01.004.