Establishing Existence of Paths between Ranked Phylogenetic Networks

by

Joachim de Bosch Kemper

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 17, 2024 at 13:00.

Student number:	5125103	
Project duration:	March 7, 2024 – July 18, 2024	
Thesis committee:	Dr. ir. L. J. J. van Iersel,	TU Delft, supervisor
	Dr. Y. van Gennip,	TU Delft

TU Delft

Abstract

In the study of phylogenetic trees and -networks, it is frequently desirable to have a measure of the distance between them. A potentially useful definition of this distance is the length (in steps) of the shortest path from one to the other, where this path consists of single rearrangement moves of certain types. While there have recently been developments in methods for finding the shortest path between ranked phylogenetic trees, we will instead be inspecting paths between ranked phylogenetic networks in this work. The primary difference between trees and networks here is in the presence of reticulations in the latter, which are moments where different "branches" come together. More specifically, we work out an algorithm that will provide *a* path between any two binary, level-1, phylogenetic networks with some restrictions, so as to lay the groundwork for work to find shortest paths within this space.

Contents

1 Introduction **3**

2 Preliminaries **4**
2.1 Networks 4
2.1.1 Ranked Networks 5
2.2 Rearrangement Moves 5

3 Connectedness **8**
3.1 Initial Approaches 8
3.2 Definitive Approach 10
3.2.1 Free Tree Nodes 11
3.2.2 Reducing Reticulations 13
3.2.3 Final Algorithm 17

4 Conclusion **18**

5 Discussion **19**
5.1 Interpretation of the Dummy Edge and -Node 19
5.2 Expansion to larger network spaces 19
5.3 Path- and Computational Efficiency 19

1 Introduction

Phylogenetic trees are a useful tool for mapping evolutionary processes, such as the diversification of species, or mutations within genes. Generally, their application is to summarize evolutionary history, so as to allow for more structured and comprehensive analysis. In order to find trees corresponding to a given process, it can be useful - or even necessary - to have a measure for how similar two given trees are. To this end, the distance between can be computed, measured in the number of *rearrangement moves* needed to transform one such tree into another. To concretely define such moves and the trees they transform, the most straightforward avenue to take (within the broader field of mathematics) is to make use of graph theory, describing the trees as a branching sequence of *nodes* connected by (often directed) *edges*.

Generally, we consider these trees to have a single *root node*, from which the rest of the tree descends, while every node where the tree branches out is called a *tree node*. The tree then 'ends' in a set of nodes which we call *leaves*, though we often represent these trees with the root at the top, and their leaves at the bottom - the exact inverse of 'actual' trees. Of course, there are quite a number options with regards to how *precisely* one decides to construct these phylogenetic trees. For example, an aspect that we will be applying in this work - but which is certainly not universal - is to consider a tree to be *ranked*, in essence assigning a set 'order' for the events (corresponding to the nodes within the graph representation). Outside of the choices made regarding such characteristics, there is also the matter of how these are implemented and defined.

In spite of there being quite a lot of variation in how phylogenetic trees are approached, their study has seen a decent amount of work. One noteworthy (and somewhat recent) result by Collienne and Gavryushkin [CG21] provided an algorithm for finding the shortest path between ranked phylogenetic trees under Nearest Neighbour Interchange (NNI) moves, which, when implemented, has a polynomial time complexity. This is noteworthy as, for most other tree spaces, it has previously been shown that the problem of finding shortest paths within them is NP-hard, meaning that the amount of time needed to run a program which solves these problems does not scale with the 'size' of its input - being the trees between which it must find a path - in a polynomial fashion (so the number of computational steps can be bounded by something of the form $c \cdot n^k$, where n is the size of the input, and k is some positive integer). And, if the time complexity of a program is not polynomial, one is rather likely to find that it instead scales *exponentially* with its input size. Needless to say, having a measure of distance that can be found in polynomial time instead is pretty useful, especially if you're inspecting a pair of decently large trees.

Looking more closely at most applications - primarily evolution - however, one may note that these processes at times cannot be entirely described by trees, which only ever 'branch out' over time. For instance, when a species begin to branch out, with different populations developing diverging traits, there is a window during which these different 'subspecies' can, and often do, have populations merge (again), creating new branches. Within a mathematical framework, like what can be used for the trees discussed earlier, such a convergence of different groups can be described by a *reticulation*, or, for the specific 'event' itself, a *reticulation node*. With the addition of such reticulations, we no longer refer to these graphs as trees, but rather *networks*.

For these networks, there are, of course, at least as many ways of precisely describing and characterising them (and the moves between them) as there are for trees; the precise definitions of the elements and aspects of the networks, the moves we have available to change them, and the ways in which those moves exactly interact with the different elements in the network and their definitions, among other things.

In this particular work, we will be inspecting *ranked* networks, so those where every element (corresponding to some 'event' in whatever process the network is representing) is given a unique rank. We also characterise reticulations as, effectively, one single event, in that the entire group of nodes - the two tree nodes splitting off, and the reticulation node where those branches join together - are in certain respects, such as their rank, treated as a single object. Then, we use a total of three moves; Rank Moves to change the rank of elements in our network, Horizontal Subtree (or rather, sub*network*) Prune and Regraft, or HSPR, as the primary means to structurally change the networks, and, to internally change the reticulations, Reticulation Descendant Exchange moves.

The ways in which we precisely define these different parts of our networks will be described in Section 2. After establishing these, we will move on to the core of this work: before one can begin

looking to finding a shortest path between two networks, it must first be shown that *a* path exists for any pair of networks - or, rather, what requirements a pair of networks must satisfy for this to be the case. This is precisely what we will establish in Section 3, as we will first build up a set of restrictions on the networks we consider, and then formulate an algorithm which can be applied to any pair of such networks to find a path between them, thus showing that the space of these networks is connected.

2 Preliminaries

Before we can begin formulating a suitable space of phylogenetic networks, we must first establish what the base properties of the networks we are working with are. Additionally, as we measure the distance between two networks in the length of the path between them, we will define the steps (or 'moves') that these paths will consist of.

2.1 Networks

A network N consists of a set of nodes (or vertices) $V(N)$, connected by directed edges $E(N)$ of the form $(u, v) : u, v \in V(N)$. We will be looking specifically at rooted, directed, binary, level-1 phylogenetic networks. Additionally, we require that these networks are directed acyclic graphs (DAGs). Let us build this up piece-by-piece:

First, the networks we consider are *rooted*, meaning that each network has a single *root node* which serves as an origin of sorts, and will be defined momentarily.

Furthermore, a graph G is called *directed* if each edge in $E(G)$ (denoting the set of edges of G) is directed, i.e. $(u, v) \in E(G)$ does not imply that $(v, u) \in E(G)$.

Then, a graph being a DAG means that it is a directed graph within which no directed cycles are present, a consequence of which is that a natural ordering of vertices exists. It is because of this that we can speak of one node being an *ancestor* or a *descendant* of another; a node u is and ancestor of node v (and v a descendant of u) if a (directed) path from u to v exists. We will, at times, use the terms *parents* and *children* (of a node) as well, which refer to the 'direct' ancestors and descendants of a node, i.e. an ancestor or descendant that is connected to this node by an edge. Additionally, we indicate the parent of a node x by $p(x)$. For an example of this first set of terms, see Figure 1.

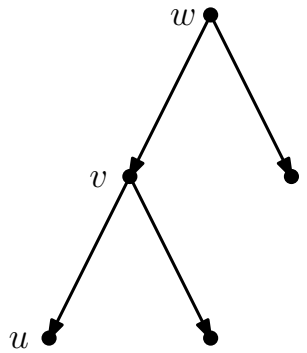

Figure 1: This graph has w as its root, of which u and v are descendants, with only the latter (v) being one of its children. Conversely, node u has ancestors v and w, with its parent $p(u)$ being v.

Next, we will utilize the definition for directed phylogenetic networks used in the work of R. Janssen [Jan21], though we use 'edges' in lieu of 'arcs', and so denote the set thereof (within a network) as E instead of A, and define the root node slightly differently, as we do not use a separate root node before the first tree node (or 'split'):

Definition 2.1. A *directed phylogenetic network* $N = (V, E, l)$ on a set of *taxa* X is a DAG (V, A) labeled with $l : L(N) \to X$, where $L(N)$ denotes the set of leaves of N, with nodes of the following types:

Root Indegree-0, outdegree-2 node, of which the network has exactly one, and for which both children have indegree-1;

Tree node Indegree-1, outdegree-2 node;

Reticulation node Indegree-2, outdegree-1 node;

Leaf Indegree-1, outdegree-0 node.

Incoming edges of reticulation nodes are called *reticulation edges* and incoming edges of leaves are called *leaf edges*.

It should be noted that this definition can be further generalized by instead requiring that the outdegrees of root and tree nodes, and the indegree of reticulation nodes, are equal to k, with $k \geq 2$. We do not do this, as we are working exclusively with *binary* networks.

We will also be referring to the parent tree nodes of reticulation nodes as *reticulation parents*, and use the term *regular* tree nodes to indicate specifically those tree nodes that are neither reticulation parents nor the root of a 'cycle'. Of course, as the actual network is a DAG - requiring that the underlying graph contains no *directed* cycles - these cycles created by the reticulations in our networks are specifically *undirected* cycles; in other words, while the network doesn't 'actually' contain any cycles, it would if its edges were not directed. Additionally, when we speak of a *reticulation grouping*, this refers to a reticulation node and both of its parent nodes. We show an example of a basic network in Figure 2, in addition to ranks and cherries, which will be discussed later.

Lastly, we also take a connected pair of definitions from Rearranging Phylogenetic Networks by R. Janssen [Jan21] for what a *level-1* network entails:

Definition 2.2. A *blob* of a phylogenetic network $N = (V, E, l)$ is the network formed by a biconnected component of the underlying DAG $D = (V, E)$ of N, together with its outgoing edges, where each leaf l is labelled with the set of leaves of N below the node corresponding to l in N.

Note that a *biconnected graph* (or, here, component) is a connected graph that cannot be divided into separate graphs by removing any single vertex (and its connected edges). Within our networks, the only such biconnected components are centered around its reticulations.

Definition 2.3. A network N is called *level-1* if every blob of N has a reticulation number of at most 1. More generally, the *level* of a network N is equal to the maximum reticulation number in the blobs of N.

2.1.1 Ranked Networks

As pointed out earlier, the fact that all of the networks that we're working with are DAGs means that the nodes in these networks can be ordered. To formalize this, we will introduce the concept of *rank* and, by applying this, *ranked networks*.

We assign a unique rank from 1 to $n-1$ to each regular tree node and reticulation grouping, and assign rank n to the root node, with $n = 1 + (t - r)$, where t and r denote the total number of tree nodes and reticulation nodes, respectively (and the $+1$ is to account for the root node, of course). Lastly, we say that the rank of each leaf node is 0.

As one might infer from the root node being assigned the highest 'available' rank n, and leaves being assigned the lowest, we assign rank in ascending order, i.e. a node must have higher rank than any of its descendants.

Formally, we use a characterization for rank in trees as described in Ranked Subtree Prune and Regraft by Collienne et al. [CWG24], modified for networks by accounting for reticulations:

Definition 2.4. For a given unranked network $N_u = (V, E, l)$, we construct a function $rank : V \to \{0, 1, \ldots, n-1, n\}$ such that: (i) $\text{rank}(u) = 0$ if and only if u is a leaf, (ii) for any two internal nodes $u \neq v$ $\text{rank}(u) = \text{rank}(v)$ if and only if u and v are part of the same reticulation grouping, and (iii) $\text{rank}(u) < \text{rank}(v)$ if v is on any path from the root of N_u to u. A pair $N = (N_u, \text{rank})$ is called a *ranked phylogenetic network*.

Going forwards, for the sake of simplicity, when we say either *ranked network* or simply *network* this refers to a *ranked, directed, binary, level-1 (phylogenetic) network*, unless stated otherwise.

Lastly, within these networks, we will later discuss *cherries*, the definition of which we loosely take from the work of Bernardini et al. [BvIJ23]: an ordered pair of leaves (a, b) of a network N, with $a \neq b$, is a *cherry* if a and b have the same parent.

2.2 Rearrangement Moves

Before we can begin working out a measure of distance between networks, we must define the ways in which we can move between different networks so as to create paths from one to another. We will be outlining a total of three different moves. An example of a path including all three moves is given in Figure 3.

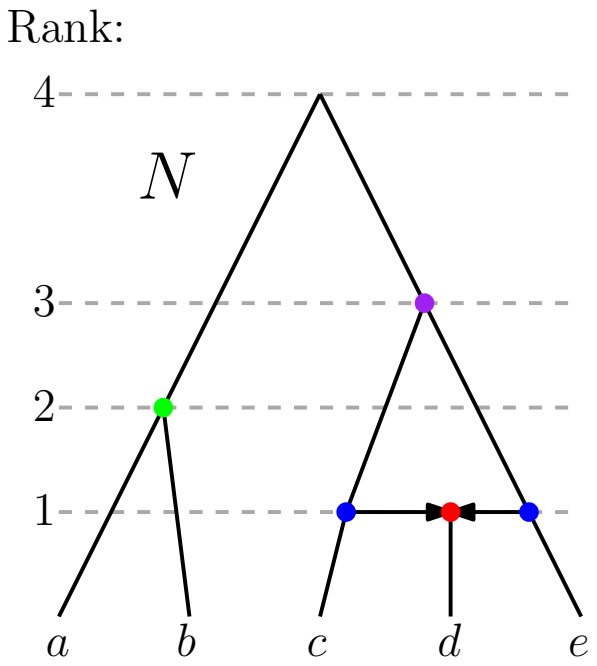

Figure 2: In this network N, we have a single reticulation grouping, consisting of the blue and red nodes at rank 1. The root of the corresponding cycle is the node at rank 3, coloured in purple. The network has four tree nodes, being the green, purple, and two blue nodes, but only a single regular tree node - the green node at rank 2, which is also the parent of a cherry. The blue nodes are reticulation parents, and (as mentioned earlier) the purple node is the root of a cycle.

Definition 2.5. Performing a *rank move* entails exchanging the rank of an unconnected pair of regular tree nodes, reticulation groupings, or a regular tree node and a reticulation grouping.

Of course, for a reticulation grouping, all three nodes that it consists of have to not be connected to the other node(s) we exchange their rank with.

Definition 2.6. A *Horizontal Subtree Prune and Regraft* or HSPR move moves the 'tail' of one of the outgoing edges of a tree node onto another edge, such that a new tree node is created at the same rank as the original. As such, it is required that the edge that is 'bisected' by this move originates from a node of higher rank that the 'moved' node.
Notably, for reticulation parents, only the edge towards the corresponding reticulation node can be moved. Additionally, the tail of one of the incoming edges of a reticulation node cannot be moved to the same 'location' as the other incoming edge's origin, as the tree node at this point is at the same rank as the moved node.

These moves can also be seen/referred to as *(horizontal) tail moves*. We often refer to these moves as moving the node from which the edge originates, with an additional qualifier to indicate which outgoing edge is moved (e.g. "We perform an HSPR move on tree node t with descendant x", indicating that t's outgoing edge which (eventually) leads to a node x descending from t has its 'tail' moved).

Definition 2.7. In a *Reticulation Descendant Exchange* or RDE move, we interchange any two of the three subtrees descending from the nodes in a reticulation grouping.

Lastly, we consider each of these moves to be *valid* if the network created through these moves is at most level-1. Correspondingly, a path between two networks is *valid* if all of the moves between its steps are valid.

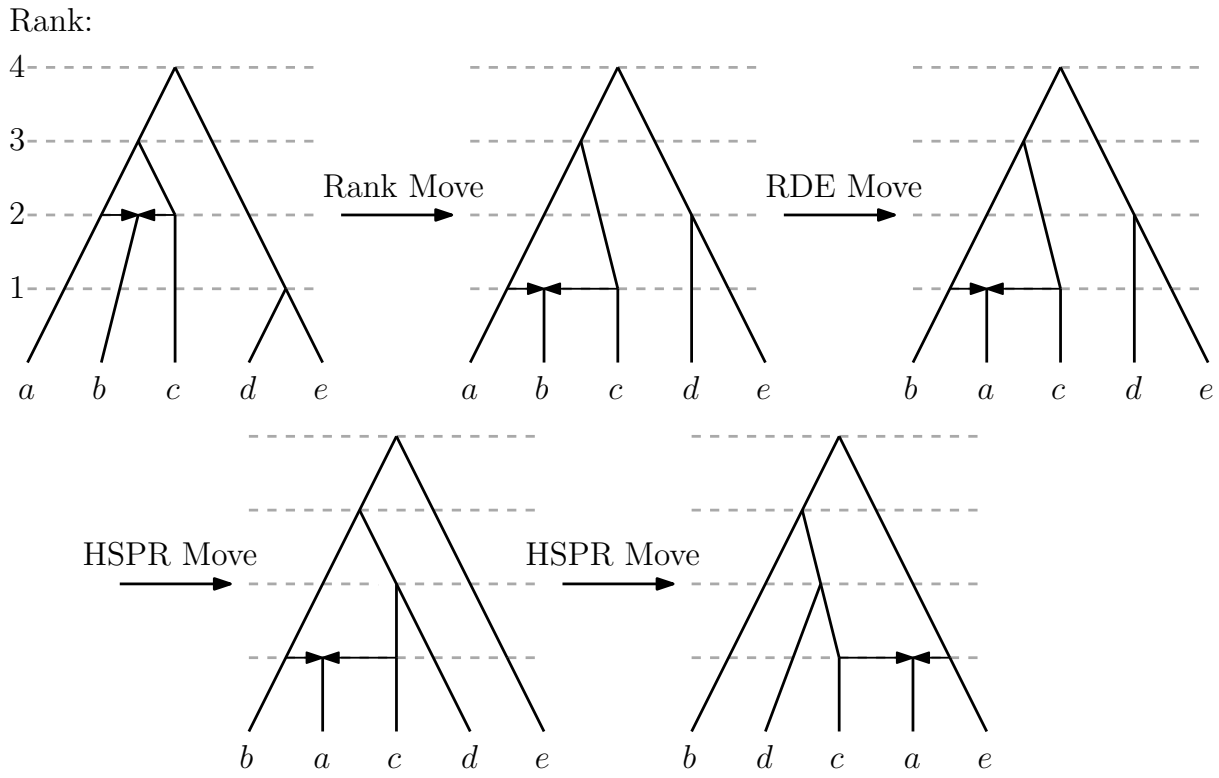

Figure 3: An example of a path including each move, on a network with leaves a through e. For the HSPR moves, the first moves the tail of the incoming edge of d, while the second moves one of the reticulation parents - specifically the one whose other descendant is leaf b - to the edge between the root and e.

3 Connectedness

Before we can consider the *shortest* paths between different networks - so as to establish a measure of distance between them -, we must first verify whether a valid path exists in the first place. To be more precise, we will try to find an as large as possible space of networks which we can show to be *connected*, indicating that valid paths exist between any two networks in this space. The initial conjectures made in the process of finding such a space turned out to not be sufficient, though we will still lay out these earlier iterations of our final result - with their negations being described in Propositions 3.1 and 3.2 - and show *why*, precisely, these requirements do not suffice.

3.1 Initial Approaches

Proposition 3.1. *There exist ranked networks N, M with an equal number of internal nodes and reticulation nodes, on the same set of leaves L, such that there is no sequence of valid moves of types*

- *Rank moves;*
- *Horizontal SPR moves;*
- *Reticulation Descendant Exchange moves*

to turn N into M.

Now, it should be noted that such a path *does* exist for a great many networks, such as those shown in Figure 4.

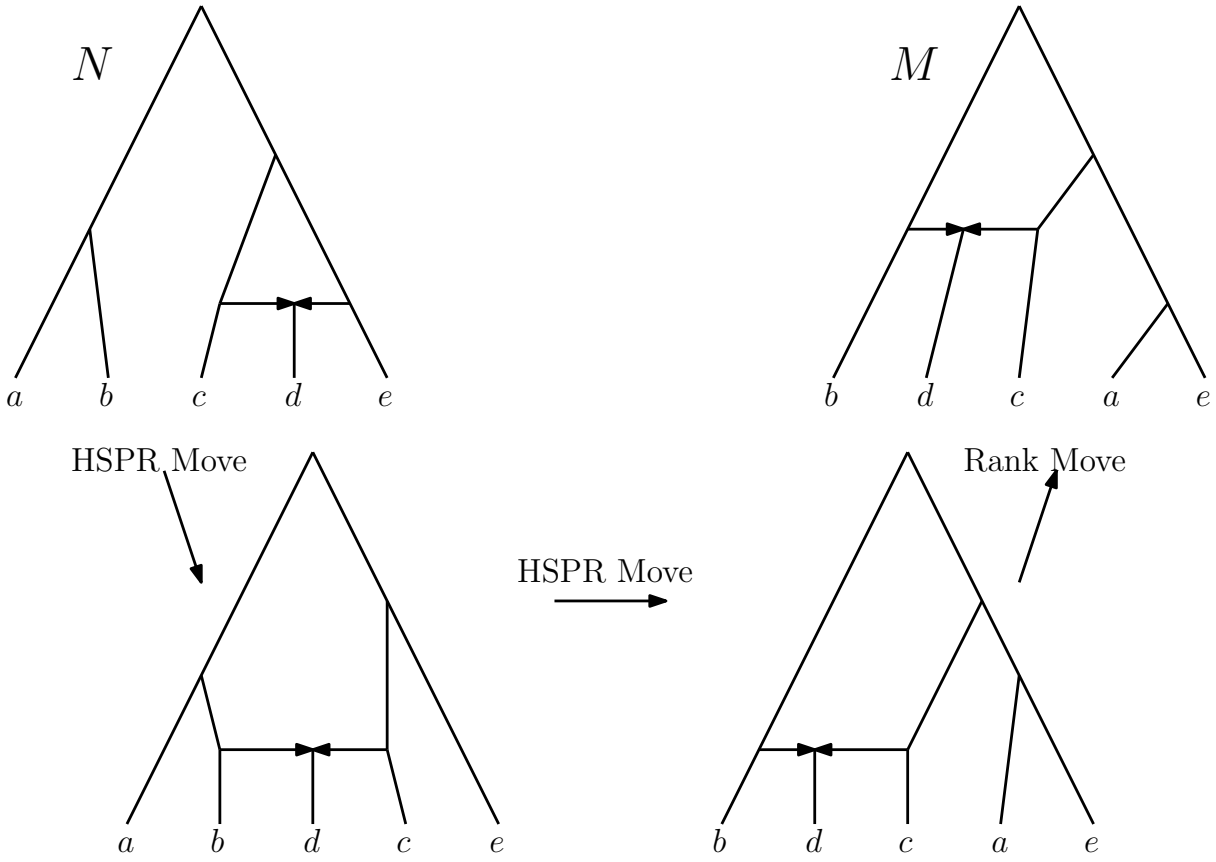

Figure 4: Here, networks N and M are connected by a path of length 3.

However, as Proposition 3.1 states, this is not universal, as is shown by cases such as the one shown in Figure 5.

As the problem in cases such as that seems to be that precisely one of the networks N, M has a reticulation grouping at the highest rank (below the root), one may initially suspect (or rather hope)

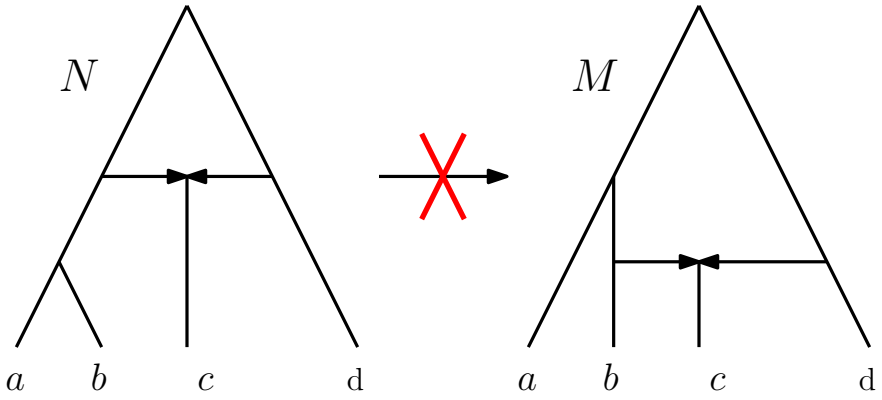

Figure 5: There is no path between the networks N and M shown above. Specifically, we 'need' to perform a rank move on the reticulation grouping and the parent tree node of a and b, but there are no edges 'available' for said tree node which are not connected to the reticulation grouping, and so no such move can be made.

that excluding such cases would suffice, but, unfortunately, there are other structures that make it impossible to create paths using the described moves and properties, as shown in Figure 6.

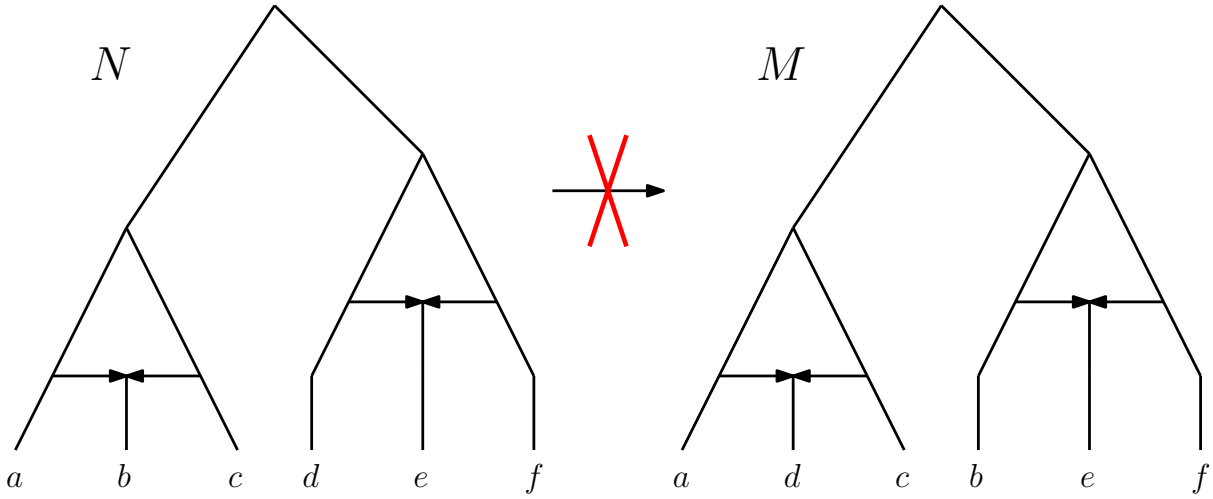

Figure 6: The reason that we cannot find a path between these networks is that, in order to move either b or d from one reticulation to another, the only available option is (something to the effect of) connecting the left reticulation to d's incoming edge using an HSPR move. Doing this would, however, turn the network into a level-2 network and, as there are no alternative paths, this means that no path exists *within the space of level-1 networks.*

As such, we looked to a broader extension to our initial set of restrictions. Because the roadblocks we have run into generally come down to not being able to move elements in or around reticulations, we sought to define a class of networks which give us the means to, in a sense, bypass these structures. However, as shown in Proposition 3.2, the expansion we decided upon did not yet suffice.

Proposition 3.2. *There exist networks N, M which satisfy the requirements laid out in Proposition 3.1, and which both have a leaf with the root node as parent, for which there is no path, consisting of Rank, HSPR, and RDE moves, from N to M.*

It should be noted that any given network can be easily modified to satisfy the property introduced above, by adding a new parent node for the original root node, with this new root node's other child being a likewise new leaf. Consequently, the original root node becomes a tree node. Essentially, this means that, because for any networks N', M' a network satisfying this requirements N, M exist (and can easily be found) , we could consider the distance between N and M - which, had our conjecture

that a path between such networks can be found been true, exists - to be analogous to that between N' and M'. Going forwards, we will, at times, refer to the newly created leaf as the *dummy leaf*, and the edge from the root to this leaf as the *dummy edge*.

As stated in the Proposition, however, the addition of a dummy edge- and leaf does not (yet) suffice, as there are sets of networks such as those shown in Figure 7 for which no path exists. The core problem with cases like this is that issues arise when we are faced with networks consisting exclusively of 'series' of 4-cycles, as we cannot move the descendants of these reticulations from or to any cycles that descend from reticulations (without stepping outside of the space of level-1 networks, at least).

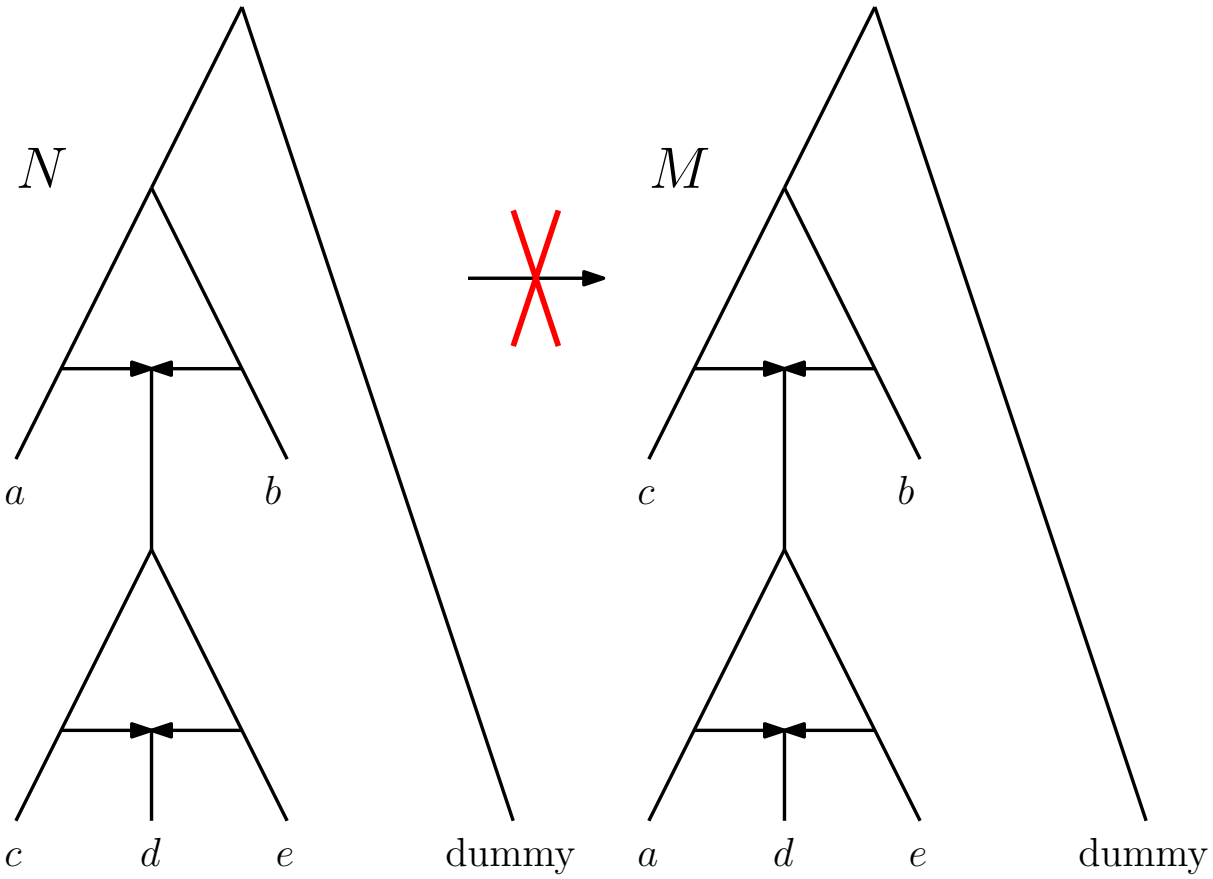

Figure 7: Similarly to Figure 6, there is no valid path between these networks N, M because the only possible paths stray outside of the space of level-1 networks. Specifically, in order to move leaf a into the lower reticulation, the only available avenue to take is to, at some point, use an HSPR move to connect one of the reticulation edges of this reticulation to either the dummy edge, or an incoming edge of a child of a node in a higher-ranked reticulation grouping. However, such a move would not be valid, as this would create a cycle which (partially) overlaps with both preexisting cycles, which puts them in the same blob, making it so the resulting network is not level-1.

3.2 Definitive Approach

In order to remedy the remaining issue with Proposition 3.2's addition, we add an additional condition, so as to exclude specifically the set of networks discussed at the end of the previous subsection, by way of requiring that $|L(N)| = |L(M)| \geq 2r + 3$, which, together with the requirements we laid out in the earlier propositions, gives the following theorem:

Theorem 3.3. *Let N, M be rooted, directed, binary, level-1, ranked phylogenetic networks with an equal number of internal nodes and reticulation nodes, on the same set of leaves L, such that $|L| \geq 2r + 3$.*

Additionally, let both N and M have a leaf with the root node as parent.
Then there exists a sequence of valid moves of types

- *Rank moves;*
- *Horizontal SPR moves;*
- *Reticulation descendant exchange moves*

to turn N into M.

3.2.1 Free Tree Nodes

The goal of this subsection is to obtain a regular tree node which has a leaf as (at least) one of its children: such a structure can be easily moved throughout the networks (especially using the dummy edge) so as to store and 'transport' descendants to and from reticulations. Due to the flexibility they present, we will refer to such nodes using the following definition:

Definition 3.1. If a tree node has a leaf as at least one of its children, this tree node is called a *free tree node.*

To begin, however, we must verify that this condition actually guarantees that networks which satisfy it have at least one regular tree node, which will later be shown to allow for the creation of a free tree node. We formalise the existence of such a prerequisite node in the following lemma:

Lemma 3.4. *Let N be a network satisfying the conditions of Theorem 3.3. Then N contains at least one regular tree node.*

Proof. Assume that, outside of the root and its leaves $L(N)$, N consists exclusively of its reticulation nodes, reticulation parents, and roots of cycles.

We note that (again excluding the network's root and leaves) all three types of nodes cannot exist within the network independently of one another; each reticulation node necessitates precisely two corresponding reticulation parents, which each must be the parent of precisely one reticulation node - as, if one tree node is in two different reticulation groupings (by being the parent of two reticulation nodes), then those two reticulations are within the same blob, and so the network would not be level-1. Lastly, each root of a cycle must be the parent of a pair of reticulation parents which are connected to the same reticulation node. This is because there can be no regular tree nodes, nor other root nodes or reticulation parents between these roots and a pair of reticulation parents descending from it. The former would violate our assumption, and the latter is not possible within the space of level-1 networks, as then multiple cycles would share at least one node, which would ensure that they are within the same blob.

Now, as there are no nodes between the roots of cycles within N and the first reticulation parents descending from them, each 4-cycle, as a structure, effectively has an indegree of 1 - the indegree of its root - and an outdegree of 3 - as the two reticulation parents and the reticulation node each have an outdegree of 1. Furthermore, such cycles are, as shown earlier, the only structures that exist within N, save for its root and leaves, in addition to being disjoint due to N being level-1. With this, we can see that the total number of leaves $|L(N)|$ of this network can be directly determined by the number of cycles (which is equal to the number of reticulations r): $|L(N)|$ is equal to the sum of the outdegrees, minus the sum of the indegrees of all elements within the network, except for its leaves. This is because the total sum of in- and outdegrees is equal, and as the leaves each have indegree-1 and outdegree-0, the difference between the in- and outdegrees of elements other than the leaves is equal to the number of leaves. The root has indegree-0, outdegree-2, while each 4-cycle (of which there are r within the network) has indegree-1, outdegree-3. As such, we get that the sum of indegrees is equal to r, the sum of outdegrees is equal to $3r + 2$, and so we have that $|L(N)| = (3r + 2) - r = 2r + 2$

This is, however, a clear contradiction of the condition, set by Theorem 3.3, that $|L(N)| \geq 2r + 3$, and so N must contain a tree node that is neither the root of a cycle nor a reticulation parent. □

Next, we will establish that, given that Lemma 3.4 holds, we can use the tree node described within it to create a free tree node using the moves outlined in Theorem 3.3, by way of the steps outlined in Algorithm 1.

Algorithm 1: Creating a free tree node

Data: t : tree node as described in Lemma 3.4, within a network satisfying Theorem 3.3
Result: A free tree node t (i.e. such that t has at least one leaf child)
if *t has no leaf child* **then**
  x, y : disjoint descendant subnetworks of t;
  HSPR move t and descendant x to the dummy edge;
  Rank move t down until it is of lower rank than a node which has a leaf child;
  HSPR move t (with x attached) to bisect the incoming edge of a leaf ; `/* Available due to the previous step */`
end

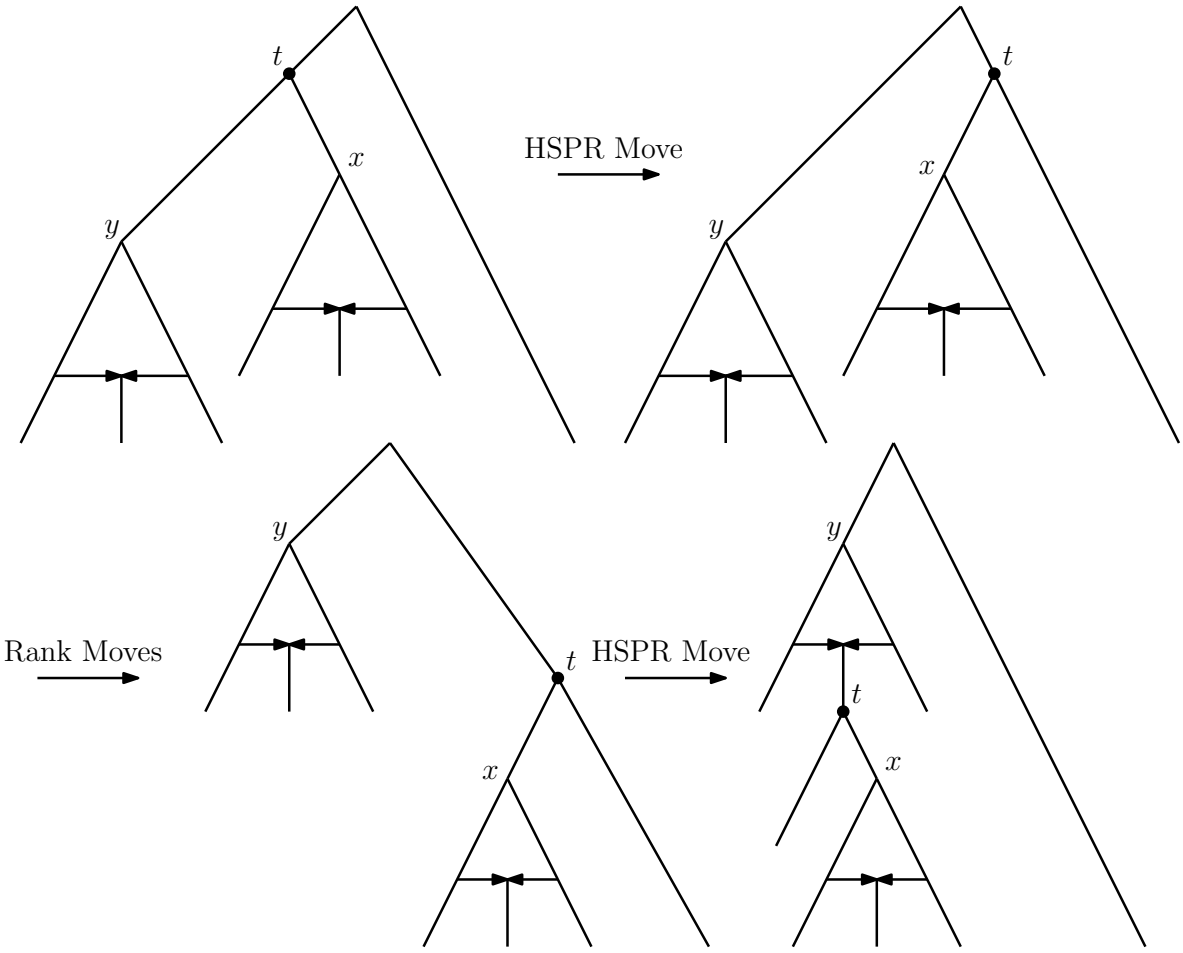

Figure 8: Here, we move subnetwork x through the dummy edge so that it descends from subnetwork y, as per Algorithm 8 As a consequence, we obtain a tree node t which has a leaf node as one of its children.

A few notes on the Algorithm 1 are in order: first, the end condition of the algorithm is that t has at least one child which is a leaf. As such, the alternative to the first condition set is that we are already 'done', hence why the algorithm ends if that condition is not met. The alternative endpoint is after t is moved to bisect a leaf's incoming edge, as then t's two descendants are a subnetwork x and some leaf, meaning that t is a free tree node (as desired).

Also, when we perform an HSPR move to remove a tree node from the dummy edge, we implicitly move the edge *not* leading to the dummy edge. This is done to both avoid the dummy node ending up somewhere somewhere within the network other than being a child of the root (which would necessitate later removal), and to maintain the existence of the dummy edge for later steps.

Last is that x, y, which are the subnetworks (where both are not leaves, due to the case they're defined in) descending from free tree node t, are disjointed due to the properties of t, and the level-1 condition. For, as t is not the root of a cycle, the two 'branches' descending from it cannot be connected (by a reticulation); if a reticulation between t's descendant subnetworks exists, then t must be its root. A reticulation's root is the lowest common ancestor of its reticulation parents, so (as t is *an* ancestor of both nodes, the root cannot be of higher rank, and if it is a node of lower rank, then it being an ancestor of two nodes in the different descendant subnetworks requires that these subnetworks are already connected (which, due to the structure of the networks we're working with, requires a reticulation 'between' nodes from these two subnetworks).

If we consider the highest-ranked reticulation grouping, the only possible root for its cycle is evidently t, which would contradict its properties given by Lemma 3.4. As this implies that the highest-ranked reticulation (connecting the descendant subnetworks of t) has no valid possible root, it cannot exist. It then follows that, as there being any reticulations implies that there is a highest-ranked reticulation (which leads to a contradiction by the above), we conclude that no such reticulations exist, and so subnetworks x and y are disjoint.

For an example of Algorithm 1 being applied, see Figure 8.

3.2.2 Reducing Reticulations

Now that we have obtained a free tree node, we can look towards using it to find a path from N to M. To this end, we will outline a series of algorithms, which, taken together, serve to effectively remove all reticulations from both networks' structures by way of contraction, so as to reduce the remaining problem to finding a path between two ranked phylogenetic trees (which, as will be mentioned later, has been established in previous works).

With the free tree node, we first look to create identical, simple reticulations within both the target- and original networks. Specifically, we define a *simple cycle* as a 4-cycle, for which all nodes within the corresponding reticulation grouping only have leaves as children (other than the reticulation node, for the two reticulation parents within the grouping). So, for such a cycle, there are no regular tree nodes between the reticulation parents within it and the root of the cycle. To make this step more concrete, we compile it into the following lemma:

Lemma 3.5. *If a network N contains at least one reticulation and a free tree node, there is a sequence of rank, HSPR, and RDE moves to create a four cycle consisting of a reticulation grouping and its reticulation parents' lowest common ancestor, and such that each node in the reticulation grouping has a leaf as its (only) child. Furthermore, this cycle can be constructed such that these leaves can be any (chosen) three leaves $\in L(N)$.*

We will prove this lemma by providing methods to create such a 4-cycle in a valid network. To make this more comprehensive, we split this process into two steps, in line with the two somewhat separate parts of Lemma 3.5; the purpose of the first algorithm - Algorithm 2 - is to create a simple cycle, while the second - Algorithm 3 - will demonstrate how we can have the corresponding reticulation grouping's descendants be exchanged with any given leaf in $L(N)$. One final thing to note before we inspect Algorithm 2 is that we specifically modify the *lowest cycle* in the network N with it, which is the cycle corresponding to the lowest-ranked reticulation grouping in the network.

Again, let us note a few things about Algorithm 2. First, we move all 'excess tree nodes' either descending or branching out from the cycle to the root node's incoming edge because, while it generally doesn't matter where we 'put' these tree nodes and their descendants, doing this can generally be

Algorithm 2: Creating a simple cycle

Data: Lowest cycle C of N, with cycle root r and reticulation grouping R
Result: Simple cycle C, i.e. C is a 4-cycle with only leaf descendants
for *regular tree nodes x in C descending from r, with Rank(x) > Rank(R)* **do**
    HSPR move x to the dummy edge (with the greatest number of descendants);
    Rank move x up to Rank$(r) + 1$;
    HSPR move x to r's incoming edge
end
for *nodes $u \in R$* **do**
    for *tree nodes x descending from u* **do**
        HSPR move x to the dummy edge (with the greatest number of descendants);
        Rank move x up to Rank$(r) + 1$;
        HSPR move x to r's incoming edge
    end
end

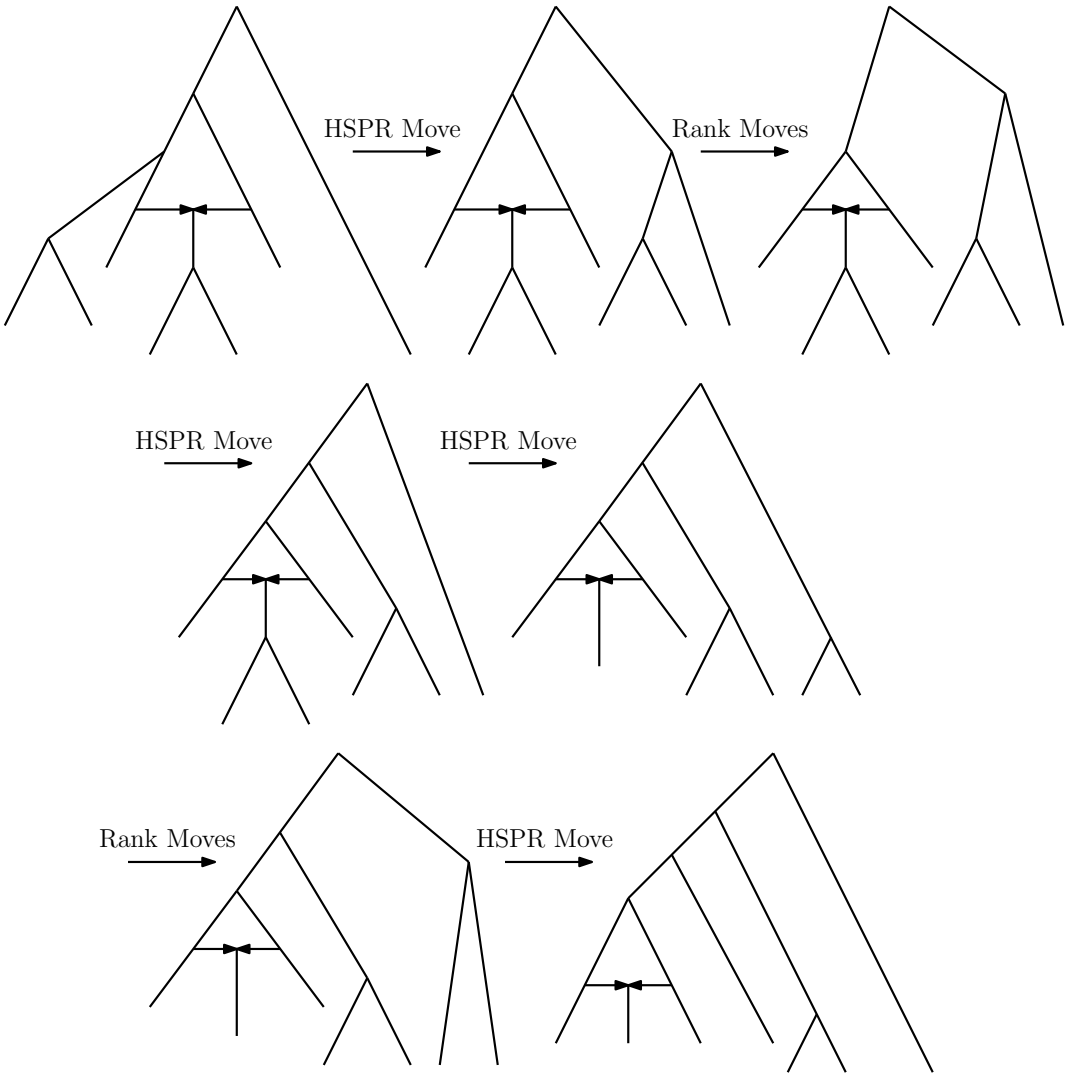

Figure 9: As per Algorithm 2, we turn the above network's only cycle into a simple cycle. We first remove the tree node 'within' the cycle, and then move the one 'excess leaf' out from the descendants of the reticulation grouping.

expected to lower the number of steps needed for further applications of this algorithm; if this incoming edge of r does not have a node within a node within another reticulation grouping as its parent (before applying this algorithm), then later applications will (or, at least, can) move the tree node it originates from entirely, moving whatever number of tree nodes were moved onto this edge as a group, rather than moving each further up in rank individually.

Additionally, the end condition for this algorithm is that all regular tree nodes originating from the root of the cycle are removed, as this leaves us with a 4-cycle (only consisting of the root and the three nodes in the reticulation grouping; any other nodes would, as this is a level-1 network, be tree nodes branching off, which we have removed), with only leaves as descendants of its reticulation grouping (as, again, removing all tree nodes leaves only leaves; as this is the lowest cycle in the network, there can be no reticulations descending from these nodes).

An example of Algorithm 2 being applied is shown in Figure 9.

With a simple cycle, we move on to getting any three desired values into the leaves descending from it - or, rather, showing that we can move any given value into them; if need be, this can then be done three times if necessary. We do this to ensure that we can create the same 4-cycle from both the original and target network (in the broader Theorem), so that we can later contract them (as will be discussed after we finish with Algorithm 3).

Algorithm 3: Moving a value (leaf) into a simple cycle

Data: Simple cycle C with reticulation grouping R and root r, desired value/leaf x, undesired value/leaf y descending from R, and free tree node t
Result: Simple cycle C s.t. x descends from it
case *x is the child of a regular tree node* **do**
  HSPR move x's incoming edge to the dummy edge
end
case *x is the child of a node within a reticulation grouping $S : S \neq R$* **do**
  HSPR move t to the dummy edge, with a child leaf;
  Rank move t to rank Rank$(S + 1)$;
  HSPR move t to the incoming edge of x;
  HSPR move t with x to the dummy edge
end
; `/* Note that, in either case (which accounts for all 'locations' of` x`), we now have` x `descending from a tree node` $p(x)$ `on the dummy edge */`
Rank move $p(x)$ to rank Rank$(R) - 1$;
HSPR move $p(x)$ to the incoming edge of y;
HSPR move $p(y)$ (now the same as $p(x)$) to the dummy edge with y;
Rank move $p(y)$ to Rank$(r) + 1$;
HSPR move $p(y)$ to r's incoming edge

For this algorithm, note that we again move certain 'excess tree nodes' to right above the root of our simple cycle. This is partially done for the same reasons as outlined for Algorithm 2, though here there is somewhat lesser need for efficiency. Notably, whatever tree node we 'store' here is a free tree node, as it has a leaf child n, and so applying this algorithm will not 'break' the condition of having a free tree node which we have set for later applications.

We can also note that none of these moves are at risk of breaking the level-1 quality of our network: most moves pose no risk, as we are moving around regular tree nodes with leaf children, but the HSPR moves applied to the edges within reticulation groupings also pose no risk, as these are only ever moved to either the incoming edges of leaves, or edges which they had previously bisected (without the descendants of said edges having been changed in the meantime).

An example for Algorithm 3 can be seen in Figure 10.

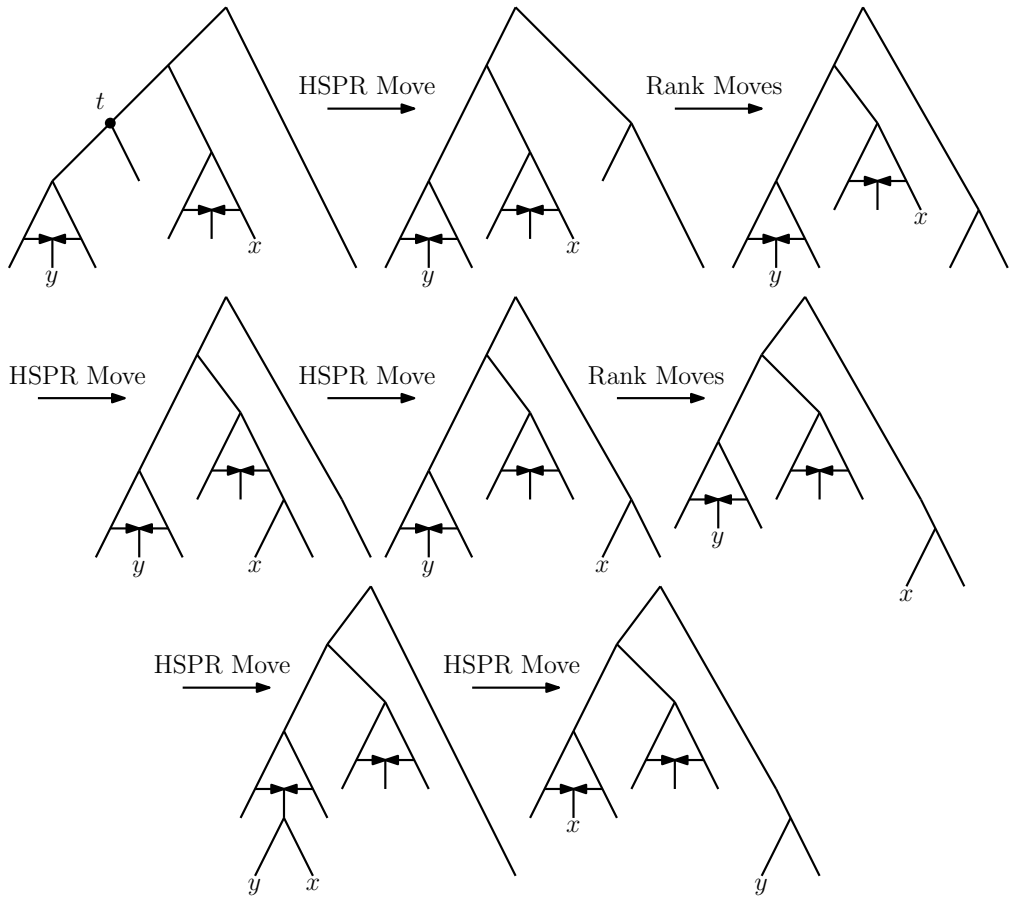

Figure 10: We first replace x and its incoming edge with the child of free tree node t (and its incoming edge), after which the original 'location ' of x no longer matters, as mentioned in Algorithm 3. We then, using the same method, replace the leaf y descending from the lowest (simple) cycle with x. Note that the last two steps of Algorithm 3 are omitted (being those regarding the 'storage ' of the unneeded leaf y), which we have done to somewhat limit the scope of this figure.

3.2.3 Final Algorithm

What we now have available is a method to isolate a reticulation within a given network. Using this, we now, in essence, aim to reduce our networks to trees. We do this by, after we have created a simple reticulation (within both our original and target networks - N and M, respectively), contracting this 4-cycle into a leaf, and then repeating this process until there are no reticulations left in both networks. Notably, any manipulation of this leaf from this point forward cannot cause us to step out of the space of level-1 networks, as, if we undo this 'substitution' at any point, we can see that the newly substituted subnetwork constitutes a blob, as removing the incoming edge of the now-replaced leaf separates it completely from the rest of the network. Consequently, so long as the network is level-1 before undoing the substitution, it will remain as such, because adding a new blob (with a reticulation number of one) does not change the reticulation number of any others, and so does not change the network's level either.

Finally, once we have a pair of trees corresponding, through substitution of simple reticulations, to modified networks obtained from N and M, we note that connectivity between ranked phylogenetic trees under rank- and HSPR moves has already been established earlier this year by Colliene, Whidden and Gavryushkin [CWG24]. However, the algorithm laid out in that work primarily uses cluster notation - a method of describing and characterising trees that is not entirely applicable to networks. This is because reticulations allow for *partial* overlap between clusters, as opposed to the still 'acceptable'/usable case of one cluster being contained in another. As such, we will not be reiterating this algorithm here - referred to as "RSPR Tree Algorithm" later -, but instead refer to it in its entirety, as we do with the algorithms laid out earlier, within the broader, 'entire' algorithm, Algorithm 4.

Algorithm 4: Finding a Path between networks

Data: Networks N, M satisfying the conditions of Theorem 3.3, with r reticulations
Result: Paths $N \to N', N' \to M', M' \to M$, which together form a path $N \to M$
if *N does not contain a free tree node* **then**
  Apply Algorithm 1 to N, recording steps taken as part of path $N \to N'$; `/* This algorithm can always be applied, as its requirements are met as per Lemma 3.4 */`
end
if *M does not contain a free tree node* **then**
  Apply Algorithm 1 to M, recording steps taken as part of path $M \to M'$
end
$i \leftarrow 1$;
$N' \leftarrow N$;
$M' \leftarrow M$;
while $i \leq r$ **do**
  Apply Algorithm 2 to N', recording steps taken as part of path $N \to N'$;
  Apply Algorithm 2 to M', recording steps taken as part of path $M \to M'$;
  for *leaf n descending from the lowest cycle of N'* **do**
    Apply Algorithm 3 to M' on its lowest cycle, with desired leaf n, recording steps taken as part of path $M \to M'$; `/* As stated in Algorithm 3, recording previous desired values, and ensuring that these are not replaced */`
  end
  RDE moves to make the order of leaves of the lowest cycle of M' identical to that of N';
  Contract (identical) lowest cycles in N', M' into leaves;
  $i \leftarrow i + 1$
end
; `/* this block contracts precisely` r `cycles into leaves, and so, as there are no available means to create new cycles or reticulations,` N' `and` M' `are now trees, for they contain no more reticulations */`
Apply the RSPR Tree Algorithm [CWG24] to N', M' to obtain a path $N' \to M'$;
Reverse the path $M \to M'$ to obtain a path $M' \to M$;
Join paths $N \to N', N' \to M', M' \to M$ to obtain a path $N \to M$

One thing to immediately note about Algorithm 4 is the step where we reverse the path $M \to M'$. We can do this reliably as all three moves we use can easily be 'undone' using the same type of move: if two objects x, y have a rank move applied to them, this swapping of ranks can be repeated if no other moves have been done afterwards. For RDE moves, any two of the outgoing edges of the reticulation group's nodes can be exchanged freely, and so, again, these moves can always be reversed. Lastly, after any HSPR move is done, whatever edge the moved node / edge tail previously bisected still originates from a node of higher rank than the moved node, and lead to a node of lower rank, and so the affected tree node can be moved back to its previous position.

Also, the contraction of simple cycles is naturally left out of the path $N \to M$. It is primarily done to be able to apply the aforementioned RSPR Tree Algorithm, but, as noted shortly before said algorithm was discussed, as this substitution does not influence or 'prevent' any moves done without it.

Lastly, we note that the contraction of a simple reticulation into a leaf effectively decreases the total number of leaves in the networks by two (as there are three leaves - those descending from the reticulation grouping - lost, and one 'gained' through the substitution). As the number of reticulations in the modified networks decreases by one through this action, we can see that, as we have that $|L(N)| \geq 2r + 3$, after one application of Algorithm 2 (and Algorithm 3, if applicable) followed by contraction of the created simple cycle to create a modified network N', we have that $|L(N')| = |L(N)| - 2 \geq (2r + 3) - 2 = 2(r - 1) + 3$, and so the condition introduced in Theorem 3.3 also holds for the new network.

In conclusion, Algorithm 4 can be applied to any pair of networks which satisfy the requirements laid out in Theorem 3.3. As such, the algorithm and its global applicability within the corresponding space constitute a proof of Theorem 3.3.

4 Conclusion

To comprehensively recap what we have now proven, let us recall Theorem 3.3 once again:

Theorem 3.3. *Let N, M be rooted, directed, binary, level-1, ranked phylogenetic networks with an equal number of internal nodes and reticulation nodes, on the same set of leaves L, such that $|L| \geq 2r + 3$. Additionally, let both N and M have a leaf with the root node as parent.*
Then there exists a sequence of valid moves of types

- *Rank moves;*
- *Horizontal SPR moves;*
- *Reticulation Descendant Exchange moves*

to turn N into M.

With this theorem, we have established connectivity for the space of level-1 networks for which the root has a leaf as one of its children. As any given network has an easily found network that satisfies the latter property - by way of taking the original network and adding an additional node whose children are the original network's root and a new 'dummy' leaf - we can use paths between such modified networks in considerations of the degree structural similarity of the original networks. Similarly, a prospective *shortest* path between modified networks can also be used in considerations of distance between the original networks.

By establishing connectivity for the described space of networks, we now have a foundation from which work can be done towards formulating either algorithms for finding shortest paths, or heuristics for finding a 'sufficiently short' path. While Algorithm 4 does not necessarily produce a particularly short or efficient path (as further discussed in section 5.3), it remains useful as being able to state/assume the existence of a path between two networks without having to take the time to show this from the ground up.

5 Discussion

5.1 Interpretation of the Dummy Edge and -Node

To begin, let us discuss "dummy edge" introduced in Proposition 3.2. Outside of the practical note made in the section this was introduced - that any pair of networks could be easily modified to satisfy it by adding a new root node, in combination with a single leaf -, we can consider how valid it is to view the distance between the modified networks as analogous to that between the original, unmodified ones.

Let us consider that, in many applications of phylogenetic networks (chief among them the mapping of evolutionary processes) we rarely, if ever, actually work with a 'whole' network, but instead look into an often rather small part of a larger one; frequently, the root of our network(s) 'actually' has a great many ancestors, and leaves are stand-ins for subnetworks. A natural consequence of the point regarding the root in specific is that there are often elements of the greater network parallel to the snippet we inspect.

This, in turn, provides an element within these applications that can function almost identically to the dummy edge and -leaf: moving a node to the dummy edge is, simply put, akin to moving it anywhere outside of the inspected part of the network. This also does not conflict with the only move we use while 'on' the dummy edge, being rank moves, which only require that the elements whose ranks we swap are not connected to one another.

5.2 Expansion to larger network spaces

Though the main issue with the requirement added in Proposition 3.2 was the inability to form a path without straying outside of the space of level-1 networks, expanding the space of networks we consider to level-2 (or higher) networks does not remedy this. We can see this if we inspect the particular counterexample in Figure 7, and add an additional reticulation descending from the lowest one in the figure in both networks. If we then need to move a leaf to or from the new lowest reticulation, we create a level-*3* network. With regards to letting go of *any* restriction on the level of the networks, while it may be possible to establish connectivity of that space, the number and complication of structures that may need to be accounted for within this space puts such considerations quite far beyond the scope of this work.

One other possible direction for expansion from this work is to expand Theorem 3.3 to higher-level networks. Combined with a dummy edge, the condition for there to be at least $2r + 3$ leaves could prove sufficient for the space of general level-k networks. To begin, suppose we generalise the blobs of level-k networks (specifically the blobs with a reticulation number of k) as containing k *nested reticulations*, i.e. k reticulations which have the same lowest common ancestor - or, rather, suppose we can find methods to transform blobs into this structure. Then, if we again ensure that we have identical (nested) reticulations in both the original and target networks, contracting such a structure containing n reticulations (with $n \leq k$) will result in the number of reticulations r obviously decreasing by n, while the number of leaves decreases by $n + 1$ (as before, while $n - 2$ leaves are deleted, the structure is replaced by one added leaf). As such, if $|L(N)| \geq 2r + 3$ holds for the original network, then $|L(N)| - (n + 1) \geq 2r + 3 - 2n = 2(r - n) + 3$ (because, for $n \in \mathbb{N} : n \geq 1$, $2n \geq n + 1$) holds for the modified network.

Furthermore, a cursory exploration of these spaces seems promising with regards to said condition - $|L(N)| \geq 2r + 3$ - implying the existence of at least one regular tree node. Of course, we have not worked out any form of a concrete argument that this holds, though for creating a free tree node (given the same type of tree node we have for level-1 networks by Lemma 3.4), an approach similar to Algorithm 1 may well work within a level-k network.

5.3 Path- and Computational Efficiency

Lastly, the existence of a (finite) path between any two networks in a space implies that there is a shortest path between them, for we have finite upper- and lower bounds in the form of the length of the path provided by Algorithm 4 and 0, respectively (with the latter being the length of a path between identical networks). However, the algorithm we have laid out in this work is not particularly efficient with regards to the length of the paths it creates. This is rather obvious, as reducing the cycles of both

networks is, in the vast majority of cases, not necessary; the reason we use this method for Algorithm 4 is primarily for the sake of universality.

For instance, this algorithm will, for a path between two networks which are connected by a single RDE move (so where one pair of descendants of a reticulation are swapped around) turn every cycle into a simple one, which will waste a number of 'steps' determined by, among other things, the number of reticulations in the network, and the number of regular tree nodes branching off from their corresponding cycles. While we have, in the process of this work, considered methods for preventing the worst of such inefficiencies, we did not find any *general* methods for doing so; while, for any given example where the described algorithm provides a path significantly longer than a short(est) path through inspection, it is generally possible to add requirements and steps to account for *that particular case*, finding more general methods or heuristics for finding, at the very least, *sufficiently short* paths proved rather more difficult. It does, however, remain an interesting (and rather useful) avenue for further study.

Adding onto the inefficiency of the paths themselves, there is also the matter of implementation. As briefly touched on in the introduction, while methods for finding the shortest paths within certain tree spaces have been found, a large number of their corresponding problems have been shown to be NP-hard. The matter of the time complexity of the algorithms we have described here were not much of a consideration - especially as they have no concrete implementation as of yet -, and it would primarily become relevant for a prospective short(est) path algorithm, rather than one showing that *a* path exists. However, one interesting note is that, although the algorithm for ranked HSPR tree spaces [CWG24] we borrowed for our algorithm does not have a polynomial time complexity, the fact that the trees we use it on are of a specific form - being that they have a dummy edge, as introduced for Proposition 3.2 - opens up the possibility for using this characteristic to find a more (computationally) efficient method.

References

[BvIJ23] G. Bernardini, L. van Iersel, and E. et al. Julien. Constructing phylogenetic networks via cherry picking and machine learning. *Algorithms Mol Biol*, 18(13), 2023.

[CG21] L. Collienne and A. Gavryushkin. Computing nearest neighbour interchange distances between ranked phylogenetic trees. *J. Math. Biol.*, 82(8), 2021.

[CWG24] L. Collienne, C. Whidden, and A. Gavryushkin. Ranked subtree prune and regraft. *Bull Math Biol*, 86(24), 2024.

[Jan21] R. Janssen. *Rearranging Phylogenetic Networks*. PhD thesis, Delft University of Technology, 2021.