Spaces of Phylogenetic Time Trees

Lena Collienne

a thesis submitted for the degree of

Doctor of Philosophy

at the University of Otago, Dunedin,

New Zealand.

31 August 2021

Abstract

Time trees are evolutionary histories (also called phylogenies) that include times of evolutionary events. They arise in many applications, including cancer and virus evolution. Of particular interest are clock-like trees, where all leaves have equal distance to the root. These can be randomly sampled using the coalescent model, and a number of software packages for reconstructing trees from sequence data do so.

Most such inference methods use tree search algorithms, which require a tree space over which the inference is performed. These typically output a distribution of trees, which needs to be interpreted. Currently, most methods use consensus trees to summarise the output. Statistical methods such as mean trees and confidence regions would be preferable, however such methods are largely undeveloped for tree spaces.

It is essential for the development of such methods to explore the geometry of tree spaces. Most tree spaces are based on tree rearrangement operations, which apply local changes to a tree to propose trees that are similar to a given tree. Popular tree rearrangements are Nearest Neighbour Interchange, Subtree Prune and Regraft, and Tree Bisection and Reconnection. For all three tree rearrangements, the problem of computing distances, which are defined as the minimum number of tree rearrangements needed to transform one tree into the other, is $\mathcal{NP}$-hard, making tree inference and comparison algorithms challenging to design in practice.

In this thesis, we introduce discrete coalescent trees, a discretisation of time trees that is motivated by the coalescent model. We then define tree rearrangement operations on discrete coalescent trees, which leads to a new tree space DCT_m. We analyse this tree space, focussing on properties that are

essential to establish statistical measures such as mean trees and confidence regions. Our results include a polynomial-time algorithm for computing shortest paths in DCT_m, making this the first tree rearrangement based tree space in which distances can be computed efficiently. We also analyse geometrical properties of our tree space DCT_m, and shortest paths within the space. As a special case of discrete coalescent trees we consider ranked trees. We also discuss unlabelled time trees and two different tree spaces that result from considering tree rearrangement operations on them, one of which can be interpreted as the unlabelled version of DCT_m.

Acknowledgements

I would like to thank my supervisors Alex Gavryushkin and David Bryant for their support during my PhD. Without their guidance I would probably have gotten lost in the woods. I would also like to acknowledge the helpful discussions with the co-authors of my papers.

I also thank my family and friends for their ongoing support. This especially includes my parents Susanne and Norbert and my sister Anja. Thank you for supporting me in all my decisions, even if this means moving to the University furthest away from home. Also Wendy – I feel really lucky to have met you and for sharing the ups and downs of a PhD with you. Last, but not least, I owe Kieran a big thank you for his support, encouragement, and patience with me in the past few years.

Contents

Chapter 1

Introduction

Evolutionary histories, also known as phylogenetic trees, display evolutionary relationships between different species or individuals of the same species. Time trees, where evolutionary events are assigned times, are of particular interest in various applications, including viral epidemiology (Ypma, Ballegooijen, and Wallinga 2013; Zhang et al. 2020), and cancer evolution (Singer et al. 2018; Alves et al. 2019; Lote et al. 2017). Reconstructing time trees from sequence data such as RNA, DNA, or protein sequences is one of the major problems in computational biology. This sequence data is often derived at the same time, so the goal is to infer clock-like time trees, where all leaves are equidistant to the root.

One popular assumption used for inferring clock-like time trees is coalescent theory (Kingman 1982). The coalescent is widely employed for inferring relationships of a sample of genes (Hudson et al. 1990; Kuhner 2009), for analysing population dynamics (Kuhner, Yamato, and Felsenstein 1998; Drummond, Rambaut, et al. 2005), and most recently also for understanding cancer phylogenetics (Posada 2020; Ohtsuki and Innan 2017). Under a coalescent model, evolution is considered backwards in time, and two lineages coalesce after a waiting time, which is to be estimated. The resulting trees are referred to as coalescent trees, which are clock-like trees where internal nodes are assigned unique times. Independent of how branch lengths are inferred, if samples are taken at the same time, then the underlying phylogeny is clock-like when branch lengths are proportional to time. This makes the coalescent a reasonable assumption for inferring clock-like genealogies.

Popular software packages using the coalescent model are based on Maximum Likelihood (Kozlov et al. 2019; Nguyen et al. 2015; Tamura et al. 2011) or Bayesian methods (Bouckaert et al. 2014; Suchard et al. 2018; Ronquist and Huelsenbeck 2003). These

methods use tree search algorithms to reconstruct phylogenetic trees from sequence data. Tree search algorithms pick a start tree and then propose a new tree, which is similar to the current tree, in every step. The proposed tree is accepted if it fulfils certain requirements. Tree search algorithms hence rely on tree proposal methods for proposing trees that are measurably similar to a given tree.

Most similarity measures that are used for tree proposals are based on tree rearrangement operations. Popular tree rearrangements are Nearest Neighbour Interchange (NNI), Subtree Prune and Regraft (SPR), and Tree Bisection and Reconnection (TBR), all of which perform a local change to a tree. Distances based on these tree rearrangement operations, which are defined as the minimum number of operations needed to transform one tree into the other, are however $\mathcal{NP}$-hard to compute (Dasgupta et al. 2000; Bordewich and Semple 2005; Hickey et al. 2008; Allen and Steel 2001).

Measuring the similarity of trees is not only needed for inference methods, but also for comparing trees inferred for the same data by different methods. One tree distance measure that does not rely on a tree rearrangement method is the Robinson-Foulds distance (Robinson and Foulds 1981). In contrast to the tree rearrangement based distances mentioned above, this distance can be computed efficiently, and is therefore widely used. A downside of this approach however is a lack of biological interpretability. The Robinson-Foulds distance is not motivated by a biological process, unlike for example SPR, where the tree rearrangement operation can be used to model hybridisation and other horizontal events. This pattern is quite common – tree distance measures that are easy to compute lack biological interpretability, while those that are biologically meaningful are often hard to compute (Whidden and Matsen 2018).

Tree rearrangement based similarity measures are hence preferable in most applications. They also usually go hand in hand with the definition of a tree space. Investigating tree spaces is necessary for developing tree inference methods (and tree proposals within these methods) as well as for analysing their output. The output of Bayesian inference software for example is a distribution of trees, which is usually summarised in a single consensus tree together with clade support values (Drummond, Suchard, et al. 2012; Ronquist and Huelsenbeck 2003). Maximum Likelihood methods compute a single tree, which is often checked for credibility using bootstrapping to indicate confidence in its clades (Felsenstein 1985; Minh, Nguyen, and von Haeseler 2013). How these clade support values can be interpreted if they do not indicate a high support, is however not obvious (Jombart et al. 2017). Further shortcomings of this approach for assessing confidence in reconstructed trees are discussed by Holmes

(2003). A different approach to analyse distributions of trees is to compute mean trees and confidence regions within a tree space (Billera, Holmes, and Vogtmann 2001; Holmes 2003). Finding a tree space that is appropriate for establishing such statistical measures for trees has however proven to be a hard task. Current efforts (Jombart et al. 2017) use distance measures that are efficiently computable, but lack biological meaningfulness.

In this thesis we introduce the tree space DCT_m of discrete clock-like time trees, where times are integer-valued. This tree space is based on tree rearrangement operations that can be interpreted as a generalisation of NNI to time trees. We show that shortest paths, and hence distances, in this tree space can be computed in time polynomial in the number of leaves, and provide an algorithm to do so. Our tree space DCT_m is hence, to our knowledge, the first tree rearrangement based space in which distances can be computed efficiently. We furthermore establish some properties of DCT_m and shortest paths within this tree space, which are desirable in a tree space aimed at analysing tree distributions.

Structure of the thesis

We start our work by providing an overview of tree spaces that have been introduced in the literature (Chapter 2). This includes the well-known BHV-space of Billera, Holmes, and Vogtmann (2001), which was defined with the goal of developing statistical methods for phylogenetic trees. We also discuss two further tree spaces, τ-space and t-space, introduced by Gavryushkin and Drummond (2016). Both are designed specifically for time trees. In our discussion of these tree spaces we focus on properties that are important for developing statistical methods. Following this discussion, we also consider tree rearrangement operations (NNI, SPR, and TBR) and in particular their induced distance metrics.

Our discussion of known tree spaces indicates that a space of time trees suitable for developing statistical methods has yet to be found. We therefore introduce a new tree space, DCT_m based on tree rearrangement operations (Chapter 3). DCT_m is a space of discrete coalescent trees, which are discretised time trees motivated by the coalescent model. The majority of our work in this thesis is investigating this tree space, which can be seen as an adaptation of NNI to time trees. As a special case, we consider a tree space on ranked trees, which we denote by RNNI ('Ranked Nearest Neighbour Interchange').

One of the major findings in this thesis is an algorithm (FindPath) that computes shortest paths, and therefore distances, under the DCT_m metric in time polynomial in the number of leaves of the given trees (Chapter 4). The significance of this result is highlighted by the fact that computing distances is known to be $\mathcal{NP}$-hard for all of the classical tree rearrangement operations NNI, SPR, and TBR.

The ability to compute shortest paths in DCT_m efficiently motivates our further studies of this tree space, which are presented in Chapter 5. We show that shortest paths maintain clusters and therefore evolutionary information shared by two trees. This property is relevant for establishing graph-based methods to compute mean trees. We furthermore investigate the subgraph of the DCT_m space induced by caterpillar trees, a special class of trees. Based on our investigations of this subgraph, we show that distances between caterpillar trees can be computed even more efficiently than by using FindPath. We moreover discuss maximal values for distances in DCT_m and uniform distributions in the RNNI space of ranked trees.

In Chapter 6 we show that the tree rearrangement operations that define DCT_m can be used on unlabelled time trees as well. The algorithm FindPath can however not be modified easily to compute shortest paths in the resulting tree space uDCT_m. We instead propose two different methods for approximating distances between unlabelled trees in uDCT_m and compare them by using simulations. Following this, we introduce and discuss a new tree space (RSS) of unlabelled ranked trees, which is also based on a tree rearrangement operation and has efficiently computable distances.

We summarise our results in Chapter 7. A list of symbols and tree spaces can be found in the appendix.

Some of the work of this thesis has been published in papers with co-authors. Parts of the introduction and the results for the RNNI space in Section 4.1 are published in (Collienne and Gavryushkin 2021). Parts of the introduction, the definition of DCT_m (Section 3.1), as well as the discussion of the Shortest Path Problem in this space (Section 4.2) and geometrical properties (Section 5.1, Section 5.2, Section 5.3) can be found in (Collienne, Elmes, et al. 2021). Some details in Section 5.2.2, especially the algorithm Caterpillar Sort, can furthermore be found in (Collienne, Elmes, et al. 2019). For all papers the research has been planned collaboratively, and the main work has been carried out by the author of this thesis.

Chapter 2

Literature review: Known tree spaces

In this chapter we provide an overview of known tree spaces. We thereby focus on time trees, as our aim is to investigate spaces of time trees and to discuss if they build a suitable basis for developing statistical methods to analyse distributions of time trees. One property that we would like a tree space to have for this purpose is that shortest paths between trees can be computed efficiently. We furthermore want shortest paths to preserve information shared by two trees in form of a split or cluster (formal definitions follow later in Section 2.3.1), as this indicates that summary trees will most likely contain this information, too.

We split our overview of existing tree spaces into three sections. At first we introduce some definitions (Section 2.1), before we discuss the three tree spaces BHV-space, τ-space, and t-space in Section 2.2. These tree spaces are related to tree rearrangement operations, which we consider in more detail in Section 2.3, where we also describe how these rearrangement operations can be used to construct tree spaces as graphs.

2.1 Basic definitions

A *phylogenetic tree* is a binary tree where leaves are uniquely labelled by elements of a set X, which we assume to be $X = \{a_1, \ldots, a_n\}$. Note that we assume throughout this thesis that all trees have a fixed number of n leaves. Most phylogenetic trees we consider here are *rooted phylogenetic trees*, which have one additional node of degree two that can be inserted on any edge of the tree. This node of degree two is called the *root*. In this chapter we call rooted phylogenetic trees simply *trees* and (unrooted) phylogenetic trees *unrooted trees*.

In this thesis we focus on *time trees*, rooted phylogenetic trees where all internal

nodes are labelled by unique positive real-valued times such that each node has time greater than its children, and all leaves are assigned time 0. We hence consider time running backwards, from the leaves of a tree towards its root. A special type of time trees are *ranked trees*, where internal nodes are assigned unique integer-valued times (also called *ranks*) in $\{1, \ldots, n-1\}$, and all leaves are labelled by time 0. A ranked tree that results from a time tree T by ignoring actual times and only keeping the order of internal nodes of T is called the *underlying ranked tree* of T (for an example see Figure 2.1). Note that it is also possible to ignore times of internal node entirely to only consider the branching process, i.e. the phylogenetic tree. For a ranked tree or time tree we call the tree that results from ignoring times (or ranks) of internal nodes the *underlying tree* (see Figure 2.1).

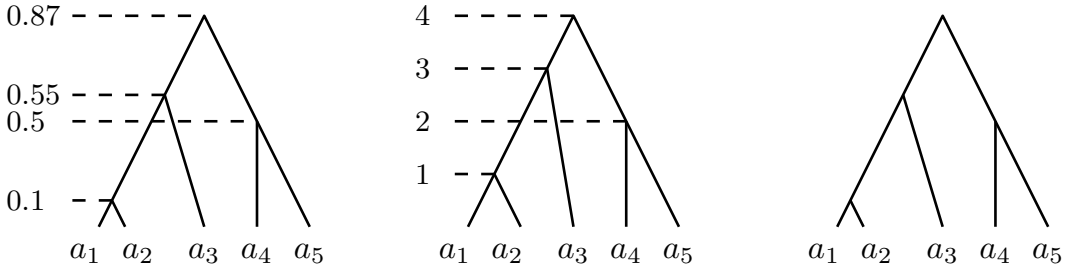


Figure 2.1: Time tree on the left, its underlying ranked tree in the middle, and the underlying (unranked) tree on the right. Note that all leaves are assigned time/rank zero in the two leftmost trees, and no nodes are assigned times in the rightmost tree.

2.2 Tree spaces

Our goal in this section is to discuss known tree spaces, especially properties that we expect these tree spaces to have to build a basis for statistical analyses of tree distributions. First we consider the BHV-space (Section 2.2.1), a popular tree space introduced by Billera, Holmes, and Vogtmann (2001). We will however see that this tree space is more suitable for trees where branch lengths display the number of mutations rather than time differences. The other two tree spaces, namely τ- and t-space (Sections 2.2.2 and 2.2.3), have been designed specifically for time trees.

2.2.1 BHV-space

The BHV-space is named after the authors who introduced it: Billera, Holmes, and Vogtmann (2001). It is probably the most commonly known tree space of the ones

discussed here. Trees in BHV-space are parameterised as follows: Every binary time tree can be identified by its underlying tree and a vector of $n-2$ real numbers that represent edge lengths of internal edges of the tree, i.e. edges that are not incident to leaves. In BHV-space each tree is associated with a positive open orthant $(0,\infty)^{n-2}$, such that a particular time tree is at the position specified by its edge length vector in the orthant associated with its underlying tree. On the boundary of an orthant are time trees that have at least one edge of length zero, i.e. non-binary trees. The BHV-space for time trees on $n = 3$ leaves is illustrated in Figure 2.2. An interesting property of the BHV-space is that trees corresponding to orthants that share a boundary face of co-dimension one are connected by an NNI move, which we will explain in more detail in Section 2.3.1.

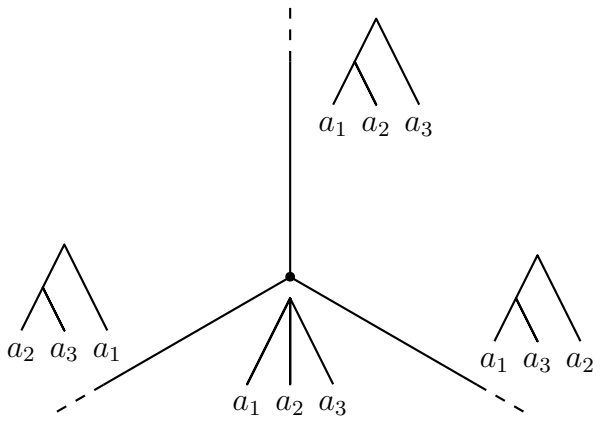


Figure 2.2: A projection of the BHV-space for trees on $n = 3$ leaves consisting of three rays that share their origin, which is associated with the star tree (after Billera, Holmes, and Vogtmann (2001)). Each ray represents a different tree and the position on the ray represent the length of the internal edge of a tree.

The distance between two trees in BHV-space is defined as the length of a shortest path connecting the two trees. For two time trees with the same underlying tree, the distance is the Euclidean distance between the edge length vectors of the given trees. For two time trees that correspond to points in different orthants, a path connecting them consists of straight segments within orthants that are connected at the points where one or more elements of the edge length vector are zero, i.e. at the boundary face. Such a path of minimum length is called *geodesic*. Billera, Holmes, and Vogtmann (2001) have proven that their tree space has unique geodesics. Owen and Provan (2011) provide an algorithm that computes shortest paths in BHV-space in time polynomial in the number of leaves. Another important property of BHV-space is that if two

trees share an edge, every tree on the geodesic between them contains this edge as well (Billera, Holmes, and Vogtmann 2001). This is a property closely related to the cluster property, which we will discuss in Section 5.1.

For time trees, especially ultrametric time trees, the parameterisation used in BHV-space is however not ideal. Since the coordinates of a tree are given by its edge lengths, changing the length of one internal edge results in changing times of all external edge lengths, i.e. the time of the parents of all leaves change (Gavryushkin and Drummond 2016). This is why this space is more suitable for trees where edge lengths represent the number of mutations rather than time differences. Another problem specifically for time trees is that subspaces of the BHV-space associated with different ranked trees have different volumes. For example a (ranked) caterpillar tree on four leaves is represented by one orthant in BHV-space, while a ranked tree on four leaves with two cherries only takes half an orthant in BHV-space. This makes it hard to introduce a probability distribution over such spaces without biasing towards certain ranked trees. A detailed discussion of this topic can be found in (Gavryushkin and Drummond 2016), where more suitable spaces for time trees, t-space and τ-space, are introduced and studied. We discuss these two tree spaces in the following sections.

2.2.2 τ-space

We now consider the τ-space, introduced by Gavryushkin and Drummond (2016). This tree space, as well as the t-space in the next section, are designed specifically for time trees. Instead of edge lengths, the actual times assigned to internal nodes are used to parameterise trees in τ-space.

More specifically, time trees in τ space are parameterised as a pair consisting of the underlying ranked tree and a list $(\tau_1, \tau_2, \ldots, \tau_{n-1})$ of real numbers, where τ_i represents the time differences of internal nodes of rank $i-1$ and i (see Figure 2.3).

In τ-space, every ranked tree is associated with an $(n-1)$-dimensional non-negative orthant $[0, \infty)^{n-1}$. Every time tree is associated with the point $(\tau_1, \ldots, \tau_{n-1})$, the vector of time differences as described above, in the orthant that corresponds to its underlying ranked tree. On the boundary of these orthants, at least one coordinate τ_i is zero. Two orthants share a boundary, if the trees associated with the points on the boundary are in both orthants. By attaching orthants like this, we receive the τ-space. If two orthants share a boundary face of co-dimension one, the corresponding ranked trees are connected by an RNNI move, which we formally introduce in Chapter 3.

Distances between time trees in τ-space are defined similarly to those in BHV-space:

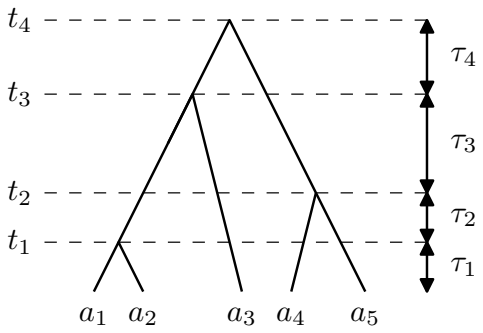


Figure 2.3: This time tree has $(\tau_1, \tau_2, \tau_3, \tau_4)$ as coordinates in τ-space. In t-space, the coordinates of this time tree are the actual node times (t_1, t_2, t_3, t_4).

For time trees within the same orthant, i.e. time trees with the same underlying ranked tree, their distance is simply the Euclidean distance of the corresponding coordinates. If two time trees are in different orthants, they can be connected by segmented paths with length equal to the sum of the length of straight segments.

Although the definitions of BHV-space and τ-space are similar, Gavryushkin and Drummond (2016) have shown that these two spaces are not isometric. There are however some properties that BHV- and τ-space have in common, which includes the uniqueness of geodesics (Gavryushkin and Drummond 2016). Furthermore, the algorithm of Owen and Provan (2011) for computing shortest paths in BHV-space can be modified to be used for computing shortest paths in τ-space in time polynomial in the number of leaves (Gavryushkin and Drummond 2016).

There are however properties of τ-space that indicate that it is not suitable for establishing statistical methods. For example, shortest paths between time trees in τ-space often contain trees that are star-tree-like (Gavryushkin and Drummond 2016), similarly to BHV-space. This can be problematic when the goal is to summarise a set of trees, for example a posterior sample, to a mean tree. If shortest paths contain star-like-trees, the mean tree can be expected to be star-like, too, which results in losing information shared by most trees in the sample, e.g. common clusters. Because of this, BHV- and τ-space are not suitable for some applications, even though shortest paths can be computed efficiently.

2.2.3 t-space

The t-space was introduced together with τ-space by Gavryushkin and Drummond (2016) and is constructed specifically for time trees. Trees in t-space are parameterised as a pair consisting of the underlying ranked tree and the vector of times of internal

nodes, ordered increasing from leaves to the root of the time tree. Instead of considering the time differences as in τ-space, the actual times assigned to internal nodes give the coordinates of time trees in t-space. The time tree in Figure 2.3 for example has (t_1, t_2, t_3, t_4) as coordinates.

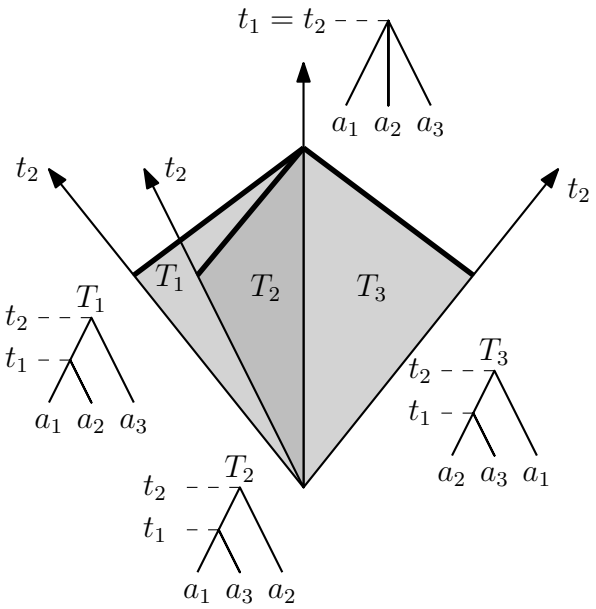


Figure 2.4: t-space for time trees on $n = 3$ leaves (after Gavryushkin and Drummond (2016)). The orthants labelled by T_1, T_2, T_3 with the corresponding time trees. All orthants corresponding to binary time trees meet at the boundary that is associated with the star tree ($t_1 = t_2$, vertical line). The upper limits of the root height H gives the upper boundaries of the simplex, highlighted as bold lines.

Unlike in BHV-space and τ-space, each ranked tree is associated with a simplex $\{(t_1, \ldots, t_{n-1}) | 0 \leq t_1 \leq t_{n-1} \leq H\}$, where H is an artificial, large enough bound of the height of the time trees. Trees on the boundaries have at least one coordinate t_i such that $t_i = t_{i+1}$. Two simplices are glued together at those boundaries if the trees at the boundaries are in both simplices. An illustration of the t-space on three leaves can be found in Figure 2.4. Note that simplices share a boundary face of co-dimension one if the associated ranked trees are connected by an RNNI move, as in τ-space.

The distance between two time trees with the same underlying ranked tree, i.e. time trees within the same simplex, is the Euclidean distance of their coordinates. Paths between time trees in different simplices consist of segmented Euclidean paths. As in the previously introduced tree spaces, geodesics in t-space are unique. The algorithm of Owen and Provan (2011) is however not applicable to t-space (Gavryushkin and Drummond 2016). There is no algorithm known to compute geodesics for t-space in

time polynomial in the number of leaves, which raises the question whether such an algorithm exists.

Finding the complexity of computing shortest paths in t-space would be beneficial, as geodesics in t-space have desirable properties that BHV- and τ-space lack. This includes in particular the fact that geodesics often are cone paths in BHV- and τ-space, but not in t-space (Gavryushkin and Drummond 2016). We later introduce the discrete coalescent tree space DCT_m (Chapter 3), which can be interpreted as a discrete version of t-space. Our results on the complexity of computing shortest paths in DCT_m might hence be useful for t-space.

2.3 Discrete tree spaces

In this section we introduce three different types of tree rearrangement operations, and discuss tree spaces that can be defined as graphs based on these operations. Tree rearrangement operations perform local changes to a tree to transform it into another tree. The three classical rearrangement operations that we discuss here are Nearest Neighbour Interchange (NNI, Section 2.3.1), Subtree Prune and Regraft (SPR, Section 2.3.2), and Tree Bisection and Reconnection (TBR, Section 2.3.3). Note that instead of considering trees with branch lengths, we now only consider the underlying tree, as the above mentioned tree rearrangement operations do not take branch lengths into account.

For any given tree rearrangement operation, we can construct a graph, where vertices represent trees. Two vertices are connected in such a graph if the associated trees can be obtained from each other by performing one tree rearrangement operation. Note that these operations are reversible, resulting in an undirected graph. This results in a definition of distances between trees as the length of shortest paths in the corresponding graph. Since all the graphs based on tree rearrangement operations that we introduce here are connected, the distance between trees in these graphs is a metric. We hence refer to these graphs as tree spaces.

2.3.1 NNI

In this section we introduce the Nearest Neighbour Interchange rearrangement operation for rooted trees. This tree rearrangement operation is of particular interest within this thesis, as it builds the basis of the rearrangement operation that we will introduce in Chapter 3 and discuss extensively throughout this thesis. Rooted and unrooted

NNI are closely related and the resulting tree spaces have similar properties. So even though NNI is usually defined for unrooted trees, we introduce NNI for rooted trees, as we focus on (rooted) time trees in this thesis.

A *Nearest Neighbour Interchange* operation (NNI *move*) connects two trees T and R if T has an edge e and R an edge f such that the (non-binary) trees resulting from contracting e and f are identical. An illustration of this can be found in Figure 2.5, where the tree in the middle results from contracting edges. Alternatively, one can think of an NNI move as contracting an edge and resolving the resulting node of degree four to receive a binary tree again. For each edge, exactly two NNI moves result in a tree that is different from the given tree. The reason for this is that there are three ways of resolving an internal node with three children into two nodes with two children each (see Figure 2.5). We say that a tree that results from an NNI move on a tree T is an NNI *neighbour* of T. Since every binary rooted tree has $n-2$ internal edges, each tree has $2(n-2)$ NNI neighbours.

The NNI *graph* hence consists of rooted (binary) trees as vertices that are connected by an edge if the trees are connected by an NNI move.

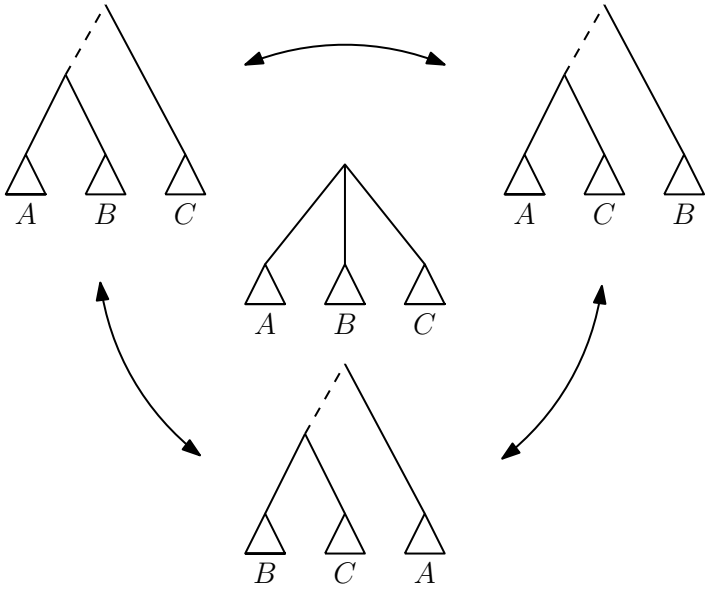


Figure 2.5: Three NNI neighbours connected by NNI moves on the dashed edges. The (non-binary) tree in the middle results from contracting these dashed edges to a node and is not a vertex in the NNI graph.

Remember that trees on boundary faces of orthants in BHV-space have at least one edge with length zero. Our definition of NNI moves on rooted trees hence implies that two orthants in BHV-space that share a boundary face of co-dimension one are

associated with trees that are NNI neighbours. Unlike in BHV-space, shortest paths in NNI are not unique. This can easily be seen when considering a tree T and two NNI moves on edges e and f that are independent, i.e. they are not incident to the same node. When considering a tree R that results from an NNI move on e and on f, it does not matter in which order the moves on e and f are performed, resulting in two possible shortest paths between T and R.

One of the most important properties of the NNI space for us is that the problem of computing distances in NNI is $\mathcal{NP}$-hard. Remarkably, it took over 25 years and a number of published erroneous attempts to prove that computing distances is $\mathcal{NP}$-hard in NNI (Dasgupta et al. 2000). Essential for the (correct) proof of Dasgupta et al. (2000) is the fact that the NNI space does not have the cluster property. We say a tree space has the *cluster property*, if for two trees that share a cluster every tree on every shortest path between them also has this cluster. More detailed definitions can be found in Section 2.1. The fact that the NNI space does not have the cluster property has been shown by Li, Tromp, and Zhang (1996).

Computing the NNI distance is also known to be fixed parameter tractable (DasGupta et al. 1999). This result is however not useful for applications, as the fixed parameter of the algorithm of DasGupta et al. (1999) is the distance. Computing the distance for trees that are far away from each other in the NNI space is hence not feasible.

The tree rearrangement operation NNI builds the basis of the tree spaces DCT_m introduced in this thesis (Chapter 3). By modifying NNI to be suitable for time trees, we receive tree rearrangement based tree spaces where distances can be computed efficiently. A detailed discussion of those tree spaces follows in Chapters 3, 4, and 5.

2.3.2 SPR

A *Subtree Prune and Regraft* operation (SPR *move*) on an unrooted tree T is defined by two edges e and f of T, where the edge e is cut, resulting in two connected components such that one of them contains the edge f. The resulting connected component T' that does not contain f has one node v of degree two. To reconnect the two connected components we introduce a new edge between v and a new node inserted on the edge f. The remaining node of degree two, which is the second degree-two node resulting from cutting e, gets suppressed, and we receive an unrooted tree R. We call this tree R SPR *neighbour* of T. In Figure 2.6 an SPR move is illustrated. The number of SPR neighbours of a tree is $2(n-2)(2n-7)$, which is quadratic in n (Allen and Steel 2001).

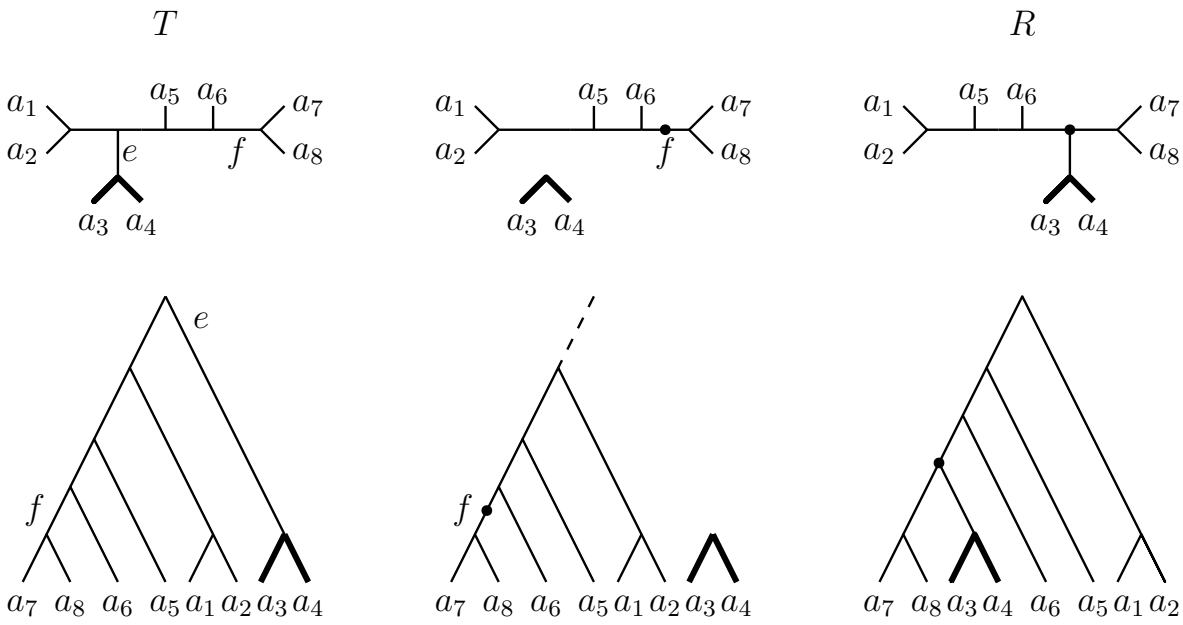


Figure 2.6: Illustration of an SPR move that transforms the trees on the left to the trees on the right. At the top is an example of unrooted trees, at the bottom rooted trees. The graphs in the middle column results from cutting the edges e, respectively, and are not trees, as they are disconnected.

We now extend SPR to rooted trees. We denote the resulting tree rearrangement operation by rSPR (rooted Subtree Prune and Regraft). Rooted and unrooted SPR are not as closely related as rooted and unrooted NNI, so we consider them separately. rSPR moves are defined like SPR moves, one just needs to consider the root edges as special cases. If the edge e that is cut in a tree T is incident to the root of T, the root has degree one after cutting e. If this is the case, the root and the edge incident to it are deleted and the resulting node of degree two, which used to be child of the root in T, now becomes the root. Furthermore, any subtree that has been cut off can be re-attached above the root of T. In this case, a new node is introduced as root of R with the root of T and the root of the subtree that has been cut off as children.

Note that both SPR and rSPR moves are reversible. Furthermore, every NNI move is an SPR move, implying that the NNI distance between two trees is greater than or equal to their SPR distance.

As for NNI, incorrect proofs for the $\mathcal{NP}$-hardness of computing SPR distances have been discussed in the literature (Hein et al. 1996; Allen and Steel 2001). Bordewich and Semple (2005) finally proved the $\mathcal{NP}$-hardness result for rooted trees and Hickey et al. (2008) utilised this proof to establish the result for unrooted trees. To facilitate

practical applications, fixed parameter tractable algorithms (Downey and Fellows 2013) for computing the SPR distance have been developed over the years (Whidden, Beiko, and Zeh 2010; Bordewich and Semple 2005; Whidden and Matsen 2018). But as in the NNI space, these algorithms remain impractical for large distances and are only applied to trees with a moderate number of leaves or those with small distances (Whidden and Matsen 2018).

2.3.3 TBR

To complete our list of tree rearrangement operations, we now introduce TBR. A *Tree Bisection and Reconnection* operation (TBR *move*) on an unrooted tree T is defined similarly to SPR. A TBR move however is more general, as the two connected components resulting from cutting an edge e can be reconnected by introducing a new node on an arbitrary edge in each of the two connected components. After reconnecting the two components by an edge between these new nodes, the remaining nodes of degree two are suppressed to get a TBR neighbour of T. An illustration of a TBR move is given in Figure 2.7.

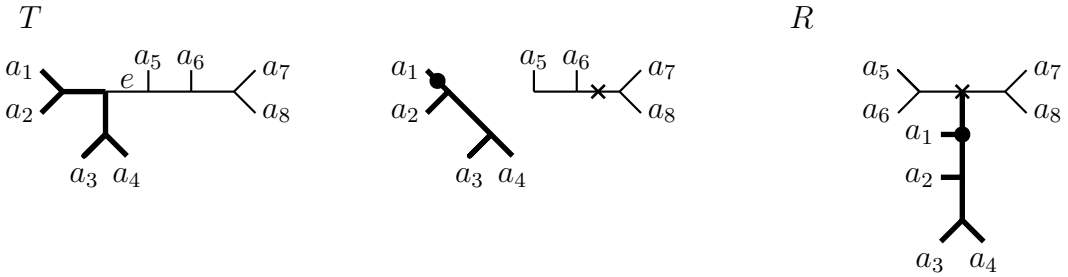


Figure 2.7: Illustration of a TBR move transforming the tree T on the left to the tree R on the right. The graphs in the middle results from cutting the edge e in T and suppressing the resulting nodes of degree one and two. After cutting e, the two resulting subtrees are reconnected between the two edges marked by a cross and a dot.

TBR is the most general tree rearrangement operation and, unlike in NNI and SPR, the neighbourhood size depends on the tree. It has however been shown by Humphries and Wu (2013) that the neighbourhood size is in $\mathcal{O}(n^3)$. Furthermore, TBR is a generalisation of both NNI and SPR moves, as every NNI and SPR move also is a TBR move, and TBR moves are reversible. Computing the TBR distance is known to be $\mathcal{NP}$-hard (Allen and Steel 2001). But in contrast to NNI and SPR, there is no intuitive definition of TBR operations on rooted trees, which makes this operation not suitable for time trees.

Chapter 3

Discretising time trees

In this chapter we introduce the tree space that we analyse throughout this thesis. We therefore define two discrete versions of time trees, discrete coalescent trees and ranked trees. After introducing discrete coalescent trees in Section 3.1, we define the tree space DCT_m on these trees. DCT_m is a tree rearrangement based tree space, just like NNI, SPR, and TBR (Section 2.3), and is hence defined as a graph. Discrete coalescent trees correspond to vertices in DCT_m and two trees are connected by an edge, if one can be transformed into the other by a tree rearrangement operation, which we define in Section 3.1. This new tree space DCT_m can be interpreted as an adaptation of NNI to time trees, or a discretisation of t-space (Section 2.2.3). The parameter m decides on how fine the discretisation of time trees in DCT_m is. When choosing the minimum value for m, we receive a tree space of ranked trees, which we call RNNI ('Ranked Nearest Neighbour Interchange', Sections 3.3 and 3.4). After introducing discrete coalescent trees, ranked trees, and the resulting tree spaces in Sections 3.1 to 3.4, we discuss some of their basic properties, such as the number of vertices and edges, in Section 3.5.

3.1 Discrete coalescent trees

At first we introduce discrete coalescent trees, a discretisation of time trees motivated by the coalescent. *Discrete coalescent trees* are rooted phylogenetic trees with a positive integer-valued *time* assigned to each node. More specifically, all n leaves $a_1, \ldots, a_n$ of a discrete coalescent tree are assigned time 0, and every internal node is assigned a unique time less than or equal to an integer m, such that every node has time greater than its children. This means that we consider time running backwards, from the leaves of a tree to its root. Throughout this thesis, we assume that trees have n leaves

labelled by $\{a_1, \ldots, a_n\}$, unless specified otherwise. For simplicity we use the word *tree* to refer to discrete coalescent trees.

We denote the time of a node v by time(v). Times of internal nodes can be used to define the *length* of an edge in a tree as the time difference of the two nodes bounding that edge. Two discrete coalescent trees are considered to be *identical* if there exists a graph isomorphism between them that preserves leaf labels and node times. We will also consider trees that are not binary. In this case we also call two trees identical if there exists a graph isomorphism between them which preserves leaf labels and node times.

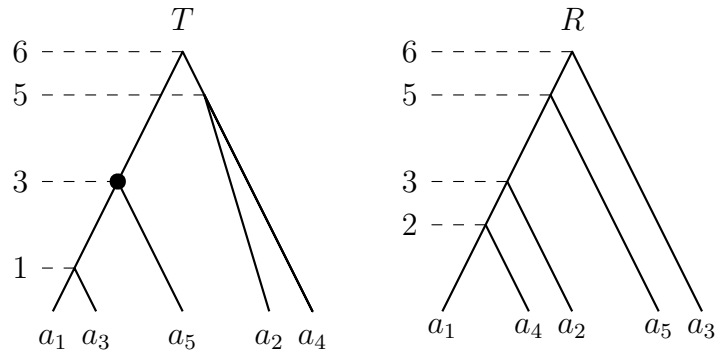


Figure 3.1: Discrete coalescent trees T and R with $n = 5$ leaves and root time $m = 6$. The highlighted node with time three in T can for example be referred to as $p(a_5)_T$, $mrca(\{a_1, a_5\})_T$, or $(T)_3$ and the cluster induced by this node is $\{a_1, a_3, a_5\}$. The tree R on the right is a caterpillar tree.

Because all nodes have unique times, we can identify every internal node in a tree T by its time $t \in \{1, \ldots, m\}$, and write $(T)_t$ to denote this node. An *interval* $[(T)_i, (T)_j]$ is defined by two internal nodes of consecutive times $i < j$, meaning that there is no node with time between i and j in T. Since every tree has $n-1$ internal nodes, every tree has $n-2$ intervals. We call a set $C \subseteq \{a_1, \ldots, a_n\}$ a *cluster* in a tree T if there is an internal node in T such that C contains exactly all leaves descending from this node. We then say that this internal node *induces* the cluster C, and that the subtree rooted at this node is *induced* by C. Let $C_1, \ldots, C_{n-1}$ be the clusters of a tree T, such that cluster C_i is induced by the internal node of time t_i in T and $t_1 < t_2 < \ldots < t_{n-1}$. T can uniquely be specified by its *cluster representation*, that is the list of pairs C_i, t_i for all $i = 1, \ldots, n-1$. We write the cluster representation as $[C_1 : t_1, C_2 : t_2, \ldots, C_{n-1} : t_{n-1}]$. For example, the cluster representation of the tree on the left of Figure 3.1 is

$$[\{a_1, a_3\} : 1, \{a_1, a_3, a_5\} : 3, \{a_2, a_4\} : 5, \{a_1, a_2, a_3, a_4, a_5\} : 6].$$

Note that the cluster $\{a_1, \ldots, a_n\}$ containing all leaves of a tree is the last cluster in the list representation of every tree on n leaves. Adding this cluster to the representations of ranked trees is not necessary, but it will be useful later in this thesis (Algorithm 2).

For a set $S \subseteq \{a_1, \ldots, a_n\}$ and a tree T we denote the *most recent common ancestor* ($mrca$) of S in T, which is the node with lowest time in T that induces a cluster containing all elements of S, by $mrca(S)_T$. We might also denote the $mrca$ of a cluster C by $mrca(C)$, if it is obvious which tree we consider. Note that $mrca(C)_T = (T)_t$ if the cluster C is induced by the node of time t in T. In the tree T in Figure 3.1 the node $(T)_3$ of time three can for example be referred to as $mrca(\{a_1, a_5\})_T$ or, using the cluster induced by this node, as $mrca(\{a_1, a_3, a_5\})_T$. The parent of a leaf a_i is denoted by $p(a_i)$, or $p(a_i)_T$ to emphasise that we mean its parent in the tree T. We call a subtree induced by a cluster $\{a_i, a_j\}$ with exactly two elements a *cherry*. A type of trees that is important throughout this thesis are *caterpillar trees*, trees where every internal node has at least one child that is a leaf. An example of a caterpillar tree is depicted on the right of Figure 3.1.

3.2 The tree space DCT_m of discrete coalescent trees

We are now ready to introduce the graph of discrete coalescent trees. This graph is called DCT_m for a fixed positive integer m. We assume throughout this thesis, unless stated otherwise, that the number of leaves of the trees in DCT_m is a fixed integer n.

The vertex set of DCT_m is the set of trees with root time less than or equal to m. Trees T and R are connected by an edge (T and R are *neighbours*) in this graph if performing one of the following (reversible) operations on T results in R (Figure 3.2):

(i) An NNI *move* connects trees T and R if there is an edge e in T and an edge f in R, both of length one, such that contracting e and f to nodes results in identical (non-binary) trees.

(ii) A *rank move* on T exchanges the times of two internal nodes with time difference one.

(iii) A *length move* on T changes the time of an internal node by one.

We first want to understand how NNI moves change discrete coalescent trees. Consider an NNI move on an edge $[(T)_k, (T)_{k+1}]$ in T such that the clusters induced by the children of $(T)_k$ are A and B and the cluster induced by the child of $(T)_{k+1}$ that is not

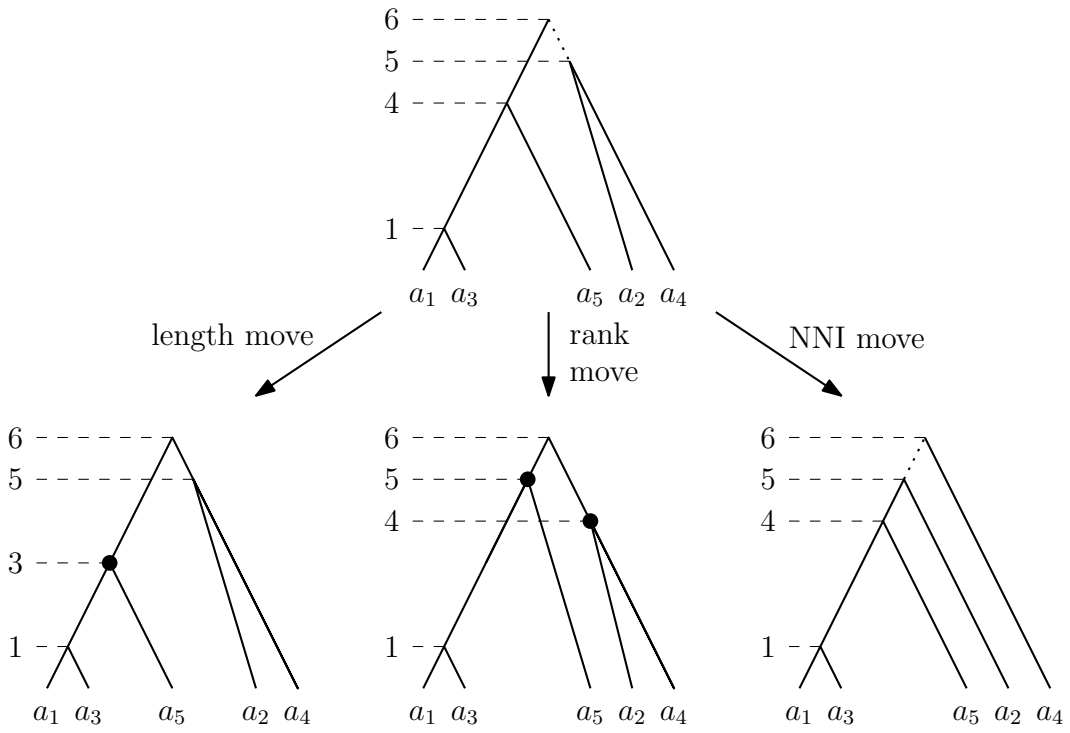


Figure 3.2: The three possible moves on a discrete coalescent tree: a length move changing the time of the highlighted node on the left, a rank move swapping the times of the highlighted nodes in the middle and an NNI move on the dotted edge on the right.

$(T)_k$ is D. We furthermore denote the cluster $A \cup B$ induced by $(T)_k$ by C. A tree T with these clusters A, B, C, D is illustrated on the top left of Figure 3.3. Contracting the edge $[(T)_k, (T)_{k+1}]$ to a single node results in a (non-binary) tree that has one node with three children, which induce the clusters A, B, and D (see the tree in the middle of Figure 3.3). Note that this non-binary tree is not in the vertex set of DCT_m. There are exactly three trees where contracting an edge results in the same non-binary tree as contracting $[(T)_k, (T)_{k+1}]$ in T:

- T itself, where the cluster induced by $(T)_k$ is $A \cup B$ (top left of Figure 3.3),
- T' that coincides with T except for the cluster induced by $(T')_k$, which is $A \cup D$ (top right of Figure 3.3), and
- T'' that coincides with T except for the cluster induced by $(T'')_k$, which is $B \cup D$ (bottom of Figure 3.3).

This implies that every edge of length one, i.e. every edge bounded by nodes of consecutive times, in a tree T gives exactly two NNI neighbours that are not identical to

T. Furthermore, there cannot be two different NNI moves on a tree that result in the same NNI neighbour. As we do not consider a tree to be an NNI neighbour of itself, the graph DCT_m is a simple graph without self-loops. A further important observation is the following:

Observation 3.1. *An* NNI *move on a tree T changes the cluster of exactly one node of T.*

The NNI move on the top left tree in Figure 3.3 for example changes the cluster induced by the node of rank k from $C = A \cup B$ to either become $A \cup D$ or $B \cup D$. This implies in particular that the time of the *mrca* of C increases by one when this NNI move is performed on T.

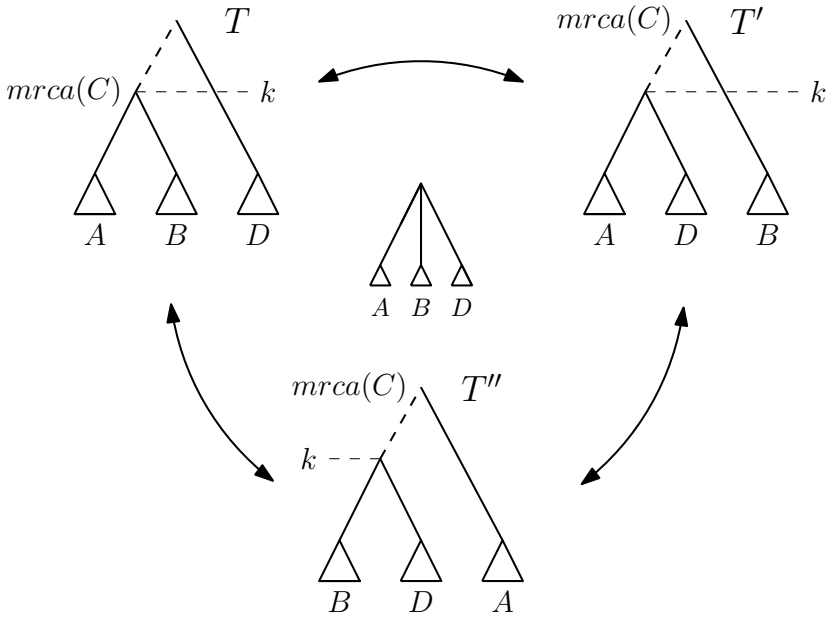


Figure 3.3: Three NNI neighbours T, T', and T'' connected by NNI moves on the dashed edges (of length one), and the tree resulting from contracting those edges to a node in the middle. Annotated are the most recent common ancestors of the cluster $C = A \cup B$.

We now consider length moves in more detail. They can only change the time of a node to become $t \in \{1, \ldots, m\}$ if there is no node with time t already. In the tree on the top of Figure 3.2 for example, there cannot be a length move changing the time of the internal node with time 5, as there already are internal nodes with time 4 and 6. Furthermore, the time of the root of a tree in DCT_m cannot be changed by a length move to become greater than m in DCT_m. Increasing the time of the root of the tree

on top of Figure 3.2, for example, would create a tree with root time 7, which would not be in DCT_6.

With the definition of DCT_m as a graph, we can use some concepts from graph theory for DCT_m. We define a *path* p between two trees T and R as a sequence $T = T_0, T_1, \ldots, T_k = R$ of trees such that T_i and T_{i+1} are connected by an edge in DCT_m for all $i = 0, \ldots, k-1$. For such a path p we say its *length* $|p|$ is k. The *distance* between T and R, denoted by $d(T, R)$, is the length of a *shortest path* between T and R, which is a path of minimal length between T and R. We will see in Section 3.5 that because DCT_m is connected, the distance between any pair of trees is well defined.

Unless stated otherwise, we assume from now on that m is an arbitrary fixed integer greater than or equal to $n-1$ (the number of internal nodes of a tree (Steel 2016)), as our results for DCT_m apply to all values of $m \geq n-1$.

A tree space similar to DCT_m has been defined by Gavryushkin, Whidden, and Matsen (2018) as DtTu_m. Our definition of DCT_m however differs from DtTu_m. In contrast to length moves in DtTu_m, length moves in DCT_m do not change the height of a tree, unless it is performed on the root, which makes our definition more appropriate for coalescent trees.

3.3 Ranked trees

Ranked trees are discrete coalescent trees with root time $m = n-1$. Internal nodes of ranked trees are hence bijectively labelled by elements of the set $\{1, \ldots, n-1\}$ (see the tree on the right of Figure 3.4). As in discrete coalescent trees, all leaves have time 0. Note that this definition of ranked trees coincides with the one in Section 2.1. For ranked trees we use the notion *rank* of a node instead of time, and hence denote the time of a node v by $\mathrm{rank}(v) = \mathrm{time}(v)$. Ranked trees can be seen as the coarsest discretisation of time trees, where instead of times only the order of internal nodes is considered. The tree in the middle of Figure 3.4 for example can be seen as discretisation of the time tree on the left of that figure.

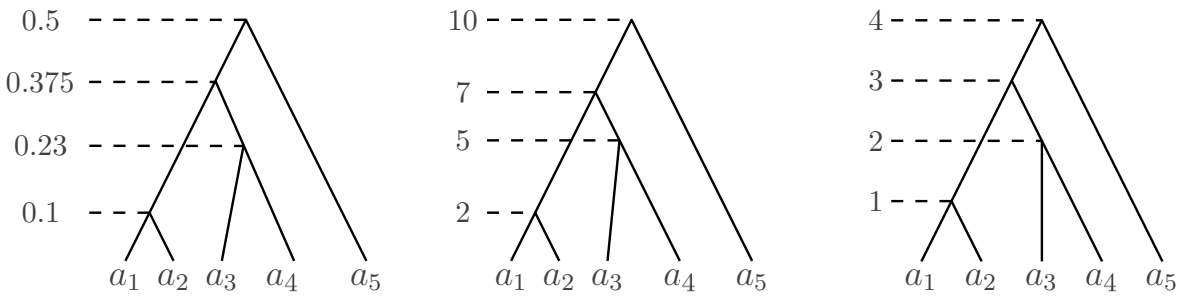


Figure 3.4: A time tree on 5 leaves on the left, in the middle a discrete coalescent tree that can be interpreted as a discretisation of that time tree, and on the right a ranked tree as the coarsest discretisation of the time tree.

Because all elements of the set $\{1, \ldots, n-1\}$ appear as ranks in a tree T, we can simplify the cluster representation for ranked trees. Instead of a list of pairs containing a cluster and the time of the node inducing it, ranked trees can be represented by a list of all clusters ordered according to the ranks of internal nodes inducing them. If the cluster C_i is induced by the node with rank i in T for all $i \in \{1, \ldots, n-1\}$, then the cluster representation of T is $[C_1, C_2, \ldots, C_{n-2}]$. For example, the cluster representation of the ranked tree on the right of Figure 3.4 is

$$[\{a_1, a_2\}, \{a_3, a_4\}, \{a_1, a_2, a_3, a_4\}, \{a_1, a_2, a_3, a_4, a_5\}].$$

3.4 The tree space RNNI of ranked trees

We now introduce the RNNI graph as a tree space of ranked trees. This graph is of particular importance in this thesis, as we often first prove statements for RNNI before we extend them to the more general DCT_m. We define the *ranked nearest neighbour interchange* (RNNI) graph as the DCT_m graph for $m = n - 1$. Note that discrete coalescent trees with root time $n - 1$, i.e. the trees of DCT_{n-1}, are ranked trees.

Because all intervals in ranked trees have length one, length moves are not possible in RNNI. We summarise the remaining two tree rearrangement operations and say RNNI *move* to mean either a rank move or an NNI move. We furthermore distinguish intervals of a ranked tree by the type of move that can be performed on the interval. If the interval is an edge, we call it *edge interval*, on which an NNI move can be performed, otherwise it is a *rank interval*, on which a rank move can be performed. An illustration of a part of the RNNI graph is provided in Figure 3.5. The tree on the top right of the figure only contains edge intervals, while the tree in the middle at the top has one rank interval between nodes of rank 1 and 2.

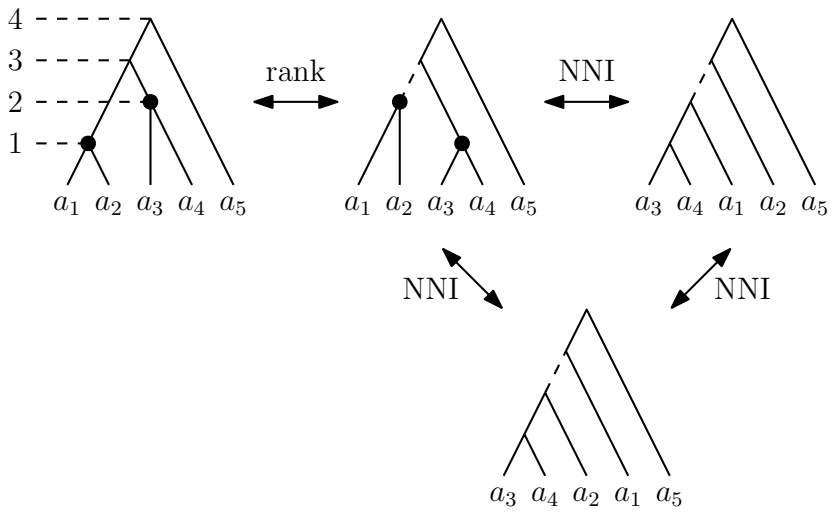


Figure 3.5: Trees in the RNNI graph with a rank move swapping ranks of the highlighted nodes on the left and three NNI moves on the dashed edges on the right.

Note that our RNNI graph is defined as RNNIu graph in (Gavryushkin, Whidden, and Matsen 2018).

3.5 Basic graph properties of DCT_m and RNNI

In this section we prove that DCT_m, and hence RNNI, is a connected graph (Theorem 3.1), which implies that the induced distances are metric. We then establish the number of ranked trees (Theorem 3.2) and discrete coalescent trees with maximum root time m (Corollary 3.3), i.e. the number of vertices in RNNI and DCT_m. We furthermore count the number of edges in the RNNI graph (Theorem 3.4).

Theorem 3.1. *The graph* DCT_m *is connected.*

Proof. To prove this theorem, we show that for every tree in DCT_m there is a path connecting it to a ranked caterpillar tree. Because a ranked caterpillar tree can be transformed into any other ranked caterpillar tree by a sequence of NNI moves, we can then conclude that DCT_m is a connected graph.

At first, we see that every tree in DCT_m is connected to a ranked tree. We therefore iterate through the internal nodes of a discrete coalescent tree from bottom to top and decrease the time of each node by length moves until it cannot be decreased further, i.e. it reaches time t such that there is another node with time $t-1$. After iterating through all nodes, we receive a ranked tree where all intervals have length one.

It hence remains to show that any ranked tree is connected to a caterpillar tree. We prove the existence of a path from any ranked tree T to a caterpillar tree by induction on the number of rank intervals in T.

As a tree with zero rank intervals is a caterpillar tree, the statement is obviously true for the base case. For the induction step we assume that T is a ranked tree with $i+1$ rank intervals, and that every tree with less than or equal to i rank intervals is connected to a caterpillar tree. We only need to prove that T is connected to a tree with i rank intervals, as this gives us the desired result by applying the induction hypothesis.

Let $[(T)_k, (T)_{k+1}]$ be the highest rank interval in T, i.e. there is no rank interval $[(T)_{k'}, (T)_{k'+1}]$ with $k' > k$. It follows that all intervals above this one are edge intervals. Therefore, the parent of $(T)_k$ has rank higher than $k+1$ and is the most recent common ancestor of $(T)_k$ and $(T)_{k+1}$. Because all intervals above the node with rank $k+1$ are edges, the rank of the most recent common ancestor of $(T)_k$ and $(T)_{k+1}$ be can decreased by NNI moves as described in the following.

Because the parent of $(T)_k$ has rank $l > k+1$ and we assumed that $[(T)_k, (T)_{k+1}]$ is the highest rank interval, there is an edge between $(T)_{l-1}$ and $(T)_l$. We can hence perform an NNI move on the interval $[(T)_{l-1}, (T)_l]$ that swaps the subtree rooted in $(T)_k$ with the subtree rooted in the child of $(T)_{l-1}$ that does not contain $(T)_{k+1}$. An example of such a move with $l = k+3$ is depicted in Figure 3.6. After this NNI move, the node with rank $l-1$ is parent of $(T)_k$, i.e. the rank of the *mrca* of $(T)_k$ and $(T)_{k+1}$ is decreased by one.

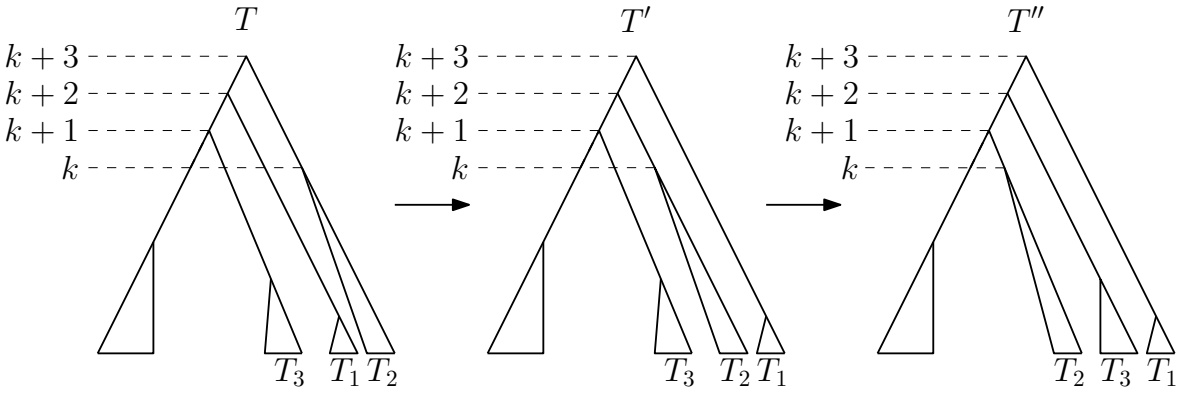


Figure 3.6: The NNI move on the left between T and T' swaps the subtrees T_1 and T_2 and thereby decreases the rank of the *mrca* of $(T)_k$ and $(T)_{k+1}$. The NNI move between T' and T'' swaps the subtrees T_2 and T_3 and results in a tree T'' with only edge intervals above rank k.

Such NNI moves can be repeated until we receive a tree T' where $(T')_k$ and $(T')_{k+1}$ share their parent with rank $k+2$. Performing a further NNI move on the edge $[(T')_{k+1}, (T')_{k+2}]$ that swaps the subtree rooted in $(T')_k$ with a subtree rooted in one of the children of $(T')_{k+1}$ results in a tree T'' where the parent of $(T'')_k$ has rank $k+1$, i.e. the interval $[(T'')_k, (T'')_{k+1}]$ is an edge. Furthermore, all intervals above this one remain edge intervals in T''. Since none of the intervals below $[(T'')_k, (T'')_{k+1}]$ have been changed on the path from T to T'', it follows that T'' has i rank intervals, one less than T. □

We can hence interpret the graphs DCT_m and RNNI as metric spaces, so we refer to them as (discrete) tree spaces (DCT_m space and RNNI space).

Theorem 3.2. *The number of ranked trees, and hence the number of vertices in the* RNNI *graph, equals*

$$\frac{n!(n-1)!}{2^{n-1}}.$$

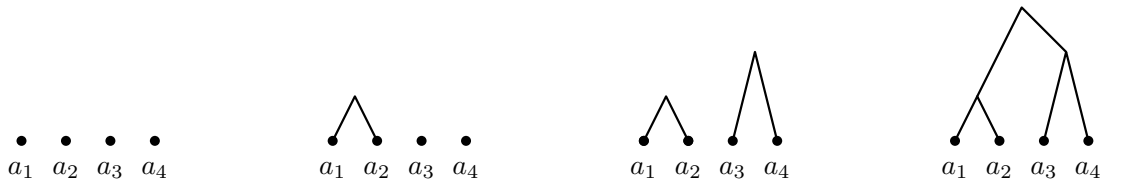
Our proof of Theorem 3.2 follows the one suggested by Steel (2016). We therefore use a model for constructing phylogenetic trees, the *coalescent process*. It is also known as *Kingman coalescent process* (Kingman 1982) and produces trees with branch lengths using a bottom-up approach, from the leaves towards the root of a tree. In every step of the process two lineages are chosen to coalesce, meaning that a new internal node is introduced as parent of two already existing nodes, until only one lineage remains, i.e. only one node (the root) is left. One important detail for us is that under this model no two coalescent events can happen at the same time, resulting in a total order on the internal nodes of the constructed tree, i.e. a ranking of internal nodes.

Algorithm 1 Coalescent Process

1: $\mathcal{S} = \{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$
2: T is a graph consisting of n nodes labelled by $a_1, \ldots, a_n$
3: **for** $i = 1, \ldots, n-1$ **do**
4: Pick two elements $X, Y \in \mathcal{S}$
5: Add new node with rank i to T with the nodes inducing X and Y as children
6: $\mathcal{S} = (\mathcal{S} \setminus X, Y) \cup (X \cup Y)$
7: **return** T

We now describe this process in more detail, as we need it in the proof of Theorem 3.4. Since we are only interested in ranked trees, we do not consider branch lengths and focus on how ranked trees are constructed under the coalescent process.

To produce a ranked tree on the leaf set $\{a_1, \ldots, a_n\}$, we start by considering the set $\mathcal{S}$ containing all leaves as one-element sets, i.e. $\mathcal{S} = \{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$. We also interpret $\mathcal{S}$ as set of clusters induced by nodes of a graph T. For $\mathcal{S} = \{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$, T consists of n nodes labelled by $a_1, \ldots, a_n$, and no edges. In each of the $n-1$ iterations of the coalescent process (Algorithm 1), two elements X, Y of the set $\mathcal{S}$ are chosen at random. These elements are removed from S and replaced with a new set $X \cup Y$. In the graph T we add a new node as parent of the nodes inducing clusters X and Y. This new node introduced in iteration $i \in \{1, \ldots, n-1\}$ is furthermore assigned rank i. After iteration $n-1$, T is a ranked tree (Steel 2016). The coalescent process as described here is hence producing a ranked tree in a bottom-up approach, from leaves to the root. Since in every iteration, any pair of lineages is allowed to coalesce, any ranked tree can result from this process. The pseudo-code for this process is given in Algorithm 1 and an example of its execution is illustrated in Figure 3.7.



$\{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}$ $\{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}$ $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ $\{\{a_1, a_2, a_3, a_4\}\}$

Figure 3.7: Building a coalescent tree on four leaves. Shown is the graph T before the first iteration (on the left), followed by the graphs after iteration one to three (from second to left to right), together with the corresponding sets $\mathcal{S}$ (at the bottom). The graph after the third iteration is a ranked tree.

To prove Theorem 3.2 it is hence sufficient to show that the number of different ranked trees that can be produced by the coalescent process is $\frac{n!(n-1)!}{2^{n-1}}$.

Proof (Theorem 3.2). We prove the theorem by showing that the coalescent process (Algorithm 1) can produce $\frac{n!(n-1)!}{2^{n-1}}$ different ranked trees. In each iteration i two sets of $\mathcal{S}$ are randomly chosen to define the children of the newly introduced node. There are $\binom{|\mathcal{S}|}{2}$ possible choices for these two sets in each iteration. Note that $\mathcal{S}$ contains n elements before the first iteration and that in every iteration the number of elements in $\mathcal{S}$ decreases by one, as two elements X and Y are replaced by one element $X \cup Y$. $\mathcal{S}$ hence contains $n - i + 1$ elements at the beginning of iteration i. Therefore, there

are $\binom{n-i+1}{2}$ two-element subsets that can be chosen as clusters induced by the children of the node of rank i.

Two ranked trees are equal if, and only if, at every iteration the exact same lineages are chosen to coalesce. In other words, the sequence of sets that unite in iterations $i = 1, \ldots, n-2$ of the coalescent process uniquely define a ranked tree. The number of ranked trees that can result from this process is hence the product of all possible different choices at rank i for $i = 1, \ldots, n-1$:

$$\prod_{i=1}^{n-1} \binom{n-i+1}{2} = \prod_{i=1}^{n-1} \frac{(n-i+1)(n-i)}{2} = \frac{n!(n-1)!}{2^{n-1}}.$$

□

From the number of ranked trees on n leaves we can infer the number of discrete coalescent trees on n leaves with maximum root time m (Corollary 3.3). Remember that Gavryushkin, Whidden, and Matsen (2018) define a tree space similar to DCT_m. While the set of trees in their tree space DtTu_m is the same as in DCT_m, the moves and hence edges in this tree space are defined in a slightly different way. Corollary 3.3 hence corrects the number of trees in DCT_m given in (Gavryushkin, Whidden, and Matsen 2018, Lemma 3).

Corollary 3.3. *The number of discrete coalescent trees on n leaves with maximum root time m, or in other words the number of vertices in* DCT_m*, is*

$$\binom{m}{n-1} \frac{n!(n-1)!}{2^{n-1}}$$

Proof. We first count the number of discrete coalescent trees with maximum root time m that have a fixed underlying ranked tree T. Since we allow a maximum root time of m, the $n-1$ internal nodes of T can have times in $\{1, \ldots, m\}$. There are hence $\binom{m}{n-1}$ possible ways of choosing the times of these internal nodes to construct a discrete coalescent tree that has T as underlying ranked tree. For each choice of T and the $(n-1)$-element subset of times of internal nodes in $\{1, \ldots, m\}$, the tree resulting from assigning these times to T is unique. Furthermore, any discrete coalescent tree is uniquely defined by its underlying ranked tree and the set of times of its internal nodes. The number of discrete coalescent trees on n leaves with maximum root time m is hence the product of the number of ranked trees and the number of possible assignments of times to their internal nodes: $\binom{m}{n-1} \frac{n!(n-1)!}{2^{n-1}}$ □

Theorem 3.4. *Let e_n be the number of edges in the* RNNI *graph corresponding to* NNI *moves and e_r the number of edges corresponding to rank moves. Then*

$$e_n = 2\ \frac{n!(n-1)!}{2^{n-1}}\ \sum_{i=2}^{n-1} \frac{1}{i} \quad \text{and} \quad e_r = \frac{1}{2}\frac{n!(n-1)!}{2^{n-1}}\left(n-2-2\ \sum_{i=2}^{n-1}\frac{1}{i}\right),$$

and the total number of edges in RNNI *is*

$$e_n + e_r = \frac{n!(n-1)!}{2^{n-1}}\left(\left(\sum_{i=2}^{n-1}\frac{1}{i}\right) + \frac{n-2}{2}\right)$$

The proof of this theorem follows a strategy similar to the one in the proof of Theorem 3.2, using the coalescent process to count trees.

Proof. We start by counting the number of edges in the RNNI graph that correspond to NNI moves. Three ranked trees are connected by an NNI move if contracting an edge of length one in each ranked tree results in the same non-binary ranked tree T^*, which contains exactly one node with three children. We assume that after contracting an edge to a node, ranks in the resulting (non-binary) ranked tree get updated such that the order of ranks is preserved from the initial ranked tree, but ranks are in $\{1, \ldots, n-2\}$. An example of this is illustrated in Figure 3.8. We call a node with three children a *trifurcation*, and a ranked tree that is binary except for one trifurcation a *ranked tree with a trifurcation*. For a fixed ranked tree with a trifurcation T^* there are exactly three (binary) ranked trees in RNNI where contracting an edge of length one results in T^*. Furthermore, if two trees are NNI neighbours in RNNI, then there is a unique edge on which the NNI move connecting the two trees is performed. We can therefore count the number of ranked trees with a trifurcation and multiply it by three to get the number of edges in RNNI that correspond to NNI moves.

For counting the number of ranked trees with a trifurcation we use a slight modification of the coalescent process (Algorithm 1) that allows trees to have exactly one trifurcation. When a trifurcation appears in a coalescent process, three lineages coalesce instead of just two. The number of possible choices for lineages to coalesce to a trifurcation is hence $\binom{i}{3}$, if i is the number of present lineages. Furthermore, the number of internal nodes of ranked trees with a trifurcation is $n-2$, one less than the number of internal nodes of ranked trees. We now summarise our previous observations to count the number of ranked trees with a trifurcation at rank k separately for each $k = 1, \ldots, n-2$ and then sum them up over all k.

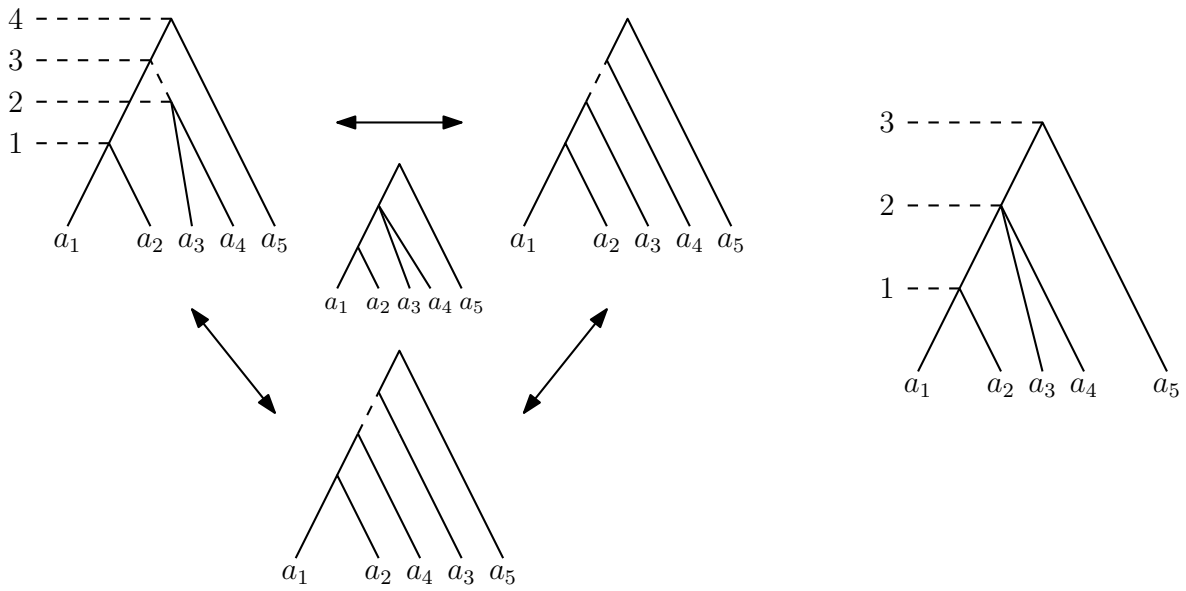


Figure 3.8: Example of three ranked trees connected by NNI moves on the dashed edges on the left. The ranked tree with a trifurcation in the middle results from contracting the dashed edge in each tree to a node. This tree is depicted larger on the right, where the ranks of internal nodes are annotated. Note that we assume that after contracting an edge the ranks are adjusted so that the resulting tree has ranks in $\{1, \ldots, n-2\}$, which in this case is $\{1, 2, 3\}$.

In every ranked tree with a trifurcation at rank k, all nodes with rank less than k result from a coalescent event of two lineages, just as in the standard coalescent process. At rank k, three lineages are chosen to coalesce, resulting in $\binom{n-k+1}{3}$ different choices for lineages to coalesce. Afterwards, the coalescent process continues as before. This means for the node of rank $k+1$ directly above the trifurcation that two out of $n-k-1$ lineages coalesce, giving $\binom{n-k-1}{2}$ possible pairs of lineages to coalesce. Similarly, for all nodes with rank j greater than k, there are $\binom{n-j}{2}$ different choices for lineages to coalesce. We conclude that the number of ranked trees with a trifurcation at rank k equals

$$
\begin{aligned}
&\binom{n}{2}\binom{n-1}{2}\cdots\binom{n-k+2}{2}\binom{n-k+1}{3}\binom{n-k-1}{2}\cdots\binom{2}{2}\\
=&\left(\prod_{i=1}^{k-1}\binom{n-i+1}{2}\right)\binom{n-k+1}{3}\left(\prod_{i=2}^{n-k-1}\binom{j}{2}\right)\\
=&\left(\prod_{i=1}^{n}\binom{i}{2}\right)\binom{n-k+1}{3}\frac{1}{\binom{n-k+1}{2}\binom{n-k}{2}}\\
=&\left(\prod_{i=1}^{n}\binom{i}{2}\right)\frac{(n-k+1)(n-k)(n-k-1)}{3\cdot 2}\frac{2\cdot 2}{(n-k+1)(n-k)(n-k)(n-k-1)}\\
=&\left(\prod_{i=1}^{n}\binom{i}{2}\right)\frac{2}{3(n-k)}
\end{aligned}
$$

The number of ranked trees with a trifurcation is the sum of all ranked trees with a trifurcation at rank k for all $k = 1, \ldots, n-2$:

$$
\begin{aligned}
&\sum_{k=1}^{n-2}\left(\prod_{i=1}^{n}\binom{i}{2}\right)\frac{2}{3(n-k)}\\
=&\frac{2}{3}\left(\prod_{i=2}^{n}\binom{i}{2}\right)\sum_{k=1}^{n-2}\frac{1}{n-k}\\
=&\frac{2}{3}\frac{n!(n-1)!}{2^{n-1}}\sum_{k=2}^{n-1}\frac{1}{k}
\end{aligned}
$$

Since each of these ranked trees with a trifurcation at rank k is associated with three NNI edges in the RNNI graph, we multiply this number by three to get the number of edges that correspond to NNI moves:

$$
e_n = 2\frac{n!(n-1)!}{2^{n-1}}\sum_{i=2}^{n-1}\frac{1}{i}
$$

For counting the total number of edges in the RNNI graph we use a strategy similar to counting NNI edges. Similar to three NNI moves corresponding to one ranked tree with a trifurcation, a rank move corresponds to a ranked tree where exactly two internal nodes have equal rank, see Figure 3.9. We refer to such ranked trees as *ranked trees with a node pair of equal ranks*. We again assume that after contracting a rank interval to get a node pair of equal rank, the internal nodes are re-labelled to have labels in $\{1, \ldots, n-2\}$ while keeping the order of the original labelling, just as in the case of trifurcations.

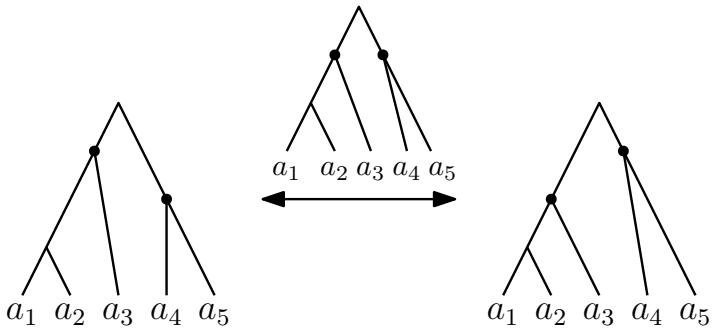


Figure 3.9: Two ranked trees (left and right) that are connected by a rank move and in the middle the ranked tree with a node pair of equal ranks corresponding to the displayed rank move.

We now establish the number of ranked trees with a node pair of equal ranks, as this number equals the number of edges in the RNNI graph that correspond to rank moves. Therefore, we first count all trees that can be received from a ranked tree by contracting an interval to a node. This interval could be an edge interval, where contracting leads to a trifurcation, or a rank interval, where contracting leads to a node pair of equal ranks. This means that we count the sum of the number of ranked trees with a trifurcation and the number of ranked trees with a node pair of equal ranks. By doing this for all $\frac{n!(n-1)!}{2^{n-1}}$ ranked trees, we count the trees corresponding to NNI moves three times, and trees corresponding to rank moves twice, i.e. we compute $3e_n + 2e_r$.

We can then derive e_r, the number of edges corresponding to rank moves. Because the number of ranked trees is $\frac{n!(n-1)!}{2^{n-1}}$ and every ranked tree has $n-2$ intervals, the total number of trees resulting from contracting an interval in a ranked tree is $(n-2)\frac{n!(n-1)!}{2^{n-1}}$. We can infer that the number of ranked trees with a node pair of equal ranks, and hence the number of edges corresponding to rank moves, is:

$$\begin{aligned} e_r &= \frac{1}{2}\left((n-2)\frac{n!(n-1)!}{2^{n-1}} - 3 \cdot \text{number of ranked trees with a trifurcation}\right) \\ &= \frac{1}{2}\left((n-2)\frac{n!(n-1)!}{2^{n-1}} - 3\frac{2}{3}\frac{n!(n-1)!}{2^{n-1}}\sum_{i=2}^{n-1}\frac{1}{i}\right) \\ &= \frac{1}{2}\frac{n!(n-1)!}{2^{n-1}}\left(n-2-2\sum_{i=2}^{n-1}\frac{1}{i}\right) \end{aligned}$$

The overall number of edges in RNNI is the sum of edges corresponding to NNI moves and edges corresponding to rank moves, and hence:

$$\begin{aligned} e_n + e_r =& 2\frac{n!(n-1)!}{2^{n-1}}\Big(\sum_{i=2}^{n-1}\frac{1}{i}\Big) + \frac{1}{2}\frac{n!(n-1)!}{2^{n-1}}\left(n-2-2\sum_{i=2}^{n-1}\frac{1}{i}\right) \\ =& \frac{n!(n-1)!}{2^{n-1}}\left(2\Big(\sum_{i=2}^{n-1}\frac{1}{i}\Big) + \frac{1}{2}\Big(n-2-2\sum_{i=2}^{n-1}\frac{1}{i}\Big)\right) \\ =& \frac{n!(n-1)!}{2^{n-1}}\left(\Big(\sum_{i=2}^{n-1}\frac{1}{i}\Big) + \frac{n-2}{2}\right) \end{aligned}$$

□

Chapter 4

The Shortest Path Problem

For most applications, including analyses of distributions of trees in a tree space, it is important to be able to compute distances between trees efficiently. Often a shortest path between trees is needed instead of just their distance, for example in graph-based summary methods. We therefore discuss the problem of computing shortest paths in our tree space DCT_m in this chapter.

The problem can be stated in its general form as follows:

Shortest Path Problem

INSTANCE: A pair of trees T and R on n leaves in tree space $\mathcal{T}$

PROBLEM: Find a path of minimal length between T and R in $\mathcal{T}$

Since the distance is defined as the length of a shortest path, the Shortest Path Problem can be seen as an extension of the problem of computing the distance between two trees. If it is $\mathcal{NP}$-hard to compute distances in a tree space $\mathcal{T}$, computing shortest paths in $\mathcal{T}$ is also $\mathcal{NP}$-hard. This implies that the Shortest Path Problem is $\mathcal{NP}$-hard in tree spaces NNI, SPR, and TBR.

In this chapter we show that the Shortest Path Problem is solvable in time polynomial in n for the RNNI space and in time polynomial in n and m for the DCT_m space. We therefore first introduce the algorithm FindPath for ranked trees in Section 4.1 and prove that it computes shortest paths in RNNI. To our knowledge, RNNI provides the first rearrangement based distance measure for trees that can be computed in polynomial time. We then explain in Section 4.2 how FindPath can be used to compute shortest paths in DCT_m as well. We furthermore generalise DCT_m to define a tree space on non-ultrametric trees, where leaves are not all assigned the same time, in Section 4.3. We show that distances between such trees can be computed in polynomial time as well.

4.1 Shortest Paths in RNNI

In this section we introduce the algorithm FindPath (Section 4.1.1) for computing paths between ranked trees. We will first discuss some properties of this algorithm before we prove that it actually computes shortest paths (Section 4.1.2). We conclude this section by showing that FindPath is an optimal algorithm for computing shortest paths in RNNI (Section 4.1.3). Our results in particular imply that the Shortest Path Problem in RNNI is in the complexity class $\mathcal{P}$, unlike NNI, SPR, and TBR, for which the Shortest Path Problem is $\mathcal{NP}$-hard (Dasgupta et al. 2000; Bordewich and Semple 2005; Hickey et al. 2008; Allen and Steel 2001). Before we introduce FindPath, we show that shortest paths in RNNI are not unique.

Proposition 4.1. *Shortest paths in* RNNI *are not unique for* $n > 3$.

We prove Proposition 4.1 by providing an example of ranked trees that have $(\frac{n-1}{2})!$ shortest paths between them in RNNI. This implies that there is no algorithm to compute all shortest paths between two trees in time polynomial in n.

Proof. We assume for this proof that $n > 3$ is odd. The proof also works for even n, one just needs to replace $n-1$ by $n-2$ throughout this proof. Let T and R be the following trees (Figure 4.1, differences in clusters are indicated in bold):

$$T = [\{a_1, \boldsymbol{a_2}\}, \{a_1, a_2, a_3\}, \{a_1, a_2, a_3, \boldsymbol{a_4}\}, \{a_1, a_2, a_3, a_4, a_5\}, \ldots, \{a_1, a_2, \ldots, a_n\}]$$
$$R = [\{a_1, \boldsymbol{a_3}\}, \{a_1, a_2, a_3\}, \{a_1, a_2, a_3, \boldsymbol{a_5}\}, \{a_1, a_2, a_3, a_4, a_5\}, \ldots, \{a_1, a_2, \ldots, a_n\}]$$

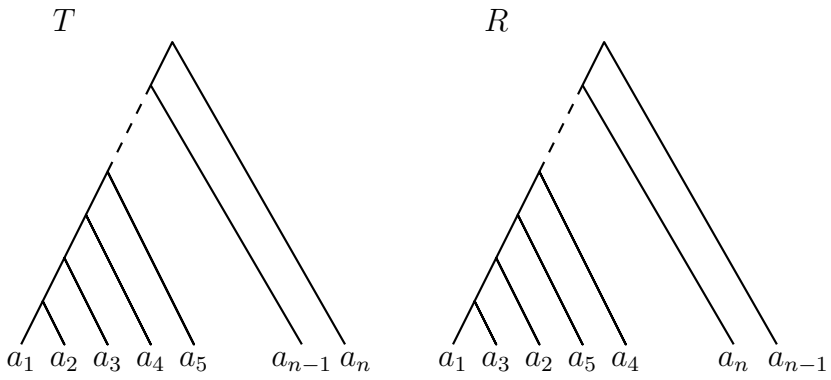


Figure 4.1: Caterpillar trees T and R as described in the proof of Proposition 4.1

Any shortest path between T and R needs to perform NNI moves to swap the leaf pairs (a_i, a_{i+1}) for all even $i < n$. The number of moves on a shortest path between T

and R is hence $\frac{n-1}{2}$. Since these NNI moves are all independent of each other, they can be performed in any order to receive a shortest paths. It follows that there are $(\frac{n-1}{2})!$ shortest paths between T and R, hence shortest paths in RNNI are not unique. □

4.1.1 The Algorithm FindPath

In this section we introduce the algorithm FindPath for computing paths between ranked trees in RNNI. After a detailed description of the algorithm, we prove that it is a correct deterministic algorithm with running time quadratic in the number of leaves (Proposition 4.3). We furthermore establish some properties of paths computed by FindPath in Lemmas 4.2, 4.4, and 4.5.

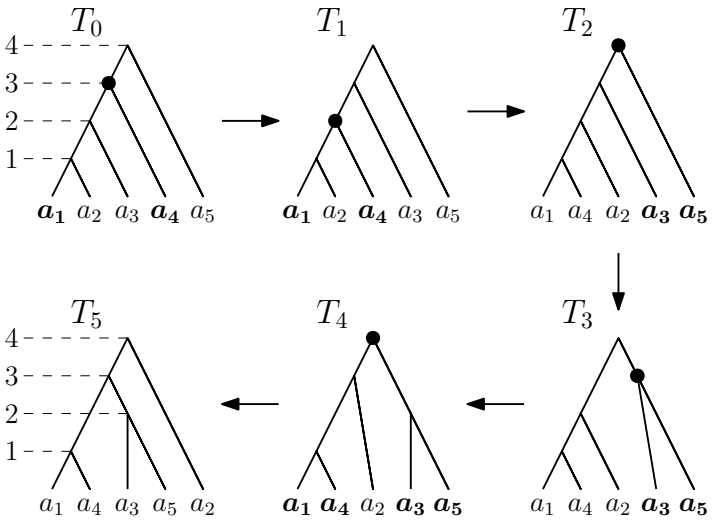
Algorithm 2 FindPath(T, R)

```
1: T_1 := T, p := [T_1], [C_1, ..., C_{n-1}] := R
2: for k = 1, ..., n − 2 do
3:    while rank(mrca(C_k)_{T_1}) > k do
4:       if mrca(C_k)_{T_1} and node u with rank one less than mrca(C_k)_{T_1} in T_1 are connected by an edge then
5:          T_2 is T_1 with the rank of mrca(C_k)_{T_1} decreased by an NNI move
6:       else
7:          T_2 is T_1 with ranks of u and mrca(C_k)_{T_1} swapped
8:       T_1 = T_2
9:       p = p + T_1
10: return p
```

The input of FindPath (Algorithm 2) is two ranked trees T and R, which we assume to be given in their cluster representation. We denote the representation of R by $[C_1, \ldots, C_{n-1}]$. The algorithm considers the clusters $C_1, \ldots, C_{n-2}$ iteratively in this order and produces a sequence p of trees which is a path from T to R when the algorithm terminates. The path p initially consists only of the tree T: $p = [T]$. In each iteration $k = 1, \ldots, n-2$, new trees are added to p if necessary (as described below). We refer to the last tree added to p as T_1. In iteration $k \in \{1, \ldots, n-2\}$ the rank of the most recent common ancestor of the cluster C_k in T_1, $mrca(C_k)_{T_1}$, is decreased by RNNI moves until it has rank k. C_k is hence induced by the node of rank k in T_1 at the end of iteration k. Since C_k is defined as the cluster induced by the node of rank k in R, all clusters induced by nodes of rank less than or equal to k after iteration k

coincide in T_1 and R. In Proposition 4.3 we show that if $mrca(C_k)_{T_1} > k$, then there is always exactly one RNNI move on T_1 decreasing the rank of $mrca(C_k)_{T_1}$, which implies that FindPath is a deterministic algorithm.

In the following we refer to the path computed by FindPath between ranked trees T and R as $\mathrm{FP}(T, R)$. An example of path computed by FindPath between two given ranked trees is illustrated in Figure 4.2.



tree	i	cluster
T_0	1	$\{a_1, a_4\}$
T_1	1	$\{a_1, a_4\}$
T_2	2	$\{a_3, a_5\}$
T_3	2	$\{a_3, a_5\}$
T_4	3	$\{a_1, a_3, a_4, a_5\}$

Figure 4.2: Left: Example for a path computed by FindPath between two trees T_0 and T_5. The leaves that are in the cluster whose $mrca$ is considered in the current tree are highlighted in bold. Right: For each tree T_k the iteration i in which the $mrca$ of a cluster of R is considered in T_k and the corresponding cluster. These most recent common ancestors are marked in the trees on the left.

A property of paths computed by FindPath that follows directly from the definition of this algorithm and is essential for establishing properties of the tree spaces RNNI and DCT_m in Chapter 5 is the following:

Lemma 4.2. *Let T and R be ranked trees in* RNNI *and let T' be the tree after iteration i of* FindPath *applied to trees T and R.*

All nodes $(T')_j$ induce the same clusters as $(R)_j$ for $j \leq i$.

Proof. This follows directly from the definition of FindPath. □

We are now ready to show that FindPath is a deterministic algorithm that computes a path between any two ranked trees T and R in RNNI.

Proposition 4.3. FINDPATH *is a correct deterministic algorithm that runs in* $\mathcal{O}(n^2)$ *time.*

Proof. To show that FINDPATH (Algorithm 2) is a deterministic algorithm, we have to prove that tree T_2 constructed in the while loop (line 3) of the algorithm always exists and is uniquely defined. If T_2 is obtained in line 7 from T_1 by a rank move, the tree exists and is unique because there always exists exactly one rank move on any particular interval that is not an edge. It remains to show that an NNI move that decreases the rank of $mrca(C_k)_{T_1}$ always exists and is unique, if the interval of which $mrca(C_k)_{T_1}$ is the upper bound is an edge. To prove this we consider cases $k = 1$ and $k > 1$ separately.

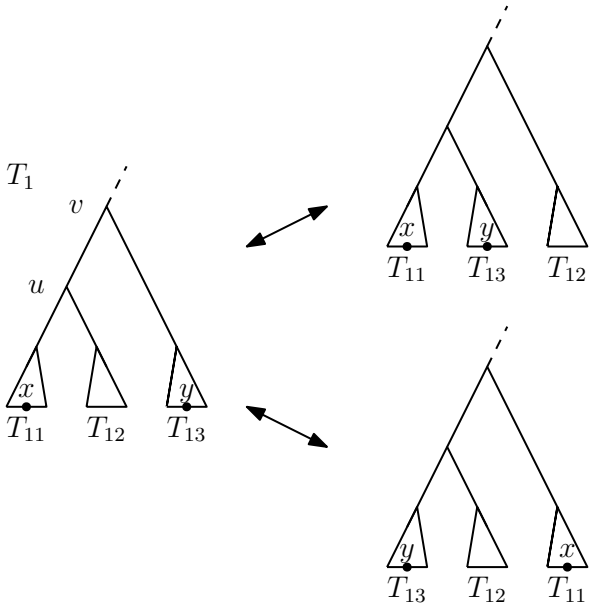


Figure 4.3: The ranked tree T_1 on the left with internal nodes u and v and subtrees T_{11}, T_{12}, T_{13} and leaves x and y as described in case $k = 1$ of the proof of Proposition 4.3. On the right are its NNI neighbours that result from an NNI moves on the edge $[u, v]$. Only in the one on the top right the rank of $mrca(\{x, y\})$ is lower than in T_1.

Case $k = 1$**.** In this case C_k consists of two leaves, i.e. $C_k = \{x, y\}$ for two leaves $x, y \in \{a_1, \ldots, a_n\}$. Since we assumed that the **while** condition is satisfied, the node $v = mrca(\{x, y\})_{T_1}$ has rank $r > 1$. Consider the node $u := (T_1)_{r-1}$ with rank $r - 1$ in T_1. Because we assumed that the interval of which $mrca(C_k)_{T_1}$ is an upper bound is an edge, u is child of v. We know that each of the two clusters induced by the children of v has either x or y in it, as v is the most recent common

ancestor of $\{x, y\}$. We can therefore assume without loss of generality that x is in the cluster induced by u, so y has to be outside this cluster. Consider the following three disjoint subtrees of T_1: the subtree T_{11} induced by a child of u and containing x, the subtree T_{12} induced by the other child of u, and the subtree T_{13} induced by a child of v and containing y. The tree T_1 with these subtrees is illustrated in Figure 4.3. Now observe that out of two NNI moves possible on the edge $[u, v]$ in T_1, only the one that swaps T_{12} and T_{13} decreases the rank of the most recent common ancestor of $\{x, y\}$. Hence T_2 exists and is unique in this case.

Case $k > 1$. In this case $C_k = C_i \cup C_j$ for $i, j < k$. Note that C_i or C_j could be leaves, which we treat as clusters containing only one element, the leaf itself, throughout this proof. FindPath considers internal nodes according to increasing ranks, so in iteration $k > i, j$ the clusters C_i and C_j have already been considered and are present in T_1 (Lemma 4.2). Therefore, the subtree of T_1 induced by $mrca(C_i)_{T_1}$ is identical to the subtree of R induced by $mrca(C_i)_R$, and the same is true for $mrca(C_j)_{T_1}$ and $mrca(C_j)_R$. We can hence reduce this case to $k = 1$ by suppressing C_i and C_j in both T_1 and R to new leaves c_i and c_j (of rank zero) respectively. As in Case $k = 1$, exactly one of two possible NNI moves decreases the rank of the most recent common ancestor of $\{c_i, c_j\}$ in T_1, so the same is true for the most recent common ancestor $mrca(C_k)_{T_1}$, and T_2 is unambiguously defined.

Thus, FindPath is a deterministic algorithm.

It remains to show that the output of FindPath is a path in RNNI between the input trees T and R. Therefore, first note that the algorithm starts by adding T to the output path. Every new tree added to the output path is an RNNI neighbour of the previously added one (see line 5 and 7). To see that the output path terminates in R, observe that after k iteration of the for loop (line 2) of the algorithm, the first k clusters of T_1 and R must coincide. It follows that after $n - 2$ iterations all clusters $C_1, \ldots, C_{n-2}$ are present in T_1. Since C_{n-1} contains all leaves in every ranked tree and trees are uniquely defined by their clusters, the tree after iteration $n - 2$ is R and a path between T and R is constructed.

The worst-case time complexity of FindPath is quadratic in the number of leaves, as there can be at most $n-2$ executions of the for loop (line 2) and in every iteration of the for loop at most $n-1-k$ while loops (line 3) are executed. Note that this assumes

that $mrca(C_k)_{T_1}$ in line 3 can be computed in constant time. Here and throughout the paper we assume that the output of FindPath is encoded as a list of RNNI moves rather than a list of trees. This is because writing out a tree on n leaves takes time linear in n (depending on the data structure) and the complexity of FindPath becomes cubic. □

An efficient implementation of the algorithm FindPath for RNNI is available on GitHub (Collienne and Berling 2021).

We now establish an important property of $\mathrm{FP}(T, R)$, for which we first need the following definition: A path between two trees T and R *preserves* a cluster C that is present in both T and R, if every tree on this path contains C as a cluster. In Lemma 4.4 we establish that the paths computed by FindPath preserve clusters in RNNI, i.e. if trees T and R share a cluster, every tree on $\mathrm{FP}(T, R)$ also has this cluster.

Lemma 4.4. *For two ranked trees T and R in* RNNI *sharing a cluster, the path $\mathrm{FP}(T, R)$ preserves this cluster.*

Proof. Let T and R be ranked trees that share a cluster C. We prove the lemma by assuming to the contrary that there is a ranked tree on $\mathrm{FP}(T, R)$ that does not contain C. Let T' be the first ranked tree on $\mathrm{FP}(T, R)$ that does not have C as a cluster. Since rank moves only change the ranks of two internal nodes, i.e. the order of clusters in the cluster representation of a ranked tree, only NNI moves can actually change clusters. And because T' is the first ranked tree on $\mathrm{FP}(T, R)$ not containing the cluster C, there is an NNI move between T' and the tree $\hat{T}$ preceding T' on $\mathrm{FP}(T, R)$. Let A and B be the clusters induced by the children of the node inducing C in $\hat{T}$. Hence $A \cup B = C$. Let furthermore D be the cluster induced by the node in $\hat{T}$ that has the same parent as the node inducing C. An illustration of $\hat{T}$ and its clusters A, B, C, and D can be found on the left of Figure 4.4. We can assume without loss of generality that the cluster C in $\hat{T}$ is replaced by the cluster $A \cup D$ in T', as depicted in Figure 4.4. We otherwise swap the names of A and B.

Every move on FindPath decreases the rank of the *mrca* of a cluster of R in the currently last tree on the path that gets created iteratively by FindPath. Therefore, the NNI move between $\hat{T}$ and T' decreases the rank of the *mrca* of a cluster C_k of R. Because the new cluster created in T' is $A \cup D$, it follows $C_k \subseteq A \cup D$. C_k therefore contains elements of both A and D, but none of B. Since every cluster is the union of either two clusters, which are induced by nodes with lower rank, a cluster and a leaf, or two leaves, it follows that any cluster containing all elements of A (except for A itself)

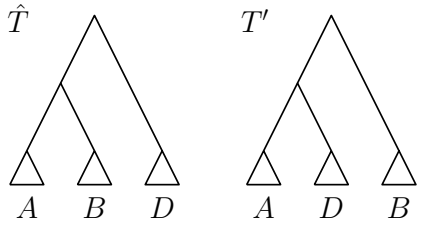


Figure 4.4: Ranked trees $\hat{T}$ and T' as described in the proof of Lemma 4.4. The cluster $C = A \cup B$ is present in $\hat{T}$, but not in T'.

also contains elements D. There can hence not be a node in R that induces $C = A \cup B$, which contradicts our assumption that T and R have the cluster C. We conclude that there cannot be a ranked tree T' on $\mathrm{FP}(T, R)$ that does not contain C. □

With Lemma 4.4 we can show that if two trees share a cluster, the iteration of FindPath that considers this cluster contains only rank moves:

Lemma 4.5. *Let T and R be ranked trees in* RNNI *that both contain a cluster C. If $(R)_i$ is the node inducing C in R, then there are only rank moves in iteration i of* $\mathrm{FP}(T, R)$.

Proof. Because FindPath preserves clusters (Lemma 4.4), C is present as cluster in every tree on $\mathrm{FP}(T, R)$. This in particular includes the tree T' at the beginning of iteration i, in which the cluster C is considered. As every cluster is the union of two clusters induced by nodes with lower rank in the tree, the two clusters that unite to C must be induced by nodes with rank less than i in R. Note that these clusters might be singletons (leaves). Since all nodes with rank less than i induce the same clusters in T' and R (Lemma 4.2), the two clusters uniting to C are induced by nodes of the same ranks in T' as in R. In other words, the children of $mrca(C)$ have rank less than i in T'. Because $mrca(C) \geq i$ in iteration i, it follows that there cannot be an edge between $mrca(C)$ and its children until $\mathrm{rank}(mrca(C)) = i$ at the end of iteration i. But since there is always exactly one RNNI move to decrease the rank of $mrca(C)$ until it reaches i (Proposition 4.3), all moves in iteration i are rank moves. □

4.1.2 FindPath finds shortest paths

In this section we prove that the Shortest Path Problem in RNNI can be solved in time polynomial in the number of leaves n. We do this by proving that FindPath computes shortest paths and therefore solves this problem.

The main idea of our proof is to show that a local property (Lemma 4.7) of the FindPath algorithm is enough to establish that the output paths are shortest. This property can intuitively be understood as FindPath always choosing the best tree possible to go to. One can hence see FindPath as a greedy algorithm for the Shortest Path Problem in RNNI.

Theorem 4.6. FindPath *computes shortest paths in* RNNI.

We later show that the running time of FindPath is in $\mathcal{O}(n^2)$, which implies with Theorem 4.6 that the Shortest Path Problem in RNNI is solvable in polynomial time (Corollary 4.9).

The following lemma reduces the proof of Theorem 4.6 to proving a local property of $\mathrm{FP}(T, R)$, the path between ranked trees T and R computed by FindPath.

Lemma 4.7. *If for all trees T, T', and R, where T and T' are* RNNI *neighbours,*

$$|\mathrm{FP}(T', R)| \geq |\mathrm{FP}(T, R)| - 1,$$

then FindPath *computes shortest paths for all pairs of ranked trees in* RNNI.

Proof. Assume to the contrary that the assumptions of the lemma are true and T and R are ranked trees with a minimum distance $d(T, R)$ such that $d(T, R) \neq |\mathrm{FP}(T, R)|$:

$$(T, R) = \operatorname*{argmin}_{\text{ranked trees } \hat{T}, \hat{R}} \{d(\hat{T}, \hat{R}) : |\mathrm{FP}(\hat{T}, \hat{R})| \neq d(\hat{T}, \hat{R})\}$$

Note that for trees $T = R$ FindPath returns the path $p = [T]$ of length zero, i.e. $|\mathrm{FP}(T, R)| = d(T, R)$. We can hence assume $d(T, R) > 0$. The minimality assumption on T and R implies that for all pairs of trees T', R' with distance less than $d(T, R)$, FindPath computes a shortest paths: $|\mathrm{FP}(T', R')| = d(T', R')$. Since FindPath computes a path in RNNI (Proposition 4.3), and the length of any path is an upper bound for the distance, we can infer from $d(T, R) \neq |\mathrm{FP}(T, R)|$ that $d(T, R) < |\mathrm{FP}(T, R)|$. Let T' be the first tree on a shortest RNNI path from T to R. Then $d(T', R) = d(T, R) - 1$, implying that the distance between T' and R is strictly smaller than that between T and R. This implies that $|\mathrm{FP}(T', R)| = d(T', R) = d(T, R) - 1 < |\mathrm{FP}(T, R)| - 1$ and hence, $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$. This however is a contradiction of the assumptions of the lemma. It follows that there cannot be trees T, R with $d(T, R) \neq |\mathrm{FP}(T, R)|$ under those assumptions. □

We are now ready to prove Theorem 4.6. At the beginning of the proof we introduce some notation, before we prove Lemma 4.8, which is essential for the following case differentiation.

Proof. From Lemma 4.7 we know that it is sufficient to show:

$$\begin{aligned} &\text{For all trees } T,\ R, \text{ and } T' \text{ such that } T' \text{ is RNNI neighbour of } T, \\ &\qquad |\mathrm{FP}(T', R)| \geq |\mathrm{FP}(T, R)| - 1 \end{aligned} \tag{4.1}$$

We will use Figure 4.5 to demonstrate our argument.

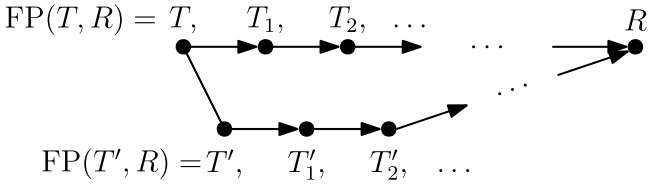


Figure 4.5: Trees T, T', and R as in inequality (4.1), i.e. T and T' are RNNI neighbours. The two paths $\mathrm{FP}(T, R) = [T, T_1, T_2, \ldots, R]$ and $\mathrm{FP}(T', R) = [T', T'_1, T'_2, \ldots, R]$ are indicated by arrows.

Assume to the contrary that T, T', and R are trees such that T' is an RNNI neighbour of T violating inequality (4.1), i.e. $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$. Out of all such triples of trees (T, T', R) choose one with minimal $|\mathrm{FP}(T, R)|$. Denote $\mathrm{FP}(T, R) = [T, T_1, T_2, \ldots, R]$ and $\mathrm{FP}(T', R) = [T', T'_1, T'_2, \ldots, R]$, and let $[(T)_t, (T)_{t+1}]$ be the interval in T on which the RNNI move connecting T and T' is performed. We furthermore assume that $[C_1, \ldots, C_{n-1}]$ is the cluster representation of R. By the definition of FindPath we know that the first move on $\mathrm{FP}(T, R)$ decreases the rank of the most recent common ancestor $mrca(C_k)_T$ of some cluster C_k of R.

To prove this theorem, we compare $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$. Therefore, we first see that the rank of C_k in T must be in $\{t, t+1, t+2\}$, where t is the rank of the lower node bounding the interval of the RNNI move between T and T':

Lemma 4.8. $\mathrm{rank}(mrca(C_k)_T) \in \{t, t+1, t+2\}$

Proof. We assume to the contrary that $\mathrm{rank}(mrca(C_k)_T) \notin \{t, t+1, t+2\}$.

At first we see that an RNNI move on an arbitrary interval on T can only change the clusters induced by nodes bounding that interval. A rank move on an interval $[(T)_r, (T)_{r+1}]$ swaps the ranks of the internal nodes $(T)_r$ and $(T)_{r+1}$ (see trees on the right of Figure 4.6) and therefore the clusters induced by these nodes. All other clusters of T stay the same. We furthermore know from Observation 3.1 that an NNI move on an edge interval $[(T)_r, (T)_{r+1}]$ only changes the cluster induced by the node with rank r. An illustration of this can be found on the top left in Figure 4.6. We can conclude that a rank move affects two clusters of a tree, while an NNI move changes one cluster.

Most importantly, only the clusters induced by the nodes that bound the interval of the RNNI move can change.

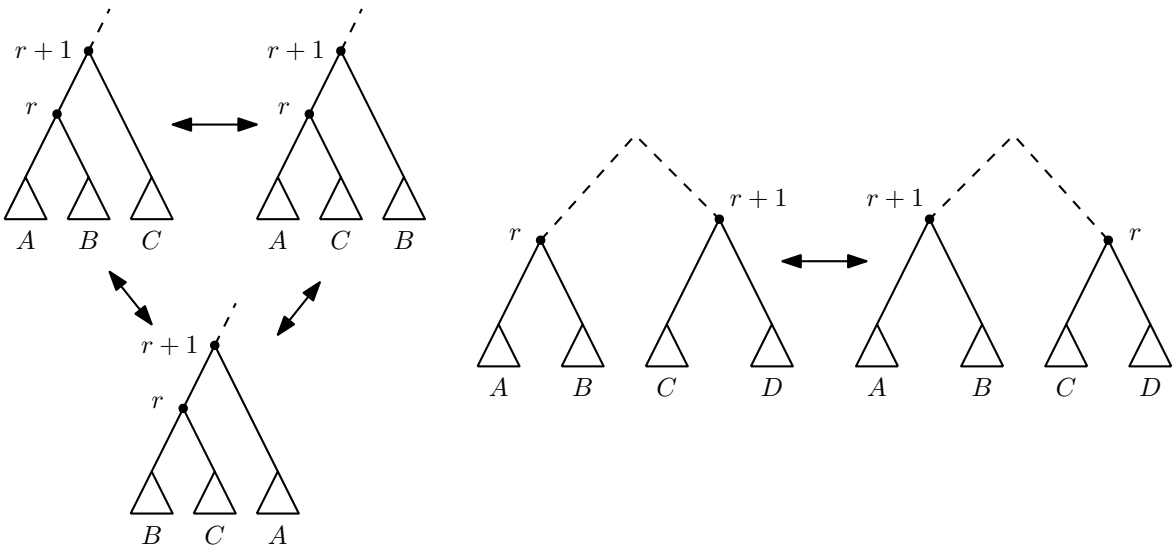


Figure 4.6: RNNI moves on interval bounded by nodes of rank r and $r+1$. NNI moves on the left and rank move on the right. The only cluster changing on the left is the cluster induced by node of rank r. On the right, the rank move swaps the two clusters induced by nodes of rank r and $r+1$. All other parts of the tree (inside the subtrees A, B, C, D and parts of trees indicated by dashed lines) are not affected by the illustrated moves.

With this in mind, we now finish the proof of this lemma. Since the first RNNI move on $\mathrm{FP}(T, R)$ decreases the rank of $mrca(C_k)_T$, it must be a move on the interval below this node (see also the proof of Proposition 4.3). By our assumption that $\mathrm{rank}(mrca(C_k)_T) \notin \{t, t+1, t+2\}$, the edge on which this move is performed cannot be $[(T)_{t-1}, (T)_t]$ (as $\mathrm{rank}(mrca(C_k)_T) \neq t$), $[(T)_t, (T)_{t+1}]$ (as $\mathrm{rank}(mrca(C_k)_T) \neq t+1$), or $[(T)_{t+1}, (T)_{t+2}]$ (as $\mathrm{rank}(mrca(C_k)_T) \neq t+2$). With the previous observation that an RNNI move on an interval can only change the cluster induced by the nodes bounding that interval, it follows that the clusters $(T)_t$ and $(T)_{t+1}$ do not change by the first move on $\mathrm{FP}(T, R)$. Remember that T and T' coincide everywhere, except for the interval $[(T)_t, (T)_{t+1}]$. Furthermore, the move that decreases the rank of $mrca(C_k)_T$ in T does not affect $(T)_t$ or $(T)_{t+1}$. Therefore, the first move on $\mathrm{FP}(T', R)$ is the same as the first move on $\mathrm{FP}(T, R)$, which decreases the rank of $mrca(C_k)$.

Let T_1 and T_1' be the trees resulting from the first moves on $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$, respectively, as in Figure 4.5. Since the first moves on $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ are the same, they change the clusters in T and T' in the same way. T_1 and T_1' are hence

connected by an RNNI move on the interval $[(T_1)_t, (T_1)_{t+1}]$, which is the same as the RNNI move between T and T'. Therefore, the trees T_1 and T_1' are RNNI neighbours. It follows that the paths $\mathrm{FP}(T_1, R)$ and $\mathrm{FP}(T_1', R)$ violate inequality (4.1) and $\mathrm{FP}(T_1, R)$ is strictly shorter than $\mathrm{FP}(T, R)$, contradicting our minimality assumption. We conclude that the first move on $\mathrm{FP}(T, R)$ has to involve an interval incident to at least one of the nodes $(T)_t$, $(T)_{t+1}$, implying $\mathrm{rank}(mrca(C_k)_T) \in \{t, t+1, t+2\}$. □

We are now ready to come to the main part of this proof, aiming to find a contradiction to our assumptions on T, T', and R. We therefore distinguish two cases: T and T' are either connected by an NNI move or by a rank move. For both rank and NNI move we further distinguish all possible moves between T and T_1. A summary of all cases we consider can be found in Table 4.1. We sometimes need to distinguish different RNNI moves, which are shown as multiple cases in one cell in Table 4.1 (e.g. cases **1.3**, **1.4**, and **1.5**), while we can at other times summarise all possible moves into one case (e.g. Case **1.1**). Note that the goal for this case differentiation is to reach a contradiction to our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$, as this proves Equation (4.1), and therefore the theorem.

		Interval of T – T_1 move		
		$t-1, t$	$t, t+1$	$t+1, t+2$
Move type T – T'	NNI	NNI **1.1** Rank **1.1**	NNI **1.2** Rank **1.2**	NNI **1.3**, **1.4** Rank **1.5**
	Rank	NNI **2.1** Rank **2.1**	NNI **2.2** Rank **2.2**	NNI **2.3** Rank **2.4**

Table 4.1: Cases we consider in the proof of Theorem 4.6. In every cell we distinguish the type of move between T and T_1 (NNI vs rank move), but these can be summarised to one case most of the times.

In all figures illustrating possible moves on $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ below, the position of the tree root is irrelevant, so we have positioned roots to simplify our figures.

Case 1. T and T' are connected by an NNI move.

Because this NNI move between T and T' is on the interval $[(T)_t, (T)_{t+1}]$, this interval is an edge in T and T'. We illustrate this in Figure 4.7, which also contains

the following notation. We denote the clusters induced by the children of $(T)_t$ by A and B and the cluster induced by the child of $(T)_{t+1}$ that is not $(T)_t$ by C. We assume without loss of generality that the NNI move between T and T' exchanges the subtrees induced by clusters B and C. Note that all clusters induced by nodes with rank less than t coincide in T and T'.

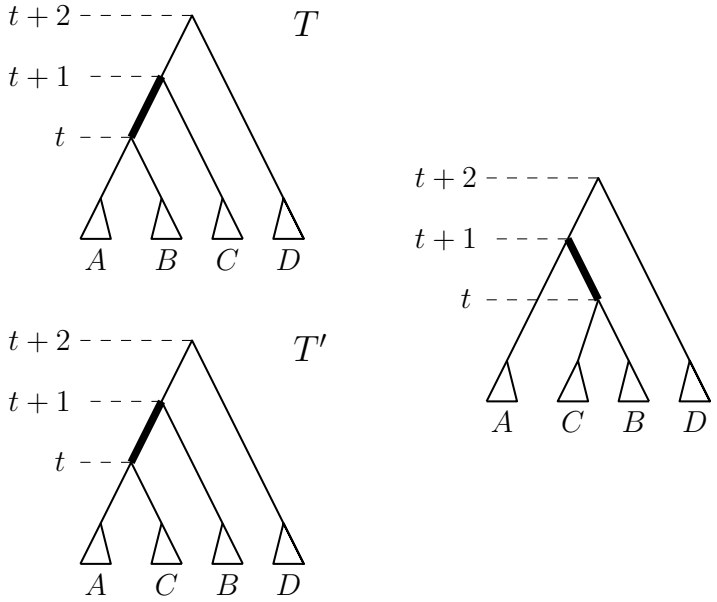


Figure 4.7: NNI move between T and T' on the edge $[(T)_t, (T)_{t+1}]$ indicated in bold, and the third RNNI neighbour resulting from a move on this edge.

We now consider all possible moves FindPath can perform to go from T to T_1 in a case differentiation. With Lemma 4.8 we know that $\text{rank}(mrca(C_k)_T) \in \{t, t+1, t+2\}$. Therefore, the move between T and T_1 is performed on one of the three intervals $[(T)_{t-1}, (T)_t]$ (case **1.1**), $[(T)_t, (T)_{t+1}]$ (case **1.2**), or $[(T)_{t+1}, (T)_{t+2}]$ (cases **1.3**, **1.4**, and **1.5**).

1.1 RNNI move (either type) on interval $[(T)_{t-1}, (T)_t]$.

Because FindPath decreases the rank of most common recent ancestors, $(T)_t$ is the most recent common ancestor of C_k in T and $(T_1)_{t-1}$ its most recent common ancestor in T_1. We can also follow from this first move on $\text{FP}(T, R)$ that the rank k of the node inducing C_k in R is less than or equal to $t-1 = \text{rank}(mrca(C_k))_{T_1}$, i.e. $k \leq t-1$. This in particular implies $k-1 < t-1$. And since all nodes with rank less than t induce the same clusters in T and T', we can infer that C_{k-1} is induced as cluster by the node of rank $k-1$ in T'. The first move on $\text{FP}(T', R)$ hence considers the cluster C_k.

Since the children of $(T)_t$ induce the clusters A and B, C_k must contain elements of both A and B. With $C_k \subseteq (A \cup B)$ we can infer that $mrca(C_k)_{T'}$ has rank $t+1$, as T' results from swapping the subtrees inducing B and C in T (see T and T' in Figure 4.7). The first move on $\mathrm{FP}(T', R)$ hence decreases the rank of $mrca(C_k)_{T'}$ from $t+1$ to t, which can only be done by an NNI move exchanging the subtrees induced by B and C (see Figure 4.7). This results in $T'_1 = T$ as the first tree after T' on $\mathrm{FP}(T', R)$, and thus $|\mathrm{FP}(T', R)| = |\mathrm{FP}(T', T)| + |\mathrm{FP}(T, R)| = 1 + |\mathrm{FP}(T, R)|$, which contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

1.2 RNNI move (either type) on interval $[(T)_t, (T)_{t+1}]$.

The interval $[(T)_t, (T)_{t+1}]$ is an edge interval, because we assume that T and T' are connected by an NNI move on this interval. The move between T and T_1 is hence an NNI move, too. If this move between T and T_1 is the same as the one between T and T', it is $T_1 = T'$ and hence $|\mathrm{FP}(T', R)| = |\mathrm{FP}(T_1, R)| = |\mathrm{FP}(T, R)| - 1$, which contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

Let us now assume that the NNI move between T and T_1 is different from the NNI move connecting T and T'. Then the cluster $B \cup C$ is induced by the node with rank t in T_1, as depicted in the tree on the right of Figure 4.7, while the node with rank t induces the cluster $A \cup C$ in T' and $A \cup B$ in T.

Because the first move on $\mathrm{FP}(T, R)$ decreases the rank of $mrca(C_k)$ and is performed on the edge bounded by nodes of rank t and $t+1$, $mrca(C_k)$ must have rank $t+1$ in T and rank t in T_1. Therefore, the rank k of the node inducing C_k in R has to be less than or equal to t. This implies $k - 1 \leq t - 1$, and since all clusters induced by nodes less than or equal to $t-1$ coincide in T and T', C_{k-1} is a cluster in T'(as in case **1.1**).

Because the $mrca$ of C_k is the node that induces the cluster $B \cup C$ in T_1, C_k contains elements of both B and C, but none of A. So $mrca(C_k)_{T'}$ is the node of rank $t+1$, as its children induce the cluster B and $A \cup C$. Therefore, FindPath applied to T' and R decreases the rank of $mrca(C_k)_{T'}$ in its first step from $t+1$ to t by swapping the subtrees induced by B and C, which results in the tree T'_1. In this tree T'_1, the node of rank t induces the cluster $B \cup C$, and the node of rank $t+1$ the cluster $A \cup B \cup C$. We have however seen before that this tree is T_1, i.e. $T'_1 = T_1$ (see Figure 4.7). This first move on $\mathrm{FP}(T', R)$ hence results in $T'_1 = T_1$, which implies $|\mathrm{FP}(T', R)| = |\mathrm{FP}(T, R)|$. This contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

We now consider the case that the first move on $\mathrm{FP}(T, R)$ is performed on the interval $[(T)_{t+1}, (T)_{t+2}]$. Since we do not know if $[(T)_{t+1}, (T)_{t+2}]$ is an edge interval or a rank interval, we consider the case that an NNI move is performed on this interval (cases **1.3**, **1.4**) separately to the case that a rank move is performed (case **1.5**). If the interval is an edge interval, we further distinguish the two possible NNI moves that could happen on this edge, resulting in the two cases **1.3** and **1.4**. We denote the cluster induced by the child of $(T)_{t+2}$ that is not $(T)_{t+1}$ by D.

1.3 NNI move on (edge) interval $[(T)_{t+1}, (T)_{t+2}]$ that swaps the subtrees induced by clusters C and D.

This move is shown in the top of Figure 4.8 by an arrow from T to T_1. Since this is the first move on $\mathrm{FP}(T, R)$, we can infer that the $mrca$ of C_k has rank $t+2$ in T and rank $t+1$ in T_1. By the definition of $mrca$, both children of $mrca(C_k)$ must contain elements of C_k. Therefore, C_k must intersect both clusters D and $A \cup B$ induced by the children of $(T_1)_{t+1}$. C_k does on the other hand not intersect with C, as no element of C descends from $mrca(C_k)$ in T_1. We now consider three cases individually: C_k intersects with both A and B, C_k intersects with A but not B, and C_k intersect with B but not A. In all three cases C_k also intersects with D. We demonstrate the three cases separately in Figures 4.8 to 4.10. Remember that T and T' are connected by an NNI move on $[(T)_t, (T)_{t+1}]$ and T and T_1 by an NNI move on $[(T)_{t+1}, (T)_{t+2}]$, which implies that these two intervals must be edges in all three trees T, T', and T_1.

1.3.1 C_k intersects A, B, and D but not C.

The $mrca$ of C_k in T_1 has rank $t+1$ and the children of this node induce the clusters C and $A \cup B$. If there was a move that decreases $mrca(C_k)_{T_1}$ further on FindPath, this move would be performed on $[(T_1)_t, (T_1)_{t+1}]$, which is an edge (see also T_1 in Figure 4.8). Such an NNI move on T_1 either swaps the nodes inducing clusters A and D, or the ones inducing cluster B and D. Neither of these moves however changes the rank of $mrca(C_k)$, as C_k intersects A, B, and D. We hence reached the end of the iteration that decreases the rank of $mrca(C_k)$. Therefore, all nodes with rank less than or equal to $t+1$ induce the same clusters in T_1 as in R. This implies in particular that $C_{k-1} = A \cup B$ and $k - 1 = t$.

Now consider the first move on $\mathrm{FP}(T', R)$. Remember that we assumed that T' is an NNI neighbour of T with a move on edge $[(T)_t, (T)_{t+1}]$ between them

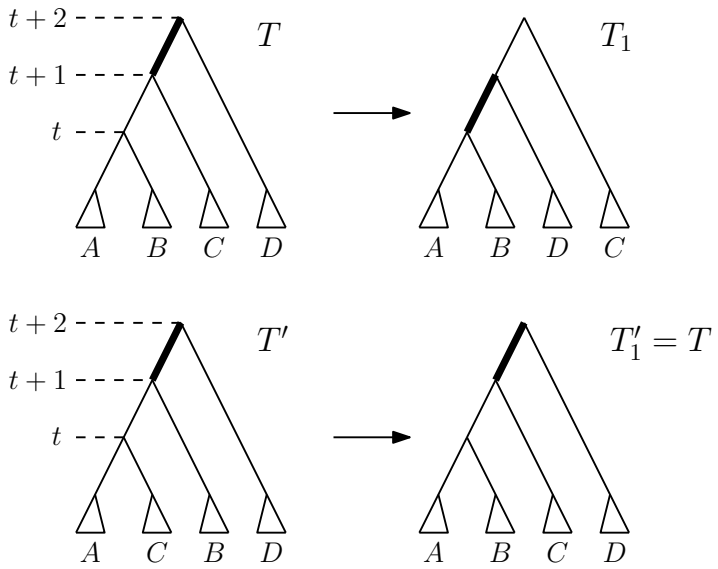


Figure 4.8: Comparison of paths $\mathrm{FP}(T, R)$ (top) and $\mathrm{FP}(T', R)$ (bottom) if T and T' are connected by an NNI move on edge $[(T)_{t+1}, (T)_{t+2}]$ and C_k intersects A, B, and D, but not C (case **1.3.1**).

such that B and C are exchanged between T and T'. It follows that the cluster $C_{k-1} = A \cup B$ is not present as cluster in T' (see also T' in Figure 4.8). Because all clusters in T' and R induced by nodes with rank less than or equal to $t-1$ coincide and $k-1 = t$, all clusters C_j with $j < k-1$ are present in T'. The first move on $\mathrm{FP}(T', R)$ hence decreases the rank of $mrca(C_{k-1}) = mrca(A \cup B)$ in T'. Because $(T')_{t+1} = mrca(A \cup B)_{T'}$, this is an NNI move on $[(T')_t, (T')_{t+1}]$ that swaps B and C, and hence results in $T'_1 = T$. T is hence the tree following T' on $\mathrm{FP}(T', R)$, meaning that $|\mathrm{FP}(T', R)| = 1 + |\mathrm{FP}(T, R)|$. This however contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

1.3.2 C_k intersects A and D but not B or C.

Remember that $mrca(C_k)_{T_1}$ has rank $t+1$, and its children induce the clusters D and $A \cup B$. Furthermore, the children of the node of rank t in T_1 induce the clusters A and B. To decrease the rank of $mrca(C_k)$ in T_1, FindPath hence performs an NNI move on $[(T_1)_t, (T_1)_{t+1}]$, exchanging the subtrees induced by clusters B and D in T_1. In the resulting tree T_2, the node of rank t induces the cluster $A \cup D$ and is the $mrca$ of C_k. This is illustrated in the top row in Figure 4.9. Note that this means that the node inducing cluster C_k in R has rank $k \leq t$. And since only clusters induced

by nodes t and $t+1$ can differ between T and T', C_{k-1}, which we know is present as cluster in T, is a cluster in T', too.

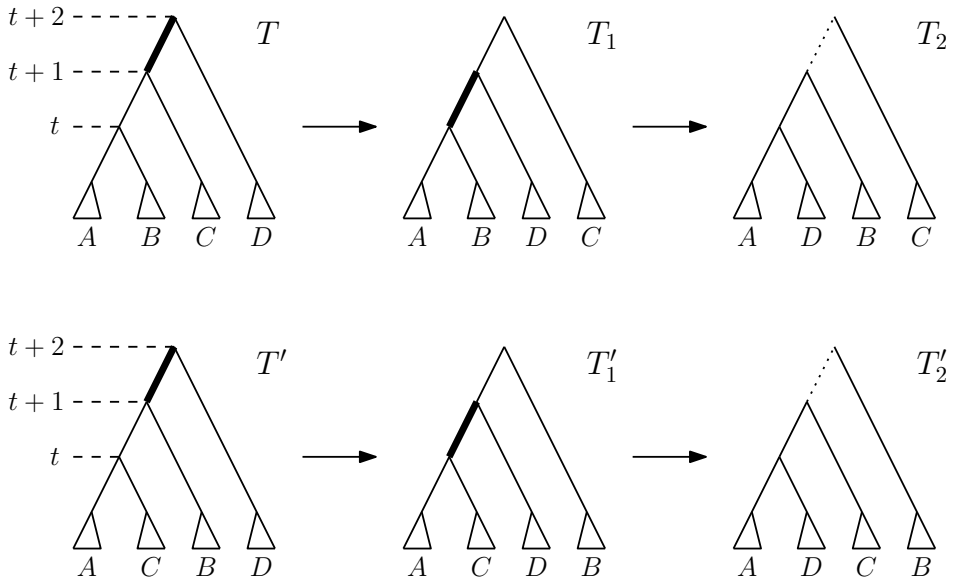


Figure 4.9: Comparison of paths $\mathrm{FP}(T, R)$ (top) and $\mathrm{FP}(T', R)$ (bottom) if T and T' are connected by an NNI move on edge $[(T)_{t+1}, (T)_{t+2}]$ and C_k intersects A and D, but not B or C (case **1.3.2**).

The first cluster considered on $\mathrm{FP}(T', R)$ is therefore C_k, which has rank $t+2$ in T', because C_k intersects both A and D and the children of $(T')_{t+2}$ induce the clusters $A \cup B \cup C$ and D (see T' in Figure 4.9). We also know that the intervals $[(T')_{t+1}, (T')_{t+2}]$ and $[(T')_t, (T')_{t+1}]$ are edges. To decrease the $mrca$ of C_k in T', the first move must hence be an NNI move that exchanges the subtrees induced by clusters D and B. The $mrca$ of C_k in the resulting tree T'_1 has rank $t+1$, and its children induce the clusters $A \cup B$ and D. Since C_k intersects A and D, but not B, the next move on T'_1 decreases the rank of $mrca(C_k)$ by swapping the subtrees induced by B and D, such that the node of rank t in T'_2 induces the cluster $A \cup D$. The path from T' to T'_2 is illustrated on the bottom of Figure 4.9.

Now compare the trees T_2 and T'_2. They originate from T and T' by moves on the intervals with ranks between $t+2$ and t. All clusters in T_2 and T'_2 with ranks $j \notin \{t, t+1, t+2\}$ hence coincide. The clusters induced by the nodes of rank $t, t+1$, and $t+2$ in T_2 are $A \cup D$, $A \cup D \cup B$, and $A \cup D \cup B \cup C$ in that order. In T'_2, these clusters are $A \cup D$, $A \cup C \cup D$, and $A \cup B \cup C \cup D$. Therefore, the subtrees induced by B and C swap position between T_2 and T'_2, implying

that these two trees are NNI neighbours. With $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T_2', R)| = |\mathrm{FP}(T', R)| - 2$, this contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

1.3.3 C_k intersects B and D but not A or C.

This case is similar to the previous one, with the roles of A and B swapped. Remember that $mrca(C_k)$ has rank $t+1$ in T_1 and has the nodes inducing clusters $A \cup B$ and D as children. The next move on $\mathrm{FP}(T, R)$ to decrease the rank of $mrca(C_k)$ therefore swaps the subtrees induced by A and D, so that the node of rank t in T_2 induces the cluster $B \cup D$. Note that this implies $k \leq t$. This segment of $\mathrm{FP}(T, R)$ is depicted on the top of Figure 4.10.

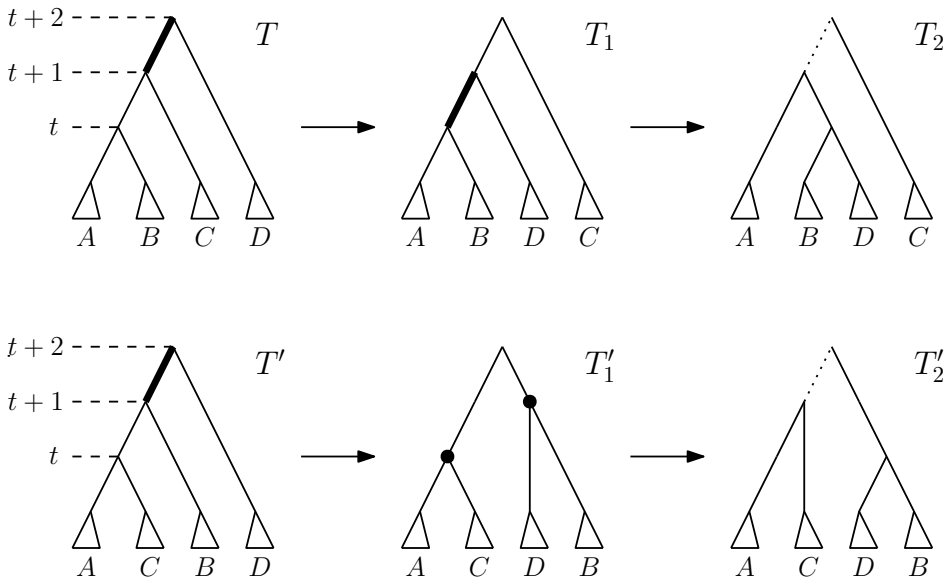


Figure 4.10: Comparison of paths $\mathrm{FP}(T, R)$ (top) and $\mathrm{FP}(T', R)$ (bottom) if T and T' are connected by an NNI move on edge $[(T)_{t+1}, (T)_{t+2}]$ and C_k intersects B and D but not A or C (case **1.3.3**).

Let us now consider the first moves on $\mathrm{FP}(T', R)$. Because $k - 1 \leq t - 1$, and all nodes with rank less than t induce the same clusters in T and T', every cluster C_j with $j \leq k - 1$ is induced by the node of rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence decreases the rank of $mrca(C_k)$. Since C_k intersects B and D, its $mrca$ in T' is the node of rank $t + 2$, as its children induce the clusters $A \cup B \cup C$ and D. In the tree T_1' after the first move on $\mathrm{FP}(T', R)$, $mrca(C_k)$ has rank $t + 1$, which implies that the NNI move between T' and T_1' swaps the subtrees induced by $A \cup C$ and D. Then the node of rank $t + 1$ in T_1' induces the cluster $B \cup D$, as depicted

in T_1' in Figure 4.10, while the node of rank t induces the cluster $A \cup C$. Because $k \leq t$, the rank of $mrca(C_k)$, which is $t+1$ in T_1', decreases further on $\mathrm{FP}(T', R)$. Note that the interval $[(T_1')_t, (T_1')_{t+1}]$ is a rank interval, as $(T_1')_{t+1}$ induces $B \cup D$ and $(T_1')_t$ induces $A \cup C$. Therefore, T_2' results from a rank swap of these two nodes in T_1', as depicted on the bottom of Figure 4.10. Now compare T_2 and T_2'. They originate from T and T' by moves on the intervals with ranks between $t+2$ and t. All clusters in T_2 and T_2' with ranks $j \notin \{t, t+1, t+2\}$ hence coincide. The clusters induced by the nodes of rank $t, t+1$, and $t+2$ in T_2 are $B \cup D$, $A \cup D \cup B$, and $A \cup D \cup B \cup C$ in that order. In T_2', these clusters are $B \cup D$, $A \cup C$, and $A \cup B \cup C \cup D$. Therefore, the subtrees induced by C and $B \cup D$ swap position between T_2 and T_2', implying that these two trees are NNI neighbours. With $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T_2', R)| = |\mathrm{FP}(T', R)| - 2$, this contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

1.4 NNI move on (edge) interval $[(T)_{t+1}, (T)_{t+2}]$ that builds a cluster $C \cup D$ in T_1.

This move is shown on the top of Figure 4.11 by an arrow from T to T_1. The cluster induced by node of rank $t+1$ in T_1, which is the *mrca* of C_k, is $C \cup D$. This implies that C_k intersects C and D but not A or B. We distinguish the following two cases: $k = t+1$ and $k < t+1$.

1.4.1 $k = t+1$

If $k = t+1$, then the ranks of $mrca(C_k)_{T_1}$ and $mrca(C_k)_R$ coincide and $C_k = C \cup D$. Because FindPath considers clusters in a bottom-up approach, C_{k-1} is induced by the node of rank $k-1 = t$ in T. Hence $C_{k-1} = A \cup B$. Because $k - 1 = t$, and all nodes with rank less than t induce the same clusters in T and T', every cluster C_j with $j < k-1$ is induced by the node of rank j in T'. The first cluster considered on $\mathrm{FP}(T', R)$ is hence C_{k-1}. With $C_{k-1} = A \cup B$ it follows that $mrca(C_k)_{T'}$ is $(T')_{t+1}$, as the children of this node induce clusters $A \cup C$ and B. The first move on $\mathrm{FP}(T', R)$ hence builds the cluster $A \cup B$ by swapping subtrees induced by cluster B and C. Remember that we assume that the NNI move on T to get T' is performed on the interval $[(T)_t, (T)_{t+1}]$ and swaps the subtrees induced by B and C (see Figure 4.11). The move between T' and T_1' is hence the reverse of the move between T and T', resulting in $T_1' = T$, and $|\mathrm{FP}(T', R)| = 1 + |\mathrm{FP}(T, R)|$, which contradicts $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

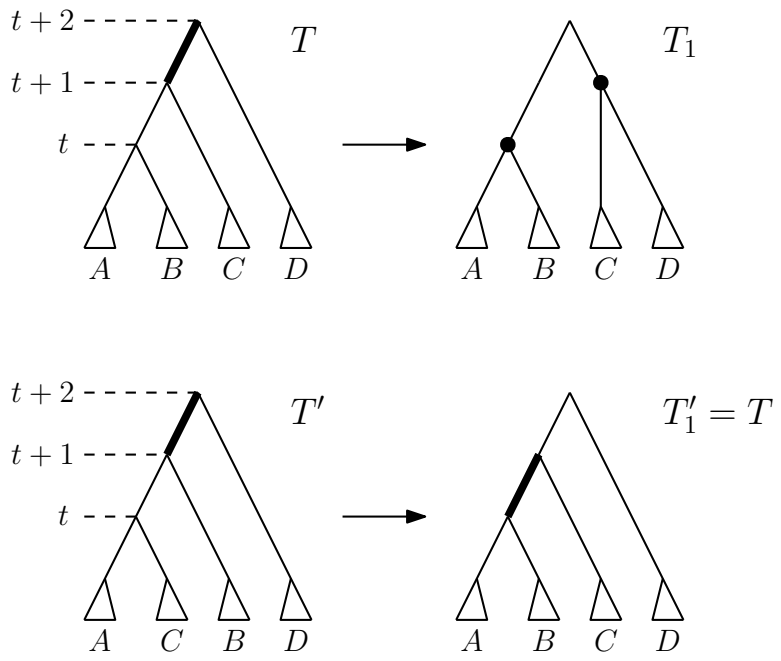


Figure 4.11: Comparison of paths $\mathrm{FP}(T, R)$ (top) and $\mathrm{FP}(T', R)$ (bottom) if T and T' are connected by a rank move on interval $[(T)_{t+1}, (T)_{t+2}]$ and $k = t + 1$ (case **1.4.1**).

1.4.2 $k < t + 1$

In this case, FindPath decreases the rank of $mrca(C_k)_{T_1}$ by a second move on $\mathrm{FP}(T, R)$. Remember that the node of rank $t + 1$ in T_1 induces the cluster $C \cup D$, and the node of rank t induces $A \cup B$. Thus, the interval $[(T_1)_t, (T_1)_{t+1}]$ is a rank interval and the move on $\mathrm{FP}(T, R)$ to decrease the rank of $C_k \subset (C \cup D)$ on that interval is a rank move. The segment of $\mathrm{FP}(T, R)$ between T and T_2 is depicted in Figure 4.12 as the path on the top.

With $k < t+1$, the cluster C_{k-1} in T is induced by a node of rank $k-1 < t$. Since all nodes with rank less than t induce the same clusters in T, T', and R, C_{k-1} is a cluster in T'. Therefore, C_k is the cluster whose $mrca$ is considered in the first move on $\mathrm{FP}(T', R)$. Since C_k intersects C and D, $mrca(C_k)$ in T' is the node of rank $t + 2$, which has the nodes inducing the clusters D and $A \cup B \cup C$ as children (see T' in Figure 4.12).

The first move on $\mathrm{FP}(T', R)$ that decreases the rank of $mrca(C_k)$ hence swaps the subtrees induced by clusters D and B. In the resulting tree T'_1 the children of the internal node of rank $t + 1$ induce the clusters D and $A \cup C$. From $k < t+1$ we know that the rank of $mrca(C_k)$ needs to decrease

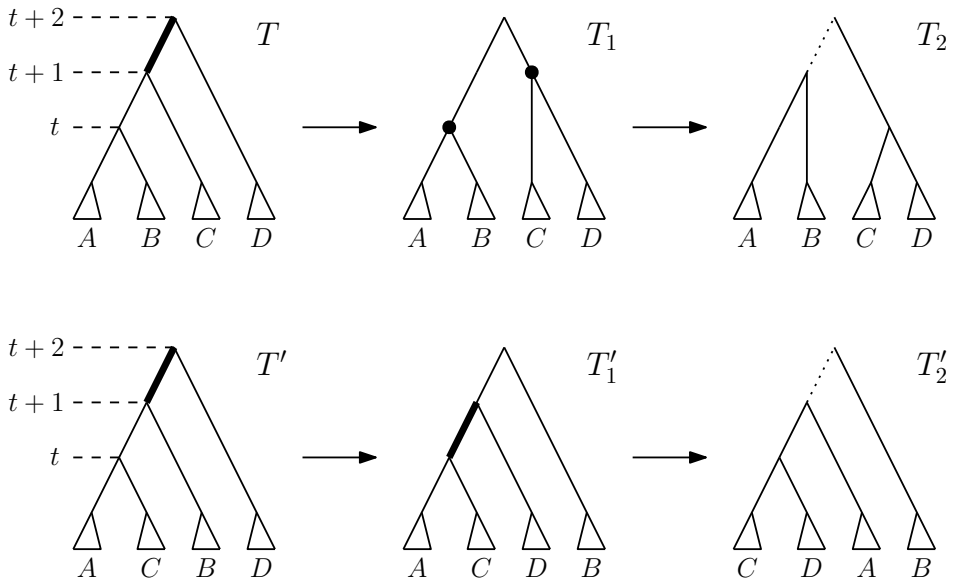


Figure 4.12: Comparison of paths $\mathrm{FP}(T, R)$ (top) and $\mathrm{FP}(T', R)$ (bottom) if T and T' are connected by a rank move on interval $[(T)_{t+1}, (T)_{t+2}]$ and $k < t + 1$ (case **1.4.2**).

further. The move on T_1' which decreases the rank of this node swaps the subtrees induced by A and D. This results in the tree T_2, which is depicted on the bottom of Figure 4.12.

Now compare T_2 and T_2'. They originate from T and T' by moves on the intervals with ranks between $t+2$ and t. Therefore, all clusters in T_2 and T_2' with ranks $j \notin \{t, t+1, t+2\}$ coincide. The clusters induced by the nodes of rank $t, t+1$, and $t+2$ in T_2 are $C \cup D$, $A \cup B$, and $A \cup B \cup C \cup D$ in that order. In T_2', these clusters are $C \cup D$, $A \cup C \cup D$, and $A \cup B \cup C \cup D$. Therefore, between T_2 and T_2' the subtrees induced by $C \cup D$ and B swap position, implying that these two trees are NNI neighbours (see the two trees on the right of Figure 4.12). This together with $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T_2', R)| = |\mathrm{FP}(T', R)| - 2$ contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

1.5 Rank move on interval $[(T)_{t+1}, (T)_{t+2}]$.

This case is analogous to the previous case **1.4** and also split into two sub-cases. The difference to the previous case is that the interval $[(T)_{t+1}, (T)_{t+2}]$ is a rank interval. Let D be the cluster induced by the node of rank $t + 2$ in T (see T on the top left of Figure 4.13). Since the first move on $\mathrm{FP}(T, R)$ decreases the rank of $mrca(C_k)$ and this move is performed on $[(T)_{t+1}, (T)_{t+2}]$, the $mrca$ of the

cluster C_k must have rank $t+2$ before this move, and it follows that $C_k \subseteq D$. We distinguish the two cases that $k = t+1$ and $k < t+1$.

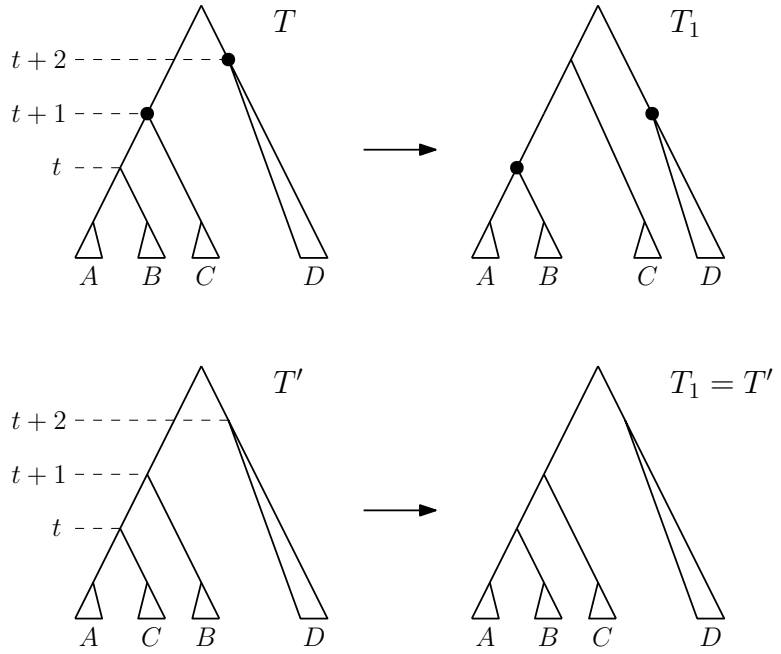


Figure 4.13: Comparison of paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ if there is an NNI move between T and T' and a rank move on the interval above this edge follows on $\mathrm{FP}(T, R)$ and $k = t+1$ (case **1.5.1**).

1.5.1 $k = t+1$

In this case the rank of $mrca(C_k)_{T_1}$ and $mrca(C_k)_R$ coincide and $C_k = D$. Because FindPath considers clusters in a bottom-up approach, C_{k-1} is induced by the node of rank $k-1 = t$ in T. Hence $C_{k-1} = A \cup B$.

Since T and T' are connected by an NNI move exchanging the subtrees induced by B and C, $A \cup B$ is not induced as cluster of a node in T' (see T' in Figure 4.13). Because $k-1 = t$, and all nodes with rank less than t induce the same clusters in T and T', every cluster C_j with $j < k-1$ is induced by the node of rank j in T'. The first cluster considered on $\mathrm{FP}(T', R)$ is hence C_{k-1}. This move decreasing the rank of $mrca(C_{k-1})$ is the reverse of the move between T and T', resulting in $T'_1 = T$ (see Figure 4.13), just as in case **1.4**. As in that case, $|\mathrm{FP}(T', R)| = 1 + |\mathrm{FP}(T, R)|$, which contradicts $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

1.5.2 $k < t + 1$

From $k < t+1$ it follows that the rank of $mrca(C_k)_{T_1}$, which is $t+1$, decreases by a second move on $\mathrm{FP}(T, R)$. The clusters induced by the nodes of t and $t + 1$ in T_1 are $A \cup B$ and D, meaning that the corresponding interval is a rank interval. Because $C_k \subseteq D$, the move on T_1 decreasing the rank of $mrca(C_k)$ is a rank swap that results in a tree T_2 with $\mathrm{rank}(C_k)_{T_2} = t$, as depicted in the top of Figure 4.14.

We know that the only difference between T' and T is the cluster of rank t, as the two trees are connected by an NNI move. We also know that C_{k-1} is a cluster in T, and with $k - 1 < t$ it must also be present in T', as all nodes with rank less than t induce the same clusters in T, T', and R. The first move on $\mathrm{FP}(T', R)$ hence decreases the rank of $mrca(C_k)$. From $C_k \subseteq D$ and $\mathrm{rank}(mrca(C_k)_{T'}) = t + 2$ we infer that there is a second move on $\mathrm{FP}(T', R)$ to decrease the rank of $mrca(C_k)$. Remember that the nodes of rank t and $t + 1$ in T' induce the clusters $A \cup C$ and $A \cup B \cup C$. The two first moves on $\mathrm{FP}(T', R)$ are hence rank swaps, where first the node inducing $A \cup B \cup C$ and then the node inducing $A \cup C$ swaps rank with the node inducing D. These moves from T' to T'_2 are depicted on the bottom of Figure 4.14.

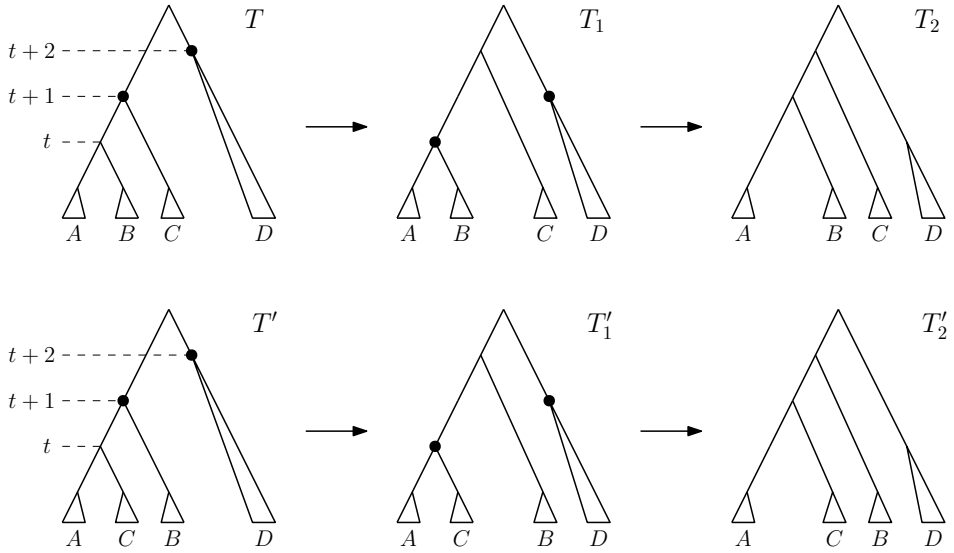


Figure 4.14: Comparison of paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ if there is an NNI move between T and T' and a rank move on the interval above this edge follows on $\mathrm{FP}(T, R)$ and $k < t + 1$ (case **1.5.2**).

We now compare the two trees T_2 and T_2'. They originate from T and T' by moves on the intervals with ranks between $t+2$ and t. All clusters in T_2 and T_2' with ranks $j \notin \{t, t+1, t+2\}$ hence coincide. The clusters induced by the nodes of rank $t, t+1$, and $t+2$ in T_2 are $D, A \cup B$, and $A \cup B \cup C$ in that order. In T_2', these clusters are $D, A \cup C$, and $A \cup B \cup C$. Therefore, between T_2 and T_2' the subtrees induced by B and C swap position, implying that these two trees are NNI neighbours (see the two trees on the right of Figure 4.14). This together with $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T_2', R)| = |\mathrm{FP}(T', R)| - 2$ contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

Case 2. T and T' are connected by a rank move.

Remember that we assume that the move between T and T' is performed on the interval $[(T)_t, (T)_{t+1}]$. Denote the cluster induced by $(T)_t$ by A, the clusters induced by the children of $(T)_t$ by A_1 and A_2, the cluster induced by $(T)_{t+1}$ by B, and the clusters induced by the children of $(T)_{t+1}$ by B_1 and B_2 – see the tree T on the top left of Figure 4.15.

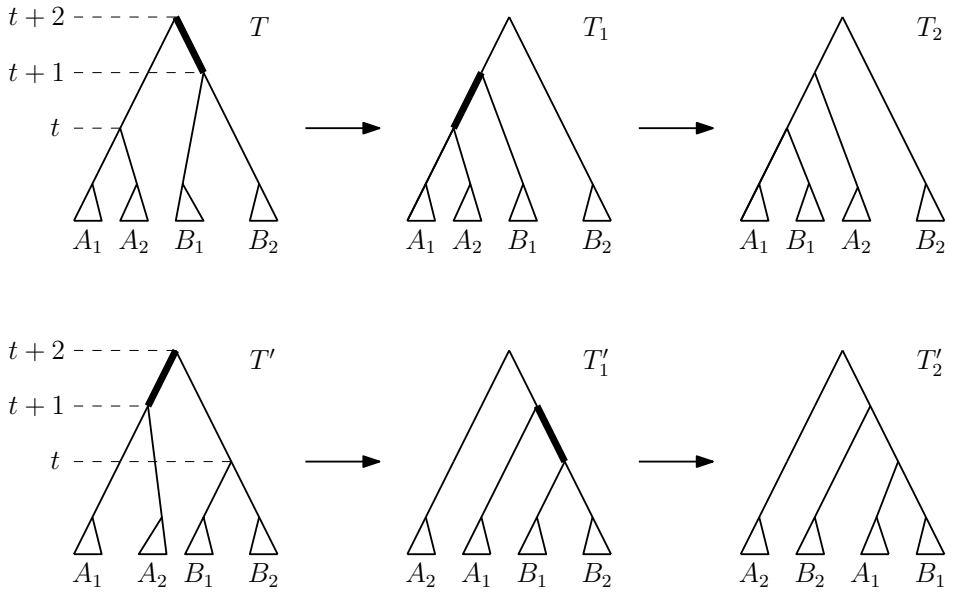


Figure 4.15: Rank move between T and T' and possible initial segments of $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ when $[(T)_{t+1}, (T)_{t+2}]$ is an edge. We use notations $A = A_1 \cup A_2$ and $B = B_1 \cup B_2$.

We again consider all possible moves FindPath might perform to go from T to T_1 on $\mathrm{FP}(T, R)$ in a case differentiation. By Lemma 4.8, $\mathrm{rank}(mrca(C_k)_T) \in \{t, t+1, t+2\}$.

Therefore, the move between T and T_1 is performed on one of the three intervals $[(T)_{t-1}, (T)_t]$ (case **2.1**), $[(T)_t, (T)_{t+1}]$ (case **2.2**), or $[(T)_{t+1}, (T)_{t+2}]$ (cases **2.3**, **2.4**).

2.1 RNNI move (either type) on interval $[(T)_{t-1}, (T)_t]$.

Since FindPath decreases the $mrca$ of a cluster C_k of R by this move, it follows that C_k is a subset of the cluster induced by $(T)_t$, which is A. We can furthermore infer $k < t-1$. Remember that the rank swap between T and T' is performed on the interval $[(T)_{t+1}, (T)_t]$, which implies that A is induced by the node of rank $t+1$ in T' (see T and T' on the left of Figure 4.15). Since all nodes with ranks less than t induce the same clusters in T and T', it follows from $k < t-1$ that all clusters C_j with $j < k$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster C_k. The first move on $\mathrm{FP}(T', R)$ must be hence be a rank swap on $[(T')_t, (T')_{t+1}]$ decreasing the rank of $mrca(C_k) = (T')_{t+1}$, resulting in $T'_1 = T$. This however contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

2.2 RNNI move (either type) on $[(T)_t, (T)_{t+1}]$.

Because T and T' are connected by a rank move on $[(T)_t, (T)_{t+1}]$, the move between T and T_1 on the same interval must be a rank move, too. And since only one tree can result from a rank move on a specific interval, $T_1 = T'$. This however contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

We now consider the case that the first move on $\mathrm{FP}(T, R)$, which is the move between T and T_1, is performed on the interval $[(T)_{t+1}, (T)_{t+2}]$. Since we do not know if $[(T)_{t+1}, (T)_{t+2}]$ is an edge interval or a rank interval, we consider the case that an NNI move is performed on this interval (cases **2.3**) separately to the case that a rank move is performed (case **2.4**).

2.3 NNI move on (edge) interval $[(T)_{t+1}, (T)_{t+2}]$.

We distinguish the case that $(T)_{t+2}$ is parent of $(T)_t$ (case **2.3.1**) from the case that it is not (case **2.3.2**).

2.3.1 $(T)_{t+2}$ is parent of $(T)_t$.

Remember that the node $(T)_{t+1}$ is parent of the nodes inducing clusters B_1 and B_2, and $(T)_t$ induces cluster A. Furthermore, an NNI move only changes the cluster induced by the lower node bounding the interval, i.e. an

NNI move on the edge $[(T)_{t+1}, (T)_{t+2}]$ changes only the cluster induced by $(T)_{t+1}$. More precisely, the node of rank $t+1$ in the tree T_1 resulting from the first move on $\mathrm{FP}(T, R)$, either induces the cluster $A \cup B_1$ or $A \cup B_2$ (see Observation 3.1 and the discussion above it). We assume without loss of generality that it is the former, otherwise we change notation. In Figure 4.15 the tree T_1 is depicted in the middle of the top row. That the first move on $\mathrm{FP}(T, R)$ builds the new cluster $A \cup B_1$ in T_1 implies that C_k intersects A and B_1 but not B_2, i.e. $C_k \subseteq (A_1 \cup A_2 \cup B_1)$. Remember $\mathrm{rank}(mrca(C_k)_{T_1}) = t+1$ and $\mathrm{rank}(mrca(C_k)_R) = k$. We now distinguish the case $k = t+1$ from $k < t+1$ (similar to case **1.5**). Note that it cannot be $t < k$, as FindPath considers clusters in a bottom-up approach.

***k* = *t* + 1**

In this case the ranks of $mrca(C_k)_{T_1}$ and $mrca(C_k)_R$ coincide. Because FindPath considers clusters in a bottom-up approach, C_{k-1} is induced by the node of rank $k-1 = t$ in T. Hence $C_{k-1} = A$. We can furthermore infer from $k = t+1$ and the fact that all clusters induced by nodes with rank less than t, that the clusters C_j with $j < k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_{k-1} = A$. Since $mrca(A)_{T'}$ is the node of rank $t+1$ in T', the first move on $\mathrm{FP}(T', R)$ decreases the rank of $mrca(C_{k-1})_{T'}$, which results in $T'_1 = T$. This however contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

***k* < *t* + 1**

If the rank of $mrca(C_k)_{T_1}$ is strictly higher than that of $mrca(C_k)_R$, then FindPath decreases the rank of $mrca(C_k)_{T_1}$ further in a second step on $\mathrm{FP}(T, R)$. The first two steps of both $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ are depicted in Figure 4.15. Remember that the clusters induced by the nodes of t and $t+1$ in T_1 are $A = A_1 \cup A_2$ and $A_1 \cup A_2 \cup B_1$, where A_1 and A_2 are the clusters induced by the children of $(T_1)_t$. Since $C_k \subseteq (A_1 \cup A_2 \cup B_1)$, the move decreasing the *mrca* of C_k in T_1 must happen on the interval $[(T_1)_t, (T_1)_{t+1}]$. Because this is an edge, there are two possible trees that result from NNI moves on this edge. The tree T_2 resulting from such an NNI move either has the cluster $A_1 \cup B_1$ or $A_2 \cup B_2$ induced by its node of rank t. Due to the symmetry we can assume that $C_k \subseteq (A_1 \cup B_1)$, which implies that $(T_2)_t$ induces $A_1 \cup B_1$, as depicted on the top right of Figure 4.15.

Now consider the first moves on $\mathrm{FP}(T', R)$, assuming $C_k \subseteq (A_1 \cup B_1)$. Since

$k < t+1$ and all nodes with rank less than t induce the same clusters in T and T', all clusters C_j with $j \leq k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_k \subseteq (A_1 \cup B_1)$. In T' the children of the node of rank t induce B_1 and B_2 and the children of the node of rank $t+1$ induce A_1 and A_2. Furthermore, $[(T')_{t+1}, (T')_{t+2}]$ is an edge interval. The first move decreasing the rank of $mrca(C_k)$ for $C_k \subseteq (A_1 \cup B_1)$ is hence an NNI move such that the resulting tree T'_1 only differs from T' in the node of rank $t+1$, which induces the cluster $A_1 \cup B_1 \cup B_2$ in T'_1. In the second step, the rank of $mrca(C_k)$, which is $t+1$ in T'_1, is decreased further by another NNI move, this time on the edge interval $[(T'_1)_t, (T'_1)_{t+1}]$.
In the resulting tree T'_2, the clusters induced by nodes with rank t, $t+1$, and $t+2$ are $A_1 \cup B_1$, $A_1 \cup B_1 \cup B_2$ and $A_1 \cup A_2 \cup B_1 \cup B_2$ (see Figure 4.15). Remember that in T_2 the clusters induced by nodes t, $t+1$, and $t+2$ are $A_1 \cup B_1$, $A_1 \cup A_2 \cup B_1$ and $A_1 \cup A_2 \cup B_1 \cup B_2$. Since only the clusters induced by nodes of rank t, $t+1$, and $t+2$ are changed by the first two moves on the paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$, all other nodes induce the same clusters in T_2 and T'_2. Therefore, T_2 and T'_2 are RNNI neighbours, as they are connected by an NNI move on the interval $[(T_2)_{t+1}, (T_2)_{t+2}]$. With $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T'_2, R)| = |\mathrm{FP}(T', R)| - 2$, this contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

2.3.2 $(T)_{t+2}$ is not parent of $(T)_t$.

Remember that the cluster induced by $(T)_t$ is A and the one induced by $(T)_{t+1}$ is $B_1 \cup B_2$, where B_1 and B_2 are the clusters induced by the children of $(T)_{t+1}$. Furthermore, $(T)_{t+2}$ is parent of $(T)_{t+1}$, as we assume that T and T_1 are connected by an NNI move on $[(T)_{t+1}, (T)_{t+2}]$. Since we assume that $(T)_{t+2}$ is not parent of $(T)_t$, $(T)_{t+2}$ is parent of another node which induces a cluster C. The tree T is depicted on the top left of Figure 4.16. Let us now consider the clusters induced by nodes with rank t, $t+1$, and $t+2$ in T', which we can infer from the rank swap connecting T and T'. The internal node of rank t induces the cluster $B_1 \cup B_2$ in T', which results from a rank swap on the interval $[(T)_t, (T)_{t+1}]$ of T. The internal node of rank $t+1$ induces A and the internal node of rank $t+2$ induces $B_1 \cup B_2 \cup C$ in T' (see Figure 4.16).

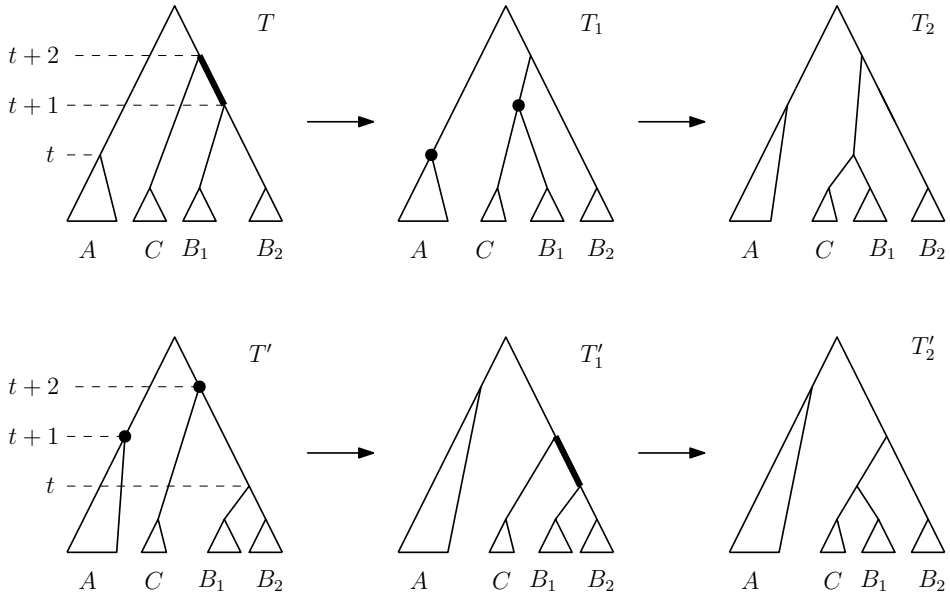


Figure 4.16: Comparison of paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ if there is a rank move between T and T' and an NNI move between T and T_1 on the edge $[(T)_{t+1}, (T)_{t+2}]$ and $(T)_{t+2}$ is not parent of $(T)_t$.

The NNI move on the interval $[(T)_{t+1}, (T)_{t+2}]$, which is the first move on $\mathrm{FP}(T, R)$, changes the cluster induced by the node of rank $t+1$ to be either $B_1 \cup C$ or $B_2 \cup C$ (a detailed explanation of NNI moves can be found above Observation 3.1). We assume without loss of generality that T_1, which results from an NNI move on the interval $[(T)_{t+1}, (T)_{t+2}]$ in T, has the cluster $B_1 \cup C$ induced by its node of rank $t+1$. We otherwise exchange the names B_1 and B_2. Remember that $\mathrm{rank}(mrca(C_k)_{T_1}) = t+1$ and $\mathrm{rank}(mrca(C_k)_R) = k$. We now distinguish the case $k = t+1$ from $k < t+1$ (similar to case **1.5**). Note that it cannot be $t < k$, as FindPath considers clusters in a bottom-up approach.

***k* = *t* + 1**

In this case the ranks of $mrca(C_k)_{T_1}$ and $mrca(C_k)_R$ coincide. Because FindPath considers clusters in a bottom-up approach, C_{k-1} is induced by the node of rank $k-1 = t$ in T. Hence $C_{k-1} = A$. We can furthermore infer from $k-1 = t$ and the fact that all clusters induced by nodes with rank less than t, that the clusters C_j with $j < k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_{k-1} = A$. Since $mrca(A)_{T'}$ is the node of rank $t+1$, the first move on $\mathrm{FP}(T', R)$ decreases the rank of $mrca(A)_{T'}$, which results in $T'_1 = T$. This however

contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

$k < t + 1$

If the rank of $mrca(C_k)_{T_1}$ is strictly higher than that of $mrca(C_k)_R$, then FindPath decreases the rank of $mrca(C_k)_{T_1}$ further in a second step on $\mathrm{FP}(T, R)$. The first two steps of both $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ are depicted in Figure 4.16. Remember that the clusters induced by the nodes of t and $t + 1$ in T_1 are A and $B_1 \cup C$. Since $C_k \subseteq (B_1 \cup C)$, the move decreasing the $mrca$ of C_k in T_1 must happen on the interval $[(T_1)_t, (T_1)_{t+1}]$. Because this is a rank interval, T_2 results from a rank move on $[(T_1)_t, (T_1)_{t+1}]$. As a result, the node of rank t in T_2 induces the cluster $B_1 \cup C$, the node of rank $t + 1$ induces A, and the node of rank $t + 2$ induces $B_1 \cup B_2 \cup C$ (see Figure 4.16).

Now consider the first moves on $\mathrm{FP}(T', R)$, assuming $C_k \subseteq (B_1 \cup C)$. Since $k < t + 1$ and all nodes with rank less than t induce the same clusters in T and T', all clusters C_j with $j \leq k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_k \subseteq (B_1 \cup C)$. In T' the internal node of rank t induces the cluster $B_1 \cup B_2$, the internal node of rank $t+1$ induces A and the internal node of rank $t+2$ induces $B_1 \cup B_2 \cup C$. The $mrca$ of $C_k \subseteq (B_1 \cup C)$ is hence $(T')_{t+2}$. Since $[(T')_{t+1}, (T')_{t+2}]$ is a rank interval, T'_1 results from a rank move on this interval. It follows that in T' the node of rank $t + 1$ induces the cluster $B_1 \cup B_2 \cup C$ and is the $mrca$ of C_k. The node of rank t in T'_1 induces the cluster $B_1 \cup B_2$, where B_1 and B_2 are the clusters induced by the children of $(T'_1)_t$, like in T'. Therefore, an NNI move on $[(T'_1)_t, (T'_1)_{t+1}]$ is the second move on $\mathrm{FP}(T', R)$ to decrease the rank of the $mrca$ of $C_k \subseteq (B_1 \cup C)$ in T'_1.

In the resulting tree T'_2 the clusters induced by nodes t, $t + 1$, and $t + 2$ are $B_1 \cup C$, $B_1 \cup B_2 \cup C$ and A (see Figure 4.16). Remember in T_2 the clusters induced by nodes t, $t+1$, and $t+2$ are $B_1 \cup C$, A and $B_1 \cup B_2 \cup C$. Since only the clusters induced by nodes of rank t, $t + 1$, and $t + 2$ are changed by the first two moves on the paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$, all other nodes induce the same clusters in T_2 and T'_2. Therefore, T_2 and T'_2 are RNNI neighbours, as they are connected by a rank move on the interval $[(T_2)_{t+1}, (T_2)_{t+2}]$. With $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T'_2, R)| = |\mathrm{FP}(T', R)| - 2$, this contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

2.4 Rank move on interval $[(T)_{t+1}, (T)_{t+2}]$.

In this case both intervals $[(T)_t, (T)_{t+1}]$ and $[(T)_{t+1}, (T)_{t+2}]$ are rank intervals. Remember that $(T)_t$ induces the cluster $A = A_1 \cup A_2$, where A_1 and A_2 are induced by the children of this node, and $(T)_{t+1}$ induces the cluster B. We again distinguish two cases: $(T)_{t+2}$ is parent of $(T)_t$ (case **2.4.1**) or is is not (case **2.4.2**).

2.4.1 $(T)_{t+2}$ is parent of $(T)_t$.

Let the child of $(T)_{t+2}$ that is not $(T)_t$ induce the cluster C. Since T_1 results from a rank move on the interval $[(T)_{t+1}, (T)_{t+2}]$, the cluster C_k whose $mrca$ decreases by this first move on $\mathrm{FP}(T, R)$ must be a subset of $A_1 \cup A_2 \cup C$, the cluster induced by $(T)_{t+2}$. The tree T_1 is depicted in the middle of the top row in Figure 4.17. Remember that $\mathrm{rank}(mrca(C_k)_{T_1}) = t + 1$ and $\mathrm{rank}(mrca(C_k)_R) = k$. We now distinguish the case $k = t+1$ from $k < t+1$ (similar to case **1.5**). Note that it cannot be $t < k$, as FindPath considers clusters in a bottom-up approach.

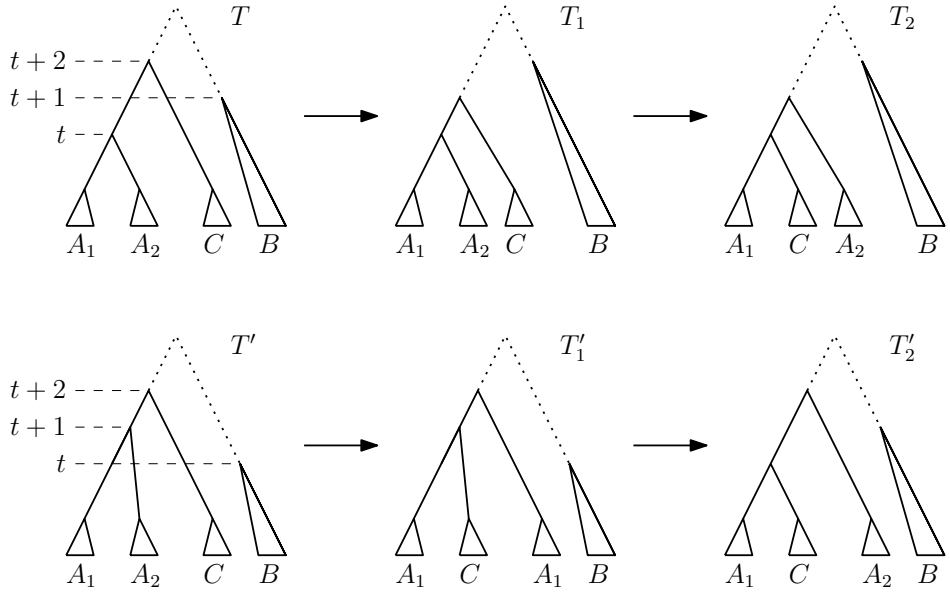


Figure 4.17: Comparison of paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ if there is a rank move between T and T' and a rank move on the edge $[(T)_{t+1}, (T)_{t+2}]$ between T and T_1 and $(T)_{t+2}$ is parent of $(T)_t$. The dotted part of the trees may look different.

***k* = *t* + 1**

In this case the ranks of $mrca(C_k)_{T_1}$ and $mrca(C_k)_R$ coincide. Because FindPath considers clusters in a bottom-up approach, C_{k-1} is induced by the node of rank $k - 1 = t$ in T. Hence $C_{k-1} = A$. We can furthermore infer

from $k-1=t$ and the fact that all clusters induced by nodes with rank less than t, that the clusters C_j with $j < k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_{k-1} = A$. Since the A is induced by the node of rank $t+1$ in T', the first move on $\mathrm{FP}(T', R)$ decreases the rank of $mrca(A)_{T'}$, which results in $T'_1 = T$. This however contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

***k* < *t* + 1**

If the rank of $mrca(C_k)_{T_1}$ is strictly higher than that of $mrca(C_k)_R$, then FindPath decreases the rank of $mrca(C_k)_{T_1}$ further in a second step on $\mathrm{FP}(T, R)$. The first two steps of both $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ are depicted in Figure 4.17. Remember that the clusters induced by the nodes of t and $t+1$ in T_1 are $A = A_1 \cup A_2$ and $A_1 \cup A_2 \cup C$, where A_1 and A_2 are the clusters induced by the children of $(T_1)_t$. Since $C_k \subseteq (A_1 \cup A_2 \cup C)$, the move decreasing the $mrca$ of C_k in T_1 must happen on the interval $[(T_1)_t, (T_1)_{t+1}]$. Because this is an edge, there are two possible trees that result from NNI moves on this edge. The tree T_2 resulting from such an NNI either has the cluster $A_1 \cup C$ or $A_2 \cup C$ induced by its node of rank t. Due to the symmetry we can assume that $C_k \subseteq (A_1 \cup C)$, which implies that $(T_2)_t$ induces $A_1 \cup C$, as depicted on the top right of Figure 4.17.

Now consider the first moves on $\mathrm{FP}(T', R)$, assuming $C_k \subseteq (A_1 \cup C)$. Since $k < t+1$ and all nodes with rank less than t induce the same clusters in T and T', all clusters C_j with $j \leq k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_k \subseteq (A_1 \cup C)$. In T' the children of the node of rank $t+2$ induce the clusters $A = A_1 \cup A_2$ and C. $(T')_{t+2}$ is hence the $mrca$ of C_k in T'. Furthermore, $[(T')_{t+1}, (T')_{t+2}]$ is an edge interval. The first move decreasing the rank of $mrca(C_k)$ for $C_k \subseteq (A_1 \cup C)$ is hence an NNI move such that the resulting tree T'_1 only differs from T' in the node of rank $t+1$, which induces the cluster $A_1 \cup C$ in T'. In the second step, the rank of $mrca(C_k)$, which is $t+1$ in T'_1, is decreased further. The interval $[(T')_t, (T')_{t+1}]$ is a rank interval, like in T. The move FindPath performs on T'_1 is hence a rank move.

In the resulting tree T'_2 the clusters induced by nodes t, $t+1$, and $t+2$ are $A_1 \cup C$, B, and $A_1 \cup A_2 \cup C$ (see Figure 4.17). Remember in T_2 the clusters induced by nodes t, $t+1$, and $t+2$ are $A_1 \cup A_2$, $A_1 \cup A_2 \cup C$, and B. Since only the clusters induced by nodes of rank t, $t+1$, and $t+2$

are changed by the first two moves on the paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$, all other nodes induce the same clusters in T_2 and T_2'. Therefore, T_2 and T_2' are RNNI neighbours, as they are connected by a rank move on the interval $[(T_2)_{t+1}, (T_2)_{t+2}]$. This together with $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T_2', R)| = |\mathrm{FP}(T', R)| - 2$ contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

2.4.2 $(T)_{t+2}$ is not a parent of $(T)_t$.

Remember that the cluster induced by $(T)_t$ is A and the one induced by $(T)_{t+1}$ is B. Let C be the cluster induced by the node $(T)_{t+2}$. The tree T is depicted on the top left of Figure 4.18. Both intervals $[(T)_t, (T)_{t+1}]$ and $[(T)_{t+1}, (T)_{t+2}]$ are rank intervals. The first move on $\mathrm{FP}(T, R)$ is hence a rank move on $[(T)_{t+1}, (T)_{t+2}]$. Since $(T)_{t+2}$ induces the cluster C, the cluster C_k of R whose *mrca* is decreased by this move must be a subset of C. Remember that $\mathrm{rank}(mrca(C_k)_{T_1}) = t + 1$ and $\mathrm{rank}(mrca(C_k)_R) = k$. We now distinguish the case $k = t + 1$ from $k < t + 1$ (similar to case **1.5**). Note that it cannot be $t < k$, as FindPath considers clusters in a bottom-up approach.

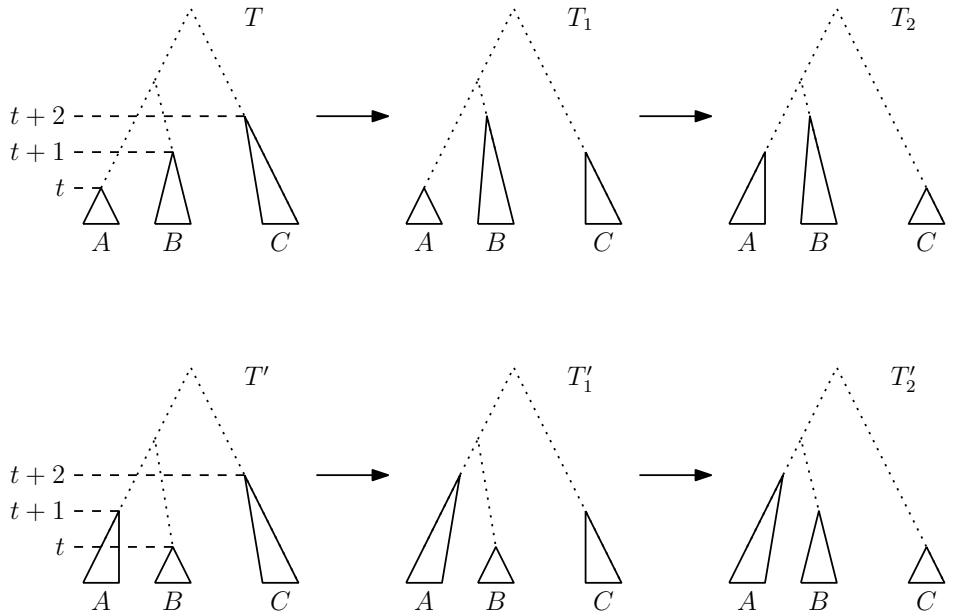


Figure 4.18: Comparison of paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ if there is a rank move between T and T' and a rank move on the edge $[(T)_{t+1}, (T)_{t+2}]$ between T and T_1 and $(T)_{t+2}$ is not parent of $(T)_t$. The dotted part of the trees may look different.

$k = t + 1$

In this case the ranks of $mrca(C_k)_{T_1}$ and $mrca(C_k)_R$ coincide. Because FINDPATH considers clusters in a bottom-up approach, C_{k-1} is induced by the node of rank $k-1 = t$ in T. Hence $C_{k-1} = A$. We can furthermore infer from $k-1 = t$ and the fact that all clusters induced by nodes with rank less than t, that the clusters C_j with $j < k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_{k-1} = A$. Since the A is induced by the node of rank $t+1$ in T', the first move on $\mathrm{FP}(T', R)$ decreases the rank of $mrca(A)_{T'}$, which results in $T'_1 = T$. This however contradicts our assumption $|\mathrm{FP}(T', R)| < |\mathrm{FP}(T, R)| - 1$.

$k < t + 1$

If the rank of $mrca(C_k)_{T_1}$ is strictly higher than that of $mrca(C_k)_R$, then FINDPATH decreases the rank of $mrca(C_k)_{T_1}$ further in a second step on $\mathrm{FP}(T, R)$. The first two steps of both $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$ are depicted in Figure 4.18. Remember that the clusters induced by the nodes of t and $t+1$ in T_1 are A and C. Since $C_k \subseteq C$, the move decreasing the $mrca$ of C_k in T_1 must happen on the interval $[(T_1)_t, (T_1)_{t+1}]$. Because this is a rank interval, T_2 results from a rank move on $[(T_1)_t, (T_1)_{t+1}]$. As a result, the node of rank t in T_2 induces the cluster C, the node of rank $t+1$ induces A, and the node of rank $t+2$ induces B (see Figure 4.18).

Now consider the first moves on $\mathrm{FP}(T', R)$, assuming $C_k \subseteq C$. Since $k < t+1$ and all nodes with rank less than t induce the same clusters in T and T', all clusters C_j with $j \leq k-1$ are induced by nodes with rank j in T'. The first move on $\mathrm{FP}(T', R)$ hence considers the cluster $C_k \subseteq C$. In T' the internal node of rank t induces the cluster B, the internal node of rank $t+1$ induces A and the internal node of rank $t+2$ induces C. The $mrca$ of $C_k \subseteq C$ is hence $(T')_{t+2}$. Since $[(T')_{t+1}, (T')_{t+2}]$ is a rank interval, T'_1 results from a rank move on this interval. It follows that in T' the node of rank $t+1$ induces the cluster C and is the $mrca$ of C_k. Since $[(T'_1)_t, (T'_1)_{t+1}]$ also is a rank interval, the move that FINDPATH performs on T'_1 is another rank move.

In the resulting tree T'_2, the clusters induced by nodes t, $t+1$, and $t+2$ are C, B and A (see Figure 4.18). Remember that in T_2 the clusters induced by nodes t, $t+1$, and $t+2$ are C, A and B. Since only the clusters induced by nodes of rank t, $t+1$, and $t+2$ are changed by the first two

moves on the paths $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R)$, all other nodes induce the same clusters in T_2 and T_2'. Therefore, T_2 and T_2' are RNNI neighbours, as they are connected by a rank move on the interval $[(T_2)_{t+1}, (T_2)_{t+2}]$. With $|\mathrm{FP}(T_2, R)| = |\mathrm{FP}(T, R)| - 2$ and $|\mathrm{FP}(T_2', R)| = |\mathrm{FP}(T', R)| - 2$, this contradicts the assumption that $\mathrm{FP}(T, R)$ is a path of minimal length violating inequality (4.1).

Since all possible cases result in a contradiction, we conclude that inequality (4.1) is true for all trees, which completes the proof of the theorem. □

4.1.3 FindPath is optimal

We now show that no algorithm that solves the Shortest Path Problem in RNNI has strictly lower worst-case time complexity than FindPath. We therefore assume that the output of an algorithm for solving the Shortest Path Problem in RNNI is a list of RNNI moves. Requiring the output to be a list of trees would result in at least cubic complexity (if each tree can be returned in linear time).

Corollary 4.9. *The time-complexity of the Shortest Path Problem in* RNNI *is* $\Omega(n^2)$.

Proof. We prove this by establishing the lower bound on the output size to the problem, the length of a shortest paths. Note that the running time of FindPath, and therefore the maximum length of a shortest path in RNNI, is in $\mathcal{O}(n^2)$ (Proposition 4.3). The time-complexity of the Shortest Path Problem in RNNI is hence at most quadratic in the number of leaves. To prove that it is actually $\Omega(n^2)$, consider the following two caterpillar trees T and R (Figure 4.19):

$$T = [\{a_1, a_2\}, \{a_1, a_2, a_3\}, \ldots, \{a_1, a_2, \ldots, a_n\}]$$
$$R = [\{a_1, a_n\}, \{a_1, a_n, a_{n-1}\}, \ldots, \{a_1, a_n, \ldots, a_2\}]$$

Applied to these trees, FindPath decreases the rank of the parent of a_{n-k-1} in iteration i. At the beginning of each iteration, $p(a_{n-k-1})$ is the root, so $n-1-k$ NNI moves are needed in iteration k to move this node down until it has rank k. In other words, FindPath executes an NNI move in each of the $n-k-1$ while loops (line 3) in every iteration k of the for loop (line 2). The length of the output path of FindPath is hence $\sum_{k=1}^{n-2} k = \frac{(n-1)(n-2)}{2}$ and therefore quadratic in n. Theorem 4.6 then implies that

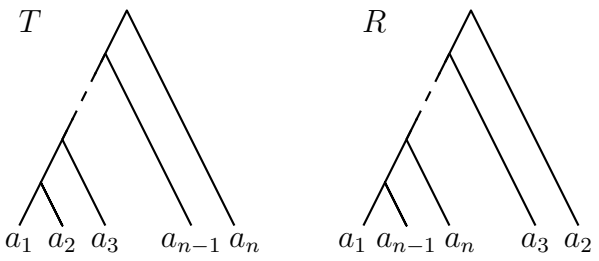


Figure 4.19: Caterpillar trees T and R in RNNI with distance $\frac{(n-1)(n-2)}{2}$ (Corollary 4.9)

this path is a shortest path. It follows that the worst-case size of the output to the Shortest Path Problem in RNNI is quadratic. □

4.2 Shortest Paths in DCT_m

In the previous section we have seen that shortest paths, and therefore distances, between ranked trees in the tree space RNNI can be computed in time quadratic in the number of leaves, using the algorithm FindPath. Since RNNI is a special case of DCT_m for $m = n-1$, the question arises whether the Shortest Path Problem in DCT_m can be solved in polynomial time for any $m \geq n-1$. Note that $m \geq n-1$, because m is defined as the maximum root time, which must be greater than or equal to the number of internal nodes, $n-1$. We answer this question in this section by showing that FindPath can actually be used for computing shortest paths between discrete coalescent trees as well.

We first see that for all $m \geq n-1$, shortest paths in DCT_m are not unique (Proposition 4.10). We then present an algorithm (Algorithm 3) to convert trees in DCT_m on n leaves into ranked trees on $m+2$ leaves in Section 4.2.1. After this conversion, we can apply FindPath to compute shortest paths, an hence distances, between the ranked trees on $m+2$ leaves. In Corollary 4.13 we show that the resulting RNNI distance equals the distance between the original discrete coalescent trees in DCT_m. We conclude the discussion of the Shortest Path Problem in DCT_m by providing a modification of FindPath in Section 4.2.2 that can be applied to discrete coalescent trees directly without first transforming them to ranked trees.

Proposition 4.10. *Shortest paths in* DCT_m *are not unique for* $n > 3$.

Proof. We assume that n , the number of leaves, is odd. The proof also works for even n, one just needs to replace $n-1$ by $n-2$ throughout this proof. Consider the following trees T and R, which equal the ranked trees in Proposition 4.1 (see also Figure 4.1),

but are now given in the cluster representation for discrete coalescent trees:

$$T = [\{a_1, a_2\} : 1, \{a_1, a_2, a_3\} : 2, \{a_1, a_2, a_3, a_4\} : 3, \ldots, \{a_1, a_2, \ldots, a_n\} : n-1]$$
$$R = [\{a_1, a_3\} : 1, \{a_1, a_2, a_3\} : 2, \{a_1, a_2, a_3, a_5\} : 3, \ldots, \{a_1, a_2, \ldots, a_n\} : n-1]$$

Note that these trees have root time less than or equal to m for any $m \geq n-1$, and are hence present in all spaces DCT_m for $m \geq n-1$. Like in RNNI (Proposition 4.1), any path between these two trees in DCT_m requires all pairs of leaves a_i, a_{i+1} for even $i < n$ to swap positions. The moves required for this are $\frac{n-1}{2}$ NNI moves, which can be performed in any order. As in Proposition 4.1, we can infer that there are $(\frac{n-1}{2})!$ shortest paths connecting T and R in DCT_m. □

4.2.1 Extending discrete coalescent trees to ranked trees

We now introduce an algorithm (Algorithm 3) to convert discrete coalescent trees on n leaves with maximum root time m into ranked trees on $m+2$ leaves. Let T be a discrete coalescent tree in DCT_m with $m > n-1$. Let $i_1 < i_2 < \ldots < i_{m-n+1}$ be the integers in $\{1, \ldots, m\}$ that are not present as times of internal nodes in T. Since we assumed $m > n-1$, there is at least one such i_j. To transform T into a ranked tree, we add internal nodes with times $i_1, \ldots, i_{m-n+1}$.

We therefore create a caterpillar tree T_r^c with leaf set $\{a_{n+1}, a_{n+2}, \ldots, a_{m+2}\}$ as follows. We first introduce an internal node v_1 with leaves a_{n+1} and a_{n+2} as children, and assign rank i_1 to v_1 (line 4 in Algorithm 3). Further internal nodes $v_2, \ldots, v_{m-n+1}$ are added iteratively (line 6 in Algorithm 3): In iteration k (for $k = 2, \ldots, m-n+1$), v_k is added as the parent of v_{k-1}, the internal node added in the previous iteration, and the leaf a_{n+1+k}. Furthermore, v_k is assigned time i_k. This means in particular that every internal node in T_r^c has at least one child that is a leaf, hence T_r^c is a caterpillar tree. In the end, we set T and T_r^c to be the children of a newly introduced root with time $m+1$, resulting in a ranked tree T_r (line 7 in Algorithm 3).

An example of this extension of a tree T to a ranked tree T_r is depicted in Figure 4.20.

For a tree T in DCT_m we call the ranked tree with $m+2$ leaves resulting from applying Algorithm 3 the *extended ranked version* of T and denote it by T_r. Moreover, we denote the subtree of T_r that is identical to T by T_r^d (d for discrete coalescent tree) and the caterpillar subtree on leaf set $\{a_{n+1}, \ldots, a_{m+2}\}$ by T_r^c.

Algorithm 3 ExtendedRankedTree(T, m)

1: $S := \{1 \leq i \leq m \mid$ no internal node in T has time $i\}$
2: $[i_1, \ldots, i_{m-n+1}] = sort(S)$
3: $T_r^d =$ copy of T
4: $T_r^c =$ tree consisting of one internal node v_1 with rank i_1 and children a_{n+1}, a_{n+2}
5: **for** $k = 2, \ldots, m - n + 1$ **do**
6: Add internal node v_k with with time i_k and children v_{k-1} and a_{n+1+k} to T_r^c
7: $T_r =$ tree with root with time $m + 1$ and children of root are roots of T_r^d and T_r^c.
8: **return** T_r

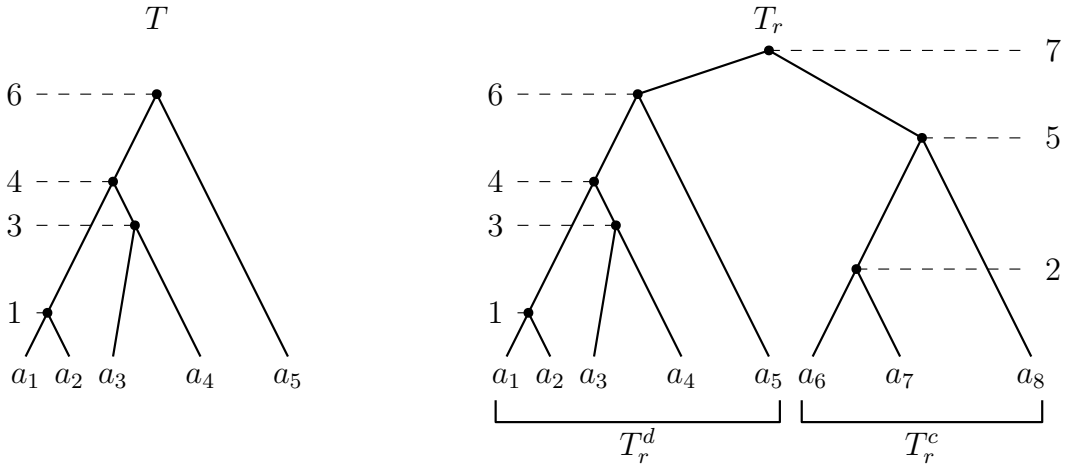


Figure 4.20: Extending a tree T on $n = 5$ leaves in DCT_6 (left) to its extended ranked version T_r with $m + 2 = 8$ leaves (right) by adding a caterpillar subtree with three leaves. The nodes v_1 and v_2 added to T as described in Algorithm 3 have ranks 2 and 5 in T_r, respectively.

Let us now consider the worst-case running time of Algorithm 3. With an appropriate data structure, the input tree T can be copied in time linear in m. The list $[i_1, \ldots, i_{m-n+1}]$ can also be derived in linear time by iterating through $1, \ldots, m$ and appending the integers to the list that are not present as times of internal nodes in T. The for loop in line 6 runs $m - n + 1$ times, and adding a new internal node in this loop can be done in constant time. The overall worst-case running time of Algorithm 3 is hence in $\mathcal{O}(m)$.

We transform discrete coalescent trees into ranked trees to be able to use FindPath to compute shortest paths. Therefore, we now show that for two trees T, R in DCT_m, the path $\mathrm{FP}(T_r, R_r)$ in RNNI can be modified to be a path in DCT_m between T and R. More specifically, we want to show that the resulting path is a shortest path in

DCT_m. To prove this, we first establish the following two lemmas, which show how paths between T and R in DCT_m relate to paths between T_r and R_r in RNNI.

Lemma 4.11. *Let p be a path in* DCT_m*. Transforming every tree on p into its extended ranked version (Algorithm 3) results in a path p_r in* RNNI *such that $|p| = |p_r|$.*

Proof. Let p a path in DCT_m and p_r the sequence of trees that result from transforming every tree on p into its extended ranked version (Algorithm 3). Let furthermore T and R be consecutive trees on p and T_r and R_r their extended ranked versions on p_r. To prove the lemma, we now show that T_r and R_r are connected by an RNNI move, from which we can infer that p_r is a valid path in RNNI. We do this by distinguishing RNNI moves from length moves between trees T and R on p and see how these correspond to moves on p_r between T_r and R_r. Note that T is identical to the subtree T_r^d of T_r, and the same is true for R and R_r^d (see line 3 of Algorithm 3).

RNNI **move.** If an NNI move or rank move is performed on T to result in R, the subtrees R_r^d and T_r^d are connected by exactly the same move, as they are identical to T and R. Neither an NNI move nor a rank move changes the set of times that are assigned to internal nodes of a tree. This implies that the set of times assigned to internal nodes in T is the same as for R. With Algorithm 3 it follows that the caterpillar trees T_r^c and R_r^c are identical. We can conclude that T_r and R_r are neighbours in RNNI, since R_r results from an RNNI move on T_r, more specifically on the subtree T_r^d.

Length move. If there is a length move on p between T and R, the time of an internal node is increased or decreased by one in R in comparison to T. Let t be the the time of that internal node in T that changes to $t + i$ for $i \in \{1, -1\}$ in R. Note that the time cannot change to become $m + 1$, as p is a path in DCT_m. There is hence a node in T that has time t, but none with time $t + i$, while the node inducing the same cluster as $(T)_t$ has time $t + i$ in R and no node with time t exists there. All other nodes of the trees T and R coincide.

For the extended ranked version T_r of T this means that there is an internal node of rank $t + i$ in the subtree T_r^c, as by the construction of the extended ranked version of a tree every integer in $\{a_1, \ldots, a_m\}$ is assigned as a time to an internal node in T_r. Similarly, there is an internal node of rank t in R_r^c, but none with rank $t + i$. All other nodes coincide in T_r^c and R_r^c and the difference between T_r^d and R_r^d is the same as between T and R, that is, the time of one internal

node that changes from t to $t+i$. We can conclude that there is a rank move between T_r and R_r swapping the ranks of the internal nodes of rank t and $t+i$ for $i \in \{1, -1\}$. An example of such a length move and the corresponding rank move is depicted in Figure 4.20. T_r and R_r are hence RNNI neighbours.

We can conclude that for every pair of consecutive trees T, R on p, their extended ranked versions T_r and R_r are connected by a rank or NNI move. Furthermore, the lengths of p and p_r coincide, which completes the proof of this lemma. □

Lemma 4.11 shows that paths between trees T and R in DCT_m can be transformed to paths in RNNI between the extended ranked version T_r and R_r of T and R. Rank and NNI moves in DCT_m translate to the same type of move in RNNI, while length moves in DCT_m correspond to rank moves in RNNI. More specifically, a length move on a discrete coalescent tree T corresponds to a rank move that swaps the rank of a node in the subtree T_r^d and a node in T_r^c. We refer to such rank swaps between nodes in T_r^d and T_r^c as *rank moves corresponding to length moves*. An example of a rank move corresponding to a length move, and the corresponding length move, is shown in Figure 4.21.

We now show that the path $\mathrm{FP}(T_r, R_r)$ between the extended ranked versions of trees T, R in DCT_m can be transformed into a path in DCT_m between T and R.

Lemma 4.12. *Let $\hat{T}$ and $\hat{R}$ be two discrete coalescent trees in DCT_m and $\hat{T}_r$ and $\hat{R}_r$ their extended ranked versions. Let p be the sequence of trees that consists of the subtrees induced by $\{a_1, \ldots, a_n\}$ in the trees in $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$, maintaining the order of the trees on the path. Then p is a path between $\hat{T}$ and $\hat{R}$ in DCT_m with $|p| = |\mathrm{FP}(\hat{T}_r, \hat{R}_r)|$.*

Proof. Let $\hat{T}$ and $\hat{R}$ be trees in DCT_m and $\hat{T}_r$ and $\hat{R}_r$ their extended ranked versions. We furthermore assume that p is the sequence of trees that results from $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ by only considering the subtrees induced by $\{a_1, \ldots, a_n\}$. We first show that every tree on p is a discrete coalescent tree on n leaves with maximum root time m.

With Lemma 4.4 we know that the cluster $\{a_1, \ldots, a_n\}$, that is shared between $\hat{T}$ and $\hat{R}$, is present in every tree on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$. This implies that the trees on p are well defined as subtrees induced by the cluster $\{a_1, \ldots, a_n\}$ of trees on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$.

To show that the trees on p are in DCT_m, it remains to prove that their root time is less than or equal to m. It follows from Algorithm 3 (line 7) that the roots of all discrete coalescent trees on p have rank $m+1$. Because the node inducing the cluster $\{a_1, \ldots, a_n\}$ is child of the root in all trees on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$, it has rank less than or

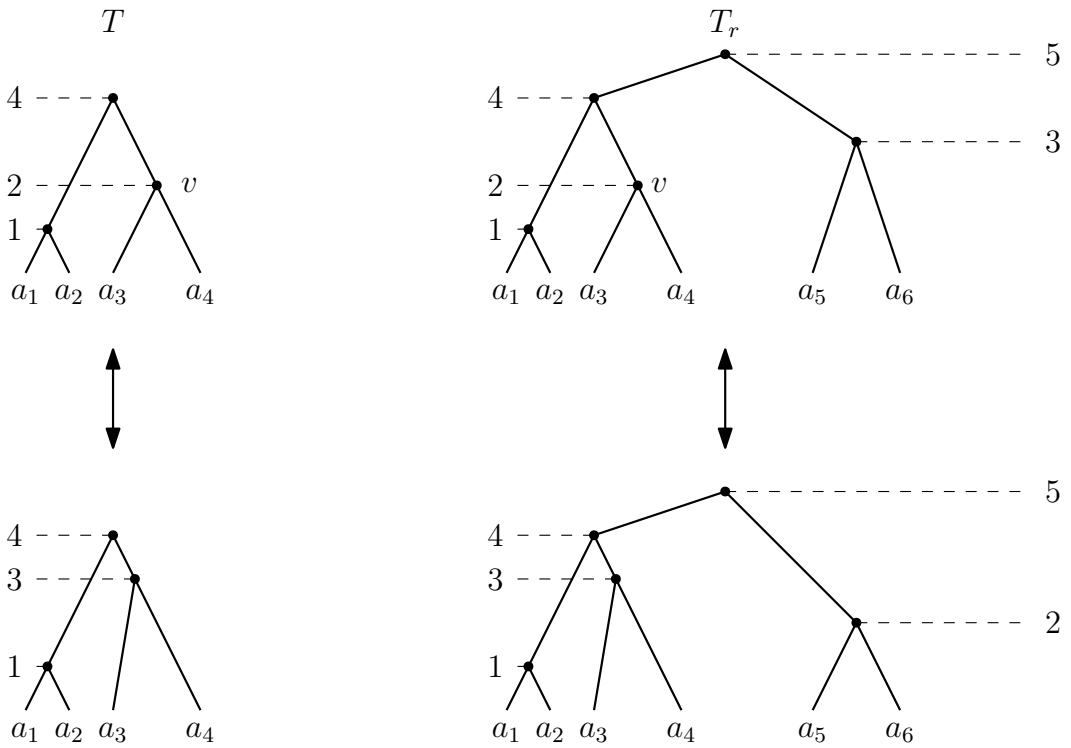


Figure 4.21: Length move on a tree T on $n = 4$ leaves in DCT_4 (left) and the corresponding rank move on the extended ranked version T_r of T (right). This move changes the time of the node v from 2 to 3.

equal to m. So the subtree induced by $\{a_1, \ldots, a_n\}$ is a tree in DCT_m for every tree on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$, i.e. p is a sequence of discrete coalescent trees in DCT_m.

Since $\hat{T}$ and $\hat{R}$ are identical to the subtrees $\hat{T}_r^d$ and $\hat{R}_r^d$, respectively, the first tree on p is $\hat{T}$ and the last tree $\hat{R}$. To finish the proof that p is a valid path in DCT_m, we need to show that every consecutive pair of trees on p is connected by an NNI move, rank move, or length move. Therefore, we assume that T_r and R_r are consecutive trees on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ and distinguish the type of move (NNI/rank move) between T_r and R_r.

NNI **move.** By the construction of the extended ranked versions $\hat{T}_r$ and $\hat{R}_r$ of $\hat{T}$ and $\hat{R}$ with Algorithm 3, all clusters in the added caterpillar subtrees $\hat{T}_r^c$ and $\hat{R}_r^c$ coincide. Note that the ranks of the nodes inducing these cluster do not need to coincide. With Lemma 4.4 there cannot be a move on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ that changes any of these clusters. This implies that there is no NNI move in the caterpillar subtree T_r^c, as NNI moves change clusters. If the move between T_r and R_r is an NNI move, it must hence be an NNI move in the subtree T_r^d. We know that the subtree T_r^d is identical to T and R_r^d is identical to R. Therefore, T and R are connected by the same NNI move in DCT_m as T_r and R_r in RNNI.

Rank move. If the move between T_r and R_r is a rank move, it can either be a rank move between two nodes in T_r^d, or between one node in T_r^d and one node in T_r^c. Note that no rank move inside T_r^c is possible, as this subtree is a caterpillar tree. If both nodes that swap ranks are inside the subtree T_r^d, all other nodes stay the same between T_r and R_r, and the discrete coalescent trees T and R are connected by the same rank move on p.

We now consider a rank move swapping the rank t of a node in the subtree T_r^d with the rank $t+i$ of a node in T_r^c, with $i \in \{1, -1\}$. The only difference between the subtrees T_r^d and R_r^d, which are identical to T and R, is the time of one internal node, which changes from t to $t+i$. T and R are therefore connected by a length move.

We can conclude that all pairs of consecutive trees T and R on p are connected by an NNI move, a rank move, or a length move in DCT_m. Thus, p is a path from $\hat{T}$ to $\hat{R}$ in DCT_m and has length $|p| = |\mathrm{FP}(\hat{T}_r, \hat{R}_r)|$, which completes this proof. □

With Lemma 4.11 and Lemma 4.12 we can now do the following to compute a path between two trees T and R in DCT_m: First extend trees T and R to their extended ranked versions T_r and R_r with $m+2$ leaves, by Algorithm 3. Then compute a shortest paths between T_r and R_r in RNNI, using FindPath. The path $\mathrm{FP}(T_r, R_r)$ then provides a path between T and R in DCT_m, when only considering the subtrees induced by $\{a_1, \ldots, a_n\}$ in all trees on $\mathrm{FP}(T_r, R_r)$.

In Corollary 4.13 we establish that the resulting path is indeed a shortest path in DCT_m. Note that for any given pair of trees T and R, we always assume m to be the maximum root time of these trees and consider a shortest path between them in DCT_m.

Corollary 4.13. *Let T and R be discrete coalescent trees and T_r and R_r their extended ranked versions. Let $d_{\mathrm{DCT}_m}(T, R)$ be the distance of T and R in DCT_m and $d_{\mathrm{RNNI}}(T_r, R_r)$ the distance between their extended ranked versions in* RNNI*, where m is greater than or equal to the maximum root time of T and R. Then*

$$d_{\mathrm{DCT}_m}(T, R) = d_{\mathrm{RNNI}}(T_r, R_r).$$

Proof. Let T and R be discrete coalescent trees in DCT_m, where m is greater than or equal to the maximum root time of T and R. Furthermore, let p be a shortest path in DCT_m connecting T and R. With Lemma 4.11 we can transform any path from T to

R in DCT_m into a path from T_r to R_r in RNNI, such that the length of the path is preserved. Especially, the path p_r in RNNI, consisting of the extended ranked versions of trees on p, has length $|p_r| = |p|$. It follows $|p| = |p_r| \geq d_{\mathrm{RNNI}}(T_r, R_r)$.

From Lemma 4.12 we also know that the path $\mathrm{FP}(T_r, R_r)$ in RNNI between the extended ranked versions T_r and R_r can be transformed into a path p' in DCT_m between trees T and R such that $|p'| = |\mathrm{FP}(T_r, R_r)| = d_{\mathrm{RNNI}}(T_r, R_r)$. This especially implies for the shortest path p between T and R in DCT_m that $|p| \leq |p'| = d_{\mathrm{RNNI}}(T_r, R_r)$.

We can conclude $|p| \geq d_{\mathrm{RNNI}}(T_r, R_r)$ and $|p| \leq d_{\mathrm{RNNI}}(T_r, R_r)$, and hence $d_{\mathrm{DCT}_m}(T, R) = |p| = d_{\mathrm{RNNI}}(T_r, R_r)$. □

Corollary 4.14. *Shortest paths between trees in* DCT_m *can be computed in time polynomial in* m*.*

Proof. Let T and R be trees on n leaves in DCT_m. To compute a shortest path between these trees, we first apply Algorithm 3 to both trees to receive their extended ranked versions T_r and R_r. This algorithm runs in time $\mathcal{O}(m)$, as discussed just after the introduction of Algorithm 3. Afterwards, FindPath can be used to compute a path between T_r and R_r. Since T_r and R_r are trees on $m + 2$ leaves in RNNI, the running time of FindPath for computing a shortest path between them is in $\mathcal{O}(m^2)$. With Lemma 4.12 and Corollary 4.13 we know that by taking the subtree induced by the set $\{a_1, \ldots, a_n\}$ for all trees on $\mathrm{FP}(T_r, R_r)$, we receive a shortest path in DCT_m. To get this subtree, one can check for each internal node if it has descending leaves outside of $\{a_1, \ldots, a_n\}$, and delete that node from the tree if this is the case. By doing this for all internal nodes, the subtree induced by the cluster $\{a_1, \ldots, a_n\}$ can be found in in $\mathcal{O}(m)$ (depending on the data structure) for each tree on $\mathrm{FP}(T_r, R_r)$. Converting the entire path to a path in DCT_m can hence be done in $\mathcal{O}(m^3)$, as the running time $\mathcal{O}(m^2)$ of FindPath is an upper bound for the length of the path $\mathrm{FP}(T_r, R_r)$ (Corollary 4.9).

We can conclude that the worst-case running time for computing a shortest path in DCT_m using the extended ranked versions of the input tree and FindPath in RNNI is $\mathcal{O}(m^3)$. □

4.2.2 FindPath$^+$ – An extension of FindPath to DCT_m

In this section we introduce a new algorithm FindPath$^+$ (Algorithm 4), which is a modification of FindPath that computes shortest paths in DCT_m more efficiently.

The idea behind FindPath$^+$ (Algorithm 4) is to modify FindPath to an algorithm that is applicable to trees in DCT_m directly without needing to transform them into

ranked trees. This saves both running time and space needed for saving trees and paths. Remember that the shortest path p between two trees $\hat{T}$ and $\hat{R}$ in DCT_m that we get by using FindPath is the restriction of all trees on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ to their subtree induced by $\{a_1, \ldots, a_n\}$. To find an algorithm that computes this path p without computing $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ first, we consider all moves possible on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ and see how they change the subtrees induced by $\{a_1, \ldots, a_n\}$. In the proof of Lemma 4.12 we have already seen that all moves inside the subtree induced by $\{a_1, \ldots, a_n\}$ happen on the path p in exactly the same way as on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$. In the same proof we have also seen that the only other moves possible on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ are rank moves corresponding to length moves.

For adapting FindPath to not need the internal nodes in the caterpillar tree that is added in the extended ranked version of the tree, we hence only need to replace the rank moves corresponding to length moves by length moves. In the following we explain how this is done.

Let T_r and R_r be two subsequent trees on a path p that results form restricting all trees on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ to their subtrees induced by $\{a_1, \ldots, a_n\}$. Let furthermore R_r result from T_r by a rank move corresponding to a length move and C be the cluster that is considered on FindPath at this point. The rank of the most recent common ancestor of C in R_r is hence one less than in T_r. We distinguish two cases:

Case 1: C is induced by a node in T_r^d.

In this case, the rank move between T_r and R_r decreases the rank of the node $mrca(C)_{T_r}$, which is in T_r^d, by one. This move translates to a length move on the discrete coalescent tree that decreases the time of the node $mrca(C)_T$ in T by one.

Case 2: C is induced by a node in T_r^c.

The rank move between T_r and R_r decreases the rank of the node $mrca(C)_{T_r}$, which is in T_r^c, by one. Therefore, the rank of the node in the subtree T_r^d that is involved in this rank move increases by one. The length move corresponding to this rank move hence increases the time of a node in T. An example of this scenario is depicted in Figure 4.21: The displayed move decreases the rank of the most recent common ancestor of $\{a_5, a_6\}$ in the tree T_r^c and increases the time of the node v in T_r^d, and therefore in T, at the same time.

We now consider iterations of FindPath that perform length moves increasing the rank of a node T_r^d (as in case 2). Let T_r be the tree in an iteration k of FindPath

applied to $\hat{T}_r, \hat{R}_r$ in DCT_m such that the cluster C of $\hat{R}_r$ considered in this iteration is in T_r^c. Let furthermore $i = \mathrm{rank}(mrca(C)_{T_r})$ be the rank of the $mrca$ of C in T_r, while k is the rank to which FindPath decreases the rank of $mrca(C)$, i.e. k is the rank of the node inducing C in R_r. Since $\hat{T}_r$ and $\hat{R}_r$ share all clusters induced by nodes in the subtree on the leaf subset $\{a_{n+1}, \ldots, a_{m+2}\}$ and FindPath preserves clusters (Lemma 4.4), all trees on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ also have these clusters. It follows that all moves decreasing the rank of $mrca(C)$ on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ from i to k must be rank moves corresponding to length moves. We have seen in case 2 above that each such rank move corresponds to a length move increasing the time of an internal node in the subtree induced by $\{a_1, \ldots, a_n\}$. This means that the sequence of rank moves decreasing the rank of $mrca(C)$ from i to k translates to length moves that increase the times of each node with time between k and i in the subtree induced by $\{a_1, \ldots, a_n\}$. An example of a sequence of such rank moves on $\mathrm{FP}(\hat{T}_r, \hat{R}_r)$ and the corresponding length moves between discrete coalescent trees $\hat{T}$ and $\hat{R}$ is depicted in Figure 4.22. Note that in this example $i = 4$ and $k = 2$.

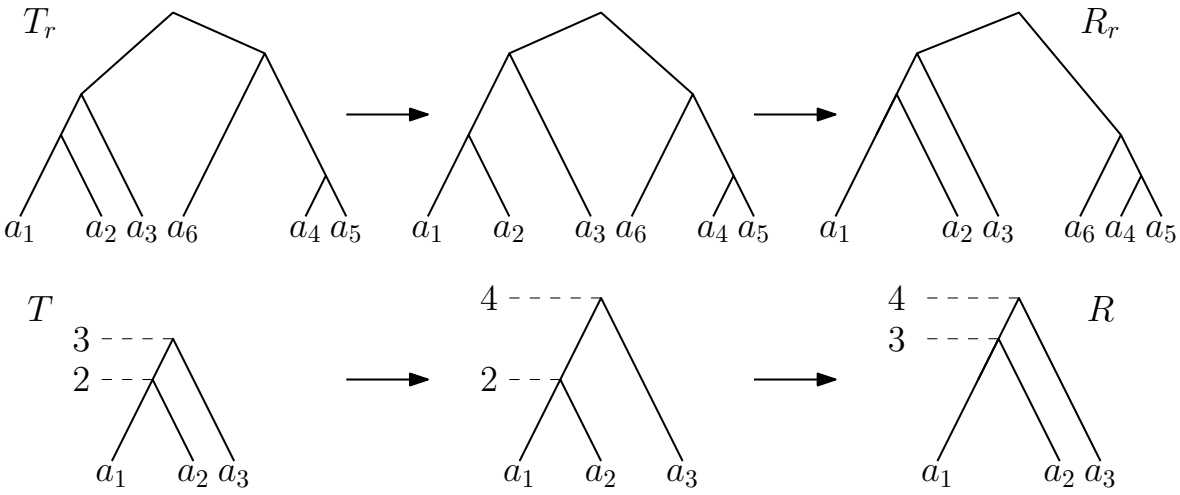


Figure 4.22: Ranked trees $\hat{T}_r$ and $\hat{R}_r$ connected by a path consisting of two rank moves decreasing the rank of $mrca(\{a_4, a_5, a_6\})$ on the top. The moves between the corresponding discrete coalescent trees are length moves increasing the time of the nodes that have time 2 and 3 in $\hat{T}$.

By replacing rank moves corresponding to length moves with the appropriate length moves and keeping all other moves of FindPath the same, we get the following algorithm FindPath$^+$ that computes shortest paths between discrete coalescent trees in DCT_m.

For input trees T, R in DCT_m, FindPath$^+$ iterates through all times $k = 1, \ldots, m$ that internal nodes could have in R to construct a path p, initially starting with $p = [T]$. If in iteration k, R has an internal node with time k that induces a cluster C, the most recent common ancestor of C in the currently last tree T_1 on p is decreased by NNI, rank, or length moves, until it reaches rank k. If there is no node with time k in R, we find for the lowest integer i that is greater than k such that there is no internal node in T_1 that is assigned the time i (line 10 in Algorithm 4). FindPath$^+$ increases the time of all internal nodes that have rank between k and i in T_1 by one, which requires length moves (line 11), ending in a tree that does not have an internal node with time k.

Algorithm 4 FindPath$^+(T, R)$

1: $T_1 := T$, $p := [T_1]$
2: **for** $k = 1, \ldots, m$ **do**
3: **if** R has a node with time k **then**
4: $C := (R)_k$
5: **while** time$(mrca(C)_{T_1}) > k$ **do**
6: T_2 is T_1 with the time of $mrca(C)_{T_1}$ decreased by an RNNI move
7: $T_1 = T_2$
8: $p = p + T_1$
9: **else if** T_1 has a node with time k **then**
10: $i := \min\{l \mid l > k$ and no node in T_1 has time $l\}$
11: **for** $j = i - 1, \ldots, k$ **do**
12: T_2 is T_1 where the time of $(T_1)_j$ is increased by one (length move)
13: $T_1 = T_2$
14: $p = p + T_1$
15: **return** p

From our construction above we can infer that every tree on the path computed by FindPath$^+$ is the same as the tree at the same position on $\mathrm{FP}(T_r, R_r)$ restricted to the subtrees induced by $\{a_1, \ldots, a_n\}$. With Corollary 4.13 it follows that FindPath$^+$ computes shortest paths in DCT_m. We denote the path resulting from applying FindPath$^+$ to trees T and R by $\mathrm{FP}^+(T, R)$.

Note that the running time of FindPath$^+$ depends on the number of moves needed on the output path. The for loop in line 2 is executed m times, but it is not obvious how often the while loop in line 5 and the for loop in line 11 are executed. In every execution of these loops a new tree is added to the computed path. The maximum running time of FindPath hence depends on the maximum length a shortest path in DCT_m can have. We will see in Section 5.3 that the maximum length of a shortest path in DCT_m, and hence the worst-case running time of FindPath$^+$, is in $\mathcal{O}(mn)$.

When computing only the distance, but not an actual shortest paths between discrete coalescent trees, one can save running time further by replacing sequences of length moves by a single move, and add the difference of times between the node that has been moved. An illustration of a case where one can improve on running time by doing this is shown in Figure 4.23. The trees T and R in that figure only differ by the time of one internal node, which changes from 75 in T to 15 in R. Instead of computing 60 length moves needed on a path from T to R one can simply compute the time difference in constant time. Using this modification for computing distances leads to a great running time improvement for $m >> n$.

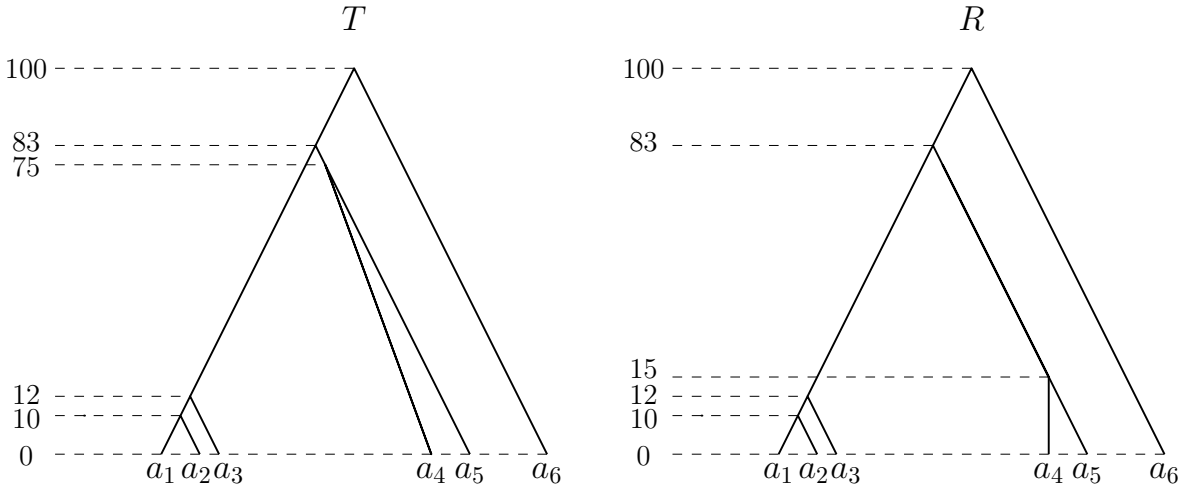


Figure 4.23: Tree R is $75 - 15 = 60$ length moves apart from T. Instead of computing every tree on this sequence of length moves, it is sufficient to compute R and add 60 to the distance.

Note that the parameter m is not needed in practice, as the distance between any two trees in $\mathrm{DCT}_{m'}$ is the same as their distance in DCT_m for any $m > m'$. This follows from the fact that in the extended ranked versions of trees T and R for $m > m'$ all clusters induced by nodes with rank greater than m' are identical in T_r and R_r, so they are preserved on $\mathrm{FP}(T_r, R_r)$. And since $d(T, R) = |\mathrm{FP}(T_r, R_r)|$ (Corollary 4.13),

the distance between T and R is the same in DCT_m for all $m > m'$. We can hence choose the maximum root time of the given trees as m to compute distances in DCT_m.

4.3 Non-ultrametric trees

In this section we provide a modification of the tree space DCT_m for trees where leaves do not have the same time (zero). These trees are of interest in applications where the data available at the leaves of the tree is not sampled at the same time. We first formally introduce non-ultrametric trees and a tree space for these trees (Section 4.3.1), and then show in Section 4.3.2 that we can use the algorithms FindPath$^+$ and FindPath to compute distances in this tree space.

A *non-ultrametric discrete coalescent tree* (or short *non-ultrametric tree*) is a rooted phylogenetic tree with positive integer-valued times uniquely assigned to each node, i.e. internal nodes and leaves. In contrast to discrete coalescent trees, leaves do not have time zero but have a unique positive integer as time. An example of a non-ultrametric tree is depicted in Figure 4.24.

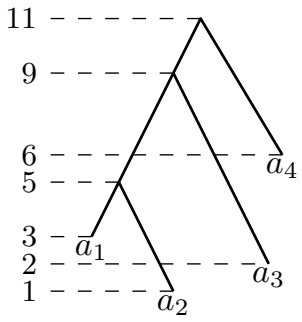


Figure 4.24: Non-ultrametric tree on 4 leaves.

We use the same terminology for non-ultrametric trees as for discrete coalescent trees but treat leaves as clusters containing a single element. The cluster representation of non-ultrametric trees hence contains $2n-1$ clusters: $n-1$ clusters for internal nodes and n single-element clusters representing leaves. The tree in Figure 4.24 in its cluster representation is:

$$\big[\{a_2\} : 1, \{a_3\} : 2, \{a_1\} : 3, \{a_1, a_2\} : 5, \{a_4\} : 6, \{a_1, a_2, a_3\} : 9, \{a_1, a_2, a_3, a_4\} : 11\big]$$

Since all nodes are assigned unique times, the root time of a non-ultrametric tree on n leaves is at least the sum of the number of leaves and the number of internal nodes: $n + n - 1 = 2n - 1$. If a tree has root time $2n - 1$, each integer in $\{1, 2, \ldots, 2n - 1\}$

is assigned as time to a node in the tree. We call these trees *non-ultrametric ranked trees*.

4.3.1 The tree space DCT_m^{nu}

We define a tree space of non-ultrametric trees with maximum root time m in the same way as DCT_m and call it DCT_m^{nu}. The vertices of DCT_m^{nu} are non-ultrametric trees on n leaves with maximum root time $m \geq 2n-1$. Two vertices in DCT_m^{nu} are connected by an edge if the corresponding trees are connected by a length move, a rank move, or an NNI move as defined in Section 3.2. Leaves can be involved in rank moves or length moves, but not in NNI moves. The reason for this is that NNI moves on edges incident to leaves cannot result in new trees. An example of a rank move and a length move changing the time of a leaf is depicted in Figure 4.25.

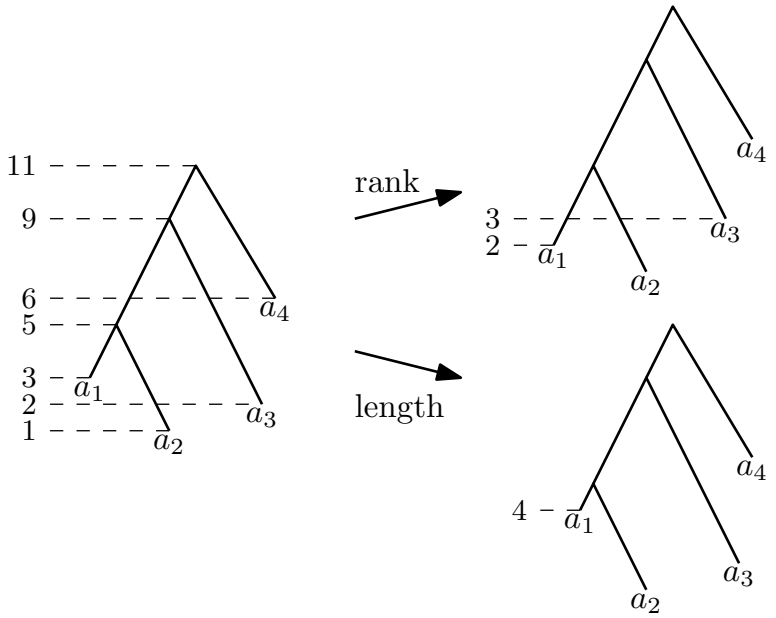


Figure 4.25: Non-ultrametric trees on 4 leaves. The trees on the left and the top right are connected by a rank move swapping the ranks of a_1 and a_3. The tree on the bottom right results from a length move increasing the time of a_1 in the tree on the left.

4.3.2 Shortest paths in DCT_m^{nu}

Let us now consider the Shortest Path Problem for non-ultrametric trees. We can directly apply the algorithms FindPath and FindPath$^+$ to DCT_m^{nu}. The input for these algorithms is then the cluster representation of non-ultrametric trees, which contains

one-element clusters for leaves. Note that the space DCT^{nu}_{2n-1} resembles RNNI in that there are no intervals of length greater than one in trees in these spaces. FindPath can hence be used in DCT^{nu}_{2n-1} while FindPath$^+$ can be applied to trees in DCT^{nu}_{m} for any $m \geq 2n-1$. Since FindPath$^+$ computes the same shortest path between ranked trees as FindPath, we only consider FindPath$^+$ in this section. To prove that the resulting paths are actually shortest paths in DCT^{nu}_{m}, we use a trick similar to the one in Section 4.2: we extend trees in DCT^{nu}_{m} on n leaves to trees in DCT_m on $2n$ leaves by adding leaves as described in the following.

We transform a non-ultrametric tree T to an ultrametric tree T' by replacing every leaf a_i in T by a cherry of two new leaves a_{i1} and a_{i2} and set the times of these new leaves to 0. The internal node of this new cherry is assigned the same time in T' as the leaf a_i in T. T' is hence an ultrametric tree with $2n$ leaves, which we call the *ultrametric version* of T. An example of a non-ultrametric tree and its ultrametric version is given in Figure 4.26. In the following lemma we show how paths between two non-ultrametric trees relate to the paths between their ultrametric versions.

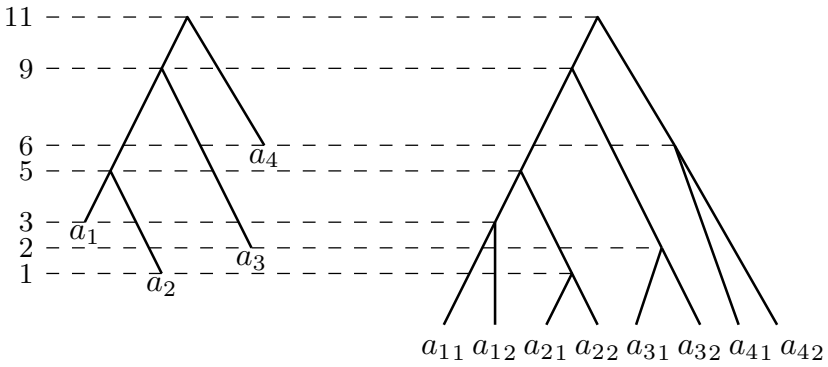


Figure 4.26: Non-ultrametric tree on 4 leaves on the left and its ultrametric version on 8 leaves on the right.

Lemma 4.15. *Let T and R be non-ultrametric trees on n leaves and p a path from T to R in* DCT^{nu}_{m}*. Furthermore, let T' and R' be the ultrametric versions of T and R on $2n$ leaves in* DCT_m*.*

The path p' that results from transforming every tree on p to its ultrametric version is a path from T' to R' in DCT_m*.*

Proof. To prove this lemma it is sufficient to show that for any pair of non-ultrametric trees T, R that are connected by an edge in DCT^{nu}_{m}, their ultrametric versions T', R' are connected by an edge in DCT_m. Then the lemma follows inductively. Let T and R be non-ultrametric trees connected by an edge in DCT^{nu}_{m} and let T' and R' be their

ultrametric versions. That T' and R' are connected by an edge in DCT_m follows from the definition of DCT_m^{nu} based on rank moves, NNI moves, and length moves, like DCT_m. Since T' contains the same nodes as T, plus $2n$ additional nodes added as leaves, any move on T is possible on T' in the same way. This implies in particular, that the move between T and R can be performed on T', too, and changes T' in the same way as it changes T. Applying this move to T' hence results in the tree R', the ultrametric version R' of R.

We conclude that for any trees T and R that are connected by an edge in DCT_m^{nu}, their ultrametric versions T' and R' are connected by an edge in DCT_m, which concludes this proof. □

Lemma 4.16. *Let T and R be non-ultrametric trees on n leaves in DCT_m^{nu}. Furthermore, let T' and R' be the ultrametric versions of T and R on $2n$ leaves in DCT_m.*

$\mathrm{FP}^+(T', R')$ is then the path that results from transforming every tree on $\mathrm{FP}^+(T, R)$ to its ultrametric version.

Proof. Let T and R be non-ultrametric trees on n leaves in DCT_m^{nu} and T' and R' their ultrametric versions on $2n$ leaves in DCT_m. Both trees T' and R' have the n cherries $\{a_{i1}, a_{i2}\}$ for $i = 1, \ldots, n$. Whenever FindPath$^+$ applied to T and R considers sets including a_i, the cluster considered on $\mathrm{FP}^+(T', R')$ is the same, except for elements a_i that are replaced by $\{a_{i1}, a_{i2}\}$. We can conclude that the path that results from transforming every tree on $\mathrm{FP}^+(T, R)$ to its ultrametric version is $\mathrm{FP}^+(T', R')$. □

An example of paths $\mathrm{FP}^+(T, R)$ and $\mathrm{FP}^+(T', R')$ for ultrametric versions T' and R' of non-ultrametric trees T and R is depicted in Figure 4.27.

Theorem 4.17. FindPath$^+$ *computes shortest paths in DCT_m^{nu}.*

Proof. Let T and R be two trees in DCT_m^{nu} on n leaves. Lemma 4.16 implies that for the ultrametric versions T' and R' of T and R, $|\mathrm{FP}^+(T, R)| = |\mathrm{FP}^+(T', R')|$.

We now prove the theorem by assuming to the contrary that there is a path p between T and R in DCT_m^{nu} that is shorter than $\mathrm{FP}^+(T, R)$. Let p' be the path that results from replacing every tree on p by its ultrametric version. Then p' is a path between T' and R' in DCT_m on $2n$ leaves with length $|p'| = |p| < |\mathrm{FP}^+(T, R)| = |\mathrm{FP}^+(T', R')|$ (Lemma 4.15). This however is a contradiction to FindPath computing shortest paths in DCT_m. We conclude that there is no path p shorter that $\mathrm{FP}^+(T, R)$ in DCT_m^{nu}, which implies that FindPath$^+$ computes shortest paths in DCT_m^{nu}. □

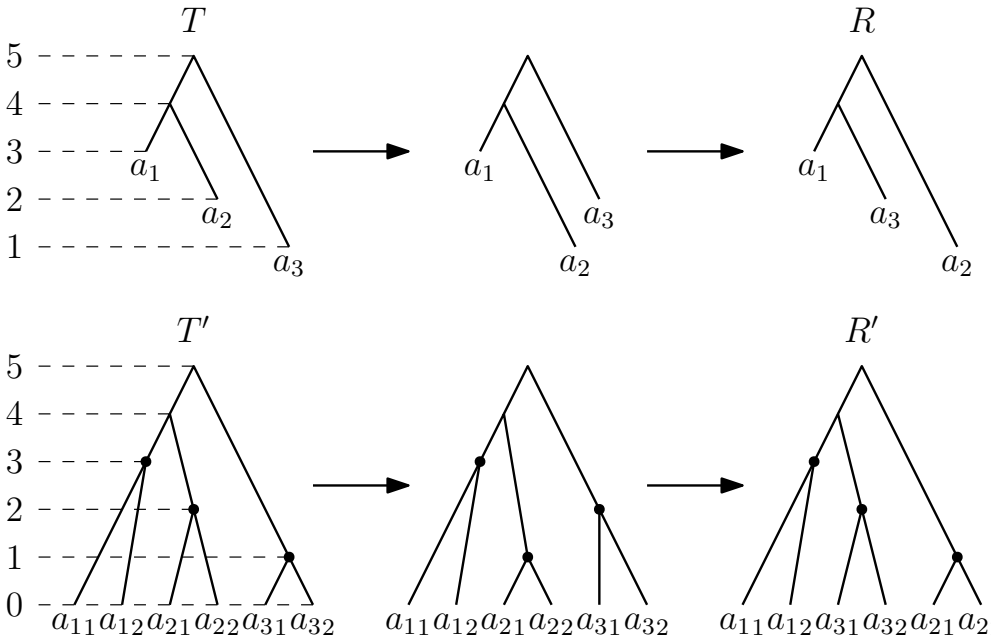


Figure 4.27: $\mathrm{FP}^+(T, R)$ between non-ultrametric trees T and R on 3 leaves in DCT_5^{nu} in the top row. $\mathrm{FP}^+(T', R')$ in DCT_5 on 6 leaves between the ultrametric versions T' and R' of T and R in the bottom row.

Chapter 5

Geometrical properties

In this chapter we investigate properties of the discrete coalescent tree spaces DCT_m, including in particular the RNNI space (DCT_{n-1}). We focus on characteristics of shortest paths in these tree spaces. In Section 5.1 we show that for two trees sharing a cluster, all trees on all shortest paths between these trees in DCT_m also have that cluster. The discussion of this property, which we call cluster property, is followed by a section analysing shortest paths between caterpillar trees (Section 5.2). There we show that every pair of caterpillar trees is connected by a shortest path consisting only of caterpillar trees in DCT_m. We use this property to present a way to compute distances between caterpillar trees in $\mathcal{O}(n\sqrt{\log n})$ time, which is more efficient than using FindPath. Afterwards we discuss diameter and radius of DCT_m in Section 5.3. We conclude this chapter with a study of the distribution of distances between ranked trees sampled from a uniform distribution (Section 5.4).

5.1 Cluster Property

A tree space has the *cluster property* if all trees on every shortest path between two trees sharing a cluster C also contain C as cluster. In other words, a tree space has the cluster property if every shortest path preserves all clusters shared between the start and end tree of that path. The NNI space does not have the cluster property (Li, Tromp, and Zhang 1996), which has been used by Dasgupta et al. (2000) to prove that computing the NNI distance is $\mathcal{NP}$-hard. Since distances in DCT_m can be computed in polynomial time (Corollary 4.14), the question whether DCT_m has the cluster property arises. The fact that the paths computed by FindPath preserve clusters (Lemma 4.4) implies that there is at least one shortest path that preserves clusters. We extend this

result to all shortest paths in the RNNI space (Theorem 5.2), before proving the same for the more general tree space DCT_m. We then prove that the cluster property holds on the space DCT_m^{nu} of non-ultrametric trees (Corollary 5.4).

For proving the cluster property for RNNI, we need the following Lemma.

Lemma 5.1. *Let T and T' be ranked trees in* RNNI *that are connected by an* RNNI *move on the interval bounded by nodes with ranks i and $i+1$. Let R be a tree such that the clusters induced by $(R)_j$ and $(T)_j$ coincide for all $j \leq i+1$, and let R' be the* RNNI *neighbour of R with $(R')_i = (T')_i$ and $(R')_{i+1} = (T')_{i+1}$.*

Then $|\mathrm{FP}(T,R)| = |\mathrm{FP}(T',R')|$.

Note that the RNNI move connecting R and R' in Lemma 5.1 is identical to the one connecting T and T'.

Proof. Let T, T', R and R' be trees as described in the lemma. We furthermore assume $T \neq T'$ and $R \neq R'$, as otherwise $|\mathrm{FP}(T,R)| = 0 = |\mathrm{FP}(T',R')|$, in which case the lemma is trivially true. We show that the first move on $\mathrm{FP}(T,R)$ changes clusters of T in the same way as the first move on $\mathrm{FP}(T',R')$ changes clusters of T'. Then it follows inductively that the moves on $\mathrm{FP}(T,R)$ are equivalent to the moves on $\mathrm{FP}(T',R')$ and hence $|\mathrm{FP}(T,R)| = |\mathrm{FP}(T',R')|$.

Because R and R' are connected by an RNNI move on $[(R)_i, (R)_{i+1}]$, all clusters induced by nodes with rank greater than $i+1$ coincide in these two trees. This implies that the cluster considered in iteration $j > i+1$ is identical on $\mathrm{FP}(T,R)$ and $\mathrm{FP}(T',R')$. There are furthermore no moves on these paths that change any of the clusters induced by nodes with rank less than or equal to $i+1$, as these clusters are the same in T and R, and the same is true for T' and R'.

Let C be the cluster induced by the node of lowest rank greater than $i+1$ in R that is not induced by the node of same rank in T. Such a cluster C exists as otherwise $T = R$, which contradicts our assumption that these trees are not identical. The first move on $\mathrm{FP}(T,R)$ and $\mathrm{FP}(T',R')$ hence decreases the rank of $mrca(C)$.

Let $[(T)_k, (T)_{k+1}]$ be the interval on which this move is performed. Note that $k \geq \mathrm{rank}(mrca(C)_R) > i+1$. Because all clusters induced by $(T)_j$ and $(T')_j$ coincide for $j > i+1$, the first move on $\mathrm{FP}(T',R')$ is the same as the one on T, if this is a rank move.

Now consider the case that the first move on $\mathrm{FP}(T,R)$ is an NNI move that decreases the rank of $mrca(C)_T$. Let A and B be the clusters induced by the subtrees that are swapped by the NNI move on T. These clusters are induced by nodes with

rank less than k in T. By showing that A and B are clusters in the T' as well, it follows that the first move on $\mathrm{FP}(T', R')$ also swaps the subtrees induced by these moves, as this is the unique move decreasing the rank of $mrca(C)$.

If T and T' are connected by a rank move, the set of clusters of the two trees is identical, hence A and B are clusters in T'. Now consider the case that T and T' are connected by an NNI move. This NNI move changes only the cluster induced by the node with rank i (Observation 3.1), which has parent with rank $i + 1 < k$. Because the parents of the nodes inducing A and B have rank k and $k + 1$, neither A nor B can be the cluster that changes between T and T', i.e. A and B are clusters in T'. We can conclude that if the first move on $\mathrm{FP}(T, R)$ is an NNI move swapping the subtrees induced by A and B, then the first move on $\mathrm{FP}(T', R')$ swaps the subtrees induced by A and B, too.

All moves on $\mathrm{FP}(T, R)$ and $\mathrm{FP}(T', R')$ hence apply the same changes to clusters, from which we conclude $|\mathrm{FP}(T, R)| = |\mathrm{FP}(T', R')|$. □

We are now ready to prove the cluster property for RNNI.

Theorem 5.2. *The* RNNI *graph has the cluster property.*

In other words, for every pair of trees sharing a cluster, this cluster is present in every tree on every shortest path between these trees.

Proof. To prove the theorem we assume to the contrary that there are trees that share a cluster and are connected by a shortest path that does not preserve that cluster. Let $\mathcal{S}$ be the set of such pairs of trees:

$$\mathcal{S} = \{(T, R) |\ T \text{ and } R \text{ share a cluster } C \text{ and there is a shortest path between } T \text{ and } R \text{ that does not preserve } C\}$$

Let $(T, R) \in \mathcal{S}$ be a pair of trees such that there is no other pair $(\hat{T}, \hat{R}) \in \mathcal{S}$ with distance $d(\hat{T}, \hat{R}) < d(T, R)$. We say that T and R give a minimal counterexample. Let p be a path between T and R on which the cluster C that is shared by T and R is not present in every tree.

Let T' be the tree following T on p. Then T' does not contain C, as otherwise (T', R) would be in $\mathcal{S}$ and $d(T', R) < d(T, R)$, which contradicts our minimality assumption on $d(T, R)$. Since NNI moves change exactly one cluster (Observation 3.1), C is the only cluster in T that is not a cluster in T' (see Figure 5.1 for an illustration of this). Let A and B be the clusters induced by the children of $mrca(C)_T$ and D the clusters

induced by the second child of the parent of $mrca(C)_T$ (see T in Figure 5.1). We assume without loss of generality that T' results from swapping the subtrees induced by B and D in T (Figure 5.1).

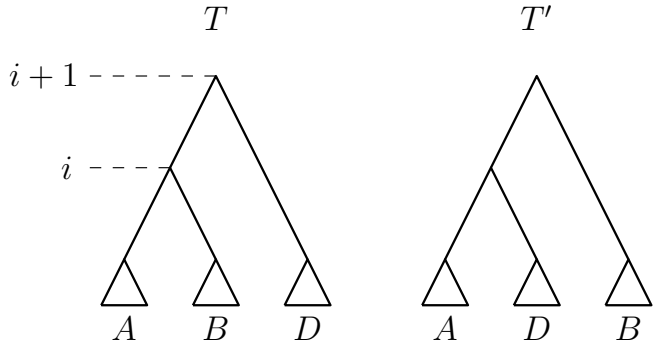


Figure 5.1: Tree T and NNI neighbour T', such that the cluster $C = A \cup B$ is present in T, but not in T'. $mrca(C)$ has rank i in T and rank $i+1$ in T'.

To compare $d(T, R)$ and $d(T', R)$ we consider the shortest paths $\mathrm{FP}(R, T)$ and $\mathrm{FP}(R, T')$ starting in R and ending in T and T', respectively. Let i be the rank of the node inducing C in T.

We first show that all clusters induced by nodes with rank less than i in T and T' coincide with those in R. Let us assume to the contrary that this is not true. Since all clusters induced by nodes with rank less than i coincide in T and T', the clusters considered in iteration $j < i$ are the same on the paths $\mathrm{FP}(R, T)$ and $\mathrm{FP}(R, T')$. Because these paths furthermore start in the same tree R, they coincide up to iteration i. The tree before iteration i on the paths $\mathrm{FP}(R, T)$ and $\mathrm{FP}(R, T')$ is hence identical, and we denote it by R'. By Lemma 4.2, all clusters induced by nodes with rank less than i in R' are the same as in T, which implies $R' \neq R$. R' furthermore contains C as cluster, because R' is on $\mathrm{FP}(T, R)$ for trees T and R that share the cluster C and FindPath preserves clusters (Lemma 4.4). From the fact that T' and R' are on a shortest path from T to R, and R' is also on a shortest path from T' to R, we can infer:

$$\begin{aligned} d(T, R') &= d(T, R) - d(R, R') \\ &= d(T, T') + d(T', R) - d(R, R') \\ &= d(T, T') + d(T', R') + d(R', R) - d(R, R') \\ &= 1 + d(T', R'). \end{aligned}$$

Therefore, T' is on a shortest path between T and R'. Since T and R' induce the cluster C, but T' does not, this results in $(T, R') \in \mathcal{S}$, which is a contradiction to the

minimality assumption of (T, R). We can hence assume that all clusters induced by nodes with rank less than i coincide in T, T', and R.

Let k be the rank of the node in R that induces C. Then $k \geq i$, as all clusters with rank less than i coincide in R and T', but C is not a cluster in T'. We now show that the cluster $C \cup D$, which is induced by the node of rank $i+1$ in T, is induced by $(R)_{k+1}$. Therefore, we assume to the contrary that $\text{rank}(mrca(C \cup D)_R) > k+1$. Because the cluster induced by the $(T')_i$ is $A \cup D$, the first move on $\text{FP}(R, T')$ decreases the rank of $mrca(A \cup D)_R$. Let $\hat{R}$ be the tree following R on $\text{FP}(R, T')$. Because $C = A \cup B$ is a cluster in R, it is $\text{rank}(mrca(A \cup D)_R) = \text{rank}(mrca(A \cup B \cup D)_R) > k+1$. With $\text{rank}(mrca(C)_R) = k$, the move decreasing the rank of $mrca(A \cup D)$ in R cannot happen on the interval incident to $mrca(C)$, hence C is a cluster in the $\hat{R}$ as well and hence $\text{rank}(mrca(A \cup D)_{\hat{R}}) = \text{rank}(mrca(A \cup B \cup D)_{\hat{R}})$. Furthermore, $d(\hat{R}, T') = d(R, T') - 1$. Because T' is on a shortest path from R to T we can infer that T' is on a shortest path from $\hat{R}$ to T and $d(\hat{R}, T) < d(T, R)$. Since $\hat{R}$ and T contain the cluster C, but T' does not, this implies $(T, \hat{R}) \in \mathcal{S}$, which is a contradiction to our minimality assumption on (T, R). Thus, the cluster induced by node with rank k in R is C and the cluster induced by the node with rank $k+1$ in R is $C \cup D$.

Because all clusters induced by nodes with rank less than i coincide in T, T', and R, the first moves on $\text{FP}(R, T)$ and $\text{FP}(R, T')$ happen in iterations i. We now compare iterations i and $i+1$ of $\text{FP}(R, T)$ and $\text{FP}(R, T')$.

On $\text{FP}(R, T)$ (depicted at the top of Figure 5.2) the cluster considered in iteration i is C, as C is induced by the node with rank i in T. Because C is induced by the node with rank k in R, we can infer with Lemma 4.5 that $k - i$ rank moves in iteration i of $\text{FP}(R, T)$ decrease the rank of $mrca(C)$. In iteration $i+1$ on $\text{FP}(R, T)$, the cluster $C \cup D$ is considered, as this is the cluster induced by the node with rank $i+1$ in T. $C \cup D$ is induced by the node with rank $k+1$ in R as well as in the tree after iteration i of $\text{FP}(R, T)$, because no move in iteration i changes the node with rank $k+1$ in R. With Lemma 4.5 it follows that $k+1-(i+1)$ rank moves in iteration $i+1$ of $\text{FP}(R, T)$ decrease the rank of $mrca(C \cup D)$. So there are in total $2k - 2i$ RNNI moves required in iterations i and $i+1$ of $\text{FP}(R, T)$.

We now consider iterations i and $i+1$ of $\text{FP}(R, T')$ (depicted at the bottom of Figure 5.2). In iteration i the cluster $A \cup D$ is considered, as it is induced by the node with rank i in T'. Remember that $(R)_k$ induces the cluster $C = A \cup B$ and $(R)_{k+1}$ induces $A \cup B \cup D$. The first move on $\text{FP}(R, T')$ hence decreases the rank of $mrca(A \cup D)$ in R by an NNI move swapping the subtrees induced by B and D. This

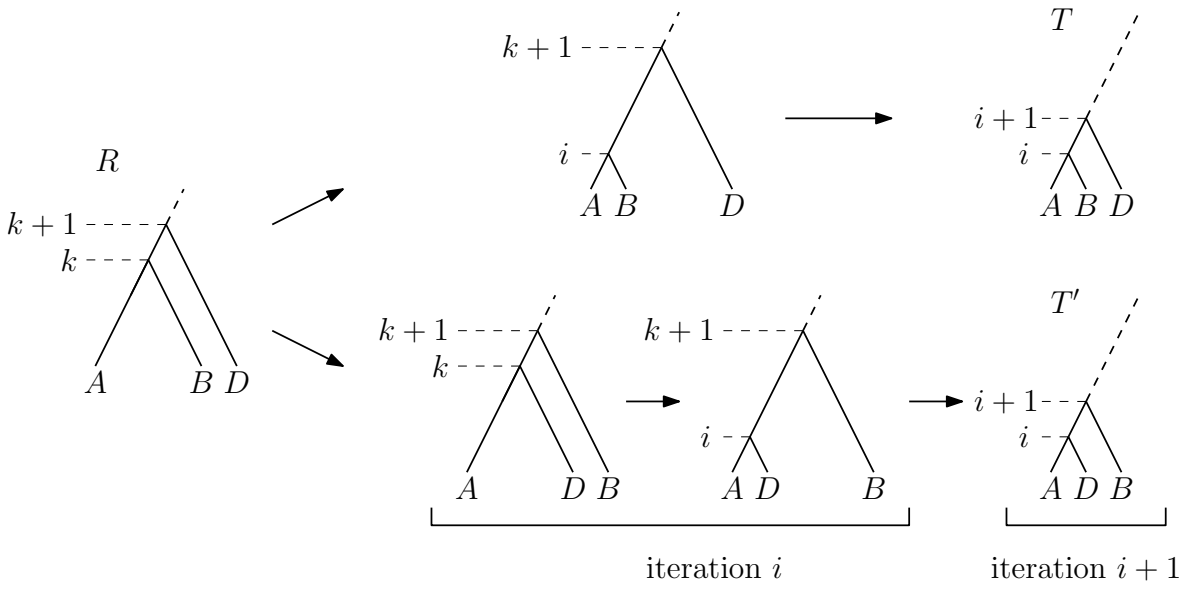


Figure 5.2: Iterations i and $i+1$ of FindPath applied to (R, T) in at the top and (R, T') at the bottom. The dashed parts of the tree might contain further nodes, which are not depicted here.

results in a tree that has clusters $A \cup D$ and $A \cup B \cup D$ induced by nodes with ranks k and $k+1$, respectively (see Figure 5.2). Because $A \cup D$ is a cluster in the tree after this NNI move, the rank of $mrca(A \cup D)$ is decreased further by rank moves (Lemma 4.5) to become i. There are hence in total $k + 1 - i$ RNNI moves needed in iteration i of $\mathrm{FP}(R, T')$, one of which is an NNI move. The cluster considered in iteration $i + 1$ of $\mathrm{FP}(R, T')$ is $A \cup B \cup D$. Because $A \cup B \cup D$ is induced by the node with rank $k + 1$ in the tree after iteration i, there are $k + 1 - (i + 1)$ rank moves needed to decrease the rank of $mrca(A \cup B \cup D)$ from $k + 1$ to $i + 1$ (Lemma 4.5). So there are in total $2k - 2i + 1$ RNNI moves needed in iterations i and $i + 1$ of $\mathrm{FP}(R, T')$, and hence one more than in these iterations on $\mathrm{FP}(T, R)$.

We now show that the trees after iteration $i+1$ on $\mathrm{FP}(R, T)$ and $\mathrm{FP}(R, T')$ are NNI neighbours on the edge bounded by nodes with rank i and $i+1$. Remember that, except for the first move on $\mathrm{FP}(R, T')$, iterations i and $i + 1$ on both paths $\mathrm{FP}(R, T)$ and $\mathrm{FP}(R, T')$ contain only rank moves to decrease two $mrca$s from k and $k+1$ to i and $i+1$, respectively. Therefore, the rank of any node that is between $i + 1$ and k is increased by two between R and the trees after iteration $i + 1$ of $\mathrm{FP}(R, T)$ and $\mathrm{FP}(R, T')$. The clusters induced by these nodes do however not change. Furthermore, the clusters induced by nodes with ranks greater than $k + 1$ are not changed in iterations i and $i + 1$ of $\mathrm{FP}(R, T)$ and $\mathrm{FP}(R, T')$ and therefore remain the same after these iterations.

The only difference between the two trees on $\mathrm{FP}(R,T)$ and $\mathrm{FP}(R,T')$ after iteration $i+1$ is hence the cluster induced by the node of rank i, i.e. these two trees are connected by an NNI move.

With Lemma 5.1 we can conclude that the remainder of $\mathrm{FP}(R,T)$ and $\mathrm{FP}(R,T')$, starting from the trees after iteration $i+1$, are of equal length. Since iterations i and $i+1$ on $\mathrm{FP}(R,T')$ together require one more move than the same iterations on $\mathrm{FP}(R,T)$, it follows $d(T',R) = d(T,R)+1$. From the definition of T' as the first tree on a shortest path from T to R we can however infer $d(T',R) = d(T,R)-1$, which leads to a contradiction. There is hence no shortest path between T and R that does not preserve C, which proves the cluster property for RNNI. □

The fact that DCT_m is a generalisation of RNNI suggests that shortest paths in RNNI and DCT_m have similar properties. Indeed, the cluster property in DCT_m follows from Theorem 5.2.

Corollary 5.3. *The graph* DCT_m *has the cluster property.*

Proof. Assume to the contrary that there are trees T and R in DCT_m that share a cluster C and that there is a shortest path p between T and R that does not preserve that cluster. With Lemma 4.11 we can transform p to a path p' between the extended ranked versions T_r and R_r of T and R, such that $|p| = |p'|$. Since the distance between T and R in DCT_m is equal to the distance between T_r and R_r in RNNI (Corollary 4.13) it follows that p' is a shortest path in RNNI. Because p does not preserve the cluster C, it follows that p' does not preserve C either. This however contradicts the cluster property of RNNI (Theorem 5.2), so there cannot be a shortest path that does not preserve clusters in DCT_m. DCT_m therefore has the cluster property. □

With Corollary 5.3 we can infer that the space DCT_m^{nu} of non-ultrametric trees (see Section 4.3) has the cluster property as well:

Corollary 5.4. *The space* DCT_m^{nu} *has the cluster property.*

Proof. We assume to the contrary that there are non-ultrametric trees T and R in DCT_m^{nu} on n leaves that share a cluster C which is not present in every tree on a shortest path p between T and R.

We can transform the path p between T and R in DCT_m^{nu} to a path p' in DCT_m on $2n$ leaves by replacing every tree by its ultrametric version (Lemma 4.15). Let T' and R' be the ultrametric versions of T and R. Consider the cluster C' that results from C

by replacing every element a_i by a_{i1} and a_{i2}. From the fact that C is present in T and R but not in every tree on p it follows that C' is present in T' and R' but not in every tree on p'. Since p is a shortest path in DCT_m^{nu}, it follows $|p| = |\mathrm{FP}^+(T, R)|$. And with Lemma 4.16 we get $|p| = |\mathrm{FP}^+(T, R)| = |\mathrm{FP}^+(T', R')|$. Together with $|p'| = |p|$ this results in $|p'| = |\mathrm{FP}^+(T', R')|$, which means that p', a path that does not preserve the cluster C', is a shortest path from T' to R' in DCT_m. This is a contradiction to DCT_m having the cluster property (Corollary 5.3). There can hence not be a path p connecting T and R that does not preserve the shared cluster C, which concludes the proof that DCT_m^{nu} has the cluster property. □

5.2 Caterpillar Trees

In this section we consider the set of caterpillar trees and establish some properties of shortest paths between those trees in DCT_m, which in particular includes RNNI. The property we want to investigate here is the convexity of the set of caterpillar trees. We say that a set of trees in a tree space is *convex*, if every pair of trees in this set is connected by a shortest path that stays within this set. Note that we only require one shortest path to stay within the set to call it convex. There may be further shortest paths that do not stay within the set. The main question we ask here is hence: Are any two caterpillar trees T and R connected by a shortest path that only consists of caterpillar trees? We call such a path only consisting of caterpillar trees a *caterpillar path*.

In Section 5.2.1 we introduce algorithms to compute caterpillar paths in RNNI and DCT_m: Caterpillar Sort and Caterpillar Sort$^+$. These algorithms allow us to prove in Section 5.2.2 that the set of caterpillar trees is convex in RNNI and more generally in DCT_m. In Corollary 5.7 we show that the result for RNNI allows us to compute the distance between caterpillar trees more efficiently than by using FindPath. This suggests a lower bound on the computational complexity of computing distances between arbitrary trees in RNNI (Conjecture 1).

5.2.1 Caterpillar Sort

To prove that the set of caterpillar trees is convex in DCT_m, and hence also in RNNI, we introduce algorithms Caterpillar Sort and Caterpillar Sort$^+$ to compute caterpillar paths. We later show in Section 5.2.2 that paths computed by these algorithms are shortest paths indeed. This section is split into two parts. We first present

a new representation of caterpillar trees and the algorithm Caterpillar Sort for RNNI. Afterwards, we do the same for caterpillar trees in DCT_m, which especially includes introducing the algorithm Caterpillar Sort$^+$.

Caterpillar Sort in RNNI

For ranked trees we define the *caterpillar list representation* of a caterpillar tree T as a list of sets where the set at position $i \in \{1, \ldots, n-1\}$ of the list contains all leaves that are children of the node with rank i in T. The first set hence contains the leaves building the cherry of T and the remaining elements of the list are sets containing single leaves, ordered according to increasing rank of their parents in T. The tree T with cluster representation $[\{a_1, a_2\}, \{a_1, a_2, a_3\}, \ldots, \{a_1, \ldots, a_n\}]$ has caterpillar list representation $[\{a_1, a_2\}, \{a_3\}, \ldots, \{a_n\}]$. Note that we might refer to the caterpillar list representation of a tree T simply by T and especially refer to the ith set in this list representation by $T[i]$, i.e. the first element of the list representing T is $T[1] = \{a_1, a_2\}$ and the ith element is $T[i] = \{a_{i+1}\}$.

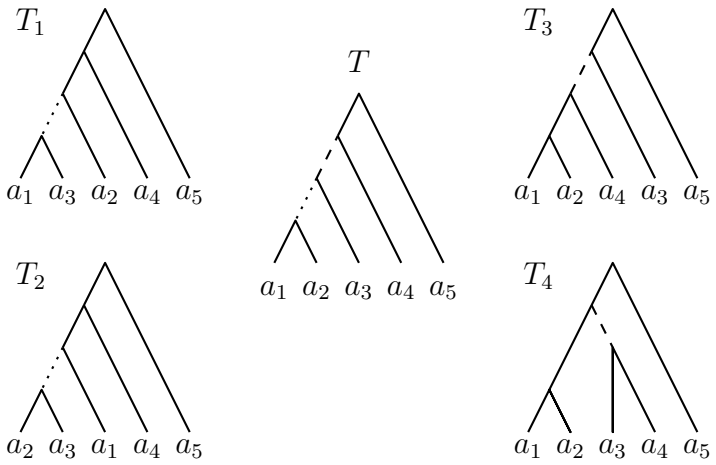


Figure 5.3: NNI moves on tree T on the dotted edge incident to the cherry lead to caterpillar trees T_1 and T_2, NNI moves on the dashed edge lead to a caterpillar tree T_3 and a non-caterpillar tree T_4.

The only move possible between two caterpillar trees is an NNI move, because caterpillar trees have no rank intervals. The edge incident to the cherry is the only one in a caterpillar tree for which two NNI moves on it result in a caterpillar tree. For all other edges, one NNI move results in a caterpillar tree, while the other one builds a new cherry. An example of this is depicted in Figure 5.3, where two NNI moves on the dotted edge incident to the cherry of T result in caterpillar trees T_1 and T_2 while one

of the two moves on the dashed edge results in a caterpillar tree T_3 and the other one in a tree T_4 with two cherries.

In the caterpillar list representation NNI moves correspond to swapping two elements of sets $T[i]$ and $T[i+1]$ that are adjacent in the list. For the pair $T[1], T[2]$ two moves are possible, because either of the two elements in $T[1]$ can swap with the element in $T[2]$. These two moves correspond to the two NNI moves on the edge incident to the cherry of T that lead to caterpillar trees. For all other pairs $T[i], T[i+1]$ with $i \in \{2, \ldots, n-2\}$ only one move is possible, as both $T[i]$ and $T[i+1]$ contain only one element. This move corresponds to the one possible NNI move on the edge $[(T)_i, (T)_{i+1}]$ that results in a caterpillar tree.

In Figure 5.3 a tree T and its caterpillar neighbours T_1, T_2, and T_3 are depicted. The caterpillar list representations of these trees are the following, where the leaves that are swapped between T and T_1, T_2, T_3, respectively, are highlighted in bold:

$$\begin{aligned} T &= [\{a_1, a_2\}, \{a_3\}, \{a_4\}, \{a_5\}] \\ T_1 &= [\{a_1, \boldsymbol{a_3}\}, \{\boldsymbol{a_2}\}, \{a_4\}, \{a_5\}] \\ T_2 &= [\{a_2, \boldsymbol{a_3}\}, \{\boldsymbol{a_1}\}, \{a_4\}, \{a_5\}] \\ T_3 &= [\{a_1, a_2\}, \{\boldsymbol{a_4}\}, \{\boldsymbol{a_3}\}, \{a_5\}] \end{aligned}$$

We are now ready to introduce the algorithm Caterpillar Sort (Algorithm 5) for computing caterpillar paths. This algorithm is a modification of the classical Bubble Sort algorithm (Knuth 1997), which sorts a given list of integers such that the integers are ordered increasingly in the end. Bubble Sort iterates through the list from first to last element and swaps two consecutive elements i and j if $i > j$. This is repeated until the integers are ordered increasingly. We now describe how Caterpillar Sort (Algorithm 5) computes paths between caterpillar trees. For input trees T and R we denote this path by CSort(T, R).

Let T and R be the input trees for which we want to compute a caterpillar path. We assume without loss of generality $R = [\{a_1, a_2\}, \{a_3\}, \ldots, \{a_n\}]$. Caterpillar Sort computes a path p from T to R iteratively so that after k steps the last k leaves of T and R coincide, i.e. $T[i] = R[i]$ for all $i = n-k, n-k+1, \ldots, n$. In iteration $k \in \{1, \ldots, n-2\}$ (line 2 in Algorithm 5) the leaf a_{n-k+1} is considered. a_{n-k+1} swaps position with its right neighbours in the caterpillar list representation of the currently last tree on p until it is at the correct position (with index $n-k$) in the list. Remember that such swaps of two elements correspond to NNI moves, i.e. a_{n-k+1} is moved by NNI

moves in iteration k until it reaches its final position (while loop, line 3 in Algorithm 5).

Algorithm 5 CATERPILLAR SORT(T, R)

1: $[\{a_1, a_2\}, \{a_3\} \ldots, \{a_n\}] := R$, $T' := T$, $p = [T']$
2: **for** $k = 1, \ldots, n-2$ **do**
3: **while** a_{n-k+1} is at position $i < n-k$ **do**
4: T'' is T' with $a_{n-k+1} \in T'[i]$ and the element in $T'[i+1]$ swapped (NNI)
5: $T' = T''$
6: $p = p + T'$
7: **return** p

An example of a path computed by CATERPILLAR SORT is given in Figure 5.4. The path depicted there consists of the following ranked trees in caterpillar list representation (leaf labels on the arrows indicate the leaf a_{n-k+1} that is moved by the corresponding move on CATERPILLAR SORT):

$$T = [\{a_1, a_4\}, \{a_2\}, \{a_5\}, \{a_3\}] \xrightarrow{a_5} [\{a_1, a_4\}, \{a_2\}, \{a_3\}, \{a_5\}]$$
$$\downarrow a_4$$
$$R = [\{a_1, a_2\}, \{a_3\}, \{a_4\}, \{a_5\}] \xleftarrow{a_4} [\{a_1, a_2\}, \{a_4\}, \{a_3\}, \{a_5\}]$$

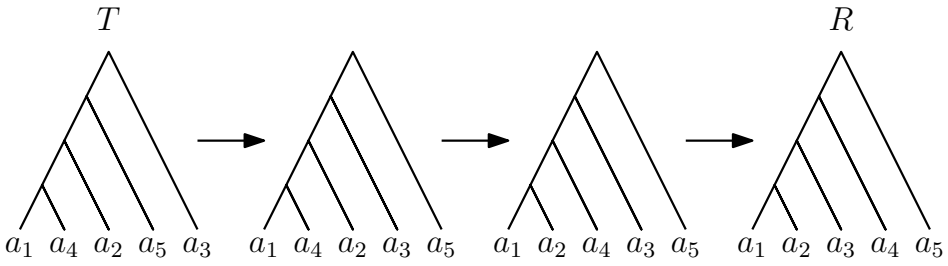


Figure 5.4: The path between caterpillar trees T (on the left) and R (on the right) computed by CATERPILLAR SORT.

CATERPILLAR SORT$^+$ in DCT$_m$

The caterpillar list representation of ranked trees can be extended to discrete coalescent caterpillar trees in DCT$_m$. Instead of a list of length $n-1$ for caterpillar trees on n leaves, a list of length m is needed, where m is the maximum root time in the tree space DCT$_m$. As for caterpillar trees in RNNI, the set with index i in the caterpillar list representation contains the leaves that are children of the node of time i. The positions

of the list representing times that do not correspond to an internal node hence contain the empty set $\{\}$. The caterpillar list representations for the trees T, T_1, and T_2 of Figure 5.5 are:

$$\begin{aligned} T &= [\{a_1, a_2\}, \{\}, \{a_3\}, \{\}, \{a_4\}, \{a_5\}] \\ T_1 &= [\{a_1, a_2\}, \{\}, \{a_3\}, \{\}, \boldsymbol{\{a_5\}}, \boldsymbol{\{a_4\}}] \\ T_2 &= [\boldsymbol{\{\}}, \boldsymbol{\{a_1, a_2\}}, \{a_3\}, \{\}, \{a_4\}, \{a_5\}] \end{aligned}$$

The sets highlighted in bold are the ones that differ in T_1 and T_2 compared to T.

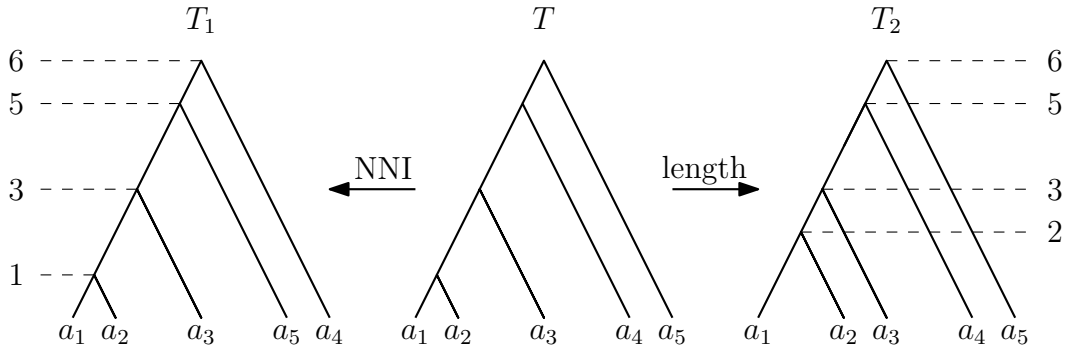


Figure 5.5: Discrete coalescent caterpillar tree T, an NNI move on T resulting in T_1, and a length move on T resulting in T_2

NNI moves between caterpillar trees in their list representation in DCT_m are the same as in the case of ranked trees. They correspond to swapping two elements of sets $T[i]$ and $T[i+1]$ for $i \in \{1, \ldots, m-1\}$. Since NNI moves are only allowed on edges of length one, neither $T[i]$ nor $T[i+1]$ can be the empty set set $\{\}$. Besides NNI moves, also length moves are possible between two caterpillar trees in DCT_m, as for example on the right of Figure 5.5. These moves correspond to swapping two sets $T[i], T[i+1]$, where exactly one of them is $\{\}$. The length move from T to T_2 in Figure 5.5 for example swaps the sets $T[1] = \{a_1, a_2\}$ and $T[2] = \{\}$ in the list representation of T and increases the time of the internal node of the cherry $\{a_1, a_2\}$ from one to two. Length moves hence swap entire sets $T[i]$ and $T[i+1]$, while NNI moves swap elements from within two sets $T[i]$ and $T[i+1]$.

With these two possible moves on caterpillar trees in DCT_m we can generalise the Caterpillar Sort algorithm from RNNI to DCT_m and call the resulting algorithm Caterpillar Sort$^+$ (Algorithm 6). Note that the difference to the algorithm Caterpillar Sort for ranked trees is that length moves need to be introduced.

The path p computed by Caterpillar Sort$^+$ initially only consists of T, and we refer to the last tree of p as T'. In each iteration $k = m, \ldots, 2$ of the algorithm

either a leaf with parent of time k in R, or, if there is no such leaf, the empty set $\{\}$ is considered. If there is a leaf with parent of rank k in R, this leaf gets moved up by NNI moves and length moves on T' until it is at position k in the list, meaning its parent has rank k. If otherwise $R[k]$ is the empty set $\{\}$, then the empty set $\{\}$ with highest index smaller than k in the caterpillar list representation of T' moves up until it is at position k. The moves needed for this are swaps of this set $\{\}$ with its right neighbours, which are length moves.

We denote the path resulting from Caterpillar Sort$^+$ applied to discrete coalescent trees T and R by CSort$^+(T, R)$.

Algorithm 6 Caterpillar Sort$^+(T, R)$

1: $T' := T$, $p = [T']$
2: **for** $k = m, \ldots, 2$ **do**
3: **if** $S := R[k] \neq T'[k]$ **then**
4: **if** $S = \{\}$ **then**
5: Let i be the maximum index of a set $\{\}$ in T' such that $i < k$
6: **while** $i < k$ **do**
7: T'' is T' with $T'[i]$ and $T'[i+1]$ swapped (length move)
8: $T' = T''$
9: $p = p + T'$
10: $i = i + 1$
11: **else if** $S = \{a_j\}$ **then**
12: Let i be the index of $\{a_j\}$ in T'
13: **while** $i < k$ **do**
14: T'' is T' with $a_j \in T'[i]$ and the element in $T'[i+1]$ swapped (NNI or length move)
15: $T' = T''$
16: $p = p + T'$
17: **return** p

Let T and R be ranked caterpillar trees in RNNI = DCT$_{n-1}$. We can apply Caterpillar Sort (Algorithm 5) or Caterpillar Sort$^+$ (Algorithm 6) to these trees and get the same caterpillar path: The case $S = \{\}$ in line 4 of Algorithm 6 is not possible for ranked trees, and neither is a length move in line 14. Therefore, the only moves performed by Caterpillar Sort$^+$ are NNI moves in line 14, which are exactly the NNI moves performed by Caterpillar Sort.

5.2.2 Convexity

In this section we show that the set of caterpillar trees is convex in DCT_m, and therefore also in $\mathrm{RNNI} = \mathrm{DCT}_{n-1}$, i.e. every pair of caterpillar trees is connected by a shortest path that only consists of caterpillar trees. To prove this we use the extended ranked versions of caterpillar trees in DCT_m. Remember that the extended ranked version of a discrete coalescent tree results from adding a subtree, which is caterpillar tree (see Algorithm 3). The caterpillar tree T_r^c that is added to a caterpillar tree T to create its extended ranked version T_r has internal nodes with rank i if the caterpillar list representation of T has an empty set at position i: $T[i] = \{\}$. For example the tree T in Figure 5.6 in its caterpillar list representation is

$$[\{a_1, a_2\}, \{\}, \{a_3\}, \{\}, \{a_4\}, \{a_5\}],$$

and the added caterpillar subtree T_r^c has internal nodes with rank two and four (see Figure 5.6).

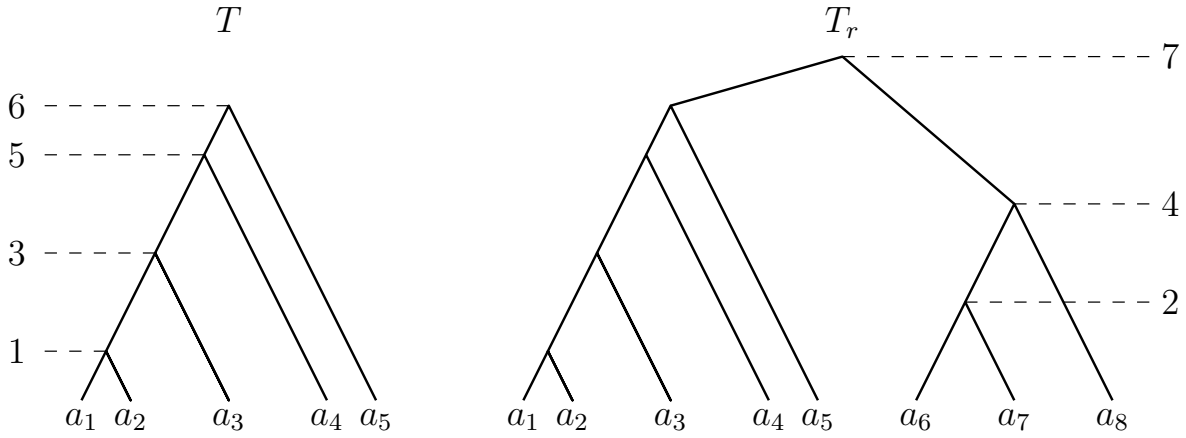


Figure 5.6: Caterpillar tree T in DCT_6 and its extended ranked version T_r

To prove that the set of caterpillar trees is convex in DCT_m, we need further notation. We say that a leaf a_i is *below* a leaf a_j in a tree T, and write $a_i \prec_T a_j$, if $\mathrm{rank}(p(a_i))_T < \mathrm{rank}(p(a_j))_T$. In this case we also say that a_j is *above* a_i. If $\mathrm{rank}(p(a_i))_T \leq \mathrm{rank}(p(a_j))_T$, we write $a_i \preceq_T a_j$. We furthermore define $r(C)_T$ as $\mathrm{rank}(mrca(C)_T)$ for a set $C \subseteq \{a_1, \ldots, a_n\}$ and a tree T.

We prove that the set of caterpillar trees is convex in DCT_m (Theorem 5.5) by showing that the algorithm Caterpillar Sort$^+$ computes shortest paths in DCT_m. Since the paths computed by Caterpillar Sort between caterpillar trees T and R are the same as the ones computed by Caterpillar Sort$^+$ between such trees, it

follows that Caterpillar Sort computes shortest paths between caterpillar trees in RNNI. This in particular also implies that the set of caterpillar trees is convex in RNNI.

Theorem 5.5. *The set of caterpillar trees is convex in* DCT_m.

Proof. We prove this theorem by showing that paths between caterpillar trees computed by Caterpillar Sort$^+$ are shortest paths. We do this by induction on the length of $\mathrm{CSort}^+(T, R)$, the caterpillar path between two caterpillar trees T and R computed by Caterpillar Sort$^+$.

We assume for the induction basis $|\mathrm{CSort}^+(T, R)| = 0$ for caterpillar trees T and R. For the caterpillar list representation this implies that there is no $k \in \{2, \ldots, m\}$ with $R[k] \neq T[k]$ (see line 3 of Algorithm 6). Therefore, T and R are identical, i.e. $d_{\mathrm{DCT}_m}(T, R) = 0$ and hence $|\mathrm{CSort}^+(T, R)| = d_{\mathrm{DCT}_m}(T, R)$.

For the induction step we assume that T and R are caterpillar trees and that $|\mathrm{CSort}^+(\hat{T}, \hat{R})| = d_{\mathrm{DCT}_m}(\hat{T}, \hat{R})$ for all caterpillar trees $\hat{T}$ and $\hat{R}$ for which $|\mathrm{CSort}^+(\hat{T}, \hat{R})| < |\mathrm{CSort}^+(T, R)|$. Let T' be the caterpillar tree after the first move on $\mathrm{CSort}^+(T, R)$ (Algorithm 6). To prove that $\mathrm{CSort}^+(T, R)$ is a shortest path between T and R, we show $d_{\mathrm{DCT}_m}(T, R) = d_{\mathrm{DCT}_m}(T', R)+1$, as we can then use the induction hypothesis to infer $|\mathrm{CSort}^+(T, R)| = 1 + |\mathrm{CSort}^+(T', R)| = 1 + d_{\mathrm{DCT}_m}(T', R) = d_{\mathrm{DCT}_m}(T, R)$. Throughout this proof we assume that T_r and R_r are the extended ranked versions of T and R in DCT_m.

Let us consider the first move of $\mathrm{CSort}^+(T, R)$ on T that results in T'. Three different types of moves are possible between T and T':

(i) swapping sets $T[i] = \{\}$ and $T[i+1] \neq \{\}$, i.e. a length move decreasing the time of the internal node with rank $i + 1$ by one (line 4 of Algorithm 6),

(ii) swapping sets $T[i] \neq \{\}$ and $T[i+1] = \{\}$, i.e. a length move increasing the time of the internal node with rank i by one (line 14 of Algorithm 6), or

(iii) swapping one element of $T[i] \neq \{\}$ with one element of $T[i+1] \neq \{\}$, i.e. an NNI move (line 14 of Algorithm 6).

Because T and T' are caterpillar trees, we can assume that the internal node with rank i, if there is one (case (ii) and (iii)), is parent of a leaf a_j and the node with rank $i + 1$, if it exists (case (i) and (iii)), has a leaf a_k as child.

We now analyse how these moves change the extended ranked versions T_r and T'_r of T and T'. Because in T_r and T'_r every integer in $\{1, \ldots, m+1\}$ is assigned as time to an

internal node, these trees have internal nodes with ranks i and $i+1$. Since furthermore the tree added to T to get its extended ranked version T_r is a caterpillar tree T_r^c, all nodes except for the root of T_r have at least one leaf as child. In each of the three cases (i), (ii), and (iii), there are hence two leaves a_j and a_k with $\text{rank}(p(a_j)_{T_r}) = i$ and $\text{rank}(p(a_k)_{T_r}) = i+1$ whose parents swap ranks by the move between T_r and T'_r: If T and T' are connected by (i) a length move decreasing the time of an internal node, then a_k is in T_r^c and a_j in T_r^d, i.e. $a_k \in \{a_{n+1}, \ldots, a_{m+2}\}$ and $a_j \in \{a_1, \ldots, a_n\}$. If T and T' are connected by (ii) an NNI move, a_k and a_j are in the subtree T_r^d, i.e. $a_k, a_j \in \{a_1, \ldots, a_n\}$. And if T' results from (iii) a length move moving the leaf a_k up then a_k is in T_r^d and a_j in T_r^c, i.e. $a_k \in \{a_1, \ldots, a_n\}$ and $a_j \in \{a_{n+1}, \ldots, a_{m+2}\}$. All three cases are depicted in Figure 5.7.

We now use FindPath to show $d_{\text{DCT}_m}(T, R) = d_{\text{DCT}_m}(T', R)+1$. By Corollary 4.13, $d_{\text{DCT}_m}(T, R) = d_{\text{RNNI}}(T_r, R_r)$, and the same is true for T' and R. We hence consider the length of the paths $\text{FP}(R_r, T_r)$ and $\text{FP}(R_r, T'_r)$ that FindPath computes from R_r to T_r and T'_r, respectively, to compare $d_{\text{DCT}_m}(T, R)$ and $d_{\text{DCT}_m}(T', R)$.

Because the caterpillar list representation of T and T' only differs in the sets at position i and $i+1$, all clusters induced by nodes of time less than i coincide in these trees. Therefore, all clusters induced by nodes with rank less than i coincide in T_r and T'_r, too. Because FindPath considers clusters in a bottom-up approach, the trees on $\text{FP}(R_r, T_r)$ and $\text{FP}(R_r, T'_r)$ coincide until the clusters of rank i in T_r and T'_r are considered. In other words, $\text{FP}(R_r, T_r)$ and $\text{FP}(R_r, T'_r)$ coincide up to the tree at the end of iteration $i-1$, which we denote by R'_r. All nodes with rank less than i hence induce the same clusters in T_r, T'_r, and R'_r. To compare $d_{\text{DCT}_m}(T, R)$ and $d_{\text{DCT}_m}(T', R)$ it is sufficient to compare $\text{FP}(R'_r, T_r)$ and $\text{FP}(R'_r, T'_r)$, because

$$d_{\text{DCT}_m}(T, R) = |\text{FP}(R_r, T_r)| = |\text{FP}(R_r, R'_r)| + |\text{FP}(R'_r, T_r)| \text{ and}$$
$$d_{\text{DCT}_m}(T', R) = |\text{FP}(R'_r, T_r)| = |\text{FP}(R_r, R'_r)| + |\text{FP}(R'_r, T'_r)|.$$

We now consider the order in which a_k and a_j appear in R'_r. Remember that the rank of $p(a_k)$ increases by the first move on on $\text{CSort}^+(T, R)$, which translates to a move exchanging the ranks of $p(a_k)$ and $p(a_j)$ between T_r and T'_r, so that $a_k \prec_{T_r} a_j$ and $a_j \prec_{T'_r} a_k$ (see Figure 5.7). This implies that the current iteration of $\text{CSort}^+(T, R)$ is moving a_k to its final position in R and we can infer $a_j \prec_{R_r} a_k$. Since FindPath considers clusters in a bottom-up approach, $a_j \prec_{T'_r} a_k$ and $a_j \prec_{R_r} a_k$ imply that $a_j \prec a_k$ is true in every tree on $\text{FP}(R_r, T'_r)$. This in particular results in $a_j \prec_{R'_r} a_k$.

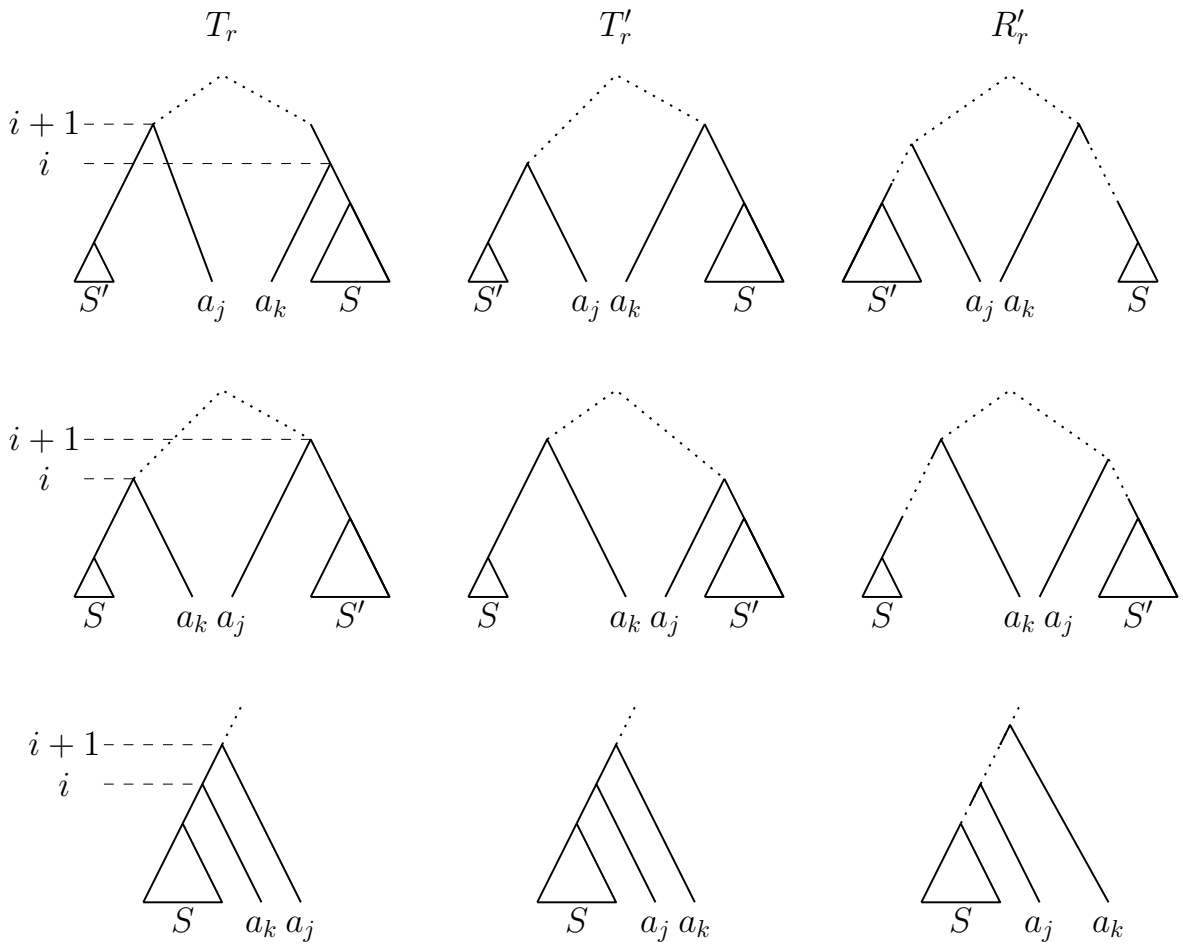


Figure 5.7: The three possible versions of trees T_r (left), T'_r (middle), and R'_r as described in the proof of Theorem 5.5. In the top row T_r and T'_r are connected by a rank move with a_k in T^c_r (case (i)), in the middle row T_r and T'_r are connected by a rank move with a_k in T^d_r (case (ii)), and in the bottom row T_r and T'_r are connected by an NNI move (case (iii)). Dotted lines indicate unknown parts of the tree.

For proving $|\mathrm{FP}(R'_r, T_r)| = |\mathrm{FP}(R'_r, T'_r)| + 1$, we now distinguish case (iii) that T and T' are connected by an NNI move from cases (i) and (ii) that T and T' are connected by a length move.

Case (i) and (ii)

With $i = \mathrm{rank}(p(a_k)_{T_r})$ we can assume that the node of rank i in T_r induces a cluster $S \cup \{a_k\}$, where S is induced by a node with rank less than i in T_r. Furthermore, $i+1 = \mathrm{rank}(p(a_j)_{T_r})$, so we can assume that the node with rank $i+1$ in T_r induces a cluster $S' \cup \{a_j\}$, where S' is induced by a node with rank less than i im T_r. In case (i) $a_j \in \{a_1, \ldots, a_n\}$ and $a_k \in \{a_{n+1}, \ldots, a_{m+2}\}$, and hence $S \subseteq \{a_{n+1}, \ldots, a_{m+2}\}$ and $S' \subseteq \{a_1, \ldots, a_n\}$. In case (ii) $a_j \in \{a_{n+1}, \ldots, a_{m+2}\}$ and $a_k \in \{a_1, \ldots, a_n\}$, and hence $S \subseteq \{a_1, \ldots, a_n\}$ and $S' \subseteq \{a_{n+1}, \ldots, a_{m+2}\}$ (see Figure 5.7). Because S and S' are induced by nodes with rank less than i in T_r, and all nodes with rank less than i induce the same clusters in T_r, T'_r, and R_r, S and S' are induced by the node of same rank in T'_r and R'_r as in T_r.

We now compare iterations i and $i+1$ of $\mathrm{FP}(R'_r, T_r)$ and $\mathrm{FP}(R'_r, T'_r)$, i.e. the first two iterations that apply changes to R'_r, which consider the clusters of T_r and T'_r that are induced by the nodes with rank i and $i+1$.

On $\mathrm{FP}(R'_r, T_r)$, $r(S \cup \{a_k\})_{R'_r} - i$ RNNI moves are required to decrease the rank of $mrca(S \cup \{a_k\})_{R'_r}$ in iteration i. Because the rank of $mrca(S' \cup \{a_j\})_{R'_r}$ increases by one when the parents of a_k and a_j swap ranks in this iteration, the following iteration for $S' \cup \{a_j\}$ needs $r(S' \cup \{a_j\})_{R'_r} + 1 - (i+1)$ RNNI moves. We can summarise that the total number of RNNI moves in iterations i and $i+1$ of $\mathrm{FP}(R'_r, T_r)$ is $r(S \cup \{a_k\})_{R'_r} + r(S' \cup \{a_j\})_{R'_r} - 2i$.

In iteration i of $\mathrm{FP}(R'_r, T'_r)$, $r(S' \cup \{a_j\})_{R'_r} - i$ RNNI moves decrease the rank of $mrca(S' \cup \{a_j\})_{R'_r}$ in R'_r. In the following iteration $i+1$, $r(S \cup \{a_k\})_{R'_r} - (i+1)$ are needed for $S \cup \{a_k\}$. We can summarise that iterations i and $i+1$ together require $r(S' \cup \{a_j\})_{R'_r} + r(S \cup \{a_k\})_{R'_r} - 2i - 1$ RNNI moves, and hence one move less than these iterations on $\mathrm{FP}(R'_r, T_r)$.

Case (iii)

Since T_r and T'_r are connected by an NNI move, we can assume that the node of rank i in T'_r induces $S \cup \{a_j\}$ and the node of rank $i+1$ induces $S \cup \{a_j\} \cup \{a_k\}$ with $S \subseteq \{a_1, \ldots, a_n\}$ (see Figure 5.7). Because S is induced by a node with rank less than i in T_r, and all nodes with rank less than i induce the same clusters in T_r, T'_r, and R_r, S is induced by the node of same rank in T'_r and R'_r as in T_r.

We now compare iterations i and $i+1$ of $\mathrm{FP}(R'_r, T_r)$ and $\mathrm{FP}(R'_r, T'_r)$, i.e. the first two iterations that apply changes to R'_r, which consider the clusters of T_r and T'_r that are induced by the nodes with rank i and $i+1$.

On $\mathrm{FP}(R'_r, T_r)$, $r(S \cup \{a_k\})_{R'_r} - i$ RNNI moves are required to decrease the rank of $mrca(S \cup \{a_k\})_{R'_r}$ in iteration i. Because the parents of a_k and a_j swap ranks in this iteration, the $mrca$ of $S \cup \{a_j\} \cup \{a_k\}$ has rank $r(S \cup \{a_j\})_{R'_r} + 1$ in the tree after iteration i. Therefore, there are $r(S \cup \{a_j\})_{R'_r} + 1 - (i+1)$ RNNI moves needed in iteration $i+1$ to decrease the rank of $mrca(S \cup \{a_j\} \cup \{a_k\})$. We can summarise that iterations i and $i+1$ together require $r(S \cup \{a_k\})_{R'_r} + r(S \cup \{a_j\})_{R'_r} - 2i$ RNNI moves.

In iteration i of $\mathrm{FP}(R'_r, T'_r)$, $r(S \cup \{a_j\})_{R'_r} - i$ RNNI moves decrease the rank of $mrca(S \cup \{a_j\})_{R'_r}$ in R'_r. In the following iteration $i+1$, $r(S \cup \{a_k\})_{R'_r} - (i+1)$ are needed for $S \cup \{a_j\} \cup \{a_k\}$, because $mrca(S \cup \{a_j\} \cup \{a_k\}) = mrca(S \cup \{a_k\})$ in R'_r as well as in the tree after iteration i (see Figure 5.7). We can summarise that iterations i and $i+1$ together require $r(S' \cup \{a_j\})_{R'_r} + r(S \cup \{a_k\})_{R'_r} - 2i - 1$ RNNI moves, and hence one move less than these iterations on $\mathrm{FP}(R'_r, T_r)$.

In all three cases (i), (ii), and (iii), the only difference between the trees on the two different paths $\mathrm{FP}(R'_r, T_r)$ and $\mathrm{FP}(R'_r, T'_r)$ after iterations i and $i+1$ as described above is the order of ranks of the parents of a_j and a_k. This is because all moves in these two iterations are the same on $\mathrm{FP}(R'_r, T_r)$ and $\mathrm{FP}(R'_r, T'_r)$, just in a different order. All clusters induced by nodes with rank less than i or greater than $i+1$ hence coincide in the trees after iteration $i+1$ on $\mathrm{FP}(R'_r, T_r)$ and $\mathrm{FP}(R'_r, T'_r)$.

We can conclude that all moves on $\mathrm{FP}(R'_r, T_r)$ and $\mathrm{FP}(R'_r, T'_r)$ after iteration $i+1$ coincide (Lemma 5.1). This implies that the difference in length of $\mathrm{FP}(R'_r, T_r)$ and $\mathrm{FP}(R'_r, T'_r)$ results from the different number of moves in iteration i and $i+1$. Since $\mathrm{FP}(R'_r, T_r)$ requires in all cases above in total one more move in iterations i and $i+1$ than $\mathrm{FP}(R'_r, T'_r)$, we conclude $d_{\mathrm{RNNI}}(R_r, T_r) = d_{\mathrm{RNNI}}(R_r, T'_r) + 1$. With the induction hypothesis applied to T' and R we get

$$|\textsc{CSort}^+(T, R)| = 1 + |\textsc{CSort}^+(T', R)| = 1 + d_{\mathrm{DCT}_m}(T', R) = d_{\mathrm{DCT}_m}(T, R),$$

which concludes this proof. □

5.2.3 More efficient distance computation

In this section we present a way to compute distances between caterpillar trees with worst-case time complexity $\mathcal{O}(n\sqrt{\log n})$ for RNNI (Corollary 5.7). This is more efficient than computing distances with FindPath, which runs in $\mathcal{O}(n^2)$. We first establish a formula to express distances between two caterpillar trees in RNNI (Theorem 5.6), and then explain how distances can be computed efficiently using this formula.

But first, we need the following definition: For two caterpillar trees T and R, we call a pair of leaves (a_i, a_j) a *transposition* in T with respect to R, if a_i is strictly below a_j in T and the opposite is true for R: $a_i \prec_T a_j$ and $a_j \prec_R a_i$.

Theorem 5.6. *Let T and R be caterpillar trees with leaf sets $\{a_1, \ldots, a_n\}$. Let $\{a_x, a_y\}$ be the cherry of R. Define*

$$P(T,R) = \{(a_i, a_j) \mid a_i \prec_T a_j \text{ and } a_j \prec_R a_i\},$$
$$M(T,R) = \{a_i \mid \text{for all } l \text{ with } a_l \preceq_T a_i \text{ it is } a_i \prec_R a_l\} \cap \{a_i \mid a_i \prec_T a_x \text{ and } a_i \prec_T a_y\}.$$

Then

$$d(T,R) = |P(T,R)| - |M(T,R)|.$$

The set $P(T,R)$ in Theorem 5.6 is the set of transpositions for the caterpillar tree T with respect to R. $M(T,R)$ contains the leaves a_i in T for which in the representation of T as a list (i) every leaf that is below a_i in T (if a_i is in the cherry, this includes the other cherry leaf) is strictly above a_i in R and (ii) no cherry leaf of R is below a_i in T.

Proof. Let T and R be caterpillar trees in RNNI as described in the theorem and let $\widehat{d}(T,R) := |P(T,R)| - |M(T,R)|$. We prove this theorem by induction.

To prove for the base case that T is equal to R if $\hat{d}(T,R) = 0$, we assume to the contrary that $\hat{d}(T,R) = 0$ and T and R are not identical. Then there is at least one element in $P(T,R)$, as otherwise T and R are identical. Since we assume $\hat{d}(T,R) = 0$, it must be $|M(T,R)| = |P(T,R)| > 0$, i.e. $M(T,R)$ contains at least one element. Let $a_j \in M(T,R)$. By the definition of $M(T,R)$, $a_j \prec_T a_x$ and $a_j \prec_T a_y$. But with a_x and a_y building the cherry of R, this implies that (a_j, a_x) and (a_j, a_y) are transpositions in T with respect to R, and hence members of the set $P(T,R)$. This means that for every element in $M(T,R)$ there are at least two elements in $P(T,R)$ and hence $|M(T,R)| < 2|P(T,R)|$, i.e. $|M(T,R)| \neq |P(T,R)|$. It follows $\hat{d}(T,R) \neq 0$, which contradicts our assumption $\hat{d}(T,R) = 0$. We can conclude that if $\hat{d}(T,R) = 0$, then T and R are identical.

For the induction step we assume that $\hat{d}(T', R') = d(T', R')$ for all caterpillar trees T', R' with $\hat{d}(T', R') \leq d$ and show that if T and R are caterpillar trees in RNNI with $\hat{d}(T, R) = d + 1$, then $\hat{d}(T, R) = d(T, R)$. We therefore consider the tree T' of T that is the first tree on CSort(T, R). By proving $\widehat{d}(T', R) = \widehat{d}(T, R) - 1$, we can infer with the induction hypothesis that $\hat{d}(T', R) = d(T', R)$. And since Caterpillar Sort(T, R) is a shortest path between T ans R in RNNI (Theorem 5.5), it follows $d(T, R) = d(T', R) + 1 = \hat{d}(T', R) + 1 = \hat{d}(T, R)$, which proves the theorem. We hence only need to prove $\widehat{d}(T', R) = \widehat{d}(T, R) - 1$.

Let a_k be the leaf with parent of highest rank in R that is not at the same position in T:

$$a_k := \underset{a_1,\ldots,a_n}{\operatorname{argmax}}\{\operatorname{rank}(p(a_j)_R) \mid \operatorname{rank}(p(a_j)_R) \neq \operatorname{rank}(p(a_j)_T)\}$$

Then T' is the tree that results from an NNI move on T swapping the leaves a_k and a_i with $\operatorname{rank}(p(a_i)_T) = \operatorname{rank}(p(a_k)_T) + 1$. This follows directly from the fact that T' is the first tree on CSort(T, R) (Algorithm 5).

To prove $\widehat{d}(T', R) = \widehat{d}(T, R) - 1$, we distinguish two cases: (i) $\operatorname{rank}(p(a_i)_T) > 1$ and (ii) $\operatorname{rank}(p(a_i)_T) = 1$, i.e. a_i is in the cherry of T.

Case (i) By definition, (a_k, a_i) is a transposition in the set $P(T, R)$. As a_k and a_i are the only leaves whose order changes between T and T', they build the only transposition that is in $P(T, R)$ but not in $P(T', R)$. Hence $|P(T', R)| = |P(T, R)| - 1$.

We now compare $M(T, R)$ with $M(T', R)$. Since a_k and a_i are the only elements whose relation changes between T and T', they are the only elements that could be one of these sets but not the other. Because the definition of a_k requires all leaves that are above a_k in R to be at the same position in T, none of the leaves below a_k in T are above a_k in R. Thus $a_k \notin M(T, R)$ and $a_k \notin M(T', R)$. If $a_i \in M(T, R)$, it follows $a_i \in M(T', R)$, as only the relationship between a_i and a_k changes and the inequalities required for a_i to be in $M(T, R)$ and $M(T', R)$ are true for a_k: $a_k \preceq_T a_i$ and $a_i \prec_R a_k$. For the same reason, if $a_i \notin M(T, R)$, then $a_i \notin M(T', R)$. We can conclude $M(T', R) = M(T, R)$ and hence:

$$\widehat{d}(T', R) = |P(T', R)| - |M(T', R)| = |P(T, R)| - 1 - |M(T, R)| = \widehat{d}(T, R) - 1$$

Case (ii) As in the previous case, (a_k, a_i) is a transposition in $P(T, R)$, but not in $P(T', R)$. There is however another transposition that could be in $P(T, R)$: the pair (a_c, a_i), where a_c is the second cherry leaf of T alongside a_k (see Figure 5.8). Because (a_c, a_i) is the cherry in T', this pair cannot be a transposition in $P(T', R)$.

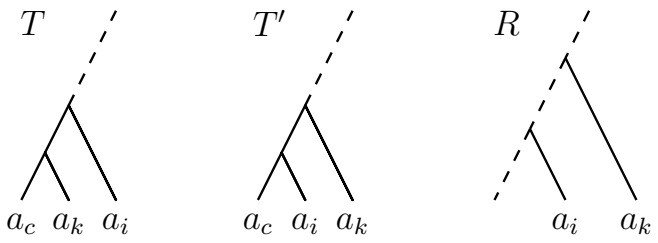


Figure 5.8: The caterpillar trees T, T', and R as described in the proof of Theorem 5.6. T and T' are neighbours and the dashed part of these two trees coincide.

We now distinguish the case (a) that (a_c, a_i) is not a transposition in $P(T, R)$ from the case (b) that (a_c, a_i) is a transposition in $P(T, R)$.

(a) If (a_c, a_i) is not a transposition in $P(T, R)$, then $|P(T', R)| = |P(T, R)| - 1$, because (a_k, a_i) is the only transposition that is in $P(T, R)$, but not in $P(T', R)$. As in the previous case (i) it also follows $|M(T, R)| = |M(T', R)|$ and we conclude

$$\widehat{d}(T', R) = |P(T', R)| - |M(T', R)| = |P(T, R)| - 1 - |M(T, R)| = \widehat{d}(T, R) - 1$$

(b) If (a_c, a_i) is a transposition in $P(T, R)$, then $|P(T', R)| = |P(T, R)| - 2$. To compare $|M(T, R)|$ with $|M(T', R)|$ it is sufficient to consider the membership of a_c, a_i, and a_k in $M(T, R)$ and $M(T', R)$. All other leaves are in $M(T, R)$ if and only if they are in $M(T', R)$.

As in case (i), $a_k \notin M(T, R)$ and $a_k \notin M(T', R)$. Furthermore, both elements a_c and a_k that are below a_i in T are above it in R, and neither a_c nor a_k is in the cherry of R, because they are above a_i in R. It follows $a_i \in M(T, R)$ and $a_i \in M(T', R)$. The leaf a_c is in $M(T, R)$, because there is only one leaf a_k that fulfils $a_k \preceq_T a_c$ and it also is $a_c \prec_R a_k$. Since (a_c, a_i) is a transposition in $P(T, R)$, we also know $a_i \prec_R a_c$. Together with $a_i \preceq_{T'} a_c$ it follows that $a_c \notin M(T', R)$. Therefore, it is $|M(T', R)| = |M(T, R)| - 1$ and we can conclude in total

$$\begin{aligned} \widehat{d}(T', R) &= |P(T', R)| - |M(T', R)| \\ &= |P(T, R)| - 2 - (|M(T, R)| - 1) \\ &= \widehat{d}(T, R) - 1. \end{aligned}$$

□

Corollary 5.7. *The distance between two caterpillar trees in* RNNI *can be computed in* $\mathcal{O}(n\sqrt{\log n})$.

Proof. By Theorem 5.6 the distance between two caterpillar trees in RNNI is the number of transpositions between two sequences of length n minus $|M(T,R)|$ as defined in Theorem 5.6.

We first show how the value $|M(T,R)|$ can be computed in time linear in n. Therefore, we assume without loss of generality that

$$T = [\{a_1, a_2\}, \{a_3\}, \{a_4\}, \ldots, \{a_n\}] \text{ and}$$
$$R = [\{a_{\pi(1)}, a_{\pi(2)}\}, \{a_{\pi(3)}\}, \{a_{\pi(4)}\}, \ldots, \{a_{\pi(n)}\}],$$

where π is a permutation of the set $\{1, 2, \ldots, n\}$. For now, we ignore a_1 and a_2, and only consider the elements $a_3, a_4, \ldots, a_n$ as potential elements in $M(T,R)$. The elements a_1 and a_2 will be considered as special case afterwards.

With our assumptions on T and R we can infer from the definition of $M(T,R)$ that an element $a_i = a_{\pi(k)}$ with $i \in \{3, \ldots, n\}$ is in the set $M(T,R)$ if $\{a_1, \ldots, a_{i-1}\} = \{a_{\pi(k+1)}, \ldots, a_{\pi(n)}\}$. Note that this implies $k = n - i$ as both sets must contain the same number of elements to be equal. We can be simplify the condition of the two sets being equal to $\sum_{j=n-i+1}^{n} \pi(j) == \sum_{j=1}^{i-1} j = \frac{i(i-1)}{2}$. This can be checked in linear time by iterating through R from top to bottom, i.e. iterating through $a_{\pi(n)}, a_{\pi(n-1)}, \ldots, a_{\pi(1)}$, and updating the sum $S = \sum_{j=n-i+1}^{n} \pi(j)$ of indices of elements that have already been considered up to the current iteration i. If $S = \sum_{j=n-i+1}^{n} \pi(j) == \frac{i(i-1)}{2}$ in iteration i, then we add the element a_i to $M(T,R)$.

We now consider the elements a_1 and a_2 that build the cherry of T. If both a_1 and a_2 are in the cherry of R, neither of these elements are in $M(T,R)$. If otherwise $a_1 \prec_R a_2$ and $a_2 \prec_R a_{\pi(1)}, a_{\pi(2)}$, then $a_2 \in M(T,R)$ and if $a_2 \prec_R a_1$ and $a_1 \prec_R a_{\pi(1)}, a_{\pi(2)}$, then $a_1 \in M(T,R)$.

Since the conditions for a_1 and a_2 to be in $M(T,R)$ can be checked simultaneously to the calculation of $M(T,R)$ as described above, $M(T,R)$ can be computed in time linear in n.

The number of transpositions of a sequence of length n (Kendall-tau distance) can be computed in time $\mathcal{O}(n\sqrt{\log n})$ (Chan and Pătraşcu 2010). This number is equal to $|P(T,R)|$, as defined in Theorem 5.6, when ignoring transpositions for the pairs of leaves sharing a parent in T and R, respectively. The worst-case running time for computing the RNNI distance between caterpillar trees is therefore $\mathcal{O}(n\sqrt{\log n})$. □

Since the problem of computing the number of transpositions has been studied extensively, it is likely that there is no algorithm with time complexity better than $\mathcal{O}(n\sqrt{\log n})$ to solve this problem. And as the problem of computing the distance between caterpillar trees is equivalent to this problem, we conjecture:

Conjecture 1. *$\mathcal{O}(n\sqrt{\log n})$ is a lower bound for the time complexity of computing the distance between two ranked trees on n leaves in* RNNI.

5.3 Diameter and Radius

In this section we investigate the *diameter* of tree spaces RNNI and DCT_m, which is the greatest distance between any pair of trees in each of these graphs, respectively, i.e. $\max_{\text{trees } T,R} d(T,R)$. We first establish the exact diameter of RNNI. By doing so we improve on the upper bound $n^2 - 3n - \frac{5}{8}$ given by Gavryushkin, Whidden, and Matsen (2018). Afterwards, we generalise this result to DCT_m. We finish this section by discussing the radius of RNNI and DCT_m.

5.3.1 Diameter

We start by establishing the diameter of the RNNI space before we discuss the diameter of the more general tree space DCT_m.

Theorem 5.8. *The diameter of* RNNI *is* $\frac{(n-1)(n-2)}{2}$.

Proof. For proving this theorem we use the fact that FindPath computes shortest paths in RNNI. Each iteration $i \in \{1, \ldots, n-2\}$ of FindPath, applied to two ranked trees T and R, decreases the rank of the most recent common ancestor of a cluster C, induced by the node of rank i in R, in the currently last tree T_1 on the already computed path (starting wth $T_1 = T$). The maximum rank of $mrca(C)_{T_1}$ at the beginning of iteration i is $n-1$, the rank of the root. As every move decreases the rank of $mrca(C)_{T_1}$ by one, there are at most $n - 1 - i$ moves in iteration i. The maximum length of a shortest path in RNNI is hence $\sum_{i=1}^{n-2} i = \frac{(n-1)(n-2)}{2}$.

We now give an example of two trees with this distance to prove that this is the actual diameter of RNNI and not just an upper bound. Consider the following two caterpillar trees T and R in their cluster representation:

$$T = [\{a_1, a_2\}, \{a_1, a_2, a_3\}, \ldots, \{a_1, \ldots, a_n\}]$$
$$R = [\{a_1, a_n\}, \{a_1, a_{n-1}, a_n\}, \ldots, \{a_1, \ldots, a_n\}]$$

In each iteration i of FindPath applied to T and R the maximum number $n-1-i$ of RNNI moves is needed. In the first the cluster under consideration is $\{a_1, a_n\}$, and the moves on FindPath decrease the rank of the parent of a_n by one in each step, until the cherry $\{a_1, a_n\}$ is present in the tree. In the following iterations $i \in \{2, \ldots, n-2\}$ the rank of the parent of leaf a_{n-i+1} decreases by NNI moves until it reaches rank i. Each iteration hence requires $n-1-i$ NNI moves. An example of this shortest path for $n = 4$ is depicted in Figure 5.9.

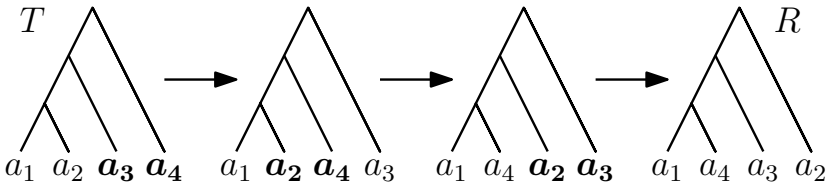


Figure 5.9: $\mathrm{FP}(T, R)$ for caterpillar trees T and R which have maximum distance $3 = \frac{(n-1)(n-2)}{2}$ (diameter) from each other, as described in the proof of Theorem 5.8. The leaves swapping positions in each move are indicated in bold.

□

We now generalise the diameter of RNNI to the tree space DCT_m.

Theorem 5.9. *The diameter of* DCT_m *is* $\frac{(n-1)(n-2)}{2} + (m-n+1)(n-1)$.

Proof. To prove the diameter of DCT_m, we consider the maximum number of moves that FindPath can perform on the extended ranked versions T_r and R_r of any two trees T and R. Remember that the extended ranked version T_r of T results from adding a caterpillar tree T_r^c on leaf set $\{a_{n+1}, \ldots, a_{m+2}\}$ and a root connecting it with T (Algorithm 3). In this extended ranked version T_r of T, the subtree that is identical to T is denoted by T_r^d. We distinguish RNNI moves in the subtrees on the leaf set $\{a_1, \ldots, a_n\}$ from the rank moves corresponding to length moves, i.e. rank moves between one node of each of the subtrees on leaf subsets $\{a_1, \ldots, a_n\}$ and $\{a_{n+1}, \ldots a_{m+2}\}$. Because FindPath preserves clusters (Lemma 4.4), the sum of the maximum number of moves in these two subtrees is an upper bound for the diameter of DCT_m.

The maximum number of RNNI moves on $\mathrm{FP}(T_r, R_r)$ inside the subtree induced by $\{a_1, \ldots, a_n\}$ follows from Theorem 5.8 and is $\frac{(n-1)(n-2)}{2}$. The maximum number of rank moves corresponding to length moves on a shortest path between T_r and R_r is reached when every internal node of the subtree T_r^c of T_r swaps rank with every internal node of the subtree T_r^d. This is $(m-n+1)(n-1)$, as there are $m-n+1$ internal nodes in

T_r^c and $n-1$ internal nodes in T_r^d. The sum of maximum number of RNNI and length moves, $\frac{(n-1)(n-2)}{2} + (m-n+1)(n-1)$, is hence an upper bound for the diameter of DCT_m.

To show that this actually is the diameter of DCT_m we give an example of trees T and R for which the path $\mathrm{FP}^+(T,R)$ has length $\frac{(n-1)(n-2)}{2} + (m-n+1)(n-1)$. Both T and R are caterpillar trees defined as follows by their cluster representation (Figure 5.10):

$$T = [\{a_1, a_2\} : m-(n-1), \{a_1, a_2, a_3\} : m-(n-2), \ldots, \{a_1, \ldots a_n\} : m-1]$$
$$R = [\{a_1, a_n\} : 1, \{a-1, a_{n-1}, a_n\} : 2, \ldots, \{a_1, \ldots, a_n\} : n-1]$$

Note that the underlying ranked trees of the subtrees T_r^d and R_r^d are identical to the example in Theorem 5.8 in RNNI and have distance $\frac{(n-1)(n-2)}{2}$, as discussed in that proof. Furthermore, T has no internal nodes with time less than $m-(n-1)$ and R has no nodes with time greater than $n-1$. Therefore, all internal nodes in T_r^c need to swap ranks with all nodes in T_r^d on every path to R_r. So T and R have the maximal distance $\frac{(n-1)(n-2)}{2} + (m-n+1)(n-1)$, which concludes this proof. □

By Proposition 4.3 the worst-case running time of FindPath in RNNI is $\mathcal{O}(n^2)$. The running time of FindPath$^+$ in DCT_m depends on its diameter, as in every execution of the loops inside the main for loop (lines 5 and 11 in Algorithm 4) a tree is added to the computed path. Therefore, the runtime of FindPath$^+$ is in $\mathcal{O}(nm)$. For computing a shortest path there is no algorithm with better worst-case running time than $\mathcal{O}(mn)$, as the running time for algorithms computing shortest paths is bounded from below by the diameter of the corresponding space (see also Corollary 4.9). There could however be more efficient algorithms for computing distances, if this is not done by computing the shortest path, but by finding an invariant that determines the distance.

5.3.2 Radius

The *eccentricity* of a vertex in a graph is the greatest distance of any other vertex to it, i.e. the eccentricity of a tree T in DCT_m is $ecc(T) = \max_R d(T,R)$. The minimum eccentricity of all vertices in a graph, i.e. $\min_T ecc(T) = \min_T \max_R d(T,R)$, is called the *radius* of a graph. In the following we show that the radius of RNNI equals its diameter. We will see afterwards that this is not true for DCT_m.

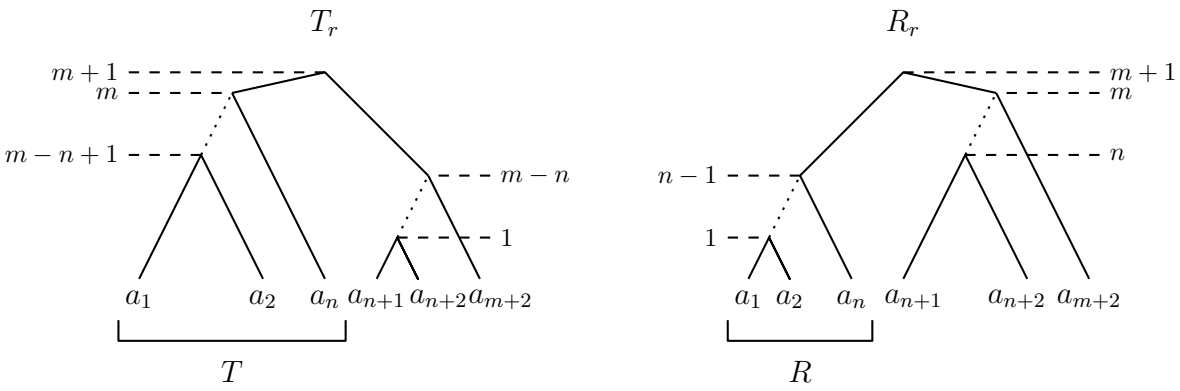


Figure 5.10: Ranked versions T_r and R_r of discrete coalescent trees T and R with distance $\frac{(n-1)(n-2)}{2} + (m-n+1)(n-1)$ as described in the proof of Theorem 5.9. T and R (which are identical to the subtrees T_r^d and R_r^d) are the subtrees on the left in the corresponding tree T_r and R_r, respectively

Theorem 5.10. *The radius of* RNNI *equals its diameter* $\frac{(n-1)(n-2)}{2}$.

Proof. We prove this theorem by showing that every ranked tree T in RNNI has a caterpillar tree R with distance $\frac{(n-1)(n-2)}{2}$ to T, using induction on the number of leaves n.

The base case $n = 3$ is trivial, as all three ranked trees in this RNNI space are caterpillar trees with distance one from each other, which equals the diameter distance. For the induction step we consider an arbitrary ranked tree T with $n+1$ leaves and assume that for every tree on less than $n+1$ leaves there exists a caterpillar tree with diameter distance from it. Let $\{x, y\}$ be the cherry of T, and let T' be the tree on n leaves resulting from deleting one of these leaves, say x, from T and suppressing the resulting degree-2 vertex. By the induction hypothesis there is a caterpillar tree R' with distance $\frac{(n-1)(n-2)}{2}$ to T'. Now consider the caterpillar tree R resulting from adding x at the top of R' such that the root of R has x and R' as children.

We now consider $\mathrm{FP}(R, T)$ and show that $d(T, R) = |\mathrm{FP}(R, T)| = \frac{n(n-1)}{2}$ to finish the proof. In the first iteration of FindPath, the rank of $mrca(\{x, y\})_R$ is decreased until it reaches rank one. Since there is always a unique move on FindPath that decreases the rank of this *mrca* (Proposition 4.3), we can infer that the following moves, which all decrease the rank of $mrca(\{x, y\})$ by one, are performed in the first iteration on $\mathrm{FP}(R, T)$: First $p(x)_R$, which is the root in R, moves down by NNI moves until it reaches rank $\mathrm{rank}(p(y)_R) + 1$. Then a further NNI move creates an internal node with children x and y, and hence a cherry $\{x, y\}$, before this node is moved down by rank moves to reach rank one. This first iteration of $\mathrm{FP}(T, R)$ is depicted

in Figure 5.11. Note that the NNI moves before and after building the cherry $\{x, y\}$ might not be necessary, depending on the position of y in R.

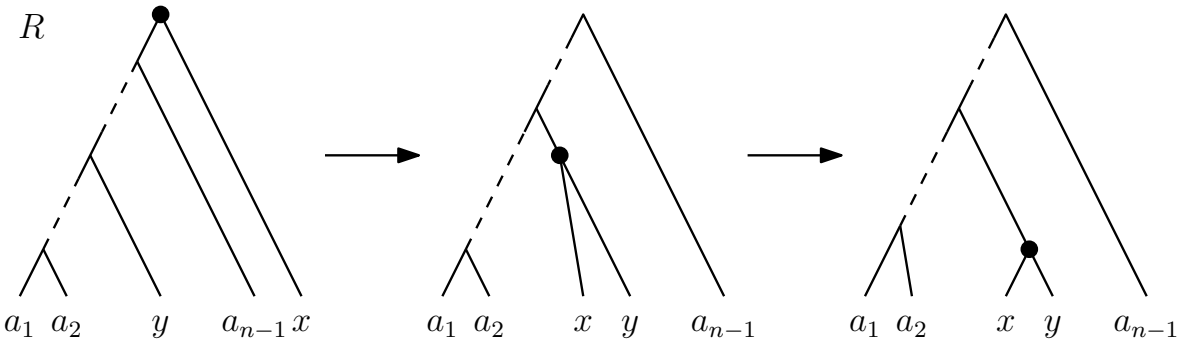


Figure 5.11: Initial $n-1$ RNNI moves of $\mathrm{FP}(R, T)$ as described in the proof of Theorem 5.10.

Altogether there are $n-1$ RNNI moves needed in the first iteration, as the rank of the parent of x decreases by one within every move, starting at the root with rank n and ending at the internal node of rank one. The tree at the end of this first iteration on $\mathrm{FP}(R, T)$ is identical to R' when removing the leaf x and suppressing its parent (the node of rank one). Since the cluster $\{x, y\}$ is not considered again in FindPath, the remaining part of $\mathrm{FP}(R, T)$ contains the same moves as $\mathrm{FP}(R', T')$, i.e. clusters are changed in the same way. Hence $|\mathrm{FP}(R, T)| = |\mathrm{FP}(R', T')| + n - 1$. Therefore, $d(T, R) = \frac{(n-1)(n-2)}{2} + n - 1 = \frac{n(n-1)}{2}$, which concludes the proof. □

In contrast to RNNI, the radius of DCT_m is not equal to its diameter. A counterexample is given by the tree $[\{a_1, a_2\} : 2, \{a_1, a_2, a_3\} : 4]$ on three leaves in DCT_4 (see Figure 5.12). The diameter of DCT_4 on three leaves is $\frac{(n-1)(n-2)}{2} + (m-n+1)(n-1) = 5$, but there is no tree with this distance from this tree. Exhaustive search shows that the maximum distance between any tree in DCT_4 and the tree in Figure 5.12 is 4.

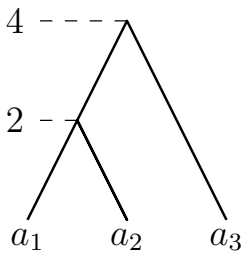


Figure 5.12: For this tree in DCT_4 on three leaves there exists no tree with diameter distance $5 = \frac{(n-1)(n-2)}{2} + (m-n+1)(n-1)$ from it.

5.4 Distributions of distance

In this section we consider ranked trees drawn from a uniform distribution and analyse distances between these trees. Most of our results are based on simulations. We are also able to prove the expected distance between a caterpillar tree and a ranked tree drawn from a uniform distribution. Our results provide an insight into the distribution of distances between trees sampled from a uniform distribution, and hence a first attempt to analyse distributions in the RNNI space.

In Section 5.4.1 we discuss uniform distributions on the set of ranked trees and the set of caterpillar trees. We then discuss some simulations of uniformly distributed ranked trees and their distances in RNNI in Section 5.4.2. More precisely, we consider the mean distance between uniformly sampled ranked trees and distances between start and end trees of random walks in RNNI. We finish this section by establishing the expected distance between caterpillar trees and ranked trees drawn from uniform distributions in Section 5.4.3 (Theorem 5.12).

5.4.1 Uniform distribution of ranked trees

In Section 3.5 we counted the number of ranked trees in RNNI using the coalescent process (Algorithm 1). Since for any tree there is exactly one sequence of coalescent events, and all coalescent events are equally likely, this process returns trees drawn from a uniform distribution of ranked trees. We can hence use the coalescent process to sample ranked trees from a uniform distribution.

Let us now consider the uniform distribution on caterpillar trees. In Section 5.2.1 we introduced the caterpillar list representation, where caterpillar trees in RNNI are represented as lists of length $n-1$. For a caterpillar tree T, the set at position i of the caterpillar list representation contains the leaves that are children of the internal node of rank i in T. All sets at position $i > 1$ of this list contain exactly one element, while the first list contains two leaves, the leaves of the cherry of T. We can hence interpret such a list as a permutation of $a_1, \ldots, a_n$, where the first two elements are considered to be equivalent. This means that for each caterpillar tree there are two permutations associated with that tree. Since there are $n!$ permutations of $a_1, \ldots, a_n$, we can conclude that there are $\frac{n!}{2}$ caterpillar trees. If Y is the uniform distribution on caterpillar trees, we get $\mathbb{P}(Y = T) = \frac{2}{n!}$ for a caterpillar tree T.

We now consider the *group elimination* property that a distribution X_n of trees on n leaves has if the following is true: A tree resulting from deleting a subtree with k

leaves from a tree drawn from X_n is distributed as X_{n-k} (Aldous 1996).

Aldous (1996) have shown that the uniform distribution of caterpillar trees has the group elimination property as well as the Yule distribution on time trees. Because the Yule model gives a uniform distribution on ranked trees (Steel and McKenzie 2001), we can infer that the uniform distribution on ranked trees has the group elimination property as well.

5.4.2 Simulations

We now use simulations to investigate RNNI distances between ranked trees sampled from a uniform distribution. After getting a general idea of the distribution of these distances, we consider the mean distance between uniformly sampled ranked trees and see how it changes with increasing number of leaves n. We finish this section by analysing the distance between start and end trees of random walks in the RNNI space.

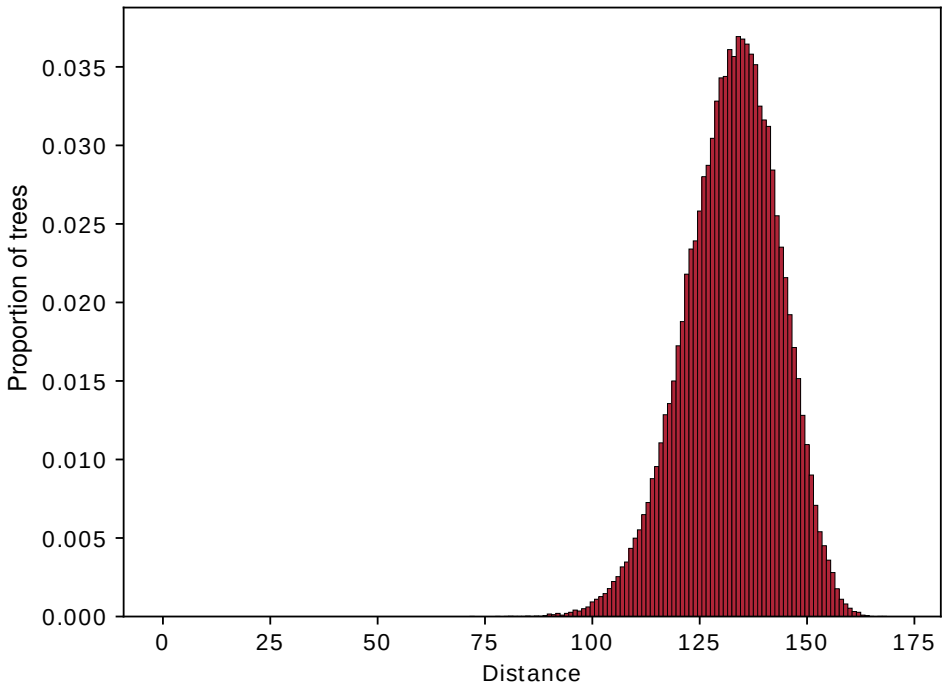


Figure 5.13: Histogram of RNNI distances between $100,000$ pairs of trees on 20 leaves

In our first simulation, we draw pairs of ranked trees from a uniform distribution. We simulate $100,000$ such pairs of ranked trees on 20 leaves, using our own implementation of the coalescent process (Algorithm 1). We then use FindPath to compute the

RNNI distance for each pair of trees. The results are illustrated in Figure 5.13, where on the x-axis all possible distances d between trees on $n = 20$ leaves are displayed. Each bar represents the fraction of the $100,000$ tree pairs with distance d. The diameter of the RNNI space for $n = 20$ is 171, distances can hence range from 0 to 171.

One can see in Figure 5.13 that most distances are between 90 and 160, centred around 135. Since the maximum RNNI distance for tree with $n = 20$ leaves is 171, most distances are between 50 and 80 percent of the diameter.

In the next step we analyse the mean distance between pairs of ranked trees drawn from a uniform distribution. We therefore repeat the simulation above for $n = 2^k$ with $k = 2, \ldots, 11$, while restricting the number of tree pairs to $10,000$ to maintain a feasible running time. For each value of n we compute the mean distance of all $10,000$ pairs of trees and plot the result in Figure 5.14. Note that the distance in this plot is divided by the diameter of the corresponding tree space, i.e. $\frac{(n-1)(n-2)}{2}$ (Theorem 5.8).

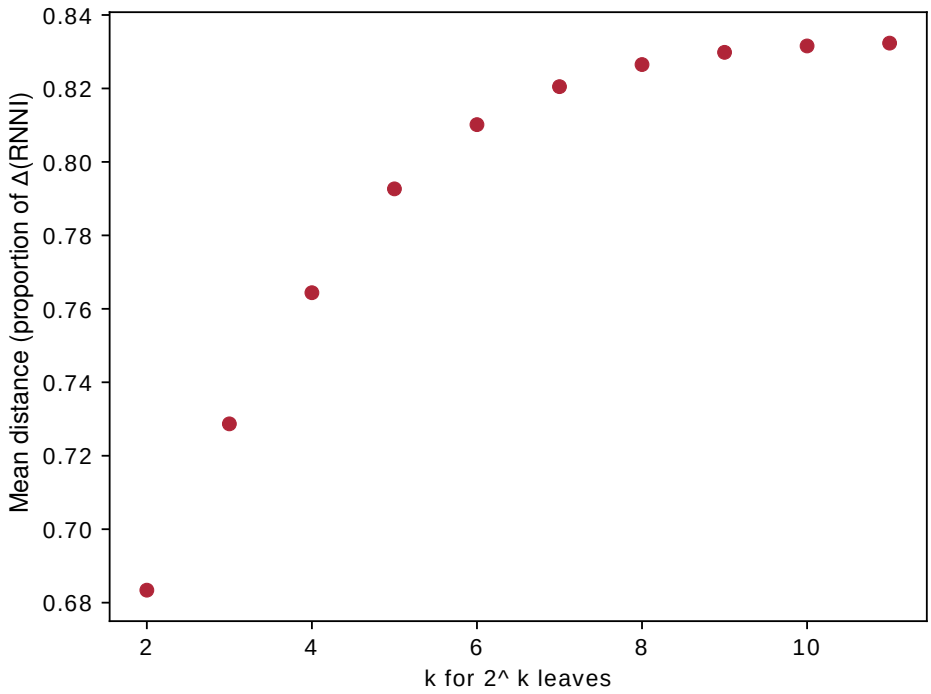


Figure 5.14: Mean RNNI distance of $10,000$ pairs of trees on $n = 2^k$ leaves with k ranging from two to 11.

The results in Figure 5.14 indicate that with increasing number of leaves, the mean RNNI distance increases. It however looks like the mean approaches an asymptote which might be at around distance 0.83 times the diameter.

We next consider random walks in RNNI. We therefore draw a start tree from the uniform distribution on ranked trees using the coalescent process (Algorithm 1). A random walk of length k starts at this start tree and is extended iteratively by an RNNI neighbour of the current tree. This neighbour is chosen randomly from a uniform distribution on the set of RNNI neighbours of the current tree. Note that we do not allow the next tree to be the current tree, i.e. one tree cannot be followed by itself on a random walk.

We simulate $10,000$ trees on $n = 10$ leaves as start trees for random walks, i.e. $10,000$ random walks. In our first simulation the random walks have length 36, which is the diameter of the RNNI space for trees with $n = 10$ leaves. In Figure 5.15 we plot the distances between start and end tree of random walks as a histogram. The height of a bar associated with distance d indicates the relative number of random walks that end in a tree with distance d from the start tree. We can see in Figure 5.15 that most random walks result in trees with distance 5 to 20 from the start tree. Since the diameter of the tree space on 10 leaves is 36, this indicates that random walks rarely result in trees with diameter distance.

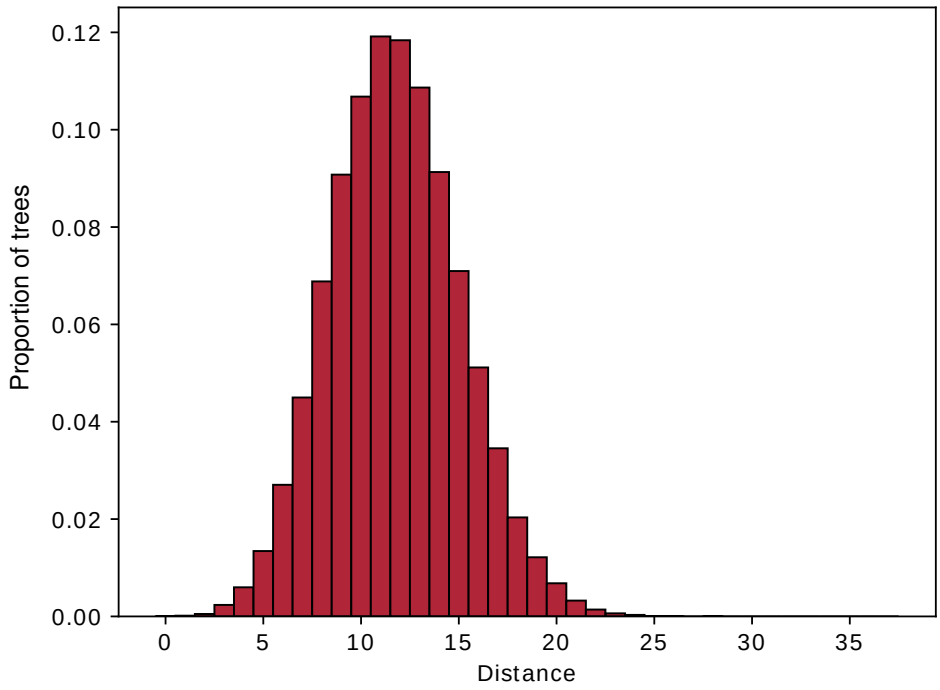


Figure 5.15: Histogram of RNNI distances between start and end tree of $10,000$ random walks of length 36 on ranked trees with $n = 10$ leaves.

To investigate the behaviour of random walks in RNNI further, we repeat our simulations of random walks for random walks with different lengths. We again fix the number of leaves to $n = 10$ and consider $10,000$ random walks for every chosen length of random walks, which is 2^k for values of k ranging from one to 15. For every value of k we compute the mean distances between start and end tree of the $10,000$ random walks and display the results in Figure 5.16. Note that distances are again divided by the diameter of the corresponding RNNI space. One can see in the plot that with increasing length of the random walk, the mean distance between start and end tree increases. From $k = 8$, and hence a random walk length of 256 (roughly seven times the diameter), the distance between start and end tree of the random walk seems to approach an asymptote at distance 0.783 of the diameter.

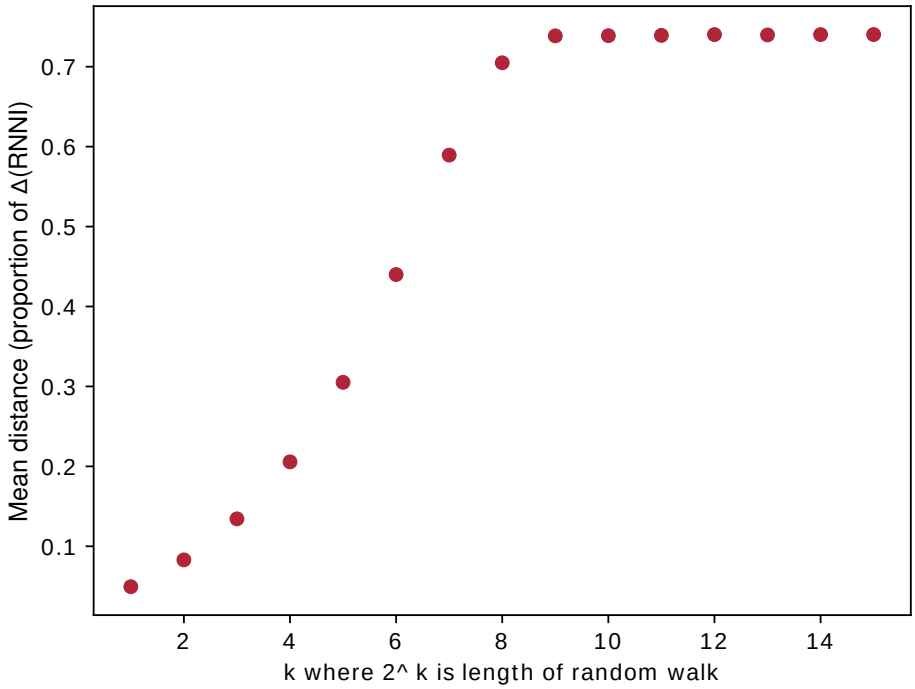


Figure 5.16: Mean RNNI distance between start and end tree of $10,000$ random walks of length 2^k on ranked trees with $n = 10$ leaves with k ranging from one to 15.

5.4.3 Caterpillar trees

In this section we consider expected distances to caterpillar trees, as we have seen in Section 5.2 that these trees build an interesting subset in the RNNI graph. More

specifically, we are interested in distances between a caterpillar tree and a ranked tree, both sampled from uniform distributions. We establish the mean distance between such trees in Theorem 5.12. We first need the following lemma.

Lemma 5.11. *Let T be a ranked caterpillar tree drawn from a uniform distribution on the set of caterpillar trees with n leaves in* RNNI *and let $x, y \in \{a_1, \ldots, a_n\}$ be two different leaves of T. The probability $p_{k,n}$ that the mrca of $\{x, y\}$ has rank k in T is*

$$p_{k,n} = \frac{2k}{n(n-1)}.$$

Proof. For proving the lemma we distinguish the two cases $k = 1$ and $k > 1$. If $k = 1$, the probability that a set $\{x, y\}$ has *mrca* of rank one equals the number of caterpillar trees with cherry $\{x, y\}$ divided by the total number of caterpillar trees. The number of caterpillar trees with cherry $\{x, y\}$ is $(n-2)!$, as every permutation of the set $\{a_1, \ldots, a_n\} \setminus \{x, y\}$ provides a different caterpillar tree with cherry $\{x, y\}$ (see also discussion of permutations and caterpillar trees in Section 5.4.1). Since the total number of caterpillar trees on n leaves is $\frac{n!}{2}$, we can infer

$$p_{1,n} = \frac{(n-2)!}{\frac{n!}{2}} = \frac{2}{n(n-1)} = \frac{2k}{n(n-1)}$$

We now consider the case $k > 1$. If the *mrca* of $\{x, y\}$ has rank $k > 1$ in T, then either x has parent with rank k and y parent with rank less than k, or the parent of y has rank k and the parent of x rank less than k, and hence:

$$p_{k,n} = \mathbb{P}(\text{rank}(p(x)) = k, \text{rank}(p(y)) < k) + \mathbb{P}(\text{rank}(p(y)) = k, \text{rank}(p(x)) < k).$$

Let X an Y be the random variables describing $\text{rank}(p(x))$ and $\text{rank}(p(y))$, respectively. With the symmetry of x and y, we can summarise

$$p_{k,n} = 2 \cdot \mathbb{P}(X = k, Y < k).$$

Now consider $\mathbb{P}(X = k, Y < k)$. We can write this using the conditional probability as follows:

$$\mathbb{P}(X = k, Y < k) = \mathbb{P}(Y < k | X = k) \cdot \mathbb{P}(X = k)$$

Since we assume that T is chosen from a uniform distribution of caterpillar trees on n leaves in RNNI, we can calculate $\mathbb{P}(X = k)$ by dividing the number of caterpillar trees for which the parent of x has rank k by the total number of caterpillar trees in RNNI. Remember that caterpillar trees can be interpreted as sequences, where the

two first element are equivalent (see Section 5.4.1). If the leaf x is positioned such that its parent has rank k, there are $n-1$ positions to be filled by the remaining leaves in a caterpillar tree. There are $(n-1)!$ possible permutations of these remaining leaves that result in possible labellings of T. Because $k > 1$, x is not in the cherry of T. Therefore, exchanging the first two elements, the cherry leaves, in each of the $(n-1)!$ permutations does not change T, which means that there are in total $\frac{(n-1)!}{2}$ caterpillar trees in RNNI in which the leaf x has parent with rank k. Since the total number of caterpillar trees is $\frac{n!}{2}$, we get

$$\mathbb{P}(X = k) = \frac{\frac{(n-1)!}{2}}{\frac{n!}{2}} = \frac{1}{n}$$

Now consider $\mathbb{P}(Y < k|X = k)$. Similar to above, we count the number of trees for which the parent of y has rank less than k if the parent of x has rank k, and divide this number by the number of trees for which the parent of x has rank k to get $\mathbb{P}(Y < k|X = k)$. If the parent of x has rank k and the parent of y has rank less than k, there are $n-2$ elements left that can be assigned as leaf labels to T in any order. We now distinguish the case that y is in the cherry of T from the case that it is not. If y is in the cherry, then every of the $(n-2)!$ permutations of the remaining leaves leads to a unique caterpillar tree. Otherwise, if y is not in the cherry, only half of those provide different trees, because swapping the first two elements, the leaves of the cherry, does not change the tree. If the parent of x has rank k, the total number of trees for which y has rank between 1 and $k-1$ is hence:

$$(n-2)! + \sum_{i=2}^{k-1} \frac{(n-2)!}{2} = (n-2)! + \frac{1}{2}(k-2)(n-2)! = \frac{k}{2}(n-2)!$$

Since the number of trees in which the parent of x has rank k is $\frac{(n-1)!}{2}$, we can summarise:

$$\mathbb{P}(Y < k|X = k) = \frac{\frac{k}{2}(n-2)!}{\frac{(n-1)!}{2}} = \frac{k}{n-1}$$

We conclude

$$\begin{aligned} p_{k,n} &= 2 \cdot \mathbb{P}(X = k, Y < k) \\ &= 2 \cdot \mathbb{P}(Y < k|X = k) \cdot \mathbb{P}(X = k) \\ &= 2 \cdot \frac{k}{n-1} \cdot \frac{1}{n} \\ &= \frac{2k}{n(n-1)}. \end{aligned}$$

□

Let $\Delta(\text{RNNI})$ denote the diameter of RNNI. We now show that the mean distance between a caterpillar tree and a ranked tree is $\frac{2}{3}\Delta(\text{RNNI})$.

Theorem 5.12. *The expected* RNNI *distance between a caterpillar tree T and a ranked tree R is*

$$\mathbb{E}(d(T,R)|T \text{ is a caterpillar tree}) = \frac{2}{3}\Delta(\text{RNNI}) = \frac{(n-1)(n-2)}{3}$$

Proof. Let T be drawn from a uniform distribution on caterpillar trees and R be drawn from a uniform distribution on ranked trees. Let furthermore T' be the tree after the first iteration of FindPath applied to (T,R). Then the cluster induced by the node with rank one in T' and R is identical. Let $\{x,y\}$ be this cluster, i.e. the lowermost cherry of T' and R. We furthermore assume without loss of generality that $\text{rank}(p(x)_T) \geq \text{rank}(p(y)_T)$.

Since the moves on $\text{FP}(T,R)$ between T and T' decrease the rank of $mrca(\{x,y\})$ and $\text{rank}(p(x)_T) \geq \text{rank}(p(y)_T)$, the first moves on this part of the path swap $p(x)$ with the node of rank one less by NNI moves, until $\text{rank}(p(x)) = \text{rank}(p(y)) + 1$ (see Figure 5.11, where the names of T and R are exchanged). In the next step, the cherry $\{x,y\}$ is built, which decreases the rank of $mrca(\{x,y\})$ by one, before the rank of this internal node is decreased by rank moves until it reaches one. Note that neither the NNI moves before building the cherry nor the rank moves afterwards are necessary in every case, depending on where x and y are in T. If T already contains the cherry $\{x,y\}$, it is $T' = T$ already and none of these moves are needed. A description of this first iteration of $\text{FP}(T,R)$ can also be found in Theorem 5.10 (the roles of T and R are swapped there).

Since $\text{FP}(T,R)$ is a shortest path between T and R (Theorem 4.6), it follows

$$\mathbb{E}(d(T,R)) = \mathbb{E}(d(T,T') + d(T',R)) = \mathbb{E}(d(T,T')) + \mathbb{E}(d(T',R))$$

We now consider $\mathbb{E}(d(T,T'))$ and $\mathbb{E}(d(T',R))$ separately. We start by establishing $\mathbb{E}(d(T,T'))$.

Remember that T' is the tree after the first iteration of FindPath. The distance between T' and T is hence $\text{rank}(mrca(\{x,y\})_T) - 1$. If $p_{k,n}$ is the probability that the *mrca* of the cherry of R has rank k in T, we can calculate the expected distance between T and T' as follows:

$$\mathbb{E}(d(T,T')) = \sum_{k=1}^{n-1}(k-1)p_{k,n}.$$

With $p_{k,n} = \frac{2k}{n(n-1)}$ (Lemma 5.11) this results in:

$$\begin{aligned}
\mathbb{E}(d(T,T')) &= \sum_{k=1}^{n-1}(k-1)p_{k,n} \\
&= \sum_{k=1}^{n-1}(k-1)\frac{2k}{n(n-1)} \\
&= \frac{2}{n(n-1)}\sum_{k=1}^{n-1}(k-1)k \\
&= \frac{2}{n(n-1)}\Big(\sum_{k=1}^{n-1}k^2 - \sum_{k=1}^{n-1}k\Big) \\
&= \frac{2}{n(n-1)}\Big(\frac{n(n-1)(2n-1)}{6} - \frac{n(n-1)}{2}\Big) \\
&= \frac{2n-1}{3} - 1
\end{aligned}$$

Now consider $\mathbb{E}(d(T',R))$. Remember that T' is either a caterpillar tree (if the cherries of T and R share at least one leaf), or a caterpillar tree with an additional cherry that has rank two (like the tree on the right of Figure 5.11). By the nature of FindPath, no move on $\mathrm{FP}(T',R)$ affects the node inducing $\{x,y\}$. Furthermore, deleting the leaf x from T' results in the same tree as deleting x from T. This especially implies that the length of $\mathrm{FP}(T',R)$ is equal to the length of $\mathrm{FP}(T|_n, R|_n)$, where $T|_n$ and $R|_n$ are the trees resulting from deleting x from T and R, respectively. Because the uniform distributions on ranked trees and caterpillar trees have the group elimination property (Section 5.4.1), $T|_n$ is drawn from a uniform distribution on caterpillar trees and $R|_n$ is drawn from a uniform distribution on ranked trees. The expected distance between T' and R hence equals the expected distance between a caterpillar tree and an arbitrary ranked tree on $n-1$ leaves.

Let $\mathbb{E}_n$ be the expected distance between a caterpillar tree and an arbitrary tree on n leaves. We can summarise our results above to:

$$\begin{aligned}
\mathbb{E}_n &= \mathbb{E}(d(T,R)) \\
&= \mathbb{E}(d(T,T')) + \mathbb{E}(d(T',R)) \\
&= \frac{2n-1}{3} - 1 + \mathbb{E}_{n-1}
\end{aligned}$$

We now prove $\mathbb{E}_n = \frac{(n-1)(n-2)}{3}$ by induction.

In the base case $n = 3$ the RNNI space consists of three caterpillar trees that have distance one from each other. The diameter of the space is hence one and the expected

distance is

$$\mathbb{E}_3 = \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 1 = \frac{2}{3} = \frac{(n-1)(n-2)}{3},$$

because each of the trees has one caterpillar tree with distance zero and two with distance one from it.

Now consider the induction step, where we assume $\mathbb{E}_{n-1} = \frac{(n-2)(n-3)}{3}$ and show $\mathbb{E}_n = \frac{(n-1)(n-2)}{3}$. We use the recursion from above and insert the induction hypothesis to finish the proof:

$$\begin{aligned}
\mathbb{E}_n &= \mathbb{E}_{n-1} + \frac{2n-1}{3} - 1 \\
&= \frac{(n-2)(n-3)}{3} + \frac{2n-1}{3} - 1 \\
&= \frac{(n-2)(n-3)}{3} + \frac{2n-4}{3} \\
&= \frac{(n-2)(n-3)}{3} + \frac{2(n-2)}{3} \\
&= \frac{(n-2)(n-3+2)}{3} \\
&= \frac{(n-2)(n-1)}{3}
\end{aligned}$$

□

Chapter 6

Unlabelled trees

In this chapter we discuss tree spaces of unlabelled trees. Our goal is to find a distance measure for unlabelled trees that is both biologically meaningful and efficiently computable. Comparing unlabelled trees, which are often called tree shapes, is of interest in applications where the goal is to compare the branching process rather than the actual evolutionary history. This is for example the case in phylodynamics, if one wants to compare the branching history of the spread of two different viruses. In the literature, such analyses are mostly done by using tree balance indices (Sackin 1972; Colless 1982) to compare unlabelled trees (Frost and Volz 2013; Poon et al. 2013).

The idea of balance indices is that the caterpillar tree is the most unbalanced tree, while a fully balanced tree receives the maximum value of a balance index. These trees are therefore considered to be most different under this approach. Many methods for comparing unlabelled trees have been proposed (Matsen 2006; Liu et al. 2020; Pompei, Loreto, and Tria 2012; Hayati, Shadgar, and Chindelevitch 2019; Kim, Rosenberg, and Palacios 2020), but so far there is no efficiently computable distance measure for unlabelled trees that is based on tree rearrangement operations.

Another option for comparing unlabelled trees is to take a distance measure that can be computed efficiently for labelled trees and add leaf labels that minimise the distance (Goloboff, Arias, and Szumik 2017; Colijn and Plazzotta 2018). Goloboff, Arias, and Szumik (2017) for example present a heuristic for labelling trees to minimise distances between labelled trees. This idea of first introducing leaf labels however has the downside that distance measures that are efficiently computable between labelled trees lack biological meaningfulness, as we discussed in Chapter 1.

Following our discussion in Chapter 1 it is sensible to use distance measures based on tree rearrangement operations. Tree rearrangement based distances for unlabelled trees

are however not typically efficiently computable. Computing the unlabelled version of the NNI distance for example is $\mathcal{NP}$-hard (Dasgupta et al. 2000). Furthermore, most of these distance measures do not take times of internal nodes into account, and are hence not suitable for unlabelled time trees.

On top of the lack of efficiently computable and biologically sensible distance measures for unlabelled trees, there are also only few distance measures for unlabelled time trees, where internal nodes are assigned times. Kim, Rosenberg, and Palacios (2020) introduce a new metric for such trees that can be computed efficiently, though their method lacks biological interpretability as some trees that are very similar (connected by one NNI move) are very distant under their metric.

We start this chapter by introducing the space uDCT_m of unlabelled discrete coalescent trees as an adaptation of DCT_m to unlabelled trees (Section 6.1). This space is based on tree rearrangement operations and therefore permits biological interpretability that most of the above mentioned methods lack. We will however see that computing shortest paths in this tree space might not be as easy as in DCT_m. In particular, we cannot directly use the algorithm FindPath$^+$ or a simple variation of it for unlabelled trees in uDCT_m. We discuss this issue and propose ways to approximate distances in uDCT_m. Afterwards, we introduce a distance measure for unlabelled trees that is similar to the subtree exchange (or subtree swap) distance for labelled trees (Höhna and Drummond 2012; Drummond, Nicholls, et al. 2002). We discuss this distance measure and show that it can be computed efficiently in Section 6.2.

6.1 The tree space uDCT_m

An *unlabelled discrete coalescent tree*, or short *unlabelled tree*, is a rooted binary tree where internal nodes are assigned unique positive integer-valued times such that every node has time greater than its children. We assume that all leaves of an unlabelled tree are assigned time zero while all other nodes have distinct integer-valued times. Two unlabelled trees are called *identical* if there is a graph isomorphism between them that preserves node times. We can derive unlabelled trees from (labelled) discrete coalescent trees by ignoring leaf labels. The unlabelled tree that results from ignoring leaf labels of a discrete coalescent tree T is the *underlying unlabelled tree* of T. Different discrete coalescent trees can have the same underlying unlabelled tree, as depicted in Figure 6.1.

We define tree rearrangement operations between unlabelled trees similar to those between labelled trees in DCT_m, as well as a tree space based on these operations. The

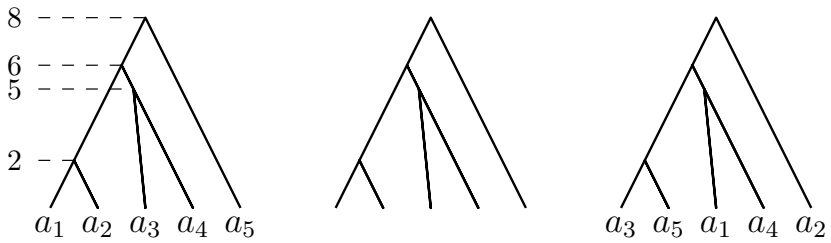


Figure 6.1: Two discrete coalescent trees on $n = 5$ leaves on the left and right. The unlabelled tree in the middle is their (shared) underlying unlabelled tree.

graph uDCT$_m$ on unlabelled trees with n leaves is defined as follows. The vertex set of uDCT$_m$ is the set of unlabelled trees with n leaves and root time less than or equal to m. Unlabelled trees T and R are connected by an edge in uDCT$_m$ if one of the following operations on T results in R:

(i) An NNI *move* connects unlabelled trees T and R if there is an edge e in T and an edge f in R, both of length one, such that contracting e and f to nodes results in identical (non-binary) unlabelled trees.

(ii) A *rank move* on T exchanges the times of two internal nodes with time difference one.

(iii) A *length move* on T changes the time of an internal node by one.

Note that these moves are defined in the same way as the moves in DCT$_m$ on labelled trees. An example of unlabelled trees and uDCT$_m$ moves between such trees is depicted in Figure 6.2.

The neighbourhood structure for uDCT$_m$ differs in important ways from that of DCT$_m$:

Observation 6.1. *Not every edge of length one in an unlabelled ranked tree T provides two* NNI *neighbours that are different from T. Furthermore, there can be rank moves on an unlabelled ranked tree T that result in the same tree T.*

Consider for example the NNI move in Figure 6.3 between the tree T in the middle and the tree on the left. This move is performed on dashed edge on T, for which there is no other NNI move that leads to a different tree. Furthermore, the rank move on T on the right of that figure results in the identical tree T.

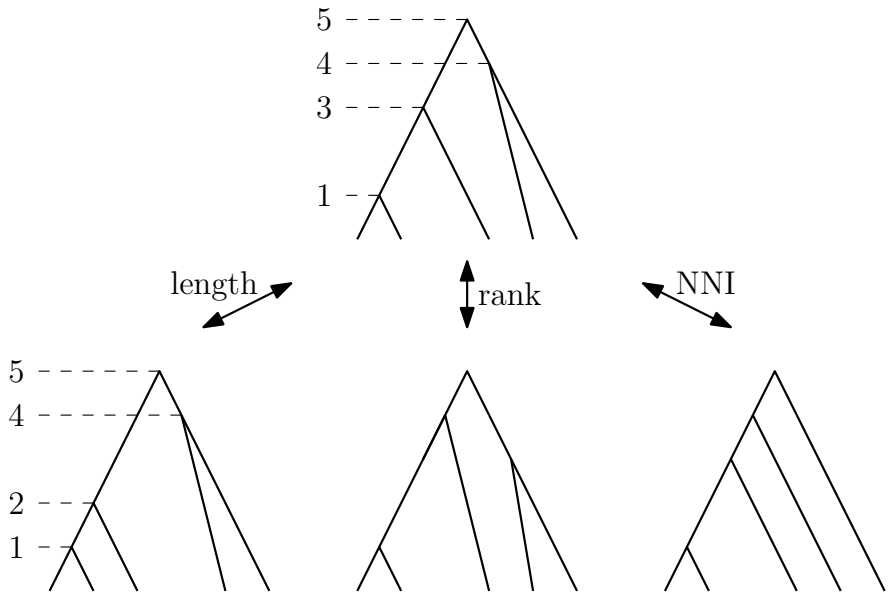


Figure 6.2: Three different move types possible on unlabelled trees in uDCT_5. A length move decreasing the time of an internal node from 3 to 2 on the left, a rank move swapping times of internal nodes with times 3 and 4 in the middle, and an NNI move on the edge connecting nodes with rank 4 and 5 on the right.

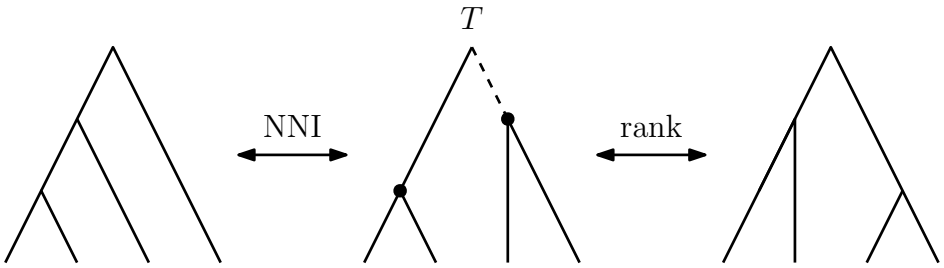


Figure 6.3: Unlabelled ranked tree T that has only one NNI neighbour on the dashed edge (tree on the left). A rank move of the two highlighted nodes results in the same tree T (on the right).

Another interesting observation is that a rank move and an NNI move on an unlabelled tree can result in the same neighbouring tree. Figure 6.4 shows one example of this. An NNI move on the dashed edge and a rank swap of the nodes with times 1 and 2 in the tree on the left result in the same tree depicted on the right.

We use the following graph theoretical terms in uDCT_m, as we do for DCT_m (see Section 3.2). A *path* in uDCT_m is a sequence $[T_0, T_1, \ldots, T_k]$ of unlabelled trees, such that T_i and T_{i+1} are connected by an NNI move, rank move, or length move for all $i = 0, \ldots, k-1$. The *length* of a path $p = [T_0, T_1, \ldots, T_k]$ is k. Furthermore, we

define the *distance* between two unlabelled trees in uDCT_m as the length of a shortest path between them. In other words, the distance between two unlabelled tree is the minimum number of tree rearrangement operations in uDCT_m (length moves, NNI moves, rank moves) needed to transform one unlabelled tree into the other.

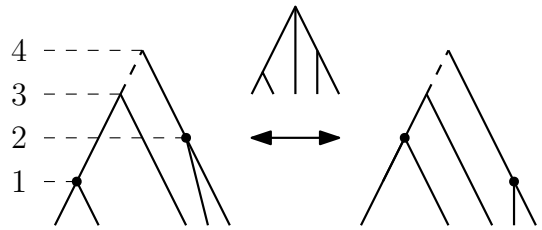


Figure 6.4: Two unlabelled trees that are connected by an NNI and a rank move. The NNI move is performed on the dashed edge and the rank move between the marked nodes with times 1 and 2. The tree depicted above the arrow results from contracting the dashed edges in the other two trees (see definition of NNI move).

Unlabelled trees with root time $n-1$ can be interpreted as ranked trees without leaf labels. These trees have integers in $\{1, \ldots, n-1\}$ assigned as times to their internal nodes and are called *unlabelled ranked trees*. We will later focus on the tree space uDCT_{n-1} of unlabelled ranked trees and denote it by uRNNI, analogously to RNNI for (labelled) ranked trees.

Because DCT_m is a connected graph (Theorem 3.1), uDCT_m is connected graph as well: for any two trees in uDCT_m we can construct a path by arbitrarily labelling these trees, taking a path between the labelled trees in DCT_m, and then deleting the leaf labels from all trees on that path. Because every move in DCT_m corresponds to a single move or no change in uDCT_m, the resulting path is a walk in uDCT_m.

Since uDCT_m is connected, it induces a metric. We therefore interpret it as (discrete) tree space and call it uDCT_m space.

6.1.1 Shortest paths in uDCT_m

We now investigate the Shortest Path Problem for unlabelled trees in uDCT_m. This tree space is similar to DCT_m, as it is based on the same tree rearrangement operations, only on a different type of time trees. Because FindPath$^+$ computes distances between trees in DCT_m the obvious first question is if it can be modified to be used for unlabelled trees in uDCT_m. Therefore, we first show that we can label two unlabelled trees so that the unlabelled uDCT_m distance between those trees equals their labelled DCT_m distance.

Theorem 6.1. *Let T an R be unlabelled trees connected by a path p in* uDCT_m*.*

All trees on p can be labelled so that the resulting sequence p' of labelled trees is a path in DCT_m*.*

Proof. Let T and R be unlabelled trees and $p = [T = T_0, T_1, \ldots, T_{d-1}, T_d = R]$ a path between them in uDCT_m. Let furthermore T' result from randomly assigning leaf labels in $\{a_1, \ldots, a_n\}$ to T, such that T' is a discrete coalescent tree in DCT_m. We inductively construct a path p of labelled trees in DCT_m which has the desired properties.

In the beginning p' contains only one tree, $T'_0 = T'$. We iterate through $i = 1, \ldots, d$ and add in each iteration i a new tree T'_i to p'. T'_i is defined as the neighbour of T'_{i-1} that results from the performing the same move on T'_{i-1} as the one between T_{i-1} and T_i on p. The path after iteration d is $p' = [T' = T'_0, T'_1, \ldots, T'_{d-1}, T'_d]$. Since all moves on p' are equivalent to the ones on p, ignoring leaf labels of the tree T'_i results in T_i for every $i = 0, \ldots, d$. This especially implies $T'_0 = T'$ and $R' = T'_d$. Deleting leaf labels from T' and R' hence results in unlabelled trees T and R, which concludes this proof. □

With Theorem 6.1 we can show that two unlabelled trees T and R can be labelled such that the distance between T and R in uDCT_m equals the distance of their labelled counterparts in DCT_m. More precisely, among all possible labellings of T and R, the one that minimises the DCT_m distance gives the uDCT_m distance:

Corollary 6.2. *Let T and R be unlabelled trees in* uDCT_m *and $d = d_{\mathrm{uDCT}_m}(T, R)$. Then*

$$d = \min\{d_{\mathrm{DCT}_m}(T', R') |\ T' \text{ and } R' \text{ are labelled versions of } T \text{ and } R\}.$$

Proof. Let T and R be unlabelled trees in uDCT_m with distance d and

$$d' = \min\{d_{\mathrm{DCT}_m}(T', R') |\ T' \text{ and } R' \text{ are labelled versions of } T \text{ and } R\}.$$

By Theorem 6.1, there is a labelling of T and R such that a path p between T and R can be converted to a path of equal length between the labelled versions of T and R. Therefore, $d \geq d'$. So it remains to prove $d \leq d'$. By the definition of d' there is a labelling of trees T and R such that the resulting trees T' and R' are connected by a path p' with length d' in DCT_m.

Let p be the path that results from p' by ignoring the leaf labels of every tree on p'. Because moves between trees in DCT_m are defined in the same way as moves in uDCT_m, two consecutive of trees on p are connected by an NNI move, rank move, or

length move like the corresponding trees on p'. By Observation 6.1 there might be two consecutive trees on p that are the same. We now assume that duplicated trees are deleted, which results in $|p| \leq |p'|$. Note that this does not change the fact that all consecutive trees on p are connected by an edge in uDCT_m.

Because p results from deleting leaf labels from the trees on p', p is a path between the underlying unlabelled trees T and R of T' and R'. Thus p is a valid path from T to R in uDCT_m and hence $|p| \geq d$. Together with $|p| \leq |p'| = d'$, we get $d \leq |p| \leq |p'| = d'$. We can conclude $d = d'$. □

Corollary 6.2 implies that it is sufficient to find a labelling of the trees that minimises their distance in DCT_m for computing the distance between unlabelled trees in uDCT_m. The problem of how to find such a labelling however remains unsolved.

6.1.2 Approximating distances in uRNNI

In this section we discuss two different strategies for approximating the uRNNI distance between unlabelled ranked trees. Both approaches rely on first labelling the given unlabelled ranked trees and then computing the distance between their labelled versions using‘ FindPath. The difference between the two approximation algorithms is the choice of labellings for the input trees. After introducing these two approaches, increasing labelling and random labelling, we compare them in a simulation study.

Increasing labelling

We now introduce a way of labelling trees that intuitively results in labelled trees that have RNNI distance close to the uRNNI distance between the given unlabelled trees. We add leaf labels $a_1, \ldots, a_n$ to an unlabelled tree T such that the parents of the leaves $p(a_1), \ldots, p(a_n)$ have increasing ranks:

$$\mathrm{rank}(p(a_1)) \leq \mathrm{rank}(p(a_2)) \leq \ldots \leq \mathrm{rank}(p(a_n)).$$

This uniquely defines a labelled tree T'. We then say that T' results from *increasingly labelling* T.

The distance between unlabelled ranked trees T and R in uRNNI does not need to equal the distance between their increasingly labelled versions T' and R' in RNNI.

Consider for example the trees depicted in Figure 6.5:

$$T' = [\{a_1, a_2\} : 1, \{a_1, a_2, a_3\} : 2, \{a_4, a_5\} : 3, \{a_1, a_2, a_3, a_4, a_5\} : 4]$$
$$R' = [\{a_1, a_2\} : 1, \{a_3, a_4\} : 2, \{a_1, a_2, a_5\} : 3, \{a_1, a_2, a_3, a_4, a_5\} : 4]$$
$$\hat{R} = [\{a_1, a_2\} : 1, \{a_4, a_5\} : 2, \{a_1, a_2, a_3\} : 3, \{a_1, a_2, a_3, a_4, a_5\} : 4]$$

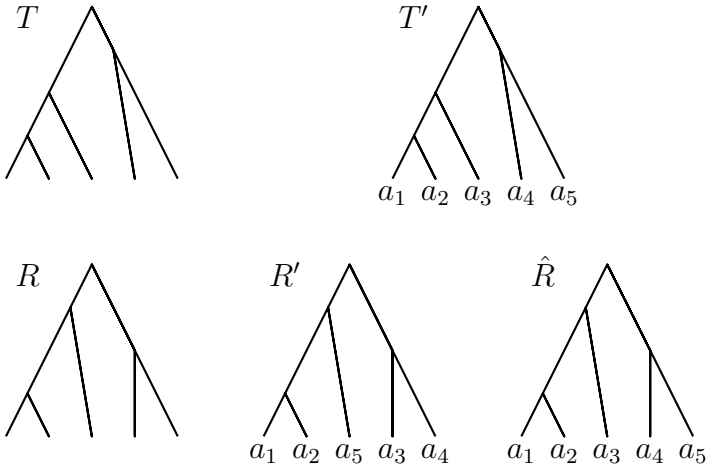


Figure 6.5: Trees T' and R' in uRNNI that result from increasingly labelling the underlying unlabelled trees T and R, and an alternative labelled version $\hat{R}$ of R.

R' and $\hat{R}$ have the same underlying unlabelled tree R. Let T be the underlying unlabelled tree of T (see Figure 6.5). T' and R' result from increasingly labelling T and R. Now consider the distances $d_{\text{RNNI}}(T', R')$ and $d_{\text{RNNI}}(T', \hat{R})$. We can compute these with FindPath and get $d_{\text{RNNI}}(T', R') = 3$ and $d_{\text{RNNI}}(T', \hat{R}) = 1$. Therefore, $d(T', \hat{R}) < d(T', R')$, i.e. the distance between unlabelled trees in uRNNI does not equal the distance between their increasingly labelled versions in RNNI. We can however use increasing labelling to approximate the uRNNI distance.

Random labelling

Our second strategy for labelling trees to use the RNNI distance between these labelled trees as approximation for uRNNI distances is random labelling, as described in the following. We label leaves of an unlabelled ranked tree randomly by iterating through the leaves of the tree and in every step assign labels randomly (at uniform) to the current leaf. For each pair of trees we repeat this M times and compute for each of the resulting M pairs of labelled trees the RNNI distance. The minimum of these M distances is then used to approximate the uRNNI distance.

Simulation

To judge the quality of the two different approaches of approximating the uRNNI distance between unlabelled ranked trees (increasing labelling and random labelling), we perform simulations. We simulate ranked trees with the coalescent process (Algorithm 1) as described in Section 3.5. Since the coalescent gives a uniform distribution on ranked trees (see discussion in Section 5.4), we uniformly sample ranked trees by this approach. In the next step we delete all leaf labels of the simulated trees to get unlabelled trees. Note that this means that we do not sample from a uniform distribution of unlabelled ranked trees. Because we are just interested in seeing whether there is a general trend that one of our approximation methods is better than the other one, we accept this shortcoming.

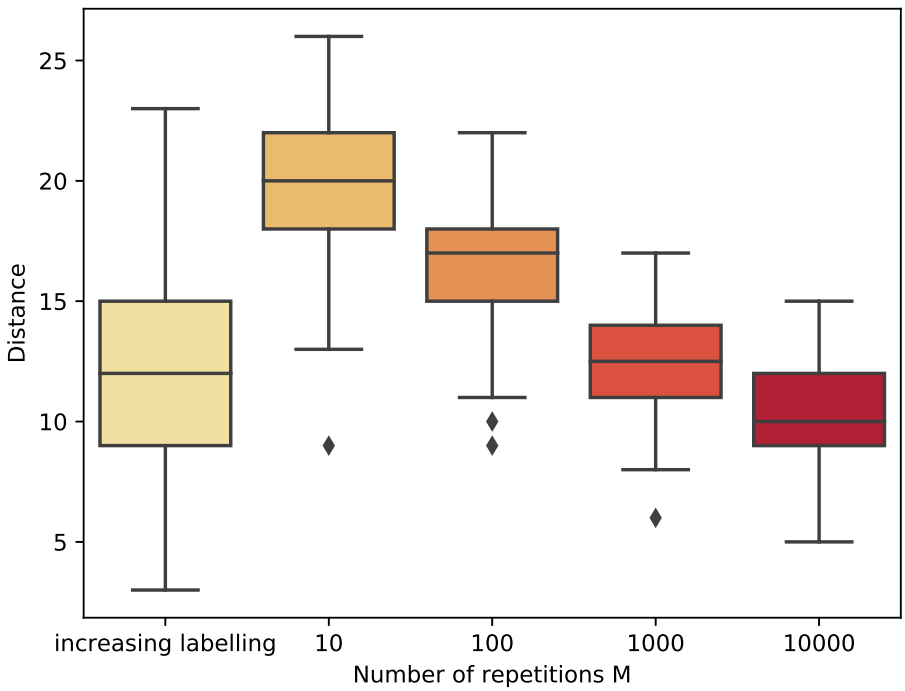


Figure 6.6: Approximated uRNNI distances for $N = 100$ trees on $n = 10$ leaves. The leftmost boxplot illustrates distances approximated by increasing labelling, the remaining ones illustrate approximations by taking the minimum over M random labellings.

We simulate $N = 100$ pairs of unlabelled ranked trees on $n = 10$ and $n = 100$ leaves as described above and compute the distance for each pair of trees using FindPath. Our results are depicted as boxplots in Figure 6.6 ($n = 10$) and Figure 6.7 ($n = 100$).

For each choice of n we compare increasing labelling to random labelling with $M = 10$, 100, 1,000, and 10,000 repetitions.

Note that we do not know the actual uRNNI distance between unlabelled ranked trees, but we know with Theorem 6.1 that it is less than or equal to our approximations. When comparing the two different approximations, the one that gives a smaller distance is hence closer to the actual distance.

In Figure 6.6 the approximated distances for unlabelled ranked trees on $n = 10$ leaves are illustrated as boxplots. For $M = 10$ and $M = 100$ the approximation of the uRNNI distance by random labelling performs worse that increasing labelling, but for $M = 10,000$, random labelling seems to outperform increasing labelling. One can see in general that with increasing M, the uRNNI distance approximated by random labelling decreases, i.e. the approximation improves. This is not surprising, as the fraction of possible labellings covered by random labelling increases with M, so the probability of finding the labelling that minimises the RNNI distance is higher for larger M.

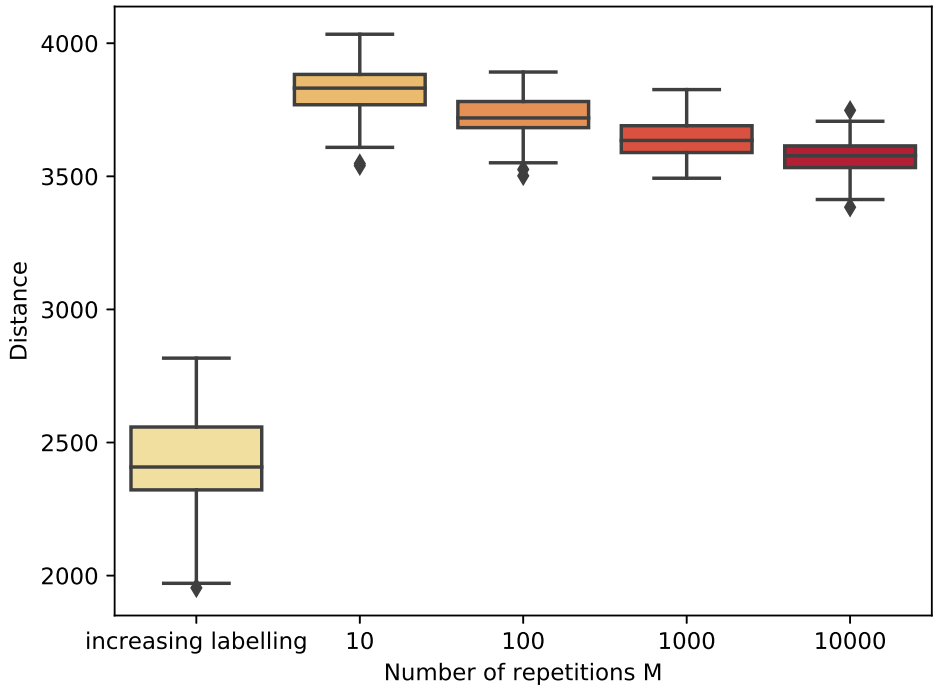


Figure 6.7: Approximated uRNNI distances for $N = 100$ trees on $n = 100$ leaves. The leftmost boxplot illustrates distances approximated by increasing labelling, the remaining ones illustrate approximations by taking the minimum over M random labellings.

This effect can also be seen for $n = 100$ (Figure 6.7). In contrast to the case $n = 10$, the distance approximated by random labelling is however significantly larger than the approximation by increasing labelling for all values of M. The reason for this difference might be that the fraction of possible labellings that are covered by $M = 10,000$ repetitions is significantly higher for $n = 10$ than for $n = 100$. The number of possible labellings for an unlabelled tree is in the order of $n!$, the number of permutations of the leaf set. For some value of M that covers a large fraction of these possible labellings, random labelling will most likely outperform increasing labelling for $n = 100$ as it does for $M = 10,000$ if $n = 10$.

Since increasing M results in an increase of runtime, our simulations indicate that it is reasonable to use increasing labelling as approximation of the uRNNI distance for large n.

6.2 The Subtree Swapping Distance

In this section we introduce a new distance measure for unlabelled trees. It is based on a tree rearrangement operation and is computable in polynomial time. We define this distance for unlabelled ranked trees.

We start by introducing a representation of unlabelled ranked trees as lists (Section 6.2.1) before we define the subtree swapping distance and show that it is a metric (Section 6.2.2). We then see in Section 6.2.3 that this distance is based on a tree rearrangement operation that can be interpreted as an unlabelled version of the subtree exchange (or subtree swap) operation as defined by Drummond, Nicholls, et al. (2002) (see also Höhna and Drummond (2012)). This operation is similar to an SPR move, but instead of moving one subtree, two subtrees are exchanged in one move. As a result, we get the tree space RSS of unlabelled ranked trees where trees are connected by an edge if they are connected by such a tree rearrangement. We furthermore discuss the diameter of this space and properties of trees with maximum subtree swapping distance (Section 6.2.4).

6.2.1 List representation of unlabelled trees

We represent an unlabelled ranked tree T by a list of multisets

$$[\{x_{2,1}, x_{2,2}\}, \{x_{3,1}, x_{3,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}]$$

where $x_{i,1}$ and $x_{i,2}$ are the ranks of the children of the node with rank i in T, noting that we only consider nodes with rank greater than one.

If a node of rank i has a child that is a leaf, then $x_{i,j} = 0$ for $j \in \{1, 2\}$. Especially, if a node of rank i has two leaves as children, then $\{x_{i,1}, x_{i,2}\} = \{0, 0\}$. There is no set representing the node of rank one in the list, as this node has two leaves as children in every tree and would hence be represented by $\{0, 0\}$ in every tree.

We call this representation the *list representation* of an unlabelled ranked tree. The list representation of the tree in Figure 6.8 is $[\{0, 0\}, \{0, 1\}, \{2, 3\}, \{0, 4\}]$.

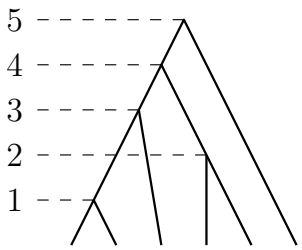


Figure 6.8: Unlabelled ranked tree $[\{0, 0\}, \{0, 1\}, \{2, 3\}, \{0, 4\}]$ on $n = 6$ leaves.

Because the list representation is a compact version of an adjacency table, trees are uniquely defined by their list representation. We hence refer to unlabelled ranked trees by their list representation, and write $T = [\{x_{2,1}, x_{2,2}\}, \{x_{3,1}, x_{3,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}]$ to refer to a tree T with this list representation. We now establish some properties of a list representing an unlabelled ranked tree:

Proposition 6.3. *A list* $[\{x_{2,1}, x_{2,2}\}, \{x_{3,1}, x_{3,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}]$ *representing an unlabelled ranked tree has the following properties:*

- $x_{i,j} \in \{0, 1, \ldots, n-2\}$ *for all* $i \in \{2, \ldots, n-1\}$ *and* $j \in \{1, 2\}$,
- *for every* $k \in \{1, \ldots, n-2\}$ *there is exactly one pair* (i, j) *with* $i \in \{2, \ldots, n-1\}$ *and* $j \in \{1, 2\}$ *such that* $x_{i,j} = k$, *and*
- $i > x_{i,j}$ *for all* $i \in \{2, \ldots, n-1\}$ *and* $j \in \{1, 2\}$.

Proof. $x_{i,j} \in \{0, 1, \ldots, n-2\}$ for all $i \in \{2, \ldots, n-1\}$ and $j \in \{1, 2\}$ follows from the fact that the integers $x_{i,j}$ represent ranks, which are integers in $\{0, \ldots, n-1\}$. More precisely, every $x_{i,j}$ is the rank of the child of an internal node, which implies that the rank $n-1$ of the root is not present in the list, i.e. $x_{i,j} \in \{0, 1, \ldots, n-2\}$.

Because every $x_{i,j}$ is the rank of an internal node, every integer in $\{1, \ldots, n-2\}$ appears exactly once in a list representing a tree, i.e. for every $k \in \{1, \ldots, n-2\}$ there is exactly one pair (i, j) with $i \in \{2, \ldots, n-1\}$ and $j \in \{1, 2\}$ such that $x_{i,j} = k$.

At last, $i > x_{i,j}$ for all $i \in \{2, \ldots, n-1\}$ and $j \in \{1, 2\}$ follows from the fact that every node in a tree has rank greater than its children. □

6.2.2 Defining the subtree swapping distance

We are now ready to define the subtree swapping distance, a distance measure for unlabelled ranked trees. We define this distance for the list representation of unlabelled ranked trees and then show in Theorem 6.4 that this distance is a metric.

Let T and R be unlabelled ranked trees with list representations

$$T = [\{x_{2,1}, x_{2,2}\}, \{x_{3,1}, x_{3,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}] \text{ and}$$
$$R = [\{y_{2,1}, y_{2,2}\}, \{y_{3,1}, y_{3,2}\}, \ldots, \{y_{n-1,1}, y_{n-1,2}\}].$$

The *subtree swapping* distance (or short *SS distance*) between T and R is defined as

$$d_{SS}(T, R) = \sum_{i=2}^{n-1} f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}),$$

with

$$f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}) = \begin{cases} 0, \text{ if } \{x_{i,1}, x_{i,2}\} = \{y_{i,1}, y_{i,2}\} \\ 2, \text{ if } \{x_{i,1}, x_{i,2}\} \cap \{y_{i,1}, y_{i,2}\} = \emptyset \\ 1, \text{ else} \end{cases}$$

Remember that $\{x_{i,1}, x_{i,2}\}$ and $\{y_{i,1}, y_{i,2}\}$ are multisets, because some nodes have two leaves as children, which have rank zero. $f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\})$ is one if the two sets have exactly one element in common. The function f can hence be interpreted as symmetric difference of multisets.

For example the trees T and R (Figure 6.9) with list representations

$$T = [\{0, 0\}, \{1, 2\}, \{0, 3\}, \{0, 4\}] \text{ and}$$
$$R = [\{0, 1\}, \{0, 0\}, \{2, 3\}, \{0, 4\}]$$

have distance

$$\begin{aligned} d_{SS}(T, R) &= f(\{0, 0\}, \{0, 1\}) + f(\{1, 2\}, \{0, 0\}) \\ &\quad + f(\{0, 3\}, \{2, 3\}) + f(\{0, 4\}, \{0, 4\}) \\ &= 1 + 2 + 1 + 0 \\ &= 4. \end{aligned}$$

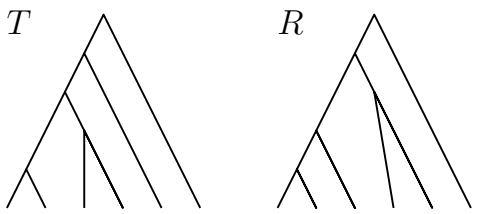


Figure 6.9: Two unlabelled ranked trees T and R with subtree swapping distance $d_{SS}(T, R) = 4$.

We now show that the subtree swapping distance is a metric.

Theorem 6.4. *The subtree swapping distance is a distance metric.*

Proof. To prove that d_{SS} is a metric, it is sufficient to show that f is a metric, as the sum of metrics is a metric. We therefore show (i) the identity of indiscernibles,(ii) symmetry, and (iii) the triangle inequality. Let $X = \{x_{i,1}, x_{i,2}\}$, $Y = \{y_{i,1}, y_{i,2}\}$, and $Z = \{z_{i,1}, z_{i,2}\}$.

(i) $f(X, Y) = 0 \Rightarrow X = Y$ follows from the definition of f.

(ii) $f(X, Y) = f(Y, X)$

This is trivially true if $f(X, Y) \in \{0, 2\}$. If $f(X, Y) = 1$, then there is one element in X that is in Y as well, while the other two elements of these sets are distinct. This implies $f(Y, X) = 1$.

(iii) $f(X, Z) \leq f(X, Y) + f(Y, Z)$

This is trivially true for $f(X, Z) = 0$.

If $f(X, Z) = 1$, then there is one element in X that is not in Z. Since $f(X, Y) = 0$ and $f(Y, Z) = 0$ would imply $X = Y = Z$ and hence $f(X, Z) = 0$, it must be $f(X, Y) + f(Y, Z) > 0$. Hence $f(X, Z) \leq f(X, Y) + f(Y, Z)$

If $f(X, Z) = 2$, we distinguish cases $f(X, Y) = 0$, $f(X, Y) = 1$, and $f(X, Y) = 2$. If $f(X, Y) = 0$, then $X = Y$ and we can infer $f(X, Z) = f(Y, Z) = 2$, i.e. $f(X, Y) + f(Y, Z) = f(X, Z)$.
If $f(X, Y) = 1$, then there is one element in Y that is also in X. And with $f(X, Z) = 2$ we know that this element cannot be in Z. Hence $f(Y, Z) \geq 1$, and it follows $f(X, Y) + f(Y, Z) \geq f(X, Z)$.
If $f(X, Y) = 2$, the triangle inequality follows trivially as f only takes values greater than or equal to zero and therefore $f(X, Y) + f(Y, Z) \geq f(X, Z)$.

□

6.2.3 The tree space RSS of unlabelled ranked trees

After establishing that the subtree swapping distance is a distance metric, we are now interested in investigating its properties. We therefore first discuss the minimum value of this distance between unlabelled ranked trees. Afterwards, we show how the subtree swapping distance between unlabelled ranked trees corresponds to tree rearrangement operations, and define the tree space RSS based on these rearrangements.

Proposition 6.5. *The minimum value of the subtree swapping distance for unlabelled trees on $n \geq 4$ leaves is two, i.e.*

$$\min_{T,R} d_{SS}(T,R) = 2.$$

Proof. Let T and R be the unlabelled ranked trees with following list representations:

$$T = [\{x_{2,1}, x_{2,2}\}, \{x_{3,1}, x_{3,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}]$$
$$R = [\{y_{2,1}, y_{2,2}\}, \{y_{3,1}, y_{3,2}\}, \ldots, \{y_{n-1,1}, y_{n-1,2}\}]$$

We first show that if there is an $i \in \{2, \ldots, n-1\}$ with $f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}) = 1$, then $d_{SS}(T,R) \geq 2$. This especially implies that there are no unlabelled ranked trees T and R with distance $d_{SS}(T,R) = 1$. We then provide an example of two unlabelled ranked trees with distance two, which proves that this is the minimum distance indeed.

Let $f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}) = 1$ for some $i \in \{2, \ldots, n-1\}$. Then $x_{i,j} = y_{i,k}$ and $x_{i,l} \neq y_{i,m}$ for $j, l, k, m \in \{1,2\}$ with $\{j,l\} = \{k,m\} = \{1,2\}$. At least one of $x_{i,l}$ and $y_{i,m}$ is not 0, as otherwise $x_{i,l} = y_{i,m}$. We assume without loss of generality that $x_{i,l}$ is the non-zero element, as the argument works the same way for $y_{i,m} = 0$. It follows $x_{i,l} \neq y_{i,k}$, as $x_{i,l} \neq 0$ and all non-zero elements appear exactly once in the list representation of a tree (see Proposition 6.3). There must hence be a set $\{y_{p,1}, y_{p,2}\}$ with $p \neq i$ and $x_{i,l} \in \{y_{p,1}, y_{p,2}\}$, as $x_{i,l} \in \{1, \ldots, n-2\}$ must also appear exactly once in R. This implies $\{x_{p,1}, x_{p,2}\} \neq \{y_{p,1}, y_{p,2}\}$, and hence $f(\{x_{p,1}, x_{p,2}\}, \{y_{p,1}, y_{p,2}\}) > 0$. Since f only takes values in $\{0,1,2\}$, we can infer $f(\{x_{p,1}, x_{p,2}\}, \{y_{p,1}, y_{p,2}\}) \geq 1$.

Together with $f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}) = 1$ it follows $d_{SS}(T,R) \geq 2$. We now provide an example of unlabelled ranked trees T and R on $n \geq 4$ leaves with distance two (Figure 6.10):

$$T = [\{0,1\}, \{0,2\}, \{0,3\}, \{0,4\}, \ldots, \{0,n-1\}]$$
$$R = [\{0,0\}, \{1,2\}, \{0,3\}, \{0,4\}, \ldots, \{0,n-1\}]$$

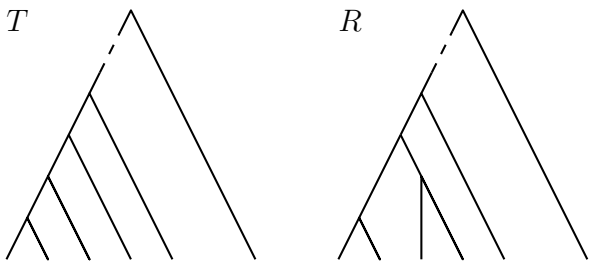


Figure 6.10: Example of unlabelled ranked trees T and R on $n \geq 4$ leaves with $d_{SS}(T, R) = 2$. The dashed parts of the trees contain further nodes in a caterpillar-like structure, i.e. every internal node has the internal node with rank one less and one leaf as child.

The sets at position i of the list representation are identical between T and R for all $i > 2$. The value of the function f is hence 0 for these i. Therefore, the distance between T and R is

$$d_{SS}(T, R) = f(\{0, 1\}, \{0, 0\}) + f(\{0, 2\}, \{1, 2\}) = 2.$$

We can conclude that the minimum subtree swapping distance $\min_{T,R} d_{SS}(T, R)$ is two. □

We now consider the difference between unlabelled ranked trees at distance two. Two unlabelled ranked trees have distance two if in their list representations two non-equal integers are swapped. Note that this implies that one of these elements needs to be different from zero. Consider the unlabelled ranked tree T with list representation

$$T = [\{x_{2,1}, x_{2,2}\}, \ldots, \{x_{i,1}, x_{i,2}\}, \ldots, \{x_{j,1}, x_{j,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}].$$

The following tree R results from swapping $x_{i,1}$ and $x_{j,1}$ for $i, j \in \{2, \ldots, n-1\}$ with $i < j$, and therefore has distance two to T:

$$R = [\{x_{2,1}, x_{2,2}\}, \ldots, \{x_{j,1}, x_{i,2}\}, \ldots, \{x_{i,1}, x_{j,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}].$$

Let us consider the difference between these unlabelled ranked trees T and R. They coincide except for the following: The node of rank i in T has children with ranks $x_{i,1}$ and $x_{i,2}$ while the node of rank i in R has children with ranks $x_{j,1}$ and $x_{i,2}$. Similarly for rank j, T has a node with children of ranks $x_{j,1}$ and $x_{j,2}$ while the node of rank j in R has children with ranks $x_{i,1}$ and $x_{j,2}$. This move can hence seen as cutting two edges in T and replacing them with new edges: We delete the edge between the node of rank $x_{i,1}$ and its parent with rank i as well as the edge connecting the nodes with

ranks $x_{j,1}$ and j. Then two new edges are added, where one connects the nodes with rank $x_{i,1}$ and j and the other one connects the nodes with ranks $x_{j,1}$ and i. We call this move *subtree swap*, as it swaps the two subtrees rooted in $x_{i,1}$ and $x_{j,1}$. An illustration of this move can be found in Figure 6.11. This move is similar to an SPR move, but instead of just moving one subtree to a new position as in SPR, two subtrees are cut and their positions are exchanged. Every NNI move is therefore a subtree swap.

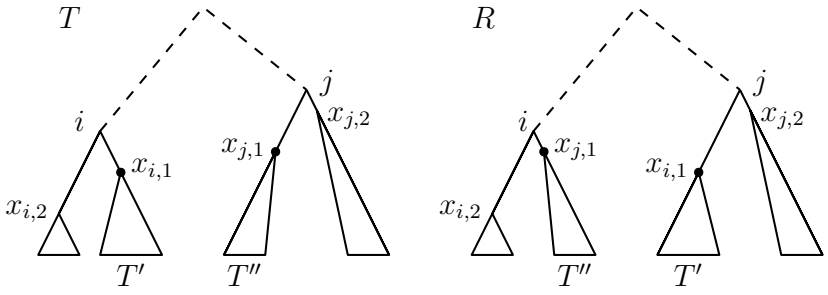


Figure 6.11: Subtree swap between unlabelled ranked trees T and R. The subtrees T' and T'' that are swapped by this move are rooted in the nodes with ranks $x_{i,1}$ and $x_{j,1}$. Dashed lines indicate the rest of the tree that may contain further nodes and is identical in T and R.

Swapping two subtrees in an unlabelled ranked tree T hence results in a tree with distance two from T. We can also swap three subtrees of T, and receive a tree with distance three from T (Figure 6.12).

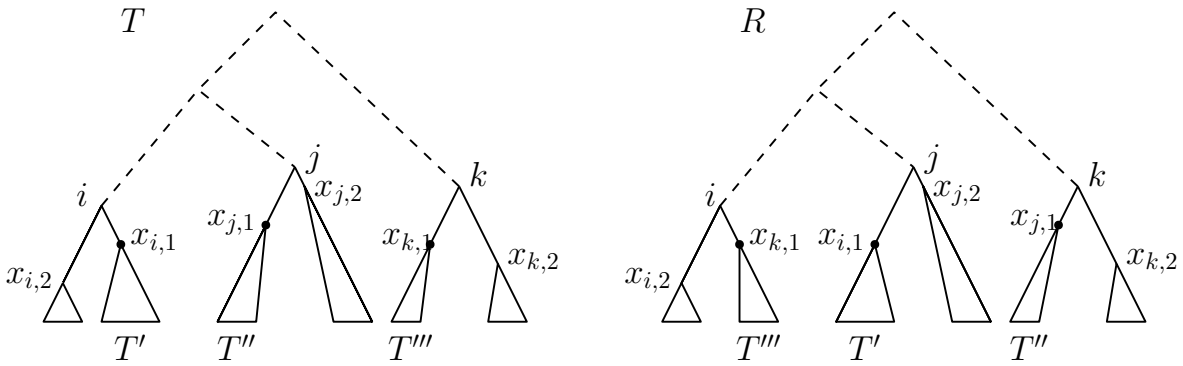


Figure 6.12: 3-subtree swap between unlabelled ranked trees T and R. The subtrees T', T'', and T''' that are swapped by this move are rooted in the nodes with ranks $x_{i,1}$, $x_{j,1}$, and $x_{k,1}$. Dashed lines indicate the rest of the tree that may contain further nodes and is identical in T and R.

With 'swapping three subtrees' we mean that three edges between nodes of ranks $x_{i,1}$, $x_{j,1}$, and $x_{k,1}$ and their parents are cut and the nodes of rank $x_{i,1}$, $x_{j,1}$, and $x_{k,1}$

then get reconnected to one of the nodes that previously has been parent of one of the other two nodes. Note that this only results in a tree with distance three if no two of the nodes of rank $x_{i,1}$, $x_{j,1}$, and $x_{k,1}$ share a parent. Furthermore, only one element in $\{x_{i,1}, x_{j,1}, x_{k,1}\}$ can be zero, and hence correspond to a leaf, to receive a tree at distance three. We call such swaps of three subtrees that result in a tree with subtree swap distance three a 3-*subtree swap*.

An example of a 3-subtree swap is shown in Figure 6.13. First, the edges connecting nodes v_0, v_1, and v_2 with their parents p_0,p_1, and p_2, respectively, are cut. To reconnect the tree, edges between v_0 and p_1, v_1 and p_2, and v_2 and p_0 are introduced. Every node in $\{v_0, v_1, v_2\}$ has therefore a different parent in R than in T, and only one of these nodes is a leaf. The subtree swapping distance between T and R is hence three.

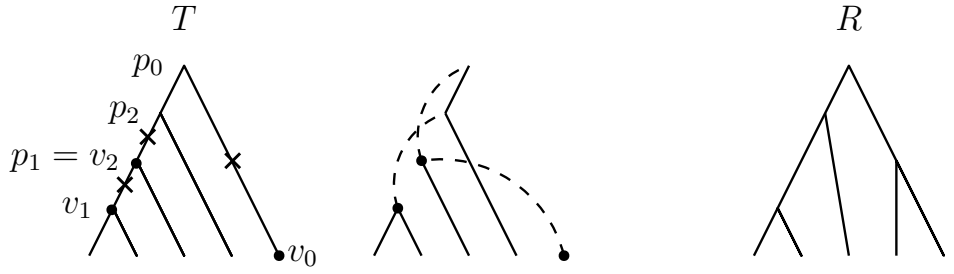


Figure 6.13: 3-subtree swap between unlabelled ranked trees T (left) and R (right). Crosses on T indicate the edges that are deleted, resulting in the graph in the middle. The dashed lines in the graph in the middle indicate where the highlighted nodes get re-attached.

$d_{SS}(T, R) = 3$ can be seen even easier when comparing the list representation of T and R:

$$T = [\{0,1\}, \{0,2\}, \{0,3\}]$$
$$R = [\{0,0\}, \{0,1\}, \{2,3\}]$$

Then

$$d_{SS}(T, R) = f(\{0,1\}, \{0,0\}) + f(\{0,2\}, \{0,1\}) + f(\{0,3\}, \{2,3\}) = 1 + 1 + 1 = 3.$$

Note that the number of subtrees that swap position can be larger then 3. If w subtrees swap and result in a tree with distance w, we call this subtree swap a *w-subtree swap*. We can define a tree space based on these subtree swaps as follows.

The *Ranked Subtree Swap space* (RSS *space*) has the set of unlabelled ranked trees as vertex set. Two of those trees are connected by an edge with weight w if they are connected by a w-subtree swap.

Every pair of unlabelled ranked trees in RSS is therefore connected by an edge, as they can be connected by cutting at most all $2n-1$ edges and reattaching all nodes, which is a $2n-1$-subtree swap. In the next section we show that this upper bound for the distance of two trees is not tight by establishing the actual diameter of the tree space RSS.

6.2.4 Diameter of RSS

We have previously seen in Proposition 6.5 that the minimum value for the subtree swapping distance is two. We now want to find the maximum value for this distance, and hence the diameter of the RSS space. The subtree swapping distance for two unlabelled ranked trees T and R in their labelled list representations

$$T = [\{x_{2,1}, x_{2,2}\}, \{x_{3,1}, x_{3,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}]$$
$$R = [\{y_{2,1}, y_{2,2}\}, \{y_{3,1}, y_{3,2}\}, \ldots, \{y_{n-1,1}, y_{n-1,2}\}]$$

is defined as $d_{SS}(T,R) = \sum_{i=2}^{n-1} f(\{x_{i,1}x_{i,2}\}, \{y_{i,1}, y_{i,2}\})$. The maximum of the function f is reached when $\{x_{i,1}x_{i,2}\} \cap \{y_{i,1}, y_{i,2}\} \neq \emptyset$.

Let $T = [\{x_{2,1}, x_{2,2}\}, \{x_{3,1}, x_{3,2}\}, \ldots, \{x_{n-1,1}, x_{n-1,2}\}]$ be an unlabelled ranked tree. With the restriction $i > x_{i,j}$ for all $i \in \{2, \ldots, n-1\}$ and $j \in \{1,2\}$ (Proposition 6.3), the only integers that can be in the first set $\{x_{2,1}, x_{2,2}\}$ of the list representation are 0 and 1. This implies that the first set of every list representation of an unlabelled ranked tree contains at least one 0.

Now consider the integer $n-2$ in the list representation of T. We know from Proposition 6.3 that there is an $x_{l,j}$ for some $l \in \{2, \ldots, n-1\}$ with $x_{l,j} = n-2$. Since $i > x_{i,j}$ for all $i \in \{2, \ldots, n-1\}$ and $j \in \{1,2\}$ (Proposition 6.3), it must be $l > x_{l,j} = n-2$, which implies $l = n-1$, as this is the maximum index in the list representation of T. The element $n-2$ is hence in the last set of the list representation of any unlabelled ranked tree with n leaves.

We conclude that every unlabelled ranked tree T has 0 in the first set of its list representation and $n-2$ in the last set of its list representation. We now show that there are trees T and R for which these are the only two elements that are in the same position in their lists representations.

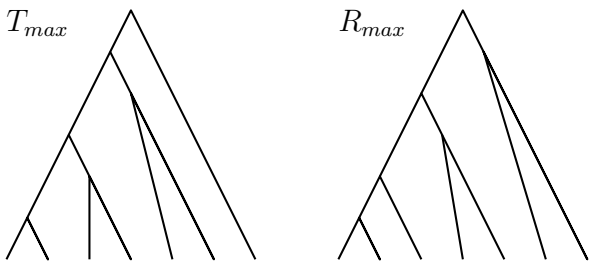


Figure 6.14: Unlabelled ranked trees T and R on $n = 7$ leaves with maximum subtree swapping distance $d_{SS}(T, R) = 2(n - 3) = 8$.

For even n let

$$T_{max} = [\{0,0\}, \{1,2\}, \{0,0\}, \{3,4\}, \ldots, \{0,0\}, \{n-3, n-2\}]$$
$$R_{max} = [\{0,1\}, \{0,0\}, \{2,3\}, \{0,0\}, \ldots, \{n-4, n-3\}, \{0, n-2\}]$$

and for odd n let

$$T_{max} = [\{0,0\}, \{1,2\}, \{0,0\}, \{3,4\}, \ldots, \{n-4, n-3\}, \{0, n-2\}]$$
$$R_{max} = [\{0,1\}, \{0,0\}, \{2,3\}, \{0,0\}, \ldots, \{0,0\}, \{n-3, n-2\}].$$

An example of these trees for $n = 7$ is depicted in Figure 6.14. Let $\{x_{i,1}, x_{i,2}\}$ and $\{y_{i,1}, y_{i,2}\}$ be the sets in the list representation of T_{max} and R_{max}, respectively, for $i \in \{2, \ldots, n-1\}$. Then $f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}) = 2$ for all $i \in \{3, \ldots, n-2\}$. Only for the first and last set the function f has value one: $f(\{0,0\}, \{0,1\}) = 1$ and $f(\{0,n\}, \{n-1,n\}) = 1$. We can conclude that T and R have maximum subtree swapping distance from each other:

Corollary 6.6. *The maximum subtree swapping distance of unlabelled ranked trees is* $2(n-3)$*, i.e.* $\max_{T,R} d_{SS}(T, R) = 2(n-3)$.

Proof. Consider the trees T_{max} and R_{max} given above. We have already seen that they maximise the subtree swapping distance. The distance between these trees is $d_{SS}(T_{max}, R_{max}) = 1 + 2(n-4) + 1 = 2(n-3)$, as $f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}) = 2$ for all $i \in \{3, \ldots, n-2\}$ and $f(\{x_{i,1}, x_{i,2}\}, \{y_{i,1}, y_{i,2}\}) = 1$ for $i \in \{2, n-1\}$. □

The unlabelled ranked trees in Figure 6.14 that are at maximum subtree swapping distance actually look quite similar. Indeed, the unlabelled trees resulting from ignoring

the ranking in those trees are connected by an SPR move. The trees that one would expect to give maximum distance are caterpillar tree and maximum balanced tree. On $n = 8$ leaves, these are the trees (Figure 6.15):

$$T_{bal} = [\{0,1\},\{0,2\},\{0,3\},\{0,4\},\{0,5\},\{0,6\}]$$
$$R_{cat} = [\{0,0\},\{0,0\},\{0,0\},\{1,3\},\{2,4\},\{5,6\}]$$

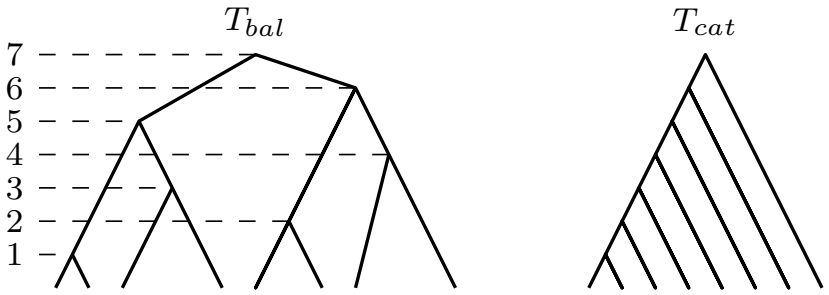


Figure 6.15: Unlabelled ranked trees T_{bal} and R_{cat} on $n = 8$ leaves. T_{bal} is a maximum balanced tree while R_{cat} is a caterpillar tree.

Balance indices, which are often used to compare unlabelled trees (see discussion at the beginning of this chapter), are defined so that these two trees have maximum distance. The subtree swapping distance between the caterpillar tree and the maximum balanced tree in Figure 6.15 however is 8, while the maximum distance for trees on $n = 8$ leaves is 10.

When using the subtree swapping distance one should bear in mind that this distance is based on the subtree swap tree rearrangement, which has some implications for trees at maximum distance as described above.

Chapter 7

Conclusion

In this thesis we explore new tree spaces and metrics defined on those, presenting new and efficient algorithms and proving fundamental properties. We discretise time trees and a define tree space DCT_m on the resulting discrete coalescent trees. As a special case of discrete coalescent trees we considered ranked trees and the corresponding tree space RNNI. After introducing these tree spaces in Chapter 3, we considered the Shortest Path Problem in DCT_m in Chapter 4. The main result there is the algorithm FindPath, which we first introduced for RNNI (Section 4.1.1). We prove that this algorithm computes shortest paths between ranked trees in RNNI (Section 4.1.2) in time polynomial in the number of leaves. This results in particular implies that the tree rearrangement based RNNI distance can be computed efficiently. The importance of this finding is highlighted by the fact that no other tree rearrangement based distance is known that is not $\mathcal{NP}$-hard to compute. For example NNI, SPR, and TBR distance are $\mathcal{NP}$-hard to compute. For the NNI space, which is closely related to the RNNI space, it took decades to prove that the Shortest Path Problem is $\mathcal{NP}$-hard.

We generalised the algorithm FindPath from the RNNI space to FindPath$^+$, which computes shortest paths in all tree spaces DCT_m (independent of m, Section 4.2). As a result, we can compute distances for time trees by first discretising them to discrete coalescent trees and applying FindPath$^+$. Note that the degree of how coarse or fine this discretisation is depends on m. For $m = n - 1$, time trees are discretised to ranked trees, which is the coarsest option. With increasing m the discretisation gets finer. Since the computational complexity of computing distances depends on m, the question on the ideal choice of m depends on the application.

Our findings that DCT_m is a tree space that is tree rearrangement based and has shortest paths that can be computed efficiently motivates us to explore these tree

spaces further in Chapter 5. There we establish that the DCT_m space has the cluster property, which means that information that is shared between two trees is shared by all trees on a shortest path between these trees. This is a desirable property of shortest paths as it is biologically reasonable and indicates that DCT_m might be suitable for developing statistical methods such as computations of mean trees that contain relevant information. We moreover show that a special subset of trees, caterpillar trees, build a convex set in DCT_m, i.e. shortest paths between those trees stay within this set (Section 5.2.2). With this result we establish a formula for calculating the distances between caterpillar trees that can be computed more efficiently than using FindPath (Section 5.2.3). This also leads to the conjecture that $\mathcal{O}(n\sqrt{\log n})$ is a lower bound for the computational complexity of computing distances in RNNI (Conjecture 1). We furthermore discuss diameter and radius of the DCT_m graph in Section 5.3. In the end of Chapter 5 we consider the uniform distribution on ranked trees and discuss distances between trees sampled from that distribution. Our results suggest that the DCT_m space might be a reasonable basis for developing statistical methods for trees.

In Chapter 6 we consider unlabelled trees. The tree rearrangement operations that define DCT_m can also be used on unlabelled trees to create a new tree space uDCT_m of unlabelled trees. We show that given two unlabelled trees, there is a labelling such that the distance between the labelled trees in DCT_m equals the distance between the unlabelled trees in uDCT_m. On this basis we introduce and compare two ways to approximate the uDCT_m distance. If the Shortest Path Problem for unlabelled trees in uDCT_m is $\mathcal{NP}$-hard however remains an open problem. We finish our discussion on unlabelled trees in Section 6.2 by introducing a different tree rearrangement based distance measure for unlabelled ranked trees. This distance can be computed in time linear in the number of leaves, and is hence to our knowledge the first rearrangement distance measure for unlabelled trees that is efficiently computable.

Bibliography

Aldous, D (1996). "Probability Distributions on Cladograms". *Random Discrete Structures*. Ed. by D Aldous and R Pemantle. The IMA Volumes in Mathematics and its Applications. New York, NY: Springer, pp. 1–18. ISBN: 978-1-4612-0719-1. DOI: 10.1007/978-1-4612-0719-1_1.

Allen, BL and M Steel (2001). "Subtree Transfer Operations and Their Induced Metrics on Evolutionary Trees". en. *Ann. Comb.* 5.1, pp. 1–15.

Alves, JM, S Prado-Lopez, JM Cameselle-Teijeiro, and D Posada (2019). "Rapid evolution and biogeographic spread in a colorectal cancer". *Nature communications* 10.1, pp. 1–7.

Billera, LJ, SP Holmes, and K Vogtmann (2001). "Geometry of the Space of Phylogenetic Trees". *Adv. Appl. Math.* 27.4, pp. 733–767.

Bordewich, M and C Semple (2005). "On the Computational Complexity of the Rooted Subtree Prune and Regraft Distance". *Ann. Comb.* 8.4, pp. 409–423.

Bouckaert, R et al. (2014). "BEAST 2: a software platform for Bayesian evolutionary analysis". *PLoS Comput. Biol.* 10.4, e1003537.

Chan, TM and M Pătraşcu (2010). "Counting inversions, offline orthogonal range counting, and related problems". *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, pp. 161–173.

Colijn, C and G Plazzotta (2018). "A Metric on Phylogenetic Tree Shapes". *Systematic Biology* 67.1, pp. 113–126. ISSN: 1063-5157. DOI: 10.1093/sysbio/syx046.

Colless, DH (1982). "Review of Phylogenetics: The Theory and Practice of Phylogenetic Systematics." *Systematic Zoology* 31.1. In collab. with EO Wiley. Publisher: [Oxford University Press, Society of Systematic Biologists, Taylor & Francis, Ltd.], pp. 100–104. ISSN: 0039-7989. DOI: 10.2307/2413420.

Collienne, L and L Berling (2021). *treeOclock*. https://github.com/bioDS/treeOclock.

Collienne, L, K Elmes, M Fischer, D Bryant, and A Gavryushkin (2019). "Geometry of Ranked Nearest Neighbour Interchange Space of Phylogenetic Trees". en. *bioRxiv*. DOI: 10.1101/2019.12.19.883603.

Collienne, L, K Elmes, M Fischer, D Bryant, and A Gavryushkin (2021). "Discrete coalescent trees". *Journal of Mathematical Biology* 83.5, p. 60. ISSN: 1432-1416. DOI: 10.1007/s00285-021-01685-0.

Collienne, L and A Gavryushkin (2021). "Computing nearest neighbour interchange distances between ranked phylogenetic trees". *Journal of Mathematical Biology* 82.1, p. 8. ISSN: 1432-1416. DOI: 10.1007/s00285-021-01567-5.

DasGupta, B, X He, T Jiang, M Li, and J Tromp (1999). "On the Linear-Cost Subtree-Transfer Distance between Phylogenetic Trees". *Algorithmica* 25.2, pp. 176–195.

Dasgupta, B et al. (2000). "On computing the nearest neighbor interchange distance". *Discrete Mathematical Problems with Medical Applications: DIMACS Workshop Discrete Mathematical Problems with Medical Applications, December 8-10, 1999, DIMACS Center*. Vol. 55. American Mathematical Soc., p. 19.

Downey, RG and MR Fellows (2013). *Fundamentals of Parameterized Complexity*. Springer, London.

Drummond, AJ, A Rambaut, B Shapiro, and OG Pybus (2005). "Bayesian coalescent inference of past population dynamics from molecular sequences". *Mol. Biol. Evol.* 22.5, pp. 1185–1192.

Drummond, AJ, GK Nicholls, AG Rodrigo, and W Solomon (2002). "Estimating Mutation Parameters, Population History and Genealogy Simultaneously From Temporally Spaced Sequence Data". *Genetics* 161.3, pp. 1307–1320. ISSN: 1943-2631. DOI: 10.1093/genetics/161.3.1307.

Drummond, AJ, MA Suchard, D Xie, and A Rambaut (2012). "Bayesian Phylogenetics with BEAUti and the BEAST 1.7". *Molecular Biology and Evolution* 29.8, pp. 1969–1973. ISSN: 0737-4038. DOI: 10.1093/molbev/mss075.

Felsenstein, J (1985). "CONFIDENCE LIMITS ON PHYLOGENIES: AN APPROACH USING THE BOOTSTRAP". *Evolution* 39.4, pp. 783–791. DOI: https://doi.org/10.1111/j.1558-5646.1985.tb00420.x.

Frost, SDW and EM Volz (2013). "Modelling tree shape and structure in viral phylodynamics". *Philosophical Transactions of the Royal Society B: Biological Sciences* 368.1614. Publisher: Royal Society, p. 20120208. DOI: 10.1098/rstb.2012.0208.

Gavryushkin, A and AJ Drummond (2016). "The space of ultrametric phylogenetic trees". en. *J. Theor. Biol.* 403, pp. 197–208.

Gavryushkin, A, C Whidden, and FA Matsen 4th (2018). "The combinatorics of discrete time-trees: theory and open problems". en. *J. Math. Biol.* 76.5, pp. 1101–1121.

Goloboff, PA, JS Arias, and CA Szumik (2017). "Comparing tree shapes: beyond symmetry". *Zoologica Scripta* 46.5, pp. 637–648. ISSN: 1463-6409. DOI: https://doi.org/10.1111/zsc.12231.

Hayati, M, B Shadgar, and L Chindelevitch (2019). "A new resolution function to evaluate tree shape statistics". *PLOS ONE* 14.11. Publisher: Public Library of Science, e0224197. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0224197.

Hein, J, T Jiang, L Wang, and K Zhang (1996). "On the complexity of comparing evolutionary trees". *Discrete Appl. Math.* 71.1, pp. 153–169.

Hickey, G, F Dehne, A Rau-Chaplin, and C Blouin (2008). "SPR distance computation for unrooted trees". en. *Evol. Bioinform. Online* 4, pp. 17–27.

Höhna, S and AJ Drummond (2012). "Guided Tree Topology Proposals for Bayesian Phylogenetic Inference". *Systematic Biology* 61.1, pp. 1–11. ISSN: 1076-836X, 1063-5157. DOI: 10.1093/sysbio/syr074.

Holmes, S (2003). "Statistics for phylogenetic trees". *Theoretical Population Biology* 63.1, pp. 17–32. ISSN: 0040-5809. DOI: 10.1016/S0040-5809(02)00005-9.

Hudson, RR et al. (1990). "Gene genealogies and the coalescent process". *Oxford surveys in evolutionary biology* 7.1, p. 44.

Humphries, PJ and T Wu (2013). "On the Neighborhoods of Trees". *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 10.3, pp. 721–728. ISSN: 1545-5963. DOI: 10.1109/TCBB.2013.66.

Jombart, T, M Kendall, J Almagro-Garcia, and C Colijn (2017). "treespace: Statistical exploration of landscapes of phylogenetic trees". *Molecular Ecology Resources* 17.6, pp. 1385–1392. ISSN: 1755-0998. DOI: 10.1111/1755-0998.12676.

Kim, J, NA Rosenberg, and JA Palacios (2020). "Distance metrics for ranked evolutionary trees". *Proceedings of the National Academy of Sciences* 117.46. Publisher: National Academy of Sciences Section: Biological Sciences, pp. 28876–28886. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1922851117.

Kingman, JFC (1982). "The coalescent". *Stochastic Process. Appl.* 13.3, pp. 235–248.

Knuth, DE (1997). *The art of computer programming*. Vol. 3. Pearson Education.

Kozlov, AM, D Darriba, T Flouri, B Morel, and A Stamatakis (2019). "RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference". en. *Bioinformatics* 35.21, pp. 4453–4455.

Kuhner, MK, J Yamato, and J Felsenstein (1998). "Maximum likelihood estimation of population growth rates based on the coalescent". en. *Genetics* 149.1, pp. 429–434.

Kuhner, MK (2009). "Coalescent genealogy samplers: windows into population history". en. *Trends Ecol. Evol.* 24.2, pp. 86–93.

Li, M, J Tromp, and L Zhang (1996). "Some notes on the nearest neighbour interchange distance". *Computing and Combinatorics*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 343–351.

Liu, P, P Biller, M Gould, and C Colijn (2020). *Polynomial Phylogenetic Analysis of Tree Shapes*. preprint. Systems Biology. DOI: 10.1101/2020.02.10.942367.

Lote, H et al. (2017). "Carbon dating cancer: defining the chronology of metastatic progression in colorectal cancer". en. *Ann. Oncol.* 28.6, pp. 1243–1249.

Matsen, FA (2006). "A Geometric Approach to Tree Shape Statistics". *Systematic Biology* 55.4, pp. 652–661. ISSN: 1063-5157. DOI: 10.1080/10635150600889617.

Minh, BQ, MAT Nguyen, and A von Haeseler (2013). "Ultrafast Approximation for Phylogenetic Bootstrap". *Molecular Biology and Evolution* 30.5. Publisher: Oxford Academic, pp. 1188–1195. ISSN: 0737-4038. DOI: 10.1093/molbev/mst024.

Nguyen, LT, HA Schmidt, A von Haeseler, and BQ Minh (2015). "IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies". en. *Mol. Biol. Evol.* 32.1, pp. 268–274.

Ohtsuki, H and H Innan (2017). "Forward and backward evolutionary processes and allele frequency spectrum in a cancer cell population". *Theor. Popul. Biol.*

Owen, M and JS Provan (2011). "A fast algorithm for computing geodesic distances in tree space". *IEEE/ACM Trans. Comput. Biol. Bioinform.* 8.1, pp. 2–13.

Pompei, S, V Loreto, and F Tria (2012). "Phylogenetic Properties of RNA Viruses". *PLOS ONE* 7.9. Publisher: Public Library of Science, e44849. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0044849.

Poon, AFY et al. (2013). "Mapping the Shapes of Phylogenetic Trees from Human and Zoonotic RNA Viruses". *PLOS ONE* 8.11. Publisher: Public Library of Science, e78122. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0078122.

Posada, D (2020). "CellCoal: Coalescent Simulation of Single-Cell Sequencing Samples". en. *Mol. Biol. Evol.* 37.5, pp. 1535–1542.

Robinson, DF and LR Foulds (1981). "Comparison of phylogenetic trees". *Math. Biosci.* 53.1, pp. 131–147.

Ronquist, F and JP Huelsenbeck (2003). "MrBayes 3: Bayesian phylogenetic inference under mixed models". *Bioinformatics* 19.12, pp. 1572–1574.

Sackin, MJ (1972). ""Good" and "Bad" Phenograms". *Systematic Biology* 21.2, pp. 225–226. ISSN: 1063-5157. DOI: 10.1093/sysbio/21.2.225.

Singer, J, J Kuipers, K Jahn, and N Beerenwinkel (2018). "Single-cell mutation identification via phylogenetic inference". en. *Nat. Commun.* 9.1, p. 5144.

Steel, M (2016). *Phylogeny: discrete and random processes in evolution*. SIAM.

Steel, M and A McKenzie (2001). "Properties of phylogenetic trees generated by Yule-type speciation models". *Mathematical Biosciences* 170.1, pp. 91–112. ISSN: 0025-5564. DOI: 10.1016/S0025-5564(00)00061-4.

Suchard, MA et al. (2018). "Bayesian phylogenetic and phylodynamic data integration using BEAST 1.10". en. *Virus Evol* 4.1, vey016.

Tamura, K et al. (2011). "MEGA5: molecular evolutionary genetics analysis using maximum likelihood, evolutionary distance, and maximum parsimony methods". en. *Mol. Biol. Evol.* 28.10, pp. 2731–2739.

Whidden, C, RG Beiko, and N Zeh (2010). "Fast FPT Algorithms for Computing Rooted Agreement Forests: Theory and Experiments". *Experimental Algorithms*. Lecture Notes in Computer Science, pp. 141–153.

Whidden, C and F Matsen (2018). "Calculating the Unrooted Subtree Prune-and-Regraft Distance". en. *IEEE/ACM Trans. Comput. Biol. Bioinform.*

Ypma, RJF, WM van Ballegooijen, and J Wallinga (2013). "Relating phylogenetic trees to transmission trees of infectious disease outbreaks". en. *Genetics* 195.3, pp. 1055–1062.

Zhang, X et al. (2020). "Viral and host factors related to the clinical outcome of COVID-19". *Nature* 583.7816, pp. 437–440. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2355-0.

Appendix

A List of Symbols

$\text{time}(v)_T$	time of the node v in T
$(T)_k$	node with time k in T
$[(T)_i, (T)_j]$	interval between nodes with times i and j with $i < j$ in T
$mrca(S)_T$	most recent common ancestor of the set S in T
$p(a_i)_T$	parent of the leaf a_i in T
$\|p\|$	length of a path p in a tree space, i.e. the number of trees of p minus one
$d_{\text{RNNI}}(T, R), d_{\text{DCT}_m}(T, R)$	distance of trees T and R in RNNI an DCT_m
$\text{rank}(v)$	rank of node v in ranked tree
$\text{FP}(T, R)$	path computed by FindPath between ranked trees T and R
T_r	extended ranked version of discrete coalescent tree T (Algorithm 3)
T_r^d	discrete coalescent tree T as subtree of its extended ranked version T_r
T_r^c	caterpillar tree that is added to receive extended ranked version for T (Algorithm 3)
$\text{FP}^+(T, R)$	path computed by FindPath between discrete coalescent trees T and R
$\text{CSort}(T, R)$	path computed by Caterpillar Sort between caterpillar trees T and R
$r(C)_T$	$\text{rank}(mrca(C)_T)$
$\Delta(\text{RNNI}_n)$	diameter of the RNNI space on n leaves
$d_{SS}(T, R)$	subtree swapping distance of unlabelled ranked trees T and R

B List of Tree Spaces

BHV-space	tree space defined by Billera, Holmes, and Vogtmann (2001)
τ-space	time tree space defined by Gavryushkin and Drummond (2016), using interval lengths as parameterisation
t-space	time tree space defined by Gavryushkin and Drummond (2016), using times of internal nodes as parameterisation
NNI	Nearest Neighbour Interchange
SPR	(unrooted) Subtree Prune and Regraft
rSPR	rooted Subtree Prune and Regraft
TBR	Tree Bisection and Reconnection
DCT_m	space of discrete coalescent trees with root time less than or equal to m
RNNI	Ranked Nearest Neighbour Interchange space of ranked trees
DCT_m^{nu}	space of non-ultrametric discrete coalescent trees with root time less than or equal to m
uDCT_m	space of unlabelled discrete coalescent trees
uRNNI	space of unlabelled ranked trees
RSS	Ranked Subtree Swap space of unlabelled ranked trees