MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master‘s Thesis

„Sequential Animation of Phylogenetic Trees“

verfasst von / submitted by

Enes Berk Sakalli BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 875
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Bioinformatik
Betreut von / Supervisor:	Univ.-Prof. Dr Arndt von Haeseler
Mitbetreut von / Co-Supervisor:	Dr. Heiko Schmidt

Contents

Abstract

Analysing sequences of phylogenetic trees is not an easy task. Especially the differences between neighbouring trees might be significant. Such sequences of trees could originate from by reconstructing trees for sliding windows along genomic alignments, e.g., to detect recombination events. Another source might be the sequence of trees found in a tree search during phylogenetic analysis. After obtaining such sequences of trees, it is tedious to manually analyse neighbouring pairs of trees and to detect major transitions, changes, or recombination events.

The main project focuses on building a tool for dynamic visualisation of phylogenetic trees like a media player. This player enables browsing dynamically through a sequence of trees. We called this tool Phylo-Movies.

Because it is hard to see differences between the two trees, we found ways to highlight changes in the tree topologies so that the user can easily recognise the significant changes. Without a beneficial visual transition using intermediates between neighbouring trees, it is hard to comprehend the differences between two trees because the bare eye has problems identifying them. The addition of colourisation makes it easier for the user to see where rearrangements happen. With the combination of colourisation and the intermediate steps, it is easier for the eyes to see apparent differences during the transition. The tool will be tested by analysing tree sequences generated by simulated and real data.

We discuss the technical realisation using Frontend-Backend structure and also the technologies used to develop our tool called Phylo-Movies.

Furthermore we analyse three different sequences of different origin and show the use ability of Phylo-Movies.

At the end we discuss some future extension that can be done, to further improve Phylo-Movies.

Zusammenfassung

Die Analyse der Sequenzen von phylogenetischen Bäumen ist keine leichte Aufgabe. Insbesondere die Unterschiede zwischen benachbarten Bäumen können erheblich sein. Solche Baumsequenzen könnten durch die Rekonstruktion von Bäumen für gleitende Fenster entlang genomischer Alignments entstehen, z. B. um Rekombinationsereignisse zu erkennen. Eine andere Quelle könnten die Baumsequenzen sein, die bei einer Baumsuche während einer phylogenetischen Analyse gefunden wurden. Nachdem man solche Baumsequenzen erhalten hat, ist es mühsam, benachbarte Baumpaare manuell zu analysieren und größere Übergänge, Veränderungen oder Rekombinationsereignisse zu erkennen.

Das Hauptprojekt konzentriert sich auf die Entwicklung eines Werkzeugs zur dynamischen Visualisierung von phylogenetischen Bäumen in Form eines Mediaplayers. Dieser Player ermöglicht es, dynamisch durch eine Sequenz von Bäumen zu blättern. Wir haben dieses Tool Phylo-Movies genannt.

Da es schwierig ist, Unterschiede zwischen den beiden Bäumen zu erkennen, haben wir Wege gefunden, Veränderungen in den Baumtopologien hervorzuheben, so dass der Benutzer die wesentlichen Veränderungen leicht erkennen kann. Ohne einen vorteilhaften visuellen Übergang mit Hilfe von Zwischenstufen zwischen benachbarten Bäumen ist es schwierig, die Unterschiede zwischen zwei Bäumen zu erkennen, da das bloße Auge Probleme hat, sie zu identifizieren. Durch die Einfärbung wird es für den Benutzer einfacher zu erkennen, wo Umschichtungen stattfinden. Durch die Kombination von Einfärbung und Zwischenschritten ist es für das Auge einfacher, offensichtliche Unterschiede beim Übergang zu erkennen. Das Tool wird anhand der Analyse von Baumsequenzen aus simulierten und realen Daten getestet.

Wir diskutieren die technische Umsetzung mit Hilfe der Frontend-Backend-Struktur und auch die Technologien, die zur Entwicklung unseres Tools namens Phylo-Movies verwendet wurden.

Weiterhin analysieren wir drei verschiedene Sequenzen unterschiedlicher Herkunft und zeigen die Einsatzmöglichkeiten von Phylo-Movies.

Zum Schluss werden einige zukünftige Erweiterungen diskutiert, die Phylo-Movies weiter verbessern können.

Acknowledgements

First, I want to thank my supervisor, Dr. Heiko Schmidt, for his invaluable advice, support, and patience during my Master's studies. His immense knowledge in the field of phylogenetics, editing of academic writings and overall experience motivated me in my academic research. Also, I want to thank him for the creative space to try and learn new methods.

I thank Prof. Dr. Arndt von Haeseler for taking on the role of main supervisor.

I thank Simon Haendler for his guidance and for reminding me of the foundations of mathematics. It was an essential experience for me to have a peer and friend to discuss the meaning of the tree topologies over a whole summer.

I thank Maximilian von der Linde for the tremendous initial start of the project working through the basics of the world of phylogenetics, and leaving me a good foundation for a good start for my Master-Thesis.

I thank the people at the CBIV, showing that there are other exciting projects outside of my scientific scope.

I want to mention my parents, Mustafa Sakalli and Gülgün Sakalli, who taught me, that science and art are the foundation for a happy life. I want to acknowledge my friends that always had time to listen to me discuss my findings.

I thank Andre Lüttig for reminding me of the importance of grammar and using the right words.

At last, I want to thank the researchers and artists in history for improving the lives of others and doing their best to be role models.

Chapter 1

Introduction

Experimental studies and analyses of the growing databases of viral gene sequences show that recombination occurs in various families of viruses, and recombination has a significant impact on their evolution, emergence, and epidemiology (Nora et al., 2007). Recombination has links to the increase of virulence (Khatchikian et al., 1989), escape from host immunity (Malim and Emerman, 2001) and the evolution of resistance to antivirals (Nora et al., 2007; Simon-Lorière and Holmes, 2011). Recombination forms new genomes from the parent genomes, which consists of recombinant DNA or RNA (Simon-Lorière and Holmes, 2011). Each fragment of the recombinant DNA is inherited from the respective parent genome. The location of the end of a fragment and the beginning of the next fragment is a recombinant breakpoint. Every fragment might represent a different evolutionary history, depending on the ancestor's genomes (Simon-Lorière and Holmes, 2011). Phylogenetic trees, diagrams representing evolutionary history among organisms, help us detect different ancestral histories. When analysing alignments of genomes containing recombinants, the generated phylogenetic trees for each fragment may be different and show a different ancestral history.

Fig. 1.1 demonstrates a simulated example with two recombinations. Taxon X is a recombinant of A2 and C1 with the recombination breakpoint at location 2000. Fig. 1.1.b shows that fragment ”a” (1-2000) of taxon X is similar to taxon C1 while fragment ”b”, ”c” and ”d” (2000-7000) closely resembles those of A2. The ancestor of B1 and B2 is a recombinant of D2 and an ancestor of A1 and A2. There are two recombination breakpoints at locations 3500 and 5000. Fragment ”c” (3500-5000) B1 and B2 resembles with D2, while the other two fragments ”a”, ”b” and ”d” B1 and B2 resembles with A1 and A2 (Fig. 1.1c).

The different histories of the fragments are typically depicted as phylogenetic trees as in Fig. 5.2. In Fig. 5.2a taxon X clusters with taxon C1, in the fragment ”b”, ”c” and ”d” taxon X clusters with taxon A2. Taxon X jumps from C1 to A2 in the trees. In Fig. 5.2a the subtree B1 and B2 clusters with the subtree A1 and A2. In Fig. 5.2c the subtree B1 and B2 clusters with D2. In Figs. 5.2b-5.2d the subtree with taxa B1, B2, are clustering with the subtree with taxa X, A1, A2 . The subtree with the taxa B1, B2 jumps from (A1, A2) to D2 in Figs. 5.2c-5.2d back.

While we knew the beginning and the end of the fragments of the examples Figs. 1.1-5.2, in reality, the recombination breakpoints are unknown and need to be found. To find the recombination breakpoints we can utilise sliding window approaches by dividing the alignment into overlapping subalignments and constructing phylogenetic trees from those to create a sequence of phylogenetic trees (Fig. 1.2) (Lemey et al., 2009, Chapter 16). Two consecutive phylogenetic trees that differ may indicate a recombination breakpoint, because of two different consecutive phylogenetic trees which are generated by two consecutive

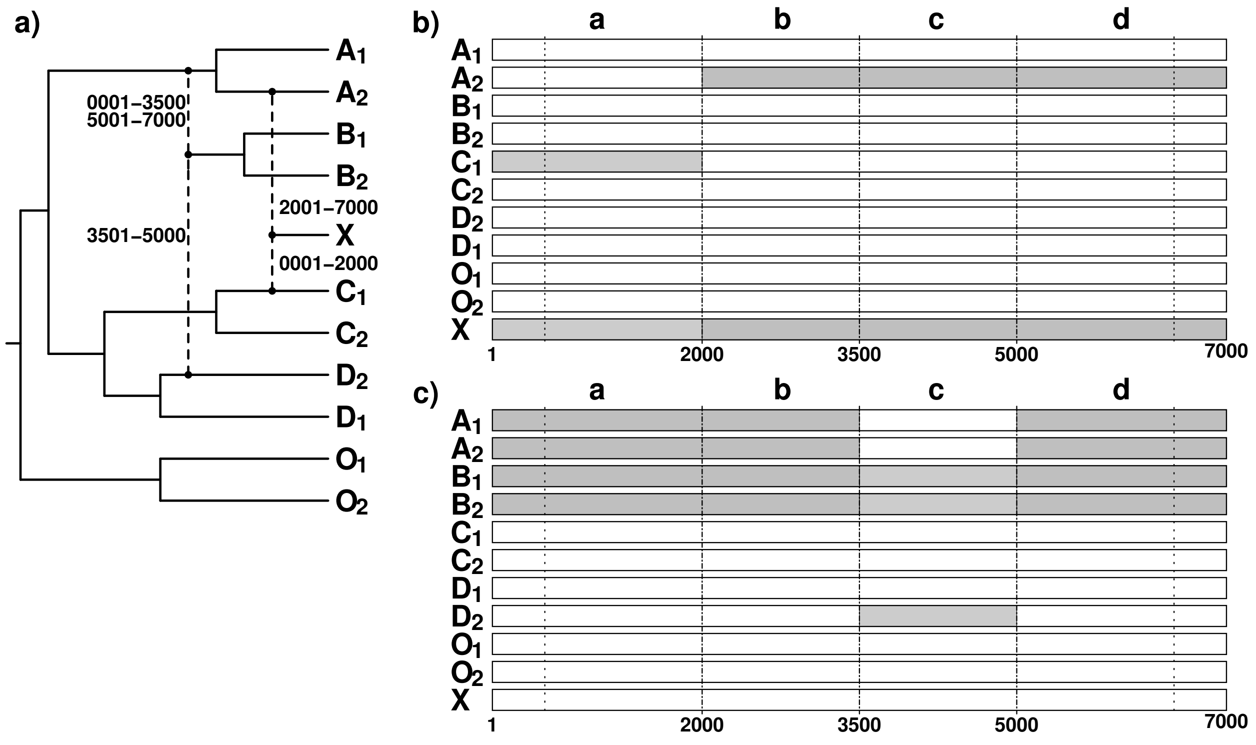


Figure 1.1: Visualisation of the relation of a simulated data set with genomes A1 - X. a): Tree of the alignment with markings where changes happen (dotted lines). b) Alignment of the genomes, grey fields show the relation of X to other genomes. c) Alignment of the genomes, grey fields show the relation of B1 and B2 to other genomes. *A1 and A2 have switched patterns. (courtesy of H.A.Schmidt)

subalignments with different ancestral histories (see Fig. 1.2). However, not all differences in phylogenetic trees need to be a consequence of recombination. The sliding window approaches may result in sequences of trees. This makes a manual comparison of changes of all consecutive trees infeasible.

Visualisation is a well-known and useful tool for converting data into comprehensible graphics and thus helping users to make sense of the original data sources or subsets of data obtained by analysis (Heer and G. Robertson, 2007). The visualisation itself should be simple and provide enough information to see the proper display of movement and change. We assume that looking at two static pictures of two different phylogenetic trees is more demanding than focusing on a transition from the previous tree to the current tree (Heer and G. Robertson, 2007; Tversky et al., 2002). Research has found that animated transitions help viewers to be oriented (G. G. Robertson et al., 1991), support learning (Bederson and Boltman, 1999) and decision making (Gonzales, 1996; Heer and G. Robertson, 2007).

Here we introduce Phylo-Movies, a bioinformatics tool that visualises differences in phylogenetic tree sequences by animation. An influence for the development of Phylo-Movies was a similar visualisation tool that has been successfully implemented and used in the field of RNA structure comparison named RNA-Movies (Kaiser et al., 2007). In the previous example (Fig. 1.1- 5.2), we have seen that manual analysis of trees changed by jumping taxa is very tedious and gets even worse the more taxa are present. With Phylo-Movies, this analysis is effortless and not even limited to detecting recombinations. Sequences of changing trees occur in different contexts, as in heuristic tree search. While searching for optimal trees, by methods for improving them as with nearest neighbour interchange, subtree pruning and regrafting or tree bisection and reconnection, sequences of trees are generated (Felsenstein, 2004, Chapter 4). Phylo-Movies can also

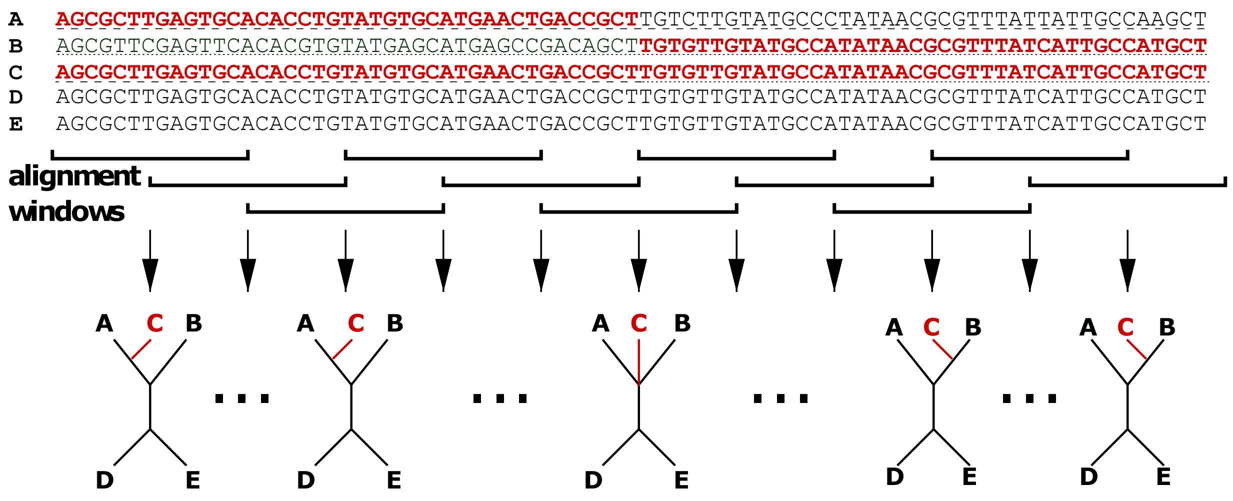

A AGCGCTTGAGTGCACACCTGTATGTGCATGAACTGACCGCTTGTCTTGTATGCCCTATAACGCGTTTATTATTGCCAAGCT
B AGCGTTCGAGTTCACACGTGTATGAGCATGAGCCGACAGCTTGTGTTGTATGCCATATAACGCGTTTATCATTGCCATGCT
C AGCGCTTGAGTGCACACCTGTATGTGCATGAACTGACCGCTTGTGTTGTATGCCATATAACGCGTTTATCATTGCCATGCT
D AGCGCTTGAGTGCACACCTGTATGTGCATGAACTGACCGCTTGTGTTGTATGCCATATAACGCGTTTATCATTGCCATGCT
E AGCGCTTGAGTGCACACCTGTATGTGCATGAACTGACCGCTTGTGTTGTATGCCATATAACGCGTTTATCATTGCCATGCT
alignment windows
A C B D E

Figure 1.2: A example for (different from Figs. 1.1-5.2). The brackets represent the sliding window (each bracket is one window). From every window one tree gets formed.(courtesy of H.A.Schmidt)

identify and visualise differences between this sequence of trees.

The following section of this introduction explains the concepts involved in detail, such as recombination, the definition of trees and data structures that depict phylogenetic trees and the tree space (Felsenstein, 2004, Chapter 4). Subsequently Chapter 2 explains how to generate animations from sequences of trees. Chapter 3 will describe algorithms we developed to identify jumping taxa. Chapter 4 is dedicated to the technical implementation. In Chapter 5 we apply Phylo-Movies to identify recombinations and changes in three different tree sequences. In Chapter 6 we discuss aspects that occurred while developing and using Phylo-Movies on specific data sets and give an outlook to future work and possible improvements.

1.1 Phylogenetics: A General Introduction

Phylogenetics is an interdisciplinary research field joining bioinformatics, biology and math that studies evolutionary relationships between or within groups of organisms called taxa. A taxon (plural taxa) is a group of one or more species considered by taxonomists to be a single unit. A phylogeny or phylogenetic tree depicts the hypothetical evolutionary relationship between selected groups of taxa. In the past, scientists constructed phylogenies on morphological characters (Lee and Palci, 2015); today, the construction of phylogenetic trees is mainly based on nucleotide and protein sequences stored in public databases such as the NCBI database (Wheeler et al., 2007). Phylogenetic trees must be inferred from a data set. For the construction of phylogenetic trees, typically multiple sequence alignments are used (Schmidt, 2003). A multiple sequence alignment is a data matrix in which molecular sequences from different species, typically with a common ancestor, are aligned (Schmidt, 2003).

1.1.1 Recombination in RNA viruses

RNA viruses exhibit high genetic variability, primarily due to high mutation rates and large population sizes (Holmes, 2009). The recombination process in RNA viruses corresponds to a new formation of segmented or unsegmented RNA from parent genomes of mixed origin (Simon-Lorière and Holmes, 2011). This recombination can occur in a single genomic segment known as RNA recombination. When this process occurs in viruses that have segmented genomes, and there is an exchange of entire genomic segments between viruses, it will be referred to as reassortment (Simon-Lorière and Holmes, 2011). The principle of RNA recombination and reassortments are two different procedures, requiring two or more viruses to infect the same host cell. We will only deal with the characterisation of recombinations, and the most widely accepted model of RNA recombination is 'copy choice' recombination (Simon-Lorière and Holmes, 2011). RNA-dependent polymerase RNA (RdpR) causes such viral replication (Simon-Lorière and Holmes, 2011). The RNA polymerase switches from one RNA template to another (might have a different ancestral history) during RNA synthesis while bound to the generated nucleic acid chain. In this way, a new mixed RNA molecule with a different ancestry is synthesised (Simon-Lorière and Holmes, 2011). Bioinformatics methods that search for differences in phylogenetic trees have identified RNA recombinations in RNA viruses probably produced by a copy choice mechanism (Simon-Lorière and Holmes, 2011).

When aligning RNA or DNA of viruses and then when we utilise the sliding window approach, we can detect RNA recombinations from different ancestors by sequentially dividing multiple sequence alignments into overlapping subalignments (Lemey et al., 2009, Chapter 16). Subsequently, we construct a phylogenetic tree for each subalignment, which leads us to generate a sequence of trees.

1.2 Mathematical Description of Phylogenetic Trees

1.2.1 Introduction into Trees

Trees, Nodes and Edges As explained before, phylogenetic trees are used to analyse the relation of species. Furthermore, they can be utilised for more advanced techniques such as building phylogenetic trees for every subalignment constructed out of a whole multiple sequence alignment to find recombination breakpoints (Fig. 1.2). Hence, we want to explain the specific structure of phylogenetic trees.

A tree can be denoted as acyclic graph $T = (V, E)$ (Schmidt, 2003). Trees denoted as an acyclic graph consists of nodes V and a set E of edges. The edges are linking two nodes. The edges in a tree are $e\ E$ directed away from the root as in a directed acyclic graph (see Fig. 1.3 for the components of tree diagrams with different layouts). We denoted edges by their adjacent nodes. The degree of a node V is the sum of outgoing and ingoing edges. Nodes without descendants are leaves. Furthermore, they have a degree of one because they only have one ingoing edge. Nodes with a degree greater than one are internal nodes (Schmidt, 2003).

Phylogenetic Trees Nodes in phylogenetic trees correspond to taxa and their ancestors. Leaves depict taxa, while internal nodes in phylogenetic trees depict hypothetical ancestors. Edges, in the context of phylogenetic trees, are called branches. They represent the development between two nodes (Schmidt, 2003), (Felsenstein, 2004, Chapter 2). Weights are assigned to edges to depict evolutionary time (Schmidt, 2003). When weights are assigned to a phylogenetic tree, they will be referred to as weighted trees. The weights assigned to the branches are depicted as branch lengths (see Fig. 1.3). In Fig. 1.3a the radial elements do not characterise evolutionary time, whereas straight lines emerging from the radials do. In Fig. 1.3b the vertical lines characterise evolutionary time, while the horizontals do not.

Phylogenetic trees can have a root. A root node r has only outgoing edges, but no ingoing ones. If a phylogenetic has a root node $r \in V$ it is referred to as a rooted tree. However, if they don't have a root,

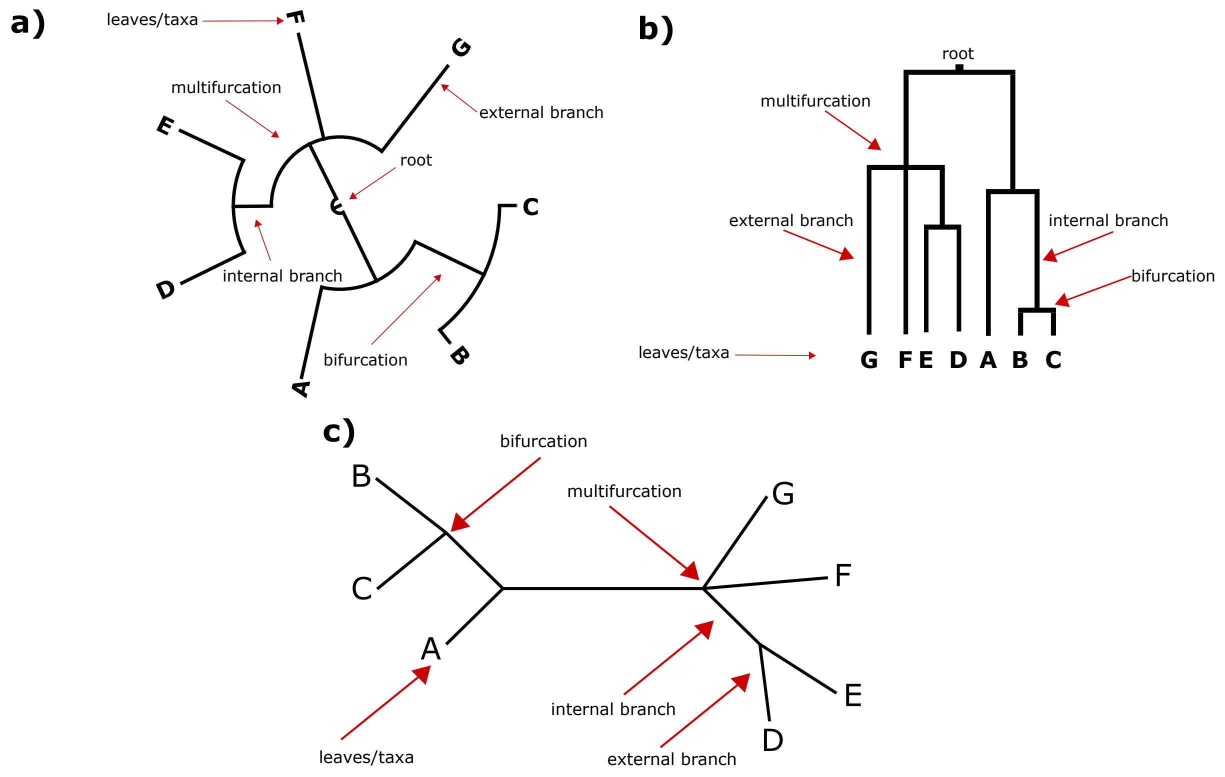


Figure 1.3: Tree notation on different tree diagrams of the same rooted tree topology (a) radial tree (b) rectangular tree (c) the unrooted toplogy of the same tree.

they will be referred to as unrooted trees. See Fig. 1.3c for an example for an unrooted. In Fig. 1.3a we can see the root node in the centre of the tree, whereas in Fig. 1.3a it is on the highest point of the tree. All internal nodes have a minimum degree of three apart from the root node. We will focus on rooted trees because Phylo-Movies can visualise and animate only rooted trees.

Additionally, nodes in phylogenetic trees that have three adjacent branches will be referred to as bifurcating, or they are bifurcations, whereas when the degree is greater than three, they are referred to as mulitfurcations (Felsenstein, 2004; Schmidt, 2003). If a tree has a multifurcation, the tree is not fully resolved (Schmidt, 2003). The knowledge about ancestral sequences is not existent, so phylogenetic trees are leaf-labelled trees (Schmidt, 2003). We use the definitions as taxon (plural taxa), species, leaf interchangeably. Hence, the lack of knowledge of ancestral sequences is the reason that evolutionary trees are leaf labelled trees (Schmidt, 2003).

Splits and Topology The term phylogenetic tree refers to a topology or the weighted tree. A topology τ is defined by a taxon set X and a set of edges. There are two types of edges: external (terminal) edges connecting leaf with inner nodes and internal edges connecting internal nodes. Suppose one deletes an edge in phylogenetic tree results in a decomposed tree connected by two components (Kupczok, 2010, Chapter 1). Thus a set of taxa is then segmented into two sets (X_1, X_2).

We term this segments as splits and define it by $X_1|X_2$. Each edge in a topology corresponds to a split. Thus a topology τ on a taxon set X identifies by the corresponding split set (Kupczok, 2010, Chapter 1).

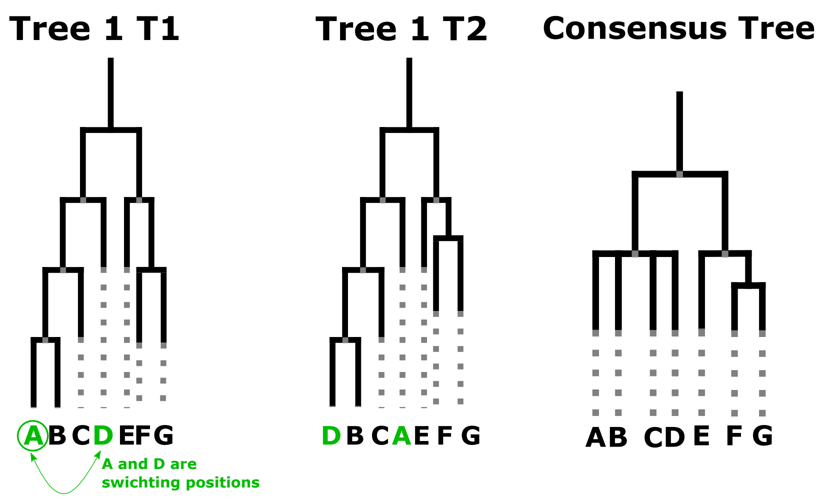


Figure 1.4: Strict consensus of trees Tree 1 and Tree 2

Furthermore, that for $n \leq |\tau| \leq 2n - 3$. Furthermore there are $m = 2^{n-1} - 1$ possible splits for $n = |X|$ taxa and a tree can only contain $2n - 3$ splits, i.e. external splits and $n - 3$ internal splits (Kupczok, 2010, Chapter 1). $|\tau|$ will be referred as count of splits in τ. The set of all splits for n taxa is denoted S_n and distinguished into n terminal splits (trivial splits) and $m - n$ internal splits (Kupczok, 2010, Chapter 1).

1.2.2 Consensus Trees

A set of trees can be summarised using consensus tree approaches (Felsenstein, 2004, Chapter 30). Here we will later use the so-called strict consensus tree method (Felsenstein, 2004, Chapter 30). A strict consensus tree of two trees T_1 and T_2 with the same taxa set is the tree that contains all splits which exist in both trees T_1 and T_2.

Splits absent in both trees cannot be resolved and lead to multifurcations. Consensus trees summarise the information contained in a set of trees. T_1 and T_2 in Fig. 1.4 share the same split $(ABCD|EFG)$,$(EFG|ABCD)$ and $(FG|ABCD)$. But the splits $(ABC|ABCEG)$, $(AB|CDEFG)$, $(BCD|AEFG)$ and $(BD|ACEFG)$ are absent in one of the trees. They are deleted and result in a multifurcation in the consensus tree, as seen in Fig. 1.4.

1.2.3 The Continuous Tree Space

Before we defined topologies and we explained that the branches of trees have lengths. For all of the examples that will occur, we assume that there are no initial trees with the branch length zero. The set of all trees with n leaves and branches with positive branch lengths are in the tree space $\mathbb{T}_n$ (Billera et al., 2001). In the tree space $\mathbb{T}_{\ltimes}$, every point represents a rooted tree with n leaves and branches with positive branch lengths (Billera et al., 2001).

In the beginning, we have two trees. We aim to transform the first tree into the second one. At the moment when the length of the branches are shrunk to zero and it reaches the same multifurcation occurring in both trees, in tree space the trees are adjacent to each other and span a subspace in the tree space (Felsenstein, 2004, Chapter 4). If we then insert a new branch that occurs in the second tree with a branch length of zero and then lengthen it, we can smoothly transition from the topology of the first tree to

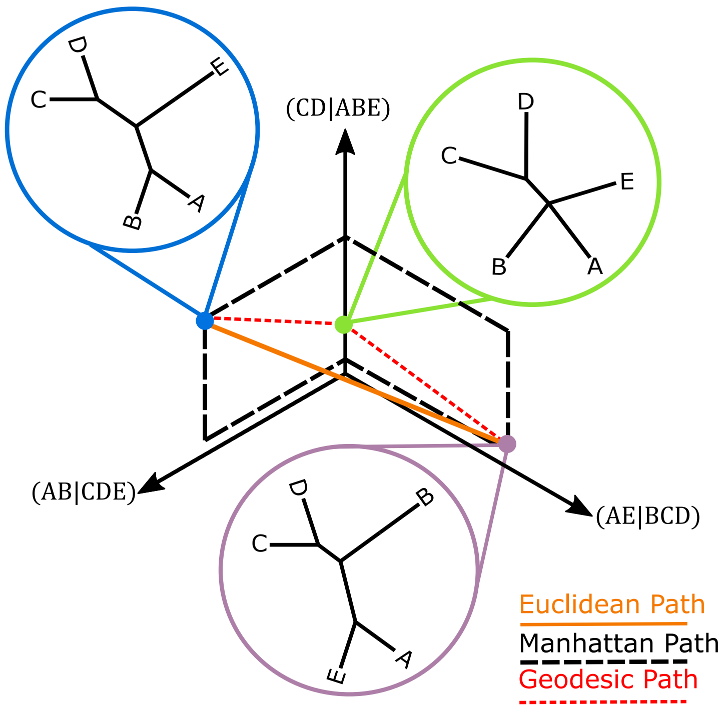


Figure 1.5: Here we depict a subspace in the tree space for a tree with five taxa. Here we depict a consensus tree lying between a segmented path leading from one tree to another. We based this figure on a previous figure explaining the tree space in Kupczok, 2010, Chapter 1

the other (Felsenstein, 2004, Chapter 4). We define the different transition forms as paths, that can lead to the second tree. The different paths will be discussed in Chapter 2. But all lead from one tree to another as shown in Fig. 1.5. We shrink an internal branch until it has zero length so that we reach a subspace of multifurcating trees (Felsenstein, 2004, Chapter 4). In Fig. 1.5 we see a subspace spanned by the splits $(CD|ABE), (AB|CDE), (AE|BCD)$. The two topologies (τ_1, τ_2) have one common split $(CD|ABE)$. The tree space is connected because each subspace includes the star tree in which no internal splits exists.

Distances

A function that measures the distance between two trees will be referred to as the distance function. A distance function quantifies how different two trees are (Kupczok, 2010, Chapter 1). A certain number of distance functions use only the topology to calculate a distance between a pair of trees, as the Robinson-Foulds distance (D. Robinson and Foulds, 1981). Other distance functions are utilising the branch length and the topology of the tree like the weighted Robinson-Foulds distance (D. Robinson and Foulds, 1979).

Robinson-Foulds Distance The Robinson-Foulds distance (D. Robinson and Foulds, 1981) is denoted by the count of splits in each topology which are absent in the other topology. The set-theoretical formulation would given by the symmetric difference denoted by $\triangle$ on the split set of τ^1 and τ^2 and the follows $RF(\tau^1, \tau^2) = |\tau^1 \triangle \tau^2| = |(\tau^1 \cup \tau^2) \setminus (\tau^1 \cap \tau^2)|$ (Kupczok, 2010, Chapter 1). The relative Robinson-Foulds distance is the Robinson-Foulds distance divided by the sum of all internal splits in τ^1 and τ^2. In Fig. 1.5

for the pair of trees the Robinson-Foulds distance is 2 because they do not have the split $(AE|BCD)$ and $(AB|CDE)$ in common.

Weighted Robinson-Foulds Distance The weighted Robinson-Foulds distance (D. Robinson and Foulds, 1979) is not only taking the topology into account but also the branch lengths (Kupczok, 2010, Chapter 1). Hence, the weighted Robinson-Foulds distance is denoted as the sum of absolute branch length differences for all splits in a pair of weighted trees (Kupczok, 2010, Chapter 1). Furthermore in b^1 all the branch lengths of each split in T_1 will be included but also the non-existent splits in T_1 existent in T_2 will assigned with zero, whereas in b^2 all the branch lengths of each split in T_2 will be included but also the non-existent splits in T_2 but existent in T_1 will assigned with zero (Felsenstein, 2004, Chapter 30). We include branch lengths of all present splits in both trees in b^1 and b^2. So the weighted Robinson-Foulds distance will be calculated as follows:

$$RF_w(T^1, T^2) = \sum\nolimits_{S \in \mathbf{S_n}} |b_S^1 - b_S^2|$$

The weighted Robinson-Foulds distance corresponds to the Manhattan distance (Kupczok, 2010, Chapter 1).

Paths A path between two trees T_1 and T_2 in $\mathbb{T}_n$ will be defined as a continuous function $p(t)$ that yields us a point on the tree space. The unit interval u is a closed interval $[0, 1]$ which is a set of all real numbers that are equal or greater than zero and less than or equal to one (Robert and Donald, 1967, Chapter 1). The variable t lies in the closed unit interval $u = [0, 1]$. Let $t \in u$ be time that we need to transition from T_1 to T_2. Hence, $p(t)$ from T_1 to T_2 is a continuous function from the closed unit interval $u = [0, 1]$ which delivers us a point on the path in $\mathbb{T}_n$. Every point on the path represents a tree, so we can say in $\mathbb{T}_n$ that $p(0) = T_1$ and $p(1) = T_2$. One can interpret the variable t as a time and the value of $p(t)$ as the point on a path at time t. In a sense we walk from T_1 to T_2 in time t.

We consider three paths $\mathbb{T}_n$: Euclidean path (see Fig. 1.5 the orange line), Geodesic path (see Fig. 1.5 the red line) and the Manhattan path (Fig. 1.5 black dashed line).

The Euclidean path (see Fig. 1.5 the orange line), is the shortest path between two trees but note that this path may not lie in the tree space and it is unclear how the intermediate states should be translated to a tree topology (Billera et al., 2001). So we cannot use the Euclidean path.

The Manhattan path is the shortest path with the length of the Manhattan distance which allows only vertical or horizontal moves (Krause, 1973). Therefore only one split and/or branch length is adjusted at a time (dashed black paths in Fig. 1.5). The length of this path, the Manhattan distance, coincides with the weighted Robinson-Foulds distance (Kupczok, 2010, Chapter 1).

The shortest possible path between two trees in the tree space is called the geodesic path, and its length is the geodesic distance (Billera et al., 2001). It can be shown that the geodesic path between two trees exists and is unique (Billera et al., 2001; Kupczok, 2010). The geodesic path is partitioned into two segments (see Fig. 1.2.3 the orange line). The midpoint of the geodesic paths will be defined as the point where the first segment ends and the second emerges that leads to T_2 (see Fig. 1.2.3). For two trees with branch length, the midpoint of a geodesic path is a consensus tree C. Therefore the path is segmented by the point where the consensus tree C lies. But calculating the geodesic path is in the worst-case exponential, so we use a special path in $\mathbb{T}_n$ known as cone path and this path can be computed in linear time (Kupczok, 2010, Chapter 1). We assume that the cone path in $\mathbb{T}_n$, is a good approximation of the geodesic distance (Kupczok, 2010, Chapter 1) for our usage. The way how the cone path is used will be described in Chapter 2. This chapter will also explain how we use the cone path for our visualisation.

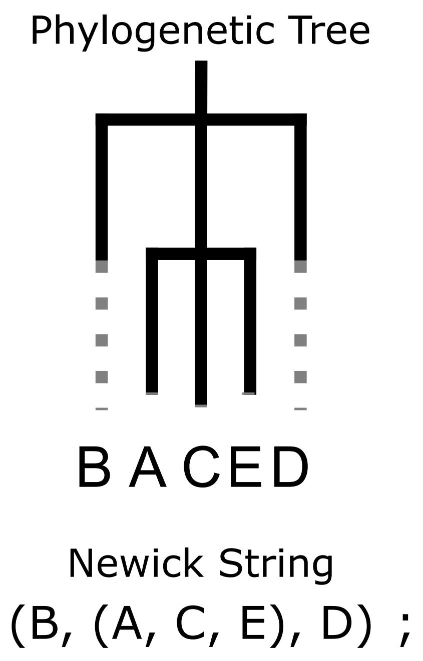


Figure 1.6: An example of how a phylogenetic tree can be described with a corresponding Newick string.

1.2.4 File Formats for Depicting Trees

We are taking the two file formats which are known as Newick (Felsenstein, 2004, Page 590) and Nexus (Maddison et al., 1997) for our visualisation of trees into account because they can define a sequence of trees. So the user can define both initial trees in Newick or Nexus in one file instead of uploading separate files for every tree.

Newick Format

The Newick format represents trees in computer-readable form (Felsenstein, 2004, Page 590). We took some parts of the explanation and the example from the website: https://evolution.genetics.washington.edu/phylip/newicktree.html. The format takes advantage of the correspondence between the hierarchy of trees and nested brackets. The Newick format represents a phylogenetic tree as a sequence of nested brackets and taxa labels. For example, the tree in Fig. 1.6 is represented by the Newick string "(B, (A, C, E), D);". Each Newick string is terminated with a semicolon. A pair of matching parentheses represent the inner nodes of a phylogenetic tree. Inside these brackets are comma-separated representations of nodes that descend directly from that inner node. In Fig. 1.6 the immediate descendants of the root are the internal node B and the leaf D. The second internal node is represented by a pair of parentheses enclosing its next descendants A, C, and E. But the descendant's A, C and E could also be inner nodes. We could do this by creating a nested of parentheses at each level. A real number indicates the branch length of a branch after a colon (Felsenstein, 2004, Page 590). Because semicolons delimit Newick Tree strings, thus files can easily contain sets of trees. This is one of the possible input formats for Phylo-Movies. (see Listing 1.1).

```
(((A:4.0,B:4.0):1.0,C:3.0):4,(D:5.0,E:4.0):3.0,F:11.0);
(((A:4.0,B:4.25):0.00000,C:3.0):4,(D:5.0,E:4.0):3.0,F:11.0);
((A:4.0,B:4.25,C:3.0):4,(D:5.0,E:4.0):3.0,F:11.0);
```

Listing 1.1: Here we can see a sequence of Newick strings. Due to the fact that we want to visualise a sequence of trees, the user has to define a set of Newick strings.

```
#NEXUS
BEGIN TAXA;
        DIMENSIONS ntax=6;
        TAXLABELS  A B C D E F;
END;
BEGIN TREES;
        TREE tree_1 = (((A:4.0,B:4.0)3:1.0,C:3.0)2:4,(D:5.0,E:4.0):3.0,F:11.0);
        TREE tree_2 = (((A:4.0,B:4.0)3:1.0,C:3.0)2:4,(D:5.0,E:4.0):3.0,F:11.0);
        TREE tree_3 = ((A:4.0,B:4.25,C:3.0):4,(D:5.0,E:4.0):3.0,F:11.0);
END;
```

Listing 1.2: In the block BEGIN_TREES we see a list of Newick Strings.

NEXUS

The second file type a user can use as input for Phylo-Movies is NEXUS (Maddison et al., 1997): The primary design feature of a NEXUS file is based on multiple blocks (see Listing 1.2). Nexus is very versatile and can contain many more blocks than those listed here. These blocks are represented by text and display, for instance, TAXA, CHARACTERS, and TREES. The tree block in a NEXUS file uses Newick strings to denote the trees (Maddison et al., 1997).

Chapter 2

Generating Intermediate Trees

In the last chapters, we explained phylogenetic trees defined in the tree space. We want to use the tree space to construct intermediate trees between a pair of trees. We use intermediate trees to create a sequence of images that describes the transition from one tree to another. Animation is a sequence of consecutive pictures that depict intermediate states between a beginning and an end (Wells, 2013). These consecutive pictures depicting the transitions, when played quickly, lead to an animation that shows the transition from one to the other.

To animate the difference between two phylogenetic trees, we need to create intermediate trees that show the transition from one tree to another. We call the first tree T_1 and the second tree T_2. So we want a transition from T_1 to T_2. We are generating four intermediate trees. These intermediate trees can be created by following a path in the tree space (see Section 1.2.3 for a description of the tree space). In further sections, we explain how we transition from one tree to another. Creating intermediate points on a path is also known as interpolation. Interpolation is the process of creating data points due to the range of a finite set of data points (Richard, Andrew, et al., 1994, Page 800). So constructing a path from the known point "a" to "b" in tree space is the same as interpolating between points a and b. By constructing a series of intermediate trees from a pair of trees, we create the animation of the transition from one tree to another. By this interpolation, we try to adhere to the principles Congruence and Apprehension of Tversky et al., 2002 for effective animations. In our animation in Phylo-Movies, we try to maintain valid graphics of trees during the transitions. We prevent misunderstandings about the animation by maintaining valid graphics, therefore preventing the user's attention disruption.

2.1 Method for Generating Dynamic Tree Visualisation

2.1.1 Layout Calculations

When visualising or drawing a tree on a canvas we need to specify coordinates for each node and draw lines between adjacent nodes. A canvas has two dimensions: a height and a width. We call a function that defines the position of each node a layout function L. The layout function correspond to $L : \mathbb{T}_n \to \mathbb{R}^2$. We consider two layouts: rectangular, circular (Figs. 2.2-2.1).

While the layouts have a different appearance, they provide the same information. The horizontal lines describing the branch length in a rectangular tree are the same as the straight lines emerging from the radial lines (curved lines). While radials in the radial tree express how related two taxa are with each other, this

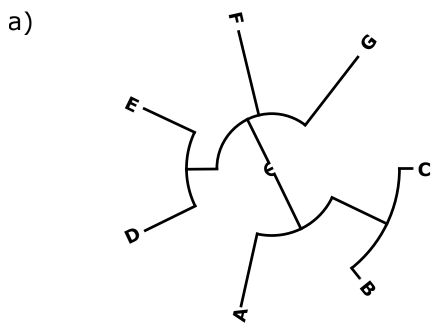


Figure 2.1: Displaying an example for a radial layout of a phylogenetic tree.

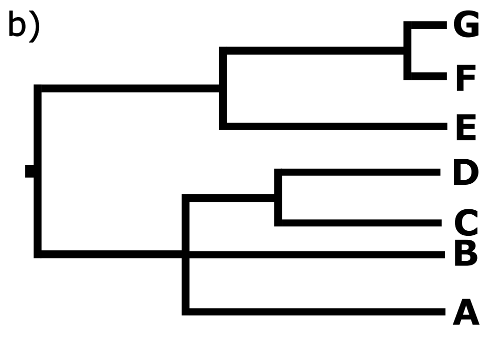


Figure 2.2: Displaying an example for a rectangular layout of a phylogenetic tree

will be explained by the vertical lines in the rectangular layout. The right side of a rectangular layout has more nodes than the left side in the rectangular layout. In contrast, the leaves in a circular layout are evenly distributed around a circle, it can be seen in Figs. 2.1-2.2. In Phylo-Movies, we decided to use the radial layout. With the radial layout, the density of nodes is better distributed than in a rectangular layout, which gives users the ability to display more taxa in a graphical, more pleasant way. Additionally, the radial layout provides us with the possibility that when we animate the branches along the radial, the movement is easier to trace and that while we are animating, the trees are valid.

We can easily describe the circular layout in polar coordinates (Richard, Andrew, et al., 1994, Chapter 11), consisting of an angle and a radial where the midpoint of the canvas (in Cartesian coordinates (0,0)) is the centre of the canvas. Instead of straight lines, circular arcs are naturally arising. The formerly horizontal lines now radiate from the centre of the circle. To calculate the radial layout of the trees, we proceed as follows: Initially, the radius of each leaf is set to the length of the width or the height of the canvas, according to which value is smaller. Then the radius of each internal node is calculated depending on their cumulative branch length from the root up to their ancestor nodes. Then all the angles are calculated. Each leaf is assigned an index according to the topological linear order. The angle of a leaf in the radial layout with the index i is $i2\pi/|\text{leaves}|$. The angle of an internal node is the mean of all its child nodes and leaves' angles. Finally, the radius is scaled based on the canvas's minimal dimension, and the polar coordinates are translated to Cartesian coordinates.

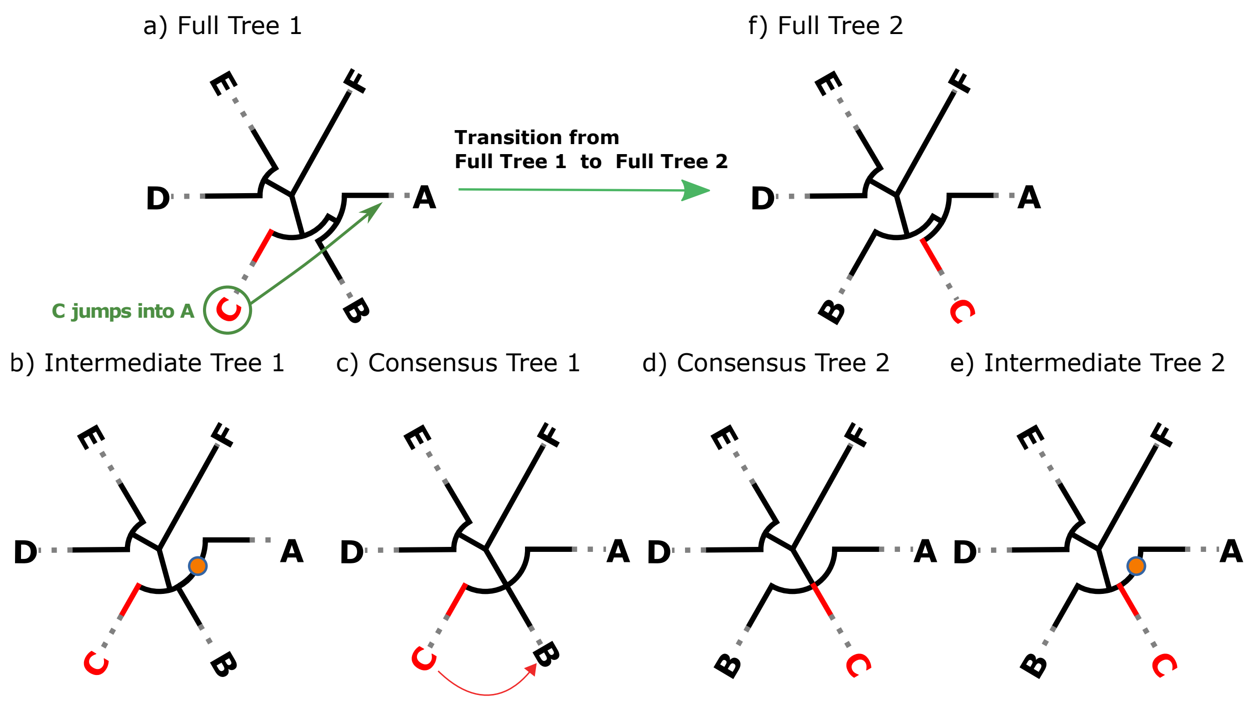


Figure 2.3: First tree of the tree pair a) where the transition is supposed to happen. f) Second tree of the tree pair. b) - e) The transition trees that are constructed by the program to simplify visualisation. The circles filled with orange and with a blue stroke are indicating where the branch with the length zero lies, which will emerge or will be deleted in the transition.

2.1.2 Combining the Cone Path with Layouting

A path from T_1 to T_2 in the tree space is only dependent on the topology and the branch lengths, the layout of a tree depends on the circular order of its taxa. The cone path is segmented into two paths, but for a continuous animation, there needs to be an intermediate step to change the circular order of leaves of T_1 to T_2. To define a continuous interpolation, we utilise four intermediate trees between the two input trees.

The intermediate trees are intermediate tree 1 IT_1, consensus tree 1 CT_1, consensus tree 2 CT_2 and intermediate tree 2 IT_2. We call the two input trees full Tree 1 (FT_1) and full Tree 2 (FT_2). For the transformation we now have three trees FT_1, C, FT_2. Recall that C is the consensus tree constructed by following the cone path. Because the circular order of the leaves of FT_1 and FT_2 may differ, one consensus tree is insufficient. So we decided to construct two consensus trees. The two consensus trees CT_1 and CT_2 are isomorphic but differ in the circular order of the taxa. The circular order of C_1 follows that of T_1 and C_2 follows that of T_2. Additionally, the layout of taxa also depends on the topology and is different depending on whether a branch length is zero or not existent. To compensate we define the trees intermediate tree 1 and 2 (IT_1 and IT_2). IT_1 has the same topology as T_1. If a branch exists in T_1 and T_2, we set the length of the branch to the average. If a branch is present in T_1 but not in T_2, the length of that branch is set to zero in IT_1 . So in the animation, we shrink the branches from their initial branch length to zero. The branches present in T_2 but not in T_1 are set to zero in IT_2. The branches existent in T_2 but are absent in T_1, will be inserted from the transition from C_2 to IT_2, with the branch length zero. Then if we transition from IT_2 to T_2, we lengthen these branches. So, T_1 is to linear IT_1 and IT_2 is linear to T_2, but C_1 is not linear to C_2 because only the order of the leaves is different.

2.1.3 Example on the Individual Steps

The first transition tree IT_1 (Fig. 2.3b) has the same topology as T_1, but the branch lengths of the branches present in T_1 and T_2 have the average branch length of the length which they had in T_1 and T_2. The split representing a branch that exists in both trees without the trivial splits is $(ED|ABCF)$. The existing split in T_1 but not existing in T_2 is $(AB|CDEF)$. Because there is no corresponding branch in T_2, the length is set to zero. This can be observed in the split $(AB|CDEF)$ (Fig. 2.3b). The second transition tree Fig. 2.3c is the first consensus tree C_1. As we can see, there is a multifurcation, since the split $(AB|CDEF)$ is absent in T_2 and the split $(AC|BDEF)$ is absent in T_1. The branches set to zero in the previous tree have now been deleted. The third transition tree (Fig. 2.3d) is the second consensus tree. However, this time, the order of taxa within the multifurcation is determined by the order of T_2. The fourth transition tree IT_2 (Fig. 2.3f) has the same topology as the second tree, but its branch lengths are still the intermediate lengths, and the branches that are not present in T_1 have length zero as seen in the split $(AC|BDEF)$ representing the branch which is set to zero in IT_2. Moving from tree IT_2 to T_2, we assign lengths to the branches that had length zero, and we assign the branch lengths of T_2 to the branches that are present in both trees.

Chapter 3

Highlighting Changes in Topologies

In the previous chapter, we defined how to interpolate from one tree to another to create a continuous animation. Just from the movement of subtrees and taxa in the animation it is hard to comprehend the actual transformation between two trees and to deduce the relocating taxa causing the topological changes from one tree to another. Therefore, we want to identify and highlight those taxa that move between the two trees. In this chapter we develop an algorithm to find jumping taxa in two steps. In the implementation of Phylo-Movies, this algorithm is used to highlight those jumping taxa.

3.1 Central Idea of the Algorithm

The same terminology as in Chapter 2 is used. But the input trees will be referred as full tree 1 (FT_1) and 2 (FT_2). We assume that all branches have non-negative branch lengths in FT_1 and FT_2. We call a branch with length zero a zero-branch and a branch with a length greater than zero a positive branch. We denote an **s-edge** a branch (in an intermediate tree) with positive length, and at least one child branch with length zero. Since s-edges have positive branch lengths, they exist in FT_1 and FT_2. We denote the opposite tree of FT_1, FT_2, whereas the opposite tree of FT_2 will be FT_1. The opposite tree IT_1 will be IT_2 and the opposite tree of IT_2 will be IT_1.

We categorise s-edges by their direct child branches. We call an s-edge with at least one positive child branch a **partial-s-edge** (Fig. 3.1) and s-edges with only zero child branches a **full s-edge** (Fig. 3.2). There are cases where the s-edge's corresponding edge in the opposite tree has only child branches with positive values, they will be referred to as **non-s-edges** (Fig. 3.3). We call an edge with branch length zero and which children's branch lengths are all positive an **anti s-edge** (see red branch in Fig. 3.1), as the definition of an anti s-edge (Fig. 3.1) is formally the logical negation of the definition of an s-edge.

Given an s-edge in FT_1 or FT_2, we call the same edge in the opposite tree the corresponding edge. Note that the corresponding edge does not need to be an s-edge in the opposite tree (for an example see Fig. 3.3). The leaf set of a branch is the set of all taxa in the subtree to which the branch leads. The components of a corresponding edge are the leaf sets of its adjacent positive descendant branches (e.g., colourised blocks in Fig. 3.2). The component set of a child branch of an s-edge is called one of the s-edge's arm.

A zero branch carries information about the taxa movement, indicating that a split changed and is either absent in FT_1 or FT_2. As the zero branches might be absent in one or both trees, we need a branch that exists in IT_1 or IT_2 to investigate. The ancestor of the zero branches is either a zero branch itself or an

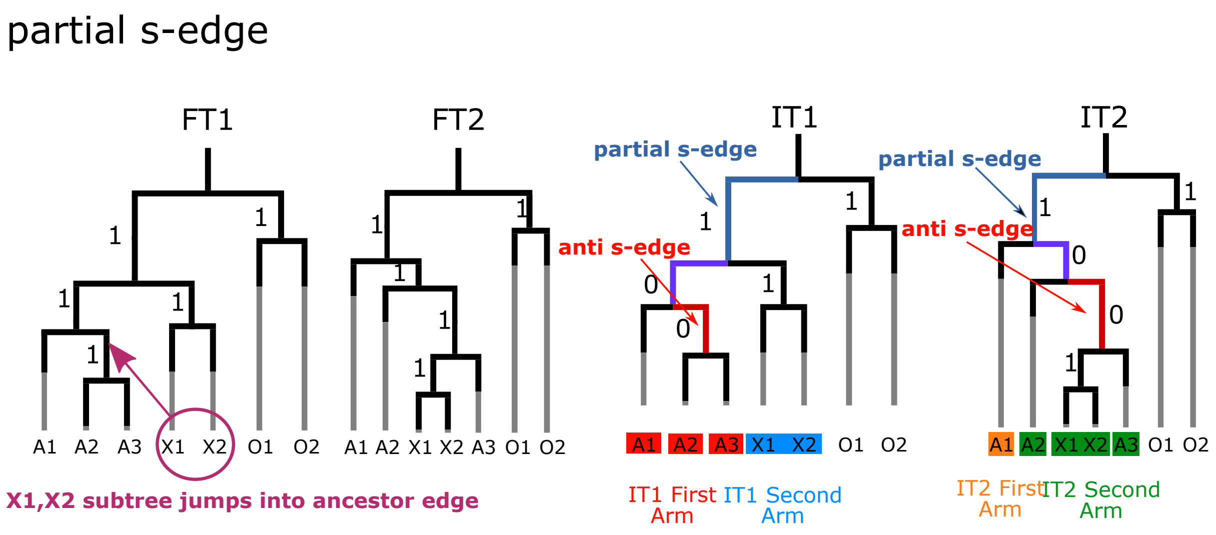


Figure 3.1: Example: A subtree (X1, X2) jumps directly to the s-edge. Here we can see when the subtree jumps directly to the s-edge, it becomes a partial-s-edge (blue edge). One of the children leading to the subtree (X1, X2) has a positive branch length, whereas the other child is zero-branch, leading to an anti s-edge.

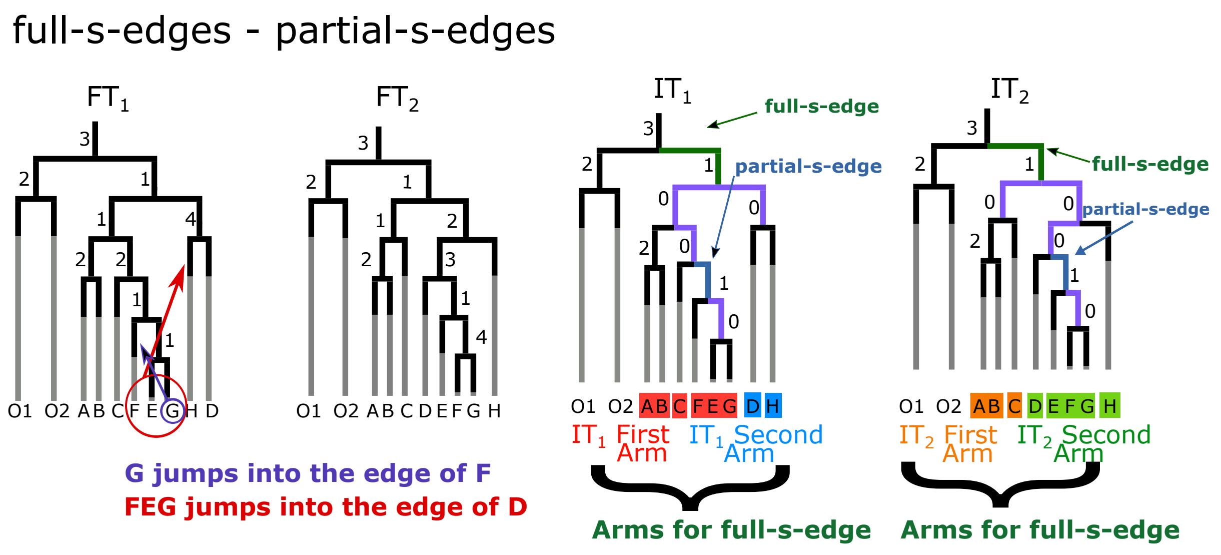


Figure 3.2: This figure shows an example of a full s-edge. E, F, and G. So one full s-edge and a partial-s-edge are emerging. As we can see, the full s-edge has only zero children branches.

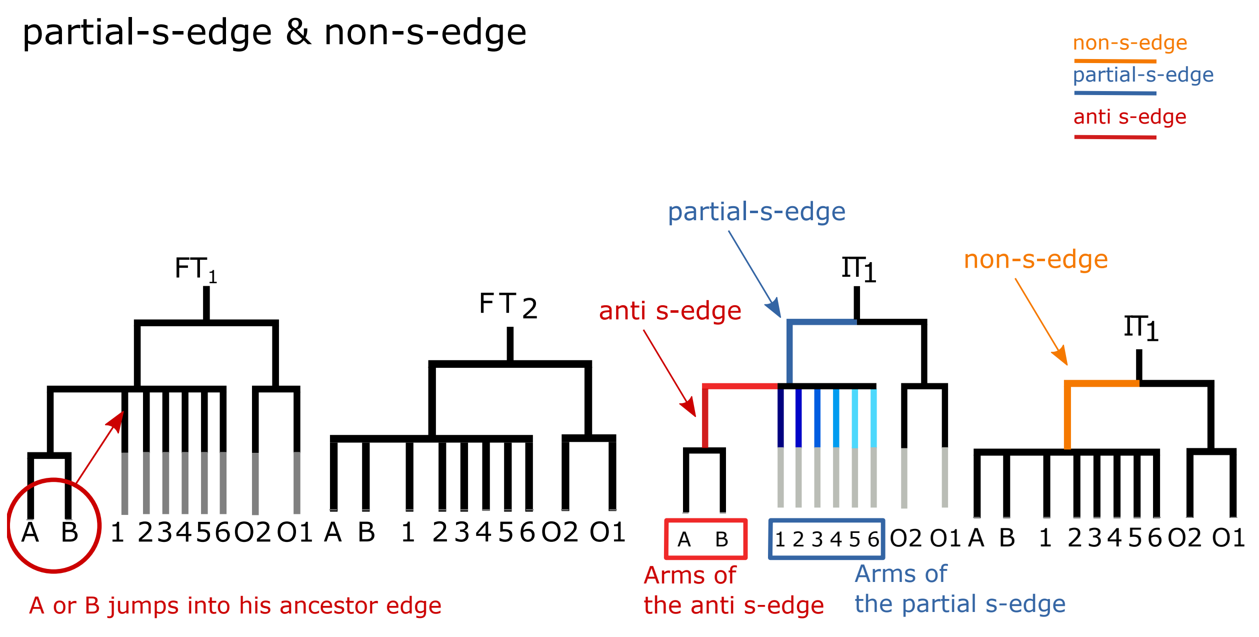


Figure 3.3: This figure shows an example of partial-s-edge 1. Either A or B jumps into his ancestor edge, which means that it results in IT_1 of an emergence of a partial-s-edge, whereas in IT_2 it leads to corresponding non-s-edge, because external branches cannot be missing, so they cannot become zero edges.

s-edge. We examine this s-edge, which exists in FT_1 and FT_2. Intuitively the jumping taxa or subtrees must have switched from one arm to another arm of the s-edge.

We utilise set operations to compare the taxa of the arms. Which set operations to use in which cases will be developed in the upcoming sections. It does not matter whether a single taxon jumps or an entire subtree. The definition of components captures both because the positive descendant branch of an s-edge can lead to a taxon or a subtree. Instead of finding jumping taxa, we want to find jumping components, which also could be a single taxon. Note that within a component, more than one jump may occur. However this does not influence that all types of the component jump as a whole.

3.2 The Initial Approach

The first version of our algorithm to identify taxa that jumped proceeds as follows. The explanation of the algorithms is denoted in pseudo-code as for example in Algorithm 1. Each declaration of a variable is followed by colons, after the colons a data type for the declared variable is set. In our algorithms, the smallest units are the components. A set of components is therefore defined as an arm. Components will be denoted by C, its arms will be denoted by $Set[C]$, and the set of arms will be defined by $Set[Set[C]]$.

In Algorithm 1 in Line 2, we declare a set called c_2, with a set of arms. We first construct IT_1 and IT_2 from the input trees FT_1 and FT_2. Then we traverse IT_1 and IT_2, collect all s-edges and their corresponding edges and store them in a set.

For each pair of corresponding edges, we traverse IT_1 and IT_2 and collect the arms and store them

Algorithm 1 Considering the intersection and symmetric difference of the component sets of the s-edges the IT1 and IT2. t_1 is input tree1, t_2 is input tree 2, e is the s-edge.

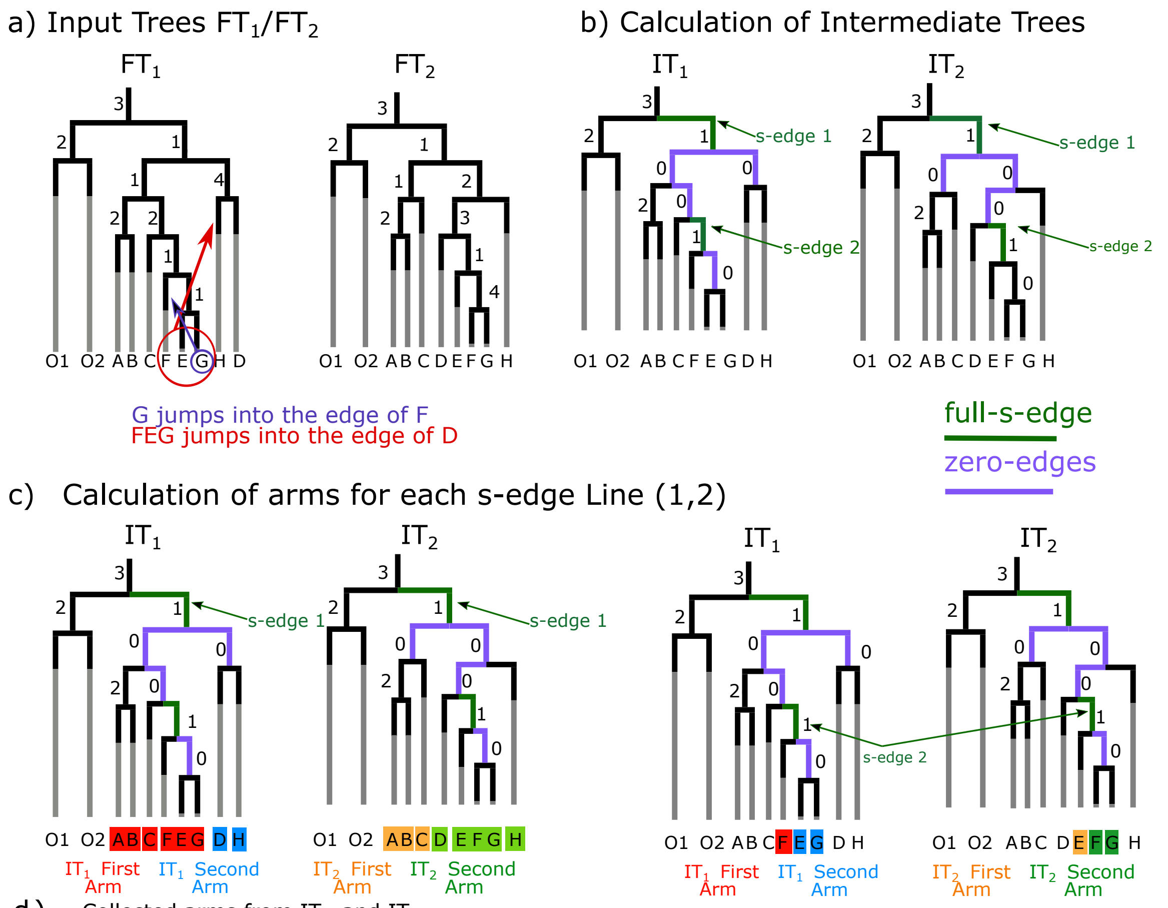
```
procedure InitialApproach($t_1$,$t_2$,$e$)
    $c_1 : Set[Set[\mathcal{C}]]$ = GET the set of arms for the s-edge $e$ in reference to $t_1$
    $c_2 : Set[Set[\mathcal{C}]]$ = GET the set of arms for the s-edge $e$ in reference to $t_2$
    $v = \{\}$
    for each $arm_1$ in $c_1$ do
        for each $arm_2$ in $c_2$ do
            ADD $arm_1 \cap arm_2$ TO $v$
            ADD $arm_1 \triangle arm_2$ TO $v$
        end for
    end for
    $m : Set[Set[\mathcal{C}]]$ = GET the component with the highest frequency in $v$
    $r : Set[Set[\mathcal{C}]]$ = GET the component with the least count of components in $m$
    $rr$ = UNITE the components of $r$
    return $rr$
end procedure
```

in c_1 and c_2 (Algorithm 1 Lines 2-3). Then, for each pair of arms $arm_1, arms_2 \in c_1 \times c_2$, we compute the intersection $arm_1 \cap arm_2$ and the symmetric difference $arm_1 \triangle arm_2$ of components and store them in v (Algorithm 1 Lines 7-8). We retain how often an element is stored in v. Subsequently we get the component set with the highest frequency in v and discard all the other component sets (Algorithm 1 Line 11). If there are multiple components sets, also referred to as candidates we get the sets with the least components (Algorithm 1 Line 12). We assume that the component set with the least components is the reason for the topological change. Afterwards each taxon of the component set is returned as the jumping taxa of this s-edge (Algorithm 1 Line 13).

Fig. 3.4 shows an example application of Algorithm 1. E, F and G jump into D and G jumps into F (Fig. 3.4a). There are two s-edges, s-edge 1 and 2 (Fig. 3.4b). For each s-edge the component sets are calculated and visualised by colourised blocks (Fig. 3.4c). The components in Fig. 3.4 for the s-edge-1 in tree IT_1 are $\{A, B\}$, $\{C\}$, $\{E, F, G\}$, $\{D\}$ and $\{H\}$, whereas the components in IT_2 are $\{A, B\}$, $\{C\}$, $\{D\}$, $\{E, F, G\}$, $\{H\}$. The various components are distributed in different arms. Each block is one component, and the colour indicates whether the component is in arm 1 or arm 2 (Figs. 3.4c and 3.4d). For s-edge 1 two arms in IT_1 and IT_2. The first arm in IT_1 has the components $\{\{A, B\}, \{C\}, \{E, F, G\}\}$, the second arm has $\{\{D\}, \{H\}\}$. For the s-edge 1 in IT_2 we have $\{\{D\}, \{E, F, G\}, \{H\}\}$ and $\{\{A, B\}, \{C\}\}$. For each pair of arms of IT_1 and IT_2, the intersections and symmetric differences are calculated (Fig. 3.4e). For example the arm $\{\{A, B\}, \{C\}, \{E, F, G\}\}$ will be intersected with the arm $\{\{D\}, \{E, F, G\}, \{H\}\}$, which results in $\{\{E, F, G\}\}$. Whereas the symmetric difference for the pairs of arms would be $\{\{A, B\}, \{C\}, \{D\}, \{H\}\}$. At last for each s-edge, the component set with the highest frequency is returned (Fig. 3.4f). In example Fig. 3.4, for s-edge 1 only one component set occurs with highest frequency, which is $\{\{E, F, G\}\}$. If there were multiple component sets with the same frequency, we would choose a component set with the least amount of components.

In some cases, this algorithm produces wrong results. The algorithm computes the intersection and symmetric differences of each corresponding s-edge and considers all components as candidates. In Fig. 3.5e, we can see how the Initial Approach chooses the wrong results. Components sets containing taxa that have not moved (in Fig. 3.5 for example $A1$) are returned. Hence we improve the part of the algorithm where the calculation of the correct candidates takes place (Fig. 3.4f) in the following sections.

Initial Approach



d) Collected arms from IT_1 and IT_2

S-Edge	Arms IT_1 1	Arms IT_1 2	S-Edge	Arms IT_2 1	Arms IT_2 2
1	{{A,B}, {C}, {E,F,G}}	{{D}, {H}}	1	{{D}, {E,F,G}, {H}}	{{A,B}, {C}}
2	{{E}, {G}}	{{F}}	2	{{G},{F}}	{{E}}

e) Compute the intersection and the symmetric difference between arm IT_1 and arm IT_2 and store them in a set (Line 4-10)

Intersections

Arms IT_2		Arms IT_1 1	Arms IT_1 2
S-Edge 1		{{A,B}, {C}, {E,F,G}}	{{D},{H}}
1	{{A,B}, {C}}	{{A,B}, {C}}	{}
2	{{D}, {E,F,G}, {H}}	{{E,F,G}}	{{D},{H}}
S-Edge 2		{{F}}	{{E}, {G}}
1	{{E}}	{}	{{E}}
2	{{F}, {G}}	{{F}}	{{G}}

Symmetric Difference

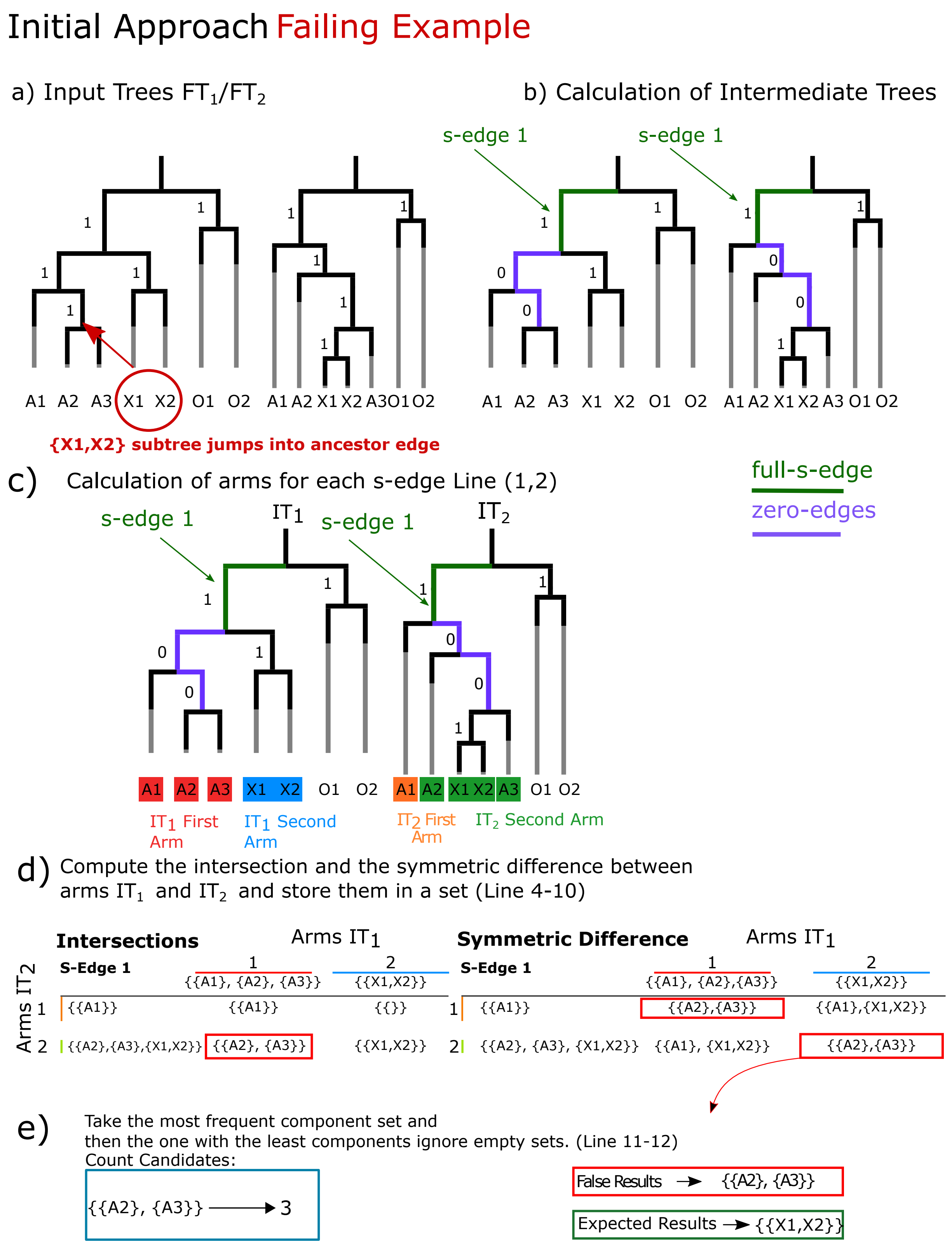
Arms IT_2		Arms IT_1 1	Arms IT_1 2
S-Edge 1		{{A,B}, {C}, {E,F,G}}	{{D}, {H}}
1	{{A,B}, {C}}	{{E,F,G}}	{{A,B},{C},{D},{H}}
2	{{D}, {E,F,G}, {H}}	{{A,B}, {C}, {D}, {H}}	{{ E,F,G }}
S-Edge 2		{{F}}	{{E}, {G}}
1	{{E}}	{{E}, {F}}	{{G}}
2	{{F}, {G}}	{{G}}	{{E}, {F}}

f) Take the most frequent component set and then the one with the least components. Ignore empty sets. (Line 11-12)

S-Edge 1 {{E,F,G}} Component frequency (3) and one component

S-Edge 2 {{G}} Component frequency (3) and one component

Figure 3.4: Here we can see the application of the Initial Approach on an example. E, F and G jump into D and G jumps into F (Fig. 3.4a)


Initial Approach Failing Example
a) Input Trees FT$_1$/FT$_2$
b) Calculation of Intermediate Trees
s-edge 1
s-edge 1
A1 A2 A3 X1 X2 O1 O2 A1 A2 X1 X2 A3 O1 O2
A1 A2 A3 X1 X2 O1 O2 A1 A2 X1 X2 A3 O1 O2
{X1,X2} subtree jumps into ancestor edge
full-s-edge
zero-edges
c) Calculation of arms for each s-edge Line (1,2)
IT$_1$
IT$_2$
s-edge 1
s-edge 1
A1 A2 A3 X1 X2 O1 O2 A1 A2 X1 X2 A3 O1 O2
IT$_1$ First Arm
IT$_1$ Second Arm
IT$_2$ First Arm
IT$_2$ Second Arm
d) Compute the intersection and the symmetric difference between arms IT$_1$ and IT$_2$ and store them in a set (Line 4-10)
e) Take the most frequent component set and then the one with the least components ignore empty sets. (Line 11-12)
Count Candidates:
{{A2}, {A3}} → 3
False Results → {{A2}, {A3}}
Expected Results → {{X1,X2}}

Intersections

S-Edge 1 (Arms IT$_2$)		Arms IT$_1$ 1: {{A1}, {A2}, {A3}}	Arms IT$_1$ 2: {{X1,X2}}
1	{{A1}}	{{A1}}	{{}}
2	{{A2},{A3},{X1,X2}}	{{A2}, {A3}}	{{X1,X2}}

Symmetric Difference

S-Edge 1 (Arms IT$_2$)		Arms IT$_1$ 1: {{A1}, {A2},{A3}}	Arms IT$_1$ 2: {{X1,X2}}
1	{{A1}}	{{A2},{A3}}	{{A1},{X1,X2}}
2	{{A2}, {A3}, {X1,X2}}	{{A1}, {X1,X2}}	{{A2},{A3}}

Figure 3.5: Example: A subtree with taxa (X1,X2) jumps directly to the s-edge. If a taxon or a subtree jumps directly to the s-edge, it is not possible for our algorithm to detect it as we can see in Fig. 3.4e

Algorithm 2 FinalApproach, differentiating between different types of s-edges and then calculating the jumping taxa with three different algorithms. t_1 is input tree1, t_2 is input tree 2, e is the s-edge.

```
1: procedure FinalApproach($t_1$,$t_2$,$e$)
2:     if isFullSedge($t_1$,e) or isFullSedge($t_2$,e) then
3:         result = InitialApproach($t_1$,$t_2$,e)
4:     end if
5:     if isPartialSedge($t_1$,e) and isPartialSedge($t_2$,e) then
6:         result = calculateResultPartialPartial($t_1$,$t_2$,e)
7:     end if
8:     if isPartialSedge($t_1$,e) and isNonSedge($t_2$,e) then
9:         result = calculateResultPartialNon($t_1$,e)
10:    end if
11:    if isNonSedge($t_1$,e) and isPartialSedge($t_2$,e) then
12:        result = calculateResultPartialNon($t_2$,e)
13:    end if
14:    return result
15: end procedure
```

3.3 Final Approach

In the Initial Approach, we treated each s-edge's components of an arm as a possible solution candidate. In simple cases (which can be correctly highlighted by the Initial Approach), all children of the s-edge have length zero. However as we can see in Fig. 3.5 this produces wrong results. We observe that if a subtree or taxon relocates itself directly to the s-edge not all child branches are becoming zero-edges (see Fig. 3.5). The branch of the subtree or leaf that jumped directly to the s-edge is a positive branch (see Fig. 3.6a). Thus, an s-edge where a subtree directly jumps to is now identified as a partial-s-edge (see Fig. 3.1). This direct jump leads our calculation in the Initial Approach to be insufficient to generate possible solutions. We look in IT_1 and IT_2 from what type of edge the components descent from. If the components descent from the expected edges in IT_1 and IT_2 , we store them, whereas the components that will be discarded (see Fig. 3.6c). By discarding the components in arms, it is possible to calculate component sets that indicate taxa which may have caused the change in a pair of trees. By our investigation, we identified as a crucial step to distinguish between the different types of s-edges that we have defined in Section 3.1.

In the Final Approach for the types of s-edges in IT_1 and IT_2 we use different strategies (see Algorithm 2). There can be alternating permutations of corresponding s-edges in two trees. For example in the tree IT_1, the s-edge can be partial whereas the corresponding edge in IT_2 can be a non-s-edge. Hence in Final Approach, we traverse IT_1 and IT_2 and collect the corresponding s-edges. Then for every corresponding s-edge in IT_1 and IT_2, we identify the type of the corresponding s-edge to determine the right strategy. It happens when finding a partial-s-edge in IT_1 (see Fig. 3.7 blue edge in IT_2) or IT_2 and its corresponding s-edge is a non-s-edge (see Fig. 3.7 orange edge in IT_2). This situation occurs when the non-s-edge's children external branches because all external branches leading to leaves have positive length. We have three procedures; if the s-edge in IT_1 or IT_2 is full s-edge, we use the Initial Approach (Algorithm 1). In the case that in both trees the corresponding s-edges are partial we use the `calculateResultPartialPartial` procedure see (Algorithm 3). In the case if in one of the two trees is a non-s-edge and in the other a partial-s-edge we are using the `calculateResultPartialNon` procedure (Algorithm 4).

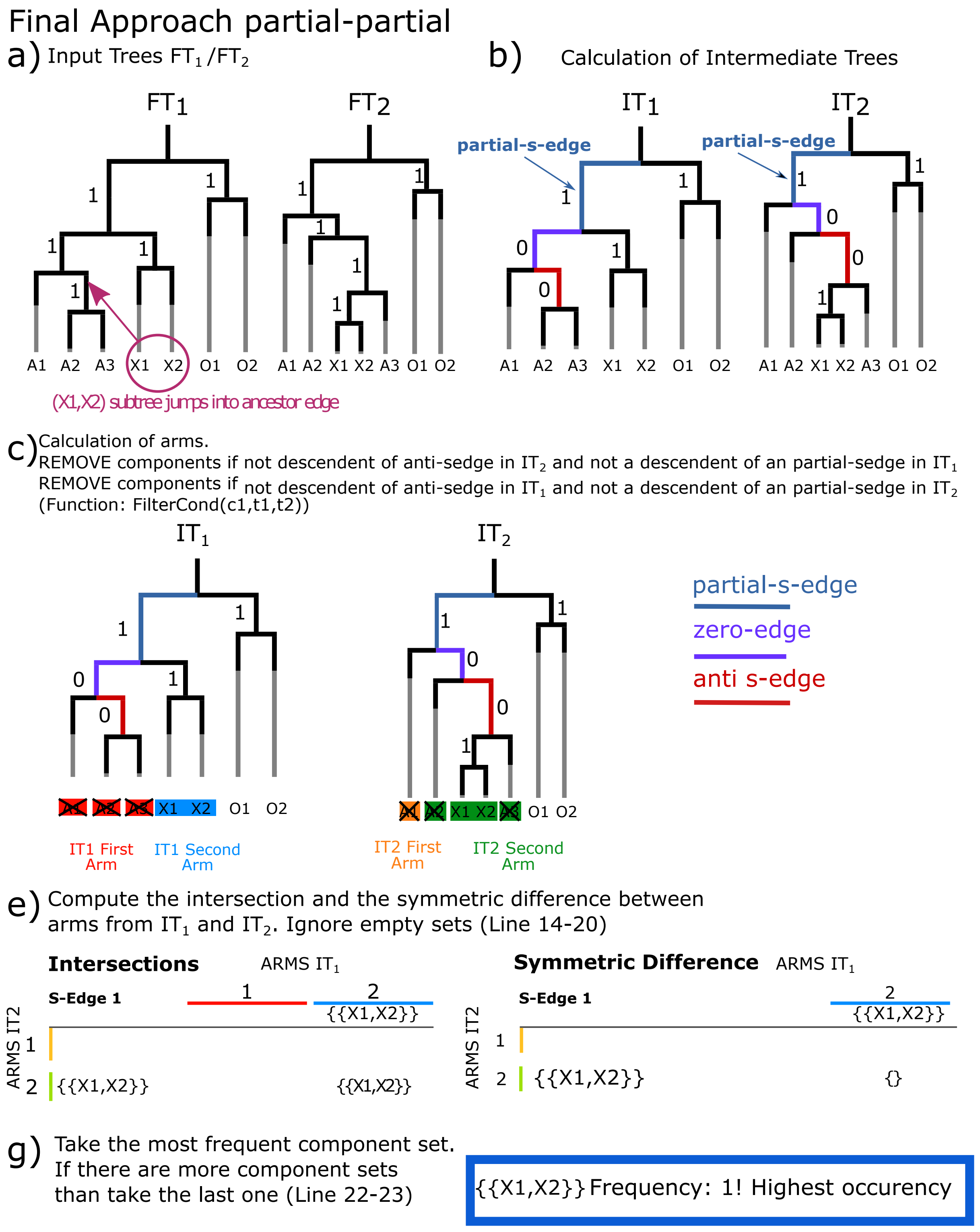
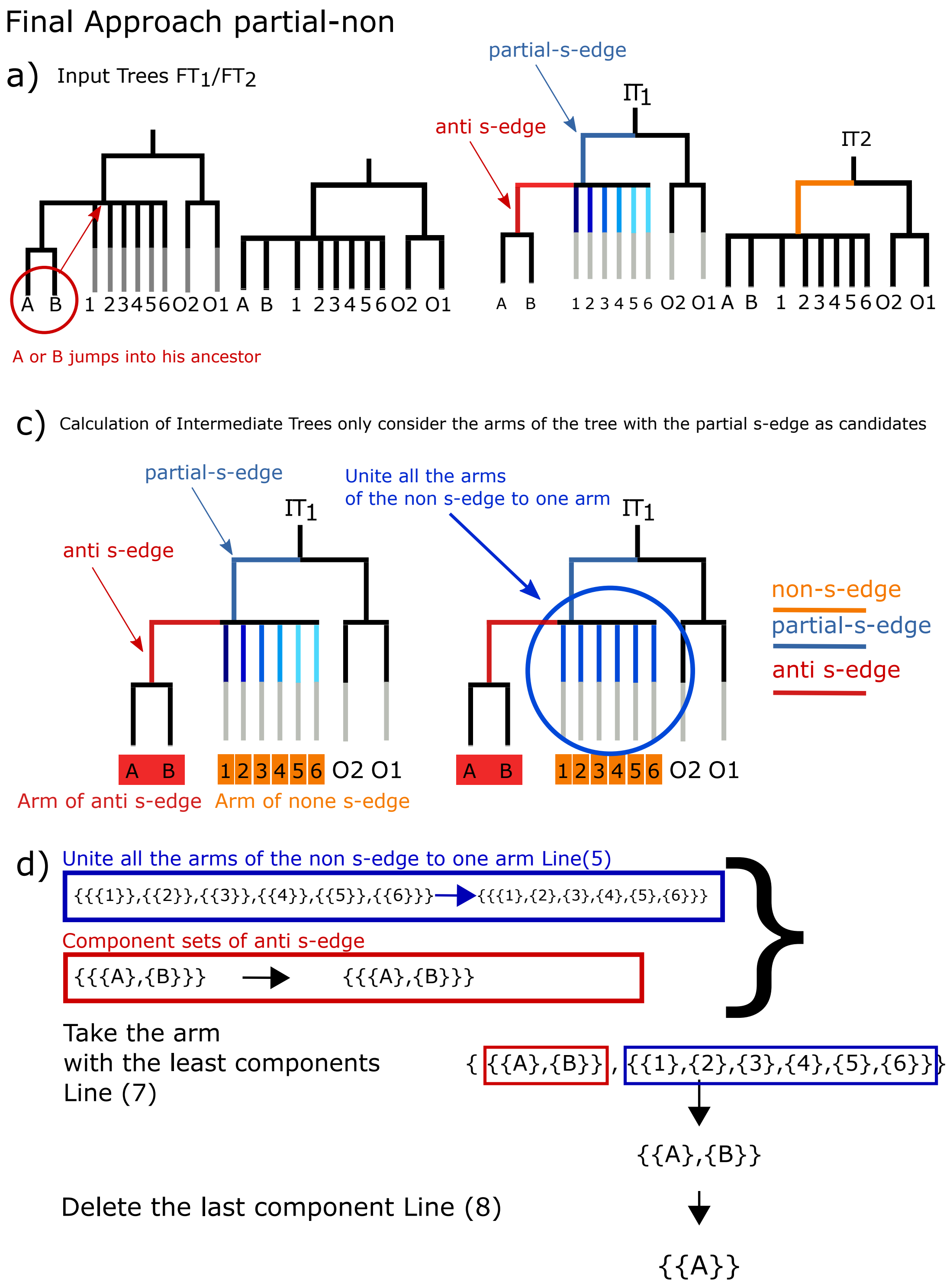

Final Approach partial-partial
a) Input Trees FT1 /FT2
FT1
FT2
A1 A2 A3 X1 X2 O1 O2
A1 A2 X1 X2 A3 O1 O2
(X1,X2) subtree jumps into ancestor edge
b) Calculation of Intermediate Trees
IT1
IT2
partial-s-edge
partial-s-edge
A1 A2 A3 X1 X2 O1 O2
A1 A2 X1 X2 A3 O1 O2
c) Calculation of arms.
REMOVE components if not descendent of anti-sedge in IT2 and not a descendent of an partial-sedge in IT1
REMOVE components if not descendent of anti-sedge in IT1 and not a descendent of an partial-sedge in IT2
(Function: FilterCond(c1,t1,t2))
IT1
IT2
partial-s-edge
zero-edge
anti s-edge
X1 X2 O1 O2
X1 X2 O1 O2
IT1 First Arm
IT1 Second Arm
IT2 First Arm
IT2 Second Arm
e) Compute the intersection and the symmetric difference between arms from IT1 and IT2. Ignore empty sets (Line 14-20)
Intersections
ARMS IT1
S-Edge 1
1
2
{{X1,X2}}
ARMS IT2
1
2
{{X1,X2}}
{{X1,X2}}
Symmetric Difference
ARMS IT1
S-Edge 1
2
{{X1,X2}}
ARMS IT2
1
2
{{X1,X2}}
{}
g) Take the most frequent component set. If there are more component sets than take the last one (Line 22-23)
{{X1,X2}} Frequency: 1! Highest occurency

Figure 3.6: Example: A subtree (X1,X2) jumps directly to the s-edge. The algorithm for the partial-partial case is able to detect the moving taxa in this case.



Final Approach partial-non
a) Input Trees FT1/FT2
partial-s-edge
anti s-edge
IT1
IT2
A B 1 23456 O2 O1 A B 1 23456 O2 O1
A B 1 2 3 4 5 6 O2 O1 A B 1 23456 O2 O1
A or B jumps into his ancestor
c) Calculation of Intermediate Trees only consider the arms of the tree with the partial s-edge as candidates
partial-s-edge
anti s-edge
IT1
Unite all the arms of the non s-edge to one arm
IT1
non-s-edge
partial-s-edge
anti s-edge
A B 1 2 3 4 5 6 O2 O1
A B 1 2 3 4 5 6 O2 O1
Arm of anti s-edge
Arm of none s-edge
d) Unite all the arms of the non s-edge to one arm Line(5)
{{{1}},{{2}},{{3}},{{4}},{{5}},{{6}}} → {{{1},{2},{3},{4},{5},{6}}}
Component sets of anti s-edge
{{{A},{B}}} → {{{A},{B}}}
Take the arm with the least components Line (7)
{ {{A},{B}}, {{1},{2},{3},{4},{5},{6}} }
{{A},{B}}
Delete the last component Line (8)
{{A}}

Figure 3.7: Example: Taxa *A* or *B* jumps directly to the s-edge. Algorithm Partial-Non is able to detect the possible jumping taxa causing the topological case.

Algorithm 3 CALCULATERESULTPARTIALPARTIAL for the case if both s-edges are from the type partial. t_1 is input tree1, t_2 is input tree 2, e is the s-edge.

```
procedure filterCond($c_1$, $t_1$, $t_2$)
    for each arm in $c_1$ do
        for each component in arm do
            REMOVE component if not descendent of anti s-edge in $t_2$ and not a descendent of an partial-sedge in $t_1$
            REMOVE component if not descendent of anti s-edge in $t_1$ and not a descendent of an partial-sedge in $t_2$
        end for
    end for
    return $c_1$
end procedure
procedure calculateResultPartialPartial($t_1$,$t_2$,$e$)
    $c_1 : Set[Set[\mathcal{C}]]$ = GET the set of arms for the edge $e$ in reference to$t_1$
    $c_2 : Set[Set[\mathcal{C}]]$ = GET the set of arms for the edge $e$ in reference to $t_2$
    $cf_1 : Set[Set[\mathcal{C}]]$ = FilterCond($c_1$, $t_1$, $t_2$)
    $cf_2 : Set[Set[\mathcal{C}]]$ = FilterCond($c_2$, $t_1$, $t_2$)
    $v =$
    for each $arm_1$ in $cf_1$ do
        for each $arm_2$ in $cf_2$ do
            ADD $arm_1 \cap arm_2$ TO $v$
            ADD $arm_1 \triangle arm_2$ TO $v$
        end for
    end for
    DELETE ALL EMPTY SETS in $v$.
    $r_h : Set[Set[\mathcal{C}]]$ = GET the arm with the highest frequency in $v$.
    $r_f : Set[Set[\mathcal{C}]]$ = REMOVE every component except the last one in $r_f$
    $rr$ = UNITE the components of $r_l$
    return $rr$
end procedure
```

3.3.1 Partial-Partial Procedure

The `calculateResultPartialPartial` procedure (see Algorithm 3) will now be explained in detail. For each corresponding s-edge of IT_1 and IT_2 we traverse IT_1 and IT_2, collecting the arms as in the Initial Approach and store them in c_1 and c_2 (see Algorithm 3 Line 11). In the partial-partial case we restrict the components. We collect the components of the arms of c_1 which are direct children of anti s-edges in IT_2 and of partial-s-edges in IT_1 to store them in cf_1. For the components in the arms of c_2, we are only taking the direct children of anti s-edges in IT_1 and of partial-s-edges in IT_2 and store them in cf_2 (see Algorithm 3 Lines 12-13). Then for each arm_1 and arm_2 in cf_1 and cf_2, we compute $arm_1 \cap arm_2$ and $arm_1 \triangle arm_2$ and store the result in v (see Algorithm 3 Lines 16)-21). Then we take the component sets with the highest frequency and store them in r_h (see Algorithm 3-Line 23). Note that, there may be several candidates, that can explain the topological change. Because we only want to indicate one solution, we consider the last element as solution and assign it to r_f (see Algorithm 3 Line 24). From the component set r_f the names of taxa will be extracted and stored in rr and then rr will be returned.

Fig. 3.6 shows an example of the usage of the `calculateResultPartialPartial` in the Final Approach, where the subtree of taxa $(X1, X2)$ jumps into $A3$ (Fig. 3.6a). There is only one corresponding s-edge in this example (Fig. 3.6b blue edges). We calculate for each s-edge the component sets, visualised by colourised blocks (Fig. 3.6c). The components for the partial-s-edge in IT_1 in Fig. 3.4c are $\{A1\}, \{A2\}, \{A3\}$ and $\{X1, X2\}$, whereas for IT_2 they are $\{A1\}$, $\{A2\}$, $\{A3\}$ and $\{X1, X2\}$. Some of the blocks are crossed out because they are not direct children of an anti s-edge in IT_1 and not direct children of a partial-s-edge in IT_2 or conversely not direct children of an anti s-edge in IT_2 and not direct children of a partial-s-edge IT_1. We can see in Fig. 3.4e in IT_1 the components in the second arm have these requirements $\{\{X1, X2\}\}$ (Fig. 3.6.c blue block), whereas no component fulfils the requirements in the second arm, so the arm and all of its components will be discarded. The same happens for the second arm in IT_2 and only the arm with the component $\{X1, X2\}$ is left, represented by Fig. 3.6.c by the not crossed out green block. Then we proceed by calculating for each arm the intersections and symmetric differences (Fig. 3.6e). So we calculate the intersection for the component in the arms of IT_1 and IT_2, which is $\{\{X1, X2\}\} \cap \{\{X1, X2\}\} = \{\{X1, X2\}\}$ and the symmetric difference $\{\{X1, X2\}\} \triangle \{\{X1, X2\}\} = \{\{\}\}$. For each s-edge the component set with the highest frequency is returned. We take the last component set if the resulting set still has more components than one. In Fig. 3.6g the only left component set is $\{\{X1, X2\}\}$, which in our case is the result.

3.3.2 Partial-Non and Non-Partial Procedure

For the situation where either in IT_1 the s-edge is partial whereas in IT_2 it's and non or vice versa we use the `calculateResultPartialNon` (see Algorithm 4). In the `calculateResultPartialNon` algorithm, we collect the arms of the tree with the partial-s-edge because we assume that the relevant information for finding the component set explaining the topological change is embedded in there. So we will only use the tree with the partial s-edge for finding the jumping taxa. We consider for this explanation that IT_1 is the tree with the partial-s-edge. So first we collect the arms in IT_1 and store them in c_1 (see Algorithm 4 Line 2).

We divide the arms c_1 that are direct children of the partial-s-edge and those of the anti s-edge and store them separately in two different variables. In Fig. 3.7c we can see that in IT_1 the partial-s-edge is marked blue and the anti s-edge red. The arms of the anti s-edge will be stored in cf_{anti} and the arm of the partial-s-edge will be stored in $cf_{partial}$ (see Algorithm 4 Lines 3 and 4). Then we unite all the arms that are stored in $cf_{partial}$ to one arm (see Algorithm 4 Line 5). Then the arms of cf_{anti} and $cf_{partial}$ will be stored in v (see Algorithm 4 Line 6). We take the arm with the least components as a result, because we assume the minimal count of jumps of the components is the reason for the topological change (see Algorithm 4 Line 8). After converting the component set to a taxa set we are returning it as a result (see Algorithm 4 Line 9).

Fig. 3.7 shows how the `calculateResultPartialNon` procedure proceeds in calculating the results in the Final Approach, where in tree FT_1 the subtree with the taxa A or B jumps into his ancestors (Fig. 3.7a). For the s-edge in IT_1 the arms with its components are collected, visualised by colourised blocks (Fig. 3.7c). We have 8 arms, two of the anti s-edge $\{\{A\}\{B\}\}$ and the other six arms of the partial-s-edge $\{\{\{1\}\}, \{\{2\}\}, \{\{3\}\}, \{\{4\}\}, \{\{5\}\}, \{\{6\}\}\}$. Then we unite the arms of the partial-s-edge $\{\{\{1\}\}, \{\{2\}\}, \{\{3\}\}, \{\{4\}\}, \{\{5\}\}, \{\{6\}\}\}$ to one arm $\{\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}\}$ (Fig. 3.7d). Then we put all arms in to one set (Fig. 3.7d). In Fig. 3.7d the combined set of arms is $\{\{\{A\}, \{B\}\}, \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}\}$. Then choose the the arm, with the least components (Fig. 3.7d), which in our example is the arm $\{\{\{A\}, \{B\}\}\}$. We take the last component as a result, because only one of the components is required to explain why the topology changed. In our example this is the component $\{A\}$.

Algorithm 4 CALCULATEPARTIALNON for the case if one of the s-edges are from the type partial and the other non. t_1 is input tree1, e is the s-edge.

```
1: procedure CALCULATERESULTPARTIALNON($t_1$,$e$)
2:     $c_1 : Set[Set[\mathcal{C}]]$ = GET the set of arms for the edge $e$ in reference to $t_1$
3:     $cf_{anti} : Set[Set[\mathcal{C}]]$ = FILTER components in $c_1$ that are no descendant of anti s-edges in $t_1$.
4:     $cf_{partial} : Set[Set[\mathcal{C}]]$ = FILTER components in $c_1$ that are no descendant of partial-s-edges in $t_1$
5:     $cf_{partial_u} : Set[Set[\mathcal{C}]]$ = UNITE all component sets in $cf_{partial}$ to one component set.
6:     $u : Set[Set[\mathcal{C}]] = cf_{partial_u} \cup cf_{anti_r}$
7:     $u_{min} : Set[Set[\mathcal{C}]]$ = GET the item with the least components sets $ut$
8:     $r : Set[Set[\mathcal{C}]]$ = REMOVE every component except the last one in $u_{min}$
9:     $rr$ = UNITE the components of $r_l$.
10:    return $rr$
11: end procedure
```

Case	Initial Approach	Final Approach
Case-1	Worked	Worked
Case-2	Worked	Worked
Case-3	Worked	Worked
Case-4	Worked	Worked
Case-5	Fail	Worked
Case-6	Fail	Worked
Case-7	Fail	Worked
Case-8	Worked	Worked

Figure 3.8: Table showing for which case the Initial approach or the Final approach worked. For a detailed view, please see the appendix.

Removing Taxa and Reiterate After identifying the taxa to be highlighted, we remove them from FT_1 and FT_2. Subsequently, we recalculate the intermediate trees and traverse the new IT_1 and IT_2 and identify further s-edges because, in an s-edge, more than one relocation of taxa may have happened. So if we find further s-edges, we repeat the Final Approach algorithm, whereas when not, we can stop. The pruning is done via the ETE3 toolkit (Huerta-Cepas, François Serra, et al., 2016b).

3.4 Testing the Algorithm

We created a test data set of tree pairs to compare the algorithms more methodically. The trees cover a range of complexities: nothing jumps, one taxon jumps, two taxa jump, one subtree jumps, two taxa swap positions in two separate subtrees, two taxa jump independently in two subtrees. Initial Approach finds taxa that jumped in the full s-edge cases, whereas when we started to construct trees where partial-s-edges occur, it was incapable of solving them. The Final Approach can detect jumping taxa in the cases of partial-s-edges. There could be phylogenetic trees that could lead to unforeseen results. There has to be further investigation on pathological cases that leads The Final Approach to produce wrong results. In the cases that we constructed, our Final Approach yields the expected results. We display the examples in Appendix A. We can see in Table 3.8 that all of the cases are solved by the Final Approach whereas, the Initial Approach is not yielding the right results in some cases. This is because the Initial Approach is not pruning the trees and is also not recalculating if there are some s-edges. Also, the differentiation between s-edges can lead to different results in both trees.

We have summarised in Appendix A a few examples that were relevant in the development of the highlighting algorithm. The pictures were generated with ETE3 (Huerta-Cepas, François Serra, et al., 2016a). One example is shown per page and the result of the algorithm. You can see the first initial tree (FT_1) and the second one (FT_2) on the left side. But also the differentiation of the strategies for the different types of s-edges, shows the differences. As it can bee seen in the cases Sections A.5, Sections A.6, Sections A.7.

Specific taxa are marked in colour. The taxa marked in green are taxa that have been correctly identified. Taxa that are marked in blue are those that were not recognised, although they are valid candidates of jumping taxa. The taxa marked in red are mismarked or do not serve as solution candidates for the cases.

First, all the case examples and their results for the Initial Approach are shown, followed by the same examples, except that this time the results of the Final Approach are shown. All cases are numbered and can be recognised by their numbering. First, we have examples where a taxon jumps Section A.1, two taxa exchange Section A.2, and a small example where a taxon jumps in a subtree Fig. A.3. After that, we see a case where two subtrees exchange Section A.4. Then we have a case where two different s-edges appear Section A.5. Then we have a case where a subtree move toward the root Section A.6. Then we have two cases where subtrees dissolve into a multifurication and where the subtree is swapping position (Section A.7 and Section A.8). It is a representative subset of cases we tested the approaches on to better understand topologies of trees.

Chapter 4

Technical Implementation

4.1 Introduction

We want to visualise the transition from one tree to another. In Chapter 2, we defined an interpolation from one tree to another by the cone path, which is an approximation on the geodesic path (Kupczok, 2010, Chapter 1). The result is that we can represent the interpolation by six trees. Then in Chapter 3 we developed an algorithm for finding taxa that most likely caused the topological changes between a pair of trees. These two algorithms are the two basic principles along which we developed the comparison of the trees. Here we describe how to convert the interpolation to a visualisation displayed on a screen.

To display an image on a computer screen, one needs a software module to render it before it is displayed on the screen (Grosskurth and Godfrey, 2005). To access a module enabling rendering a picture on a screen, one can either write their software or use software the user already has. Because one of our goals is accessibility, we opt for the second choice and base Phylo-Movies on a widely installed software basis. Such installed software that could be used to display animations are either web browsers or movie players. We want our software to be interactive, but since movie players do not offer interactivity, we decided to develop Phylo-Movies as a web application displayed by a web browser. This means that Phylo-Movies must contain a web server that speaks the `HTTP` protocol (Nielsen et al., 1999). `HTTP` is the protocol that one can use to interact with the browser (Nielsen et al., 1999). Generally, one does not need to program a completely new `HTTP` server but can use frameworks that work as plugins to existing `HTTP` servers.

There are many frameworks for different programming languages to develop the backend part, but we decided to base Phylo-Movies on `Flask` (Grinberg, 2014, Chapter 1). `Flask` is a widely used framework that provides basic functionality for building web applications without relying on external libraries (Grinberg, 2014, Chapter 1). There are three relevant approchaes in `HTTP` that have the capability to display animations: (a) sequence of static images (like `JPEGs` (Stewart et al., 1998) or `PNG`s (Boutell, 1997)), (b) a movie format (like `MP4` (Lim and Singer, 2006) or animated `GIF` (Alvestrand, 1998)) or (c) web page. From these three approaches, only web pages offer interaction capability.

Web pages consist of three complementary parts: Hypertext Markup Language (`HTML`) (Berners-Lee and Connolly, 1995) is the standard markup language for documents intended for display in a web browser and defines the content of the web page. Cascading Style Sheet (`CSS`) (Flanagan, 2016) is a text format that defines the styling of the content of the web page (Grosskurth and Godfrey, 2005). Finally `JavaScript` is a dynamic programming language, which is generally used to define the interactions of the web page.

A subpart of HTML is Scaleable Vector Graphics (SVG) (Brownlee and IAB, 2016), which is a vector image format, that meets our criteria of usability and high fidelity.

All three parts, HTML, CSS, JavaScript operate on the Document Object Model (Berners-Lee and Connolly, 1995) also knows as DOM. An Application Programming Interface API that represents the web page as a tree of DOM nodes (Grosskurth and Godfrey, 2005). DOM nodes include text elements (like paragraphs or headings), structural elements (like spans or divs) and graphical elements (like SVGs). HTML defines the content of the web page by creating DOM nodes, CSS defines the style of the content by defining how each DOM node is visualised and JavaScript defines the interactions of the web page by defining how each DOM node reacts on mouse clicks or keyboard events.

In Phylo-Movies we generate the HTML documents by the templating library of Flask, which is called Jinja (Grinberg, 2014, Chapter 3). D3 (Bostock et al., 2011) was used for the visualisation of the data. While there are many JavaScript frameworks available, D3 is the state of the art in interactive web visualisations (Bostock et al., 2011). HTML, CSS and JavaScript are called the frontend, while the Python server is called the backend (see Fig. 4.1 for visual representation of the structure). The data which should be visualised in the frontend is transmitted from the backend to the frontend via the JavaScript Object Notation JSON (Bray, 2017) also referred as JSON format. In the next few sections several aspects of DOM, CSS, SVG, JavaScript and templating will be discussed and how every technology contributed to the development of Phylo-Movies.

4.2 Backend and Frontend

As discussed in the previous section, we split Phylo-Movies into two separate parts interacting with each other. The two parts are backend (server) and frontend (client). See Fig. 4.1 for an explanation how specific tasks are executed in the backend and which are executed in the frontend. The frontend gets a sequence of phylogenetic trees in the Newick or Nexus format as input and sends them to the backend. The backend computes the consensus and intermediate trees and converts them into hierarchical JSON objects; these computed trees are then sent to the frontend. After the backend completes the computation, it redirects the browser to the visualisation page. As mentioned before, the backend is implemented in Flask (Grinberg, 2014, Chapter 1), a micro web framework. A web framework is based on a set of components. This components allow one to build an application without repeatedly implementing the same parts.

Whenever the user enters a URL (Uniform Ressource Locator (Berners-Lee, Masinter, et al., 1994)), the browser sends a request to a server, which return files that the browser then processes and renders (Grinberg, 2014, Chapter 2). Every Web application encounters problems such as mapping a specific URL to a specific function returning files or dynamically creating a web page (Grinberg, 2014, Chapter 1). One can focus on writing the code for a particular URL quickly with a web framework. The major alternative to Flask is Django (Holovaty and Kaplan-Moss, 2009), but we found that Flask is better suited for the requirements of Phylo-Movies. Django provides form validation, object-relational mappers, open authentication systems, upload mechanisms, and several other tools, which Phylo-Movies does not require and impose additional overhead. Since Flask provides us with enough utility with its basic features and its lightweight design, we decided that Flask is better suited for Phylo-Movies. For a graphical overview of how the input will be transformed into a visualisation, see Fig. 4.2. For the dependencies to run the backend on a server please see Appendix B.

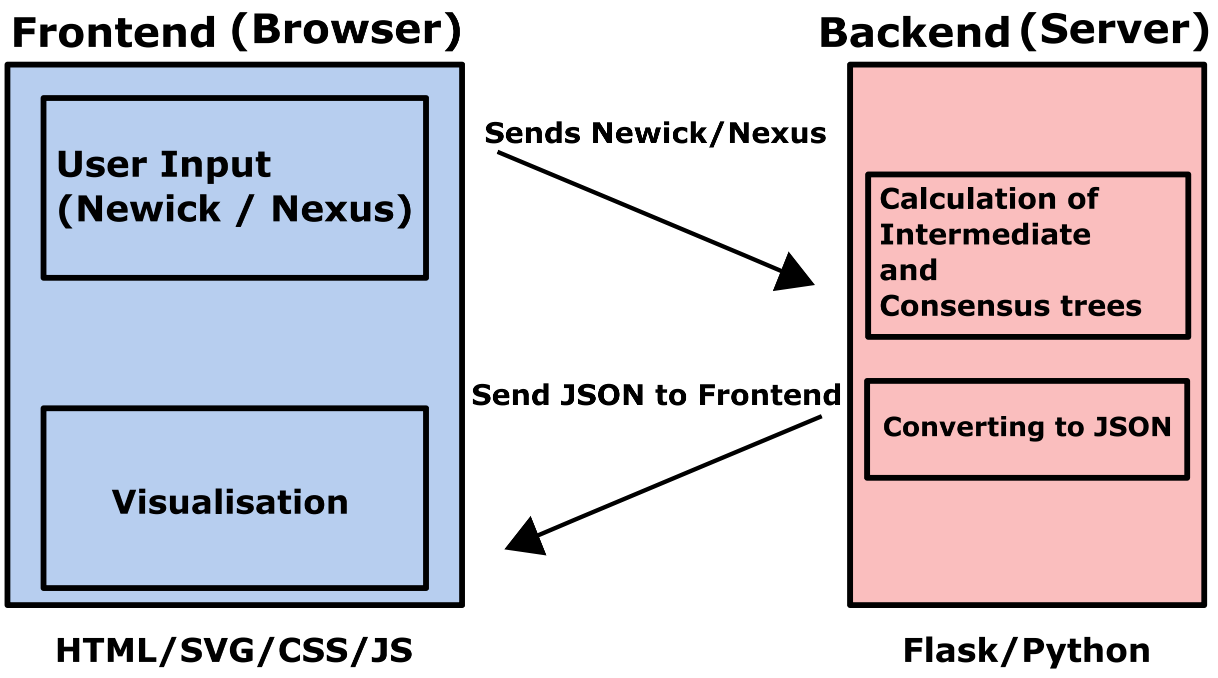


Figure 4.1: Visualising how the specific tasks are handled by the backend and frontend.

4.3 Rendering `HTML` with a sequence of trees

As mentioned before the `HTML` document will be rendered after the calculation of the intermediate trees. The intermediate tree then will be embedded into the `HTML`. `Jinja`, a template engine enables us to embed the sequence of trees into the `HTML` file, which then will be loaded into the browser of the user (Grinberg, 2014, Chapter 3). A `Jinja` template is simply a text file (Grinberg, 2014, Chapter 3). `Jinja` can produce any text-based format (`HTML`, `XML`, `CSV`, `LaTeX`, etc.). A template contains variables and/or expressions that are replaced with values when a template is rendered, as well as tags that control the logic of the template (Grinberg, 2014, Chapter 3). The string `{{X}}` is expanded into the value of the variable X. Phylo-Movies includes the computed trees as `JSON` in the `HTML` file, stored in the variable `list_of_trees` (see Listing.4.1). The list of `list_of_trees` variable consists of the originals trees and the interpolated trees (see Chapter 2 for the sequential order of the trees) which were then transformed into a list of `JSON` objects (see how a transformed Newick or Nexus tree into `JSON` object looks like in Table 4.1). Also for every transition from one tree to another the to be highlighted taxa (stored in the variable `to_highlighted_taxa` in Listing 4.1) were previously calculated in the backend. This then will be used for colouring the labels, circles which represent the taxa and the branches leading to the taxa.

4.4 Transforming `JSON` to `D3` Nodes

The further mentioned sequence of trees (`list_of_trees`) in Listing 4.1 now have to be visualised. We use `D3` for the visualisation, `D3` (Data-Driven Documents) is a novel approach for data visualisation on the web (Bostock et al., 2011). `D3` was developed to build web-based data visualisations (Bostock et al., 2011) and

```
<html lang="en">
    <body>
        <script language="js">
            let list_of_trees = {{ list_of_trees }}
            let to_be_highlighted_taxa = {{ to_be_highlighted_taxa }}
        </script>
    </body>
</html>
```

Listing 4.1: Phylo-Movies includes the computed trees as `JSON` in the `HTML` file, assigned in the variable `list_of_trees` and the to be highlighted taxa for each tree will be assigned to `to_be_highlighted_taxa`

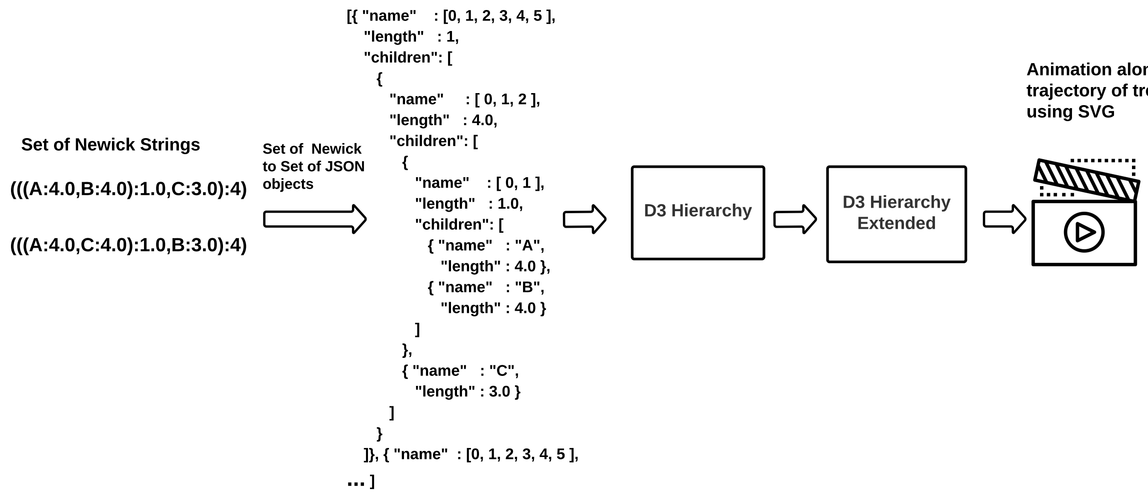
JSON Property	Type	Description
name	`string`	union of child names
length	`integer`	branch lengths
children	`<JSON>`	children of nodes

Table 4.1: The structure of a `JSON` Object which was transformed from a Newick. Here we can see the properties of the `JSON` object.

provides a tool to create interactive and animated content based on data by linking data to existing web page elements (see Fig. 4.5). Since we are using functions from `D3`, we need to use the appropriate data structures that the framework provides. All `D3` functions operate on `D3` Node instances, but our data is in the format of `JSON` (see for its properties Table 4.1).

But `D3` offers a function to transform a `JSON` object into a `D3` node instances, called `d3.hierarchy()` (Janert, 2019, Chapter 9). We convert each tree including the intermediate trees and the consensus tree into `D3` Nodes instances with `d3.hierarchy(tree)` function, where the variable `tree` is a `JSON` object. `D3` node instances have the following additional properties: parent, height, children and depth (see Table 4.2). The parent is every node's ancestor, the height is the total height from the root node and can be calculated from the length of all parents and depth is the number of parents one node has. The property children includes the descendants of one node. In our `JSON` object, this information is not included directly but can be calculated. In addition, each `D3` node instance may have associated data stored in `node.data()`. Note that access to `D3` Node instance properties is done via accessors (Janert, 2019, Chapter 9).

`D3` Nodes offer many utility functions; we shortly describe those functions which are used in Phylo-Movies. In this case, the "node" namespace defines the object we are operating on, which is an instance of the `D3` Node class. The function transforming the `JSON` to `D3` Node instances is embedded in the `TreeConstructor` module in Phylo-Movies. The `node.each()` accessor returns a list of nodes in level order. The `tree.nodes()` accessor returns an array of nodes connected to the root node. The `tree.leaves()` accessor returns the array of node that have no children in traversal order. The `tree.links()` accessor returns an array of tuples representing the branches from the parent to the children for each node. These tuples contain as a first element the outgoing node and as a second the ingoing one (`(the parent node,the child node)`). In `D3` the parent node will be referred to as sources, whereas the child will be referred to as targets. We mentioned some of the accessors but there a lot more functions available, which are listed at https://github.com/d3/d3-hierarchy/blob/main/README.md.


Set of Newick Strings
(((A:4.0,B:4.0):1.0,C:3.0):4)
(((A:4.0,C:4.0):1.0,B:3.0):4)
Set of Newick to Set of JSON objects
D3 Hierarchy
D3 Hierarchy Extended
Animation along a trajectory of trees by using SVG

```
[{ "name"   : [0, 1, 2, 3, 4, 5 ],
   "length"  : 1,
   "children": [
     {
       "name"   : [ 0, 1, 2 ],
       "length"  : 4.0,
       "children": [
         {
           "name"  : [ 0, 1 ],
           "length"  : 1.0,
           "children": [
             { "name"  : "A",
               "length" : 4.0 },
             { "name"  : "B",
               "length" : 4.0 }
           ]
         },
         { "name"  : "C",
           "length" : 3.0 }
       ]
     }
 ]}, { "name"  : [0, 1, 2, 3, 4, 5 ],
... ]
```

Figure 4.2: First, we convert the input data into `JSON` objects. This is because `D3` only accepts `JSON` format. Then, `D3` converts the `JSON` object into a `D3` hierarchy object. The `TreeConstructor` module extend the `D3` hierarchy object with coordinates so that our `TreeDrawer` module can later visualise our trees. The `TreeDrawer` module then animates the trees along a sequence in an `SVG` canvas.

D3 Node instances Property	Type	Description
name	`string`	union of child names
length	`integer`	branch lengths
children	`Optional<Array<D3Hierarchy>>`	children of nodes
parent	`Optional<D3Hierarchy>`	parent of node if not root
height	`integer`	zero for leaves, and the greatest distance from descendants
depth	`integer`	zero for root, increasing by one for each descendant

Table 4.2: Properties of `D3` Node instances

4.5 Assigning Coordinates To `D3` Nodes

For rendering trees, we need a fixed layout function that specifies how the trees should be drawn. The layout functions take a data set, compute appropriate sizes and positions for each node, and add them to the input data set as numeric member variables. For creating hierarchical layouts and storing data, `D3` provides two functions `d3.cluster()` and `d3.tree()` (Janert, 2019, Chapter 9). However, the length of branches is not accounted in `d3.cluster()` and `d3.tree()`, so we neglected them. Hence, after finding out that the internal `D3` methods `d3.cluster()` and `d3.tree()` are inadequate we had to implement a new layout function.

We implemented a module that computes the positions of all nodes and stores them in the `D3` Node instances, as mentioned in Chapter 2.1.1. As before mentioned this module is called `TreeConstructor`. After the calculation, we store the positions as values of x and y coordinates, angles, and radii as additional member fields in the `D3` Node instances. For the additional property members see Table 4.3. The extended `D3` Node Class with additional members will be referred to as `D3` Node Extended. Since we only add additional fields to the class `D3` Node, it is still part of the `D3` family. So we can still use the original function of

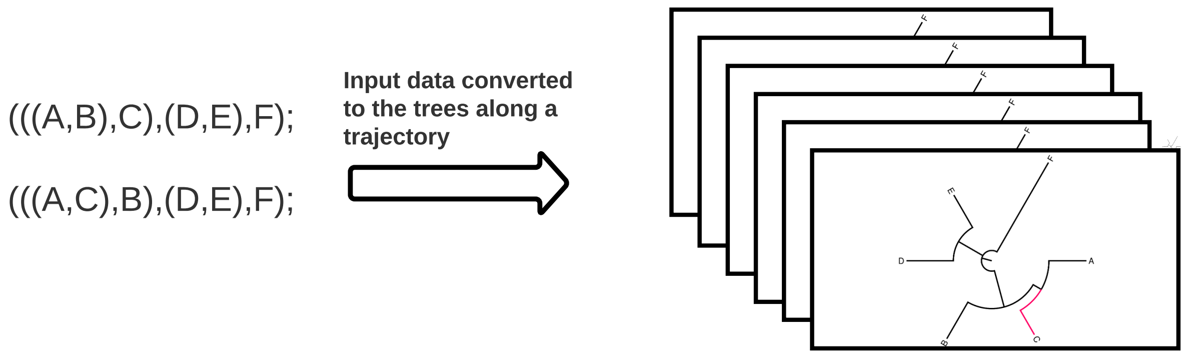


Figure 4.3: The user input will be taken and transformed to the animation of trees along a sequence.

Property	Type	D3 Node Extended Description
name	string	union of child names
length	integer	branch lengths
children	`Optional<<Array<D3Hierarchy>>>`	children of nodes
parent	`Optional<D3Hierarchy>`	parent of node if not root
height	`integer`	zero for leaves, otherwise the greatest distance from descendants
depth	`integer`	zero for root otherwise increasing by one for each descendant
x	`integer`	the computed x-coordinate of the node position
y	`integer`	the computed y-coordinate of the node position
radius	`integer`	the computed radius of the node
angle	`integer`	the computed angle of the node in the radial tree

Table 4.3: Properties of `D3` Node instances with additional fields marked in blue

the `D3` Node class. The `TreeDrawer` module then uses this described `D3` Hierarchy Extended Objects to generate graphical output by the given additional fields as the x, y coordinates. For an illustration of how the sequence and how the data types are handled and converted see Fig. 4.3.

4.5.1 `DOM` (Document Object Model)

A web page like Phylo-Movies is a dynamic data structure accessed, manipulated, and updated by the browser via the Document Object Model. The `DOM` represents the elements of a web page as a tree where the elements are the nodes (see Fig. 4.4). So `DOM` nodes can be, `circles` representing leaves in phylogenetic trees, or labels with the name of a taxon or the lines depicting branches; otherwise, in `HTML` documents, they can be `paragraphs`. The `DOM` is a object-oriented API which follows specific standards(Janert, 2019, Appendix C). The API exposes the tree like data-structure and allows one to query and manipulate it (Janert, 2019, Appendix C).

One accesses the `DOM` using different programming languages, but in our case, we consider `JavaScript` as a solution since it is the only programming language that runs in a web browser (client) (Grosskurth and Godfrey, 2005). We are using `D3` to apply the change on the `DOM` dynamically, which wraps the functionality of `DOM` and provides APIs, making it easier to implement a visualisation. `D3` thereby enables navigation

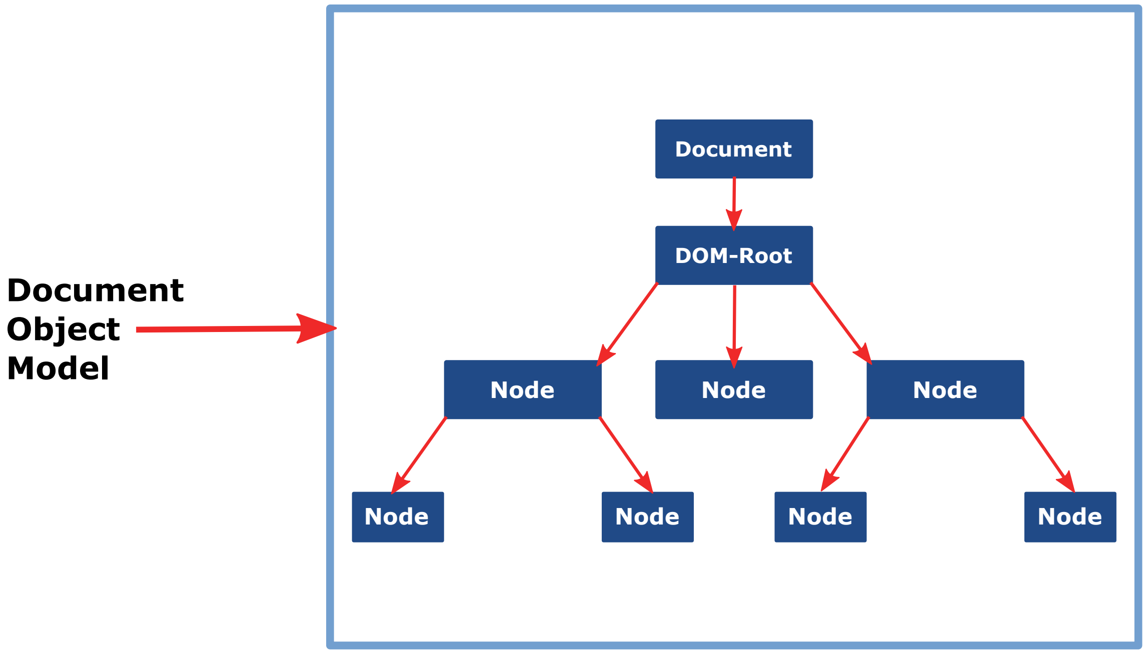


Figure 4.4: A representation how a DOM tree could look like.

through the structure and dynamic addition, modification, or deletion of elements and content (Janert, 2019, Appendix C).

4.5.2 SVG (Scaleable Vector Graphics)

Since the goal is to display trees graphically, we need a technology that is used to display and animate images in web browsers. We use SVGs (Brownlee and IAB, 2016) to draw and animate trees. SVG is a vector-based image format for creating two-dimensional graphics, that enables animation and interactivity (Ferraiolo et al., 2000). The format of SVGs is based on the hierarchical structure of XML (Ferraiolo et al., 2000; Rose et al., 2003). SVGs can be standalone but also included in web pages, as in Phylo-Movies.

In an SVG document the graphical elements can be styled, transformed, grouped and arranged (Ferraiolo et al., 2000). Modern browsers can render the input of SVG elements. Three types of graphic objects are depicted by SVGs: vector graphic shapes (straight lines and curves), pixel-based images, and text (Ferraiolo et al., 2000). Writing applications based on SVG is possible by using an additional scripting language like JavaScript. We wrote a JavaScript module which uses D3 to animate SVGs named TreeDrawer. SVG drawings can be interactive and dynamic. One can define animations declaratively or via scripts. An SVG document fragment may contain no element, it may be a very simple SVG document fragment containing a single SVG graphical element such as a rectangle, or a complex, deeply nested collection of lines, circles and labels such as our visualisation of trees (Ferraiolo et al., 2000).

SVG Paths

The graphical output of branches of trees in Phylo-Movies is implemented using SVG path technology. One can define with SVG clip paths a shape. It is based on the analogy of drawing on paper with a pen (Ferraiolo

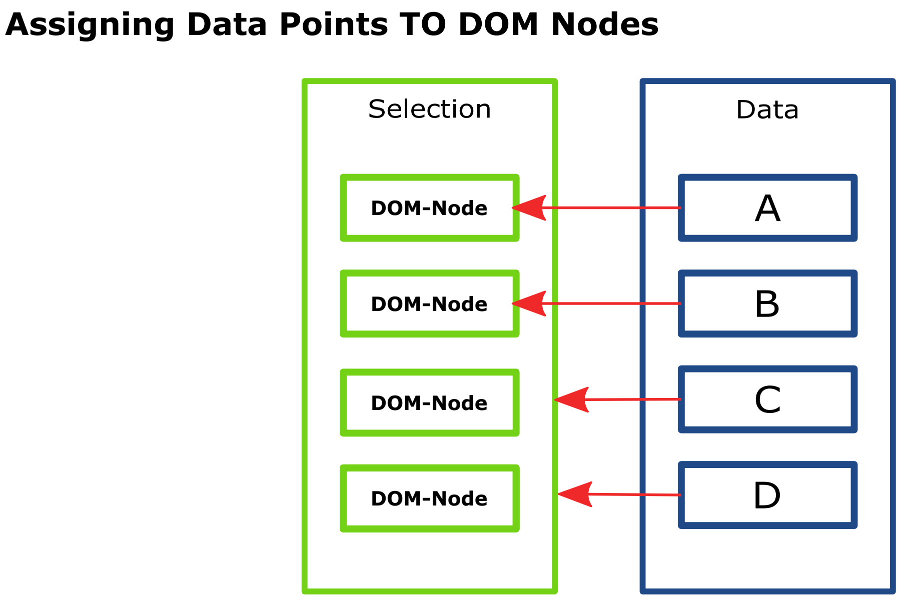


Figure 4.5: The letters which are in this case data points are assigned to selected DOM nodes. In this case the count of data points and DOM nodes is the same. So we see assign to every letter a DOM Node.

```
<svg>
    <path d="   M -174.37500000000014, -302.02635956982294
                A  348.75, 348.75
                0  0  0
               -348.75, 4.2709557120263946e-14
                L -558,    6.833529139242231e-14">
    </path>
</svg>
```

Listing 4.2: Example of a path in an SVG environment

et al., 2000). Hence, the current position is analogue to the position of the pen's tip (Ferraiolo et al., 2000). One can change the position of the pen. The outline of a shape is drawn by guiding the pen in straight lines or curves. A `path` is defined by inserting a path element into an SVG canvas, where the `d` property specifies the outline of the path data. All instructions are expressed as one character (for example, a "move to" instruction is expressed by an M). The path data contains the `moveto`, `lineto` as "L" (draw a straight line), `curveto` as "C" (draw a curve using a cubic Bézier), arc as "A" (elliptical or circular arc), and close path as "Z" (close the current shape by connecting to the last "move to" instruction) instructions (Ferraiolo et al., 2000). See Table 4.4 for more information about the instruction. The `d` properties of a clip-path can only be interpolated if the structure stays the same (Ferraiolo et al., 2000). Please see for an example, of a command generating a line in a SVG canvas depicting a branch in Listing 4.2. The output of the example in Listing 4.2, is depicted in Fig. 4.6. For a detailed description of the commands for generating a shape please refer to the Table 4.4. The commands are translated into a visualisation in a SVG canvas.

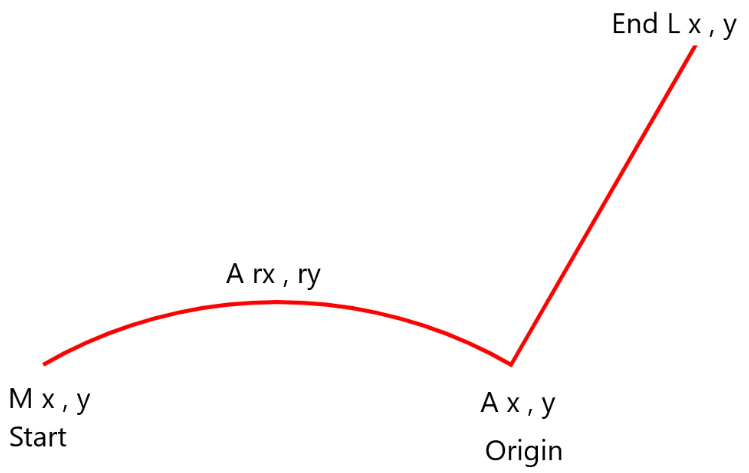


Figure 4.6: SVG Path Drawn, with Commands M x,y A rx,ry x-axis-rotation, large-arc-flag, sweep-flag, x, y L x, y

Command	Name	Parameters	Description
M	moveto	(x, y)+	Begin a path at the given absolute coordinates.;
L	lineto	(x, y)+	A line from the current point to the given (x, y) coordinate
A	elliptical arc	(rx ry)+	Elliptical arc from the current point to (x, y). Two radii (rx, ry) define orientation and size of the elliptical arc
		x-axis-rotation	indicates how the ellipse as a whole is rotated
		large-arc-flag	contributes to arc calculations
		sweep-flag	help determine how the arc is drawn
		x y	Target Point

Table 4.4: Commands and their explanation for defining a path in an SVG environment. The statements and explanations of the statements were taken from Ferraiolo et al., 2000

4.6 Assigning Classes, IDs and Aesthetics

Cascading Style Sheets (CSS) and JavaScript code rely on classes and IDs to identify and query for DOM nodes (Murray, 2017). The HTML/SVG class attribute is used to specify a single or multiple class names for a DOM element. In our use case, we distinguish between nodes and branches by labelling the nodes with the class "node" and the branches with "link". This makes it easier to access the element through selectors (see Listing 4.4). Selectors are methods to refer to DOM elements and apply changes on the specific DOM element as changing the colour or width. Selectors will be explained in detail in the further sections. IDs are assigned to a DOM element, but each ID must be unique, so there can only be one element per ID (Listing 4.3). Uniqueness in the context of web pages means that there can only be one element with a given ID in each HTML document (Murray, 2017, Chapter 1). As an example of a taxon, there can only be one label with the ID of the taxon's name.

Selectors are patterns used to select elements in HTML or SVG documents and manipulate them. D3 uses these selectors derived from CSS (Bostock et al., 2011). Selectors are a declarative language for applying

```
<div class="tree-1">
    <div id="node-1"></div>
    <div id="node-2"></div>
</div>
```

Listing 4.3: An example how unique ids are assigned to DOM elements.

```
<div class="tree">
    <div class="node"></div>
    <div class="node"></div>
</div>
```

Listing 4.4: An example how classes are assigned to DOM elements.

aesthetic features such as fonts and colours on DOM elements (Bostock et al., 2011). So that elements can be identified and then changes can be applied on them (see Listing 4.7) (Bostock et al., 2011). The elements of a HTML document corresponding to a selector are called the selector's subjects (Bos et al., 2005). There are many different types of selectors. Each selector matches any element that satisfies its requirements (Bos et al., 2005). JavaScript based selectors provide the dynamic ability, in addition to CSS, to compute styles based on user events or changing data (Janert, 2019, Appendix C).

Class selectors match the elements of any type that have been assigned a specific class (Murray, 2017). Selecting a class is done by a period followed by the name of the class, as shown in Listing 4.5, whereas when selecting an element with a specific ID, a hash mark precedes the ID name (see Listing 4.6). The DOM is dynamic; classes and IDs can be added and removed. Because of the dynamic behaviour, CSS rules are defined that only apply in certain scenarios (Janert, 2019, Appendix C). As mentioned, properties can also be used to change the style of an element. For changing the style, we have to assign them specific values. Each property expects different inputs. For example, the colour property requires a string defining the colour in hexadecimal or RGBA values; the margin property requires a measurement (pixel) for an example, see Listing 4.7.

As a reminder HTML and SVG attributes are special keywords within the opening tag to control the behaviour of the element as it can be seen in Listing 4.8. An HTML document consists of simple DOM elements, such as text elements or shapes in an SVG canvas. Some attributes are assigned to HTML and SVG elements by including property/value pairs in the opening tags as classes and IDs (Janert, 2019, Appendix C). See Listing 4.8) for an example. The property name is followed by an equal sign. Double quotes enclose the value and one can assign different attributes as IDs and classes to HTML or SVG elements (Janert, 2019, Appendix C). For example, one can assign a value for the radius or colour (fill) to a circle element(see Listing 4.8).

```
.node   /* Selects elements with class "node" */
.link   /* Selects elements with class "link" */
```

Listing 4.5: Here we are selecting DOM elements with the assigned class ”node” or ”link”. A period precedes the Classes

```
#tree-1 /* Selects element with ID "tree-1" */
#node-1 /* Selects element with ID "node-1" */
#link-1 /* Selects element with ID "link-1" */
```

Listing 4.6: ID selectors match a single element with a unique ID. A hash mark precedes the IDs. Here we are selecting DOM element with the assigned IDs with the prefixes "tree", "node", and "link". IDs have to be unique, whereas classes don't need to be unique.

```
.node{
    margin: 10 px;
    padding: 25 px;
    color: pink;
    font-family: Arial;
}
```

Listing 4.7: Assigning CSS properties. Here we assigning the DOM element with the assigned class different properties, as colouring the DOM element pink.

4.7 Using D3 to Dynamically Select DOM Nodes and Applying Aesthetic Properties

D3 offers functionalities to specify where on the DOM tree changes should be applied (Janert, 2019, Chapter 3). To associate individual data points with specific nodes of the DOM tree is an important aspect of how we visualise our tree. So that we can represent the change in position and colour of DOM nodes according to the data points they are linked to. The two selection methods are methods for querying specific DOM nodes, for example by their ID or class name and changing their appearance without iterating over their nodes. D3 uses Selectors used in CSS to identify document elements for selection (Bostock et al., 2011).

One of the functions is `d3.selectAll(selectorString)` (see Listing.4.9), which is a D3 selection method. Both the `d3.select(selectorString)` and `d3.selectAll(selectorString)` methods accept the previously described `CSS selector` patterns. Hence, we give the selection method as input a string the matched DOM nodes are returned.

Selections can be made with `d3.select(selectorString)`, which selects the first matched element, or by `d3.selectAll(selectorString)` which returns all matching DOM nodes. Example: We have a set of SVG circles in our SVG canvas labelled with the class name "leaf" and we want to select them, then we use `d3.selectAll(".leaf")` (see for a visual representation Fig. 4.7). If there is no matching element then an empty collection will be returned (Janert, 2019, Chapter 3).

After we have selected a group of elements, we can modify the appearance of those elements to visualise the change in the data. We want to change the position of the leaves, delete branches or the colour of branches. The D3 function `d3.attr(property, value)` sets DOM attributes and their value for an element.

```
<svg width="400px" height="110px">
    <circle class=leaf radius="20px" rx="0" ry="0" fill="blue" />
</svg>
```

Listing 4.8: Example of the creation a rectangle in an SVG environment. The SVG canvas has the width of 400px and height of 110px. In the canvas we are creating a circle with the radius of 20x.

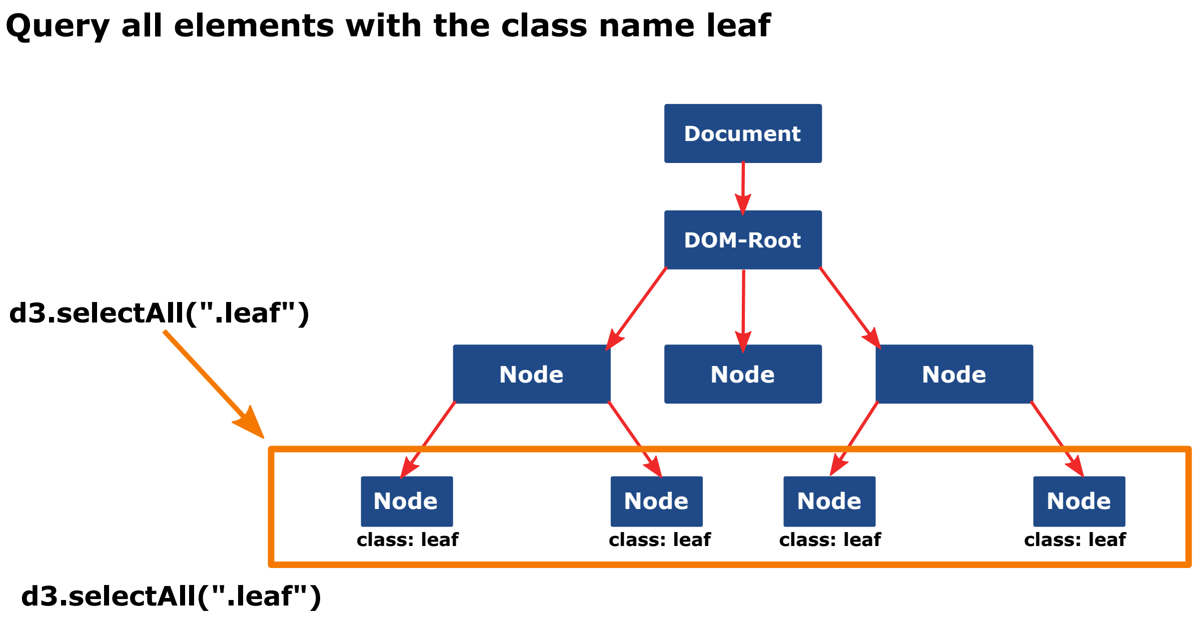


Figure 4.7: A representation how a selection works on the nodes of a DOM tree. We are only selecting the DOM nodes, with the class name ”leaf”.

We can use the `d3.attr(property, value)` function to access the properties or set the attribute on an DOM node. With the `d3.attr(property, value)` function we can set values like the ID or the class and a specific style attribute (Murray, 2017).

```
d3.selectAll('.leaf').attr("fill","red");
```

Listing 4.9: We are selecting all DOM elements with the class ”leaf” and assigning them the colour red.

After the selection of DOM nodes by their class or ID, we have to bind each data point to DOM nodes. For instance, we bind the leafs in the JSON data set to circles and labels. Whereas we bind the branches generated by the `node.links()` function to SVG paths (see Fig. 4.8). Recall that SVG paths are drawn lines in our canvas used for representing the ingoing and outgoing branches in our phylogenetic trees. SVG paths should not be mistaken with the paths in the tree space (Explained in Section.1.2.3), although they share the same name. Using the data binding functions of D3, we bind our data (nodes, branches) to associated nodes in our DOM. Data Binding assigns a data point to a particular DOM node (Janert, 2019, Chapter 3). Later, D3 associate each data point as identifier to apply mapping rules (see Fig. 4.5). Without this crucial step, we are incapable of referencing SVG elements to data points. The binding, drawing and construction of the animation are done by the module `TreeDrawer`, which provides several methods for updating branches and leaves and their labels. The further sections explain how the branches and leaves will be updated. The function `d3.data(dataPoints, key)` accepts a set of arbitrary data and assigns the entries of the set to selected DOM nodes. If no key function is specified, the elements in the array are assigned to DOM nodes based on their index in the set (Janert, 2019, Chapter 3). The first element in the set is assigned to the first selected DOM node, the second to the second selected DOM node, and so on (see Fig. 4.5). If a key function is

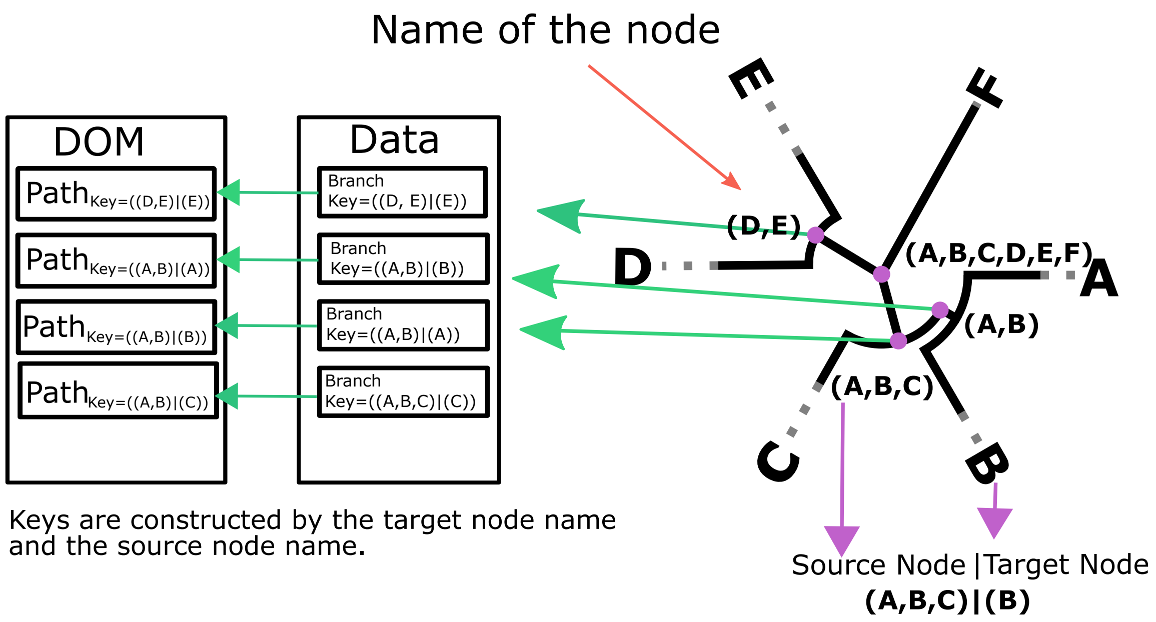


Figure 4.8: Binding the data of branches generated by the function `d3.links()` to `DOM` nodes. As an identifier for the assignment of data point to a `DOM` node, we are using the outgoing internal node's (source) name and the of the ingoing one (target).

defined, the default algorithm is replaced by a function that associates values of data points with `DOM` nodes. We use as a key the leaf set of a branch for the association for the path elements and the branches in the dataset, whereas for the leaves and the labels, we use the taxon name (see Fig. 4.8). Recall that the leaf set of a branch is a set of the names that the branch is leading to.

The `d3.data(dataPoints, key)` method returns a new selection object containing the elements that were successfully bound to entries in the data set (Janert, 2019, Chapter 3). The `d3.data(dataPoints, key)` method also returns the "enter" and "exit" selections. The "enter" selection contains the unmatched data points, while the "exit" selection contains the unmatched `DOM` nodes (Janert, 2019, Chapter 3). This different types of selections will be discussed in the further sections.

4.7.1 `Update Pattern`

Now we focus on representing the change in data sets through the `DOM`. We use the `D3 Update Pattern` (Janert, 2019, Chapter 3) in the module `TreeDrawer` for constructing the animation. If the number of data points (branches) and `DOM` nodes (paths representing the branches) are not equal, there will be an excess of unassigned data points or `DOM` nodes. The methods for creating and updating the branches is implemented in the module `TreeDrawer` in the function `updateLinks()`. The methods for updating the position of the circles representing leaves is implemented in `updatesLeaves()`.

The enter selection `d3.enter()` (see Fig. 4.9) for an illustration, returns placeholder nodes for data that has no corresponding `DOM` nodes in the selection (Janert, 2019, Chapter 3). We can use these placeholders to create `DOM` nodes for our unmatched data points. At the initialisation of our visualisation, there are no `DOM` nodes representing our branches or our leaf labels. So at the initialisation for every leaf and branch, we

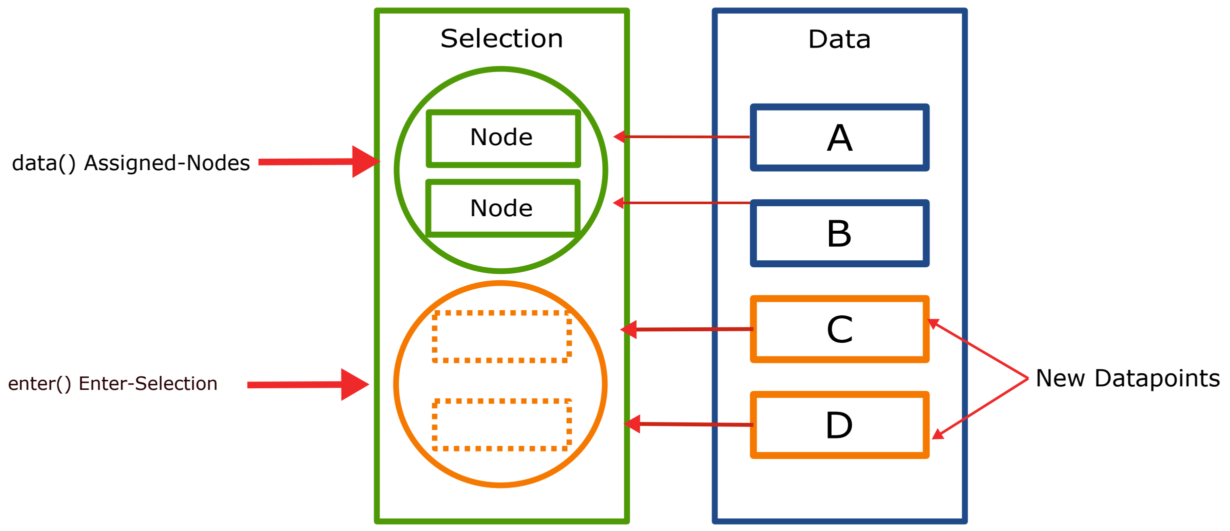


Figure 4.9: Example of how D3 binds data to a selection of DOM nodes when the number of data points is higher than the selected DOM nodes (Janert, 2019, Chapter 4).

create placeholders. This placeholder translates D3 into DOM elements for the leaf labels, and D3 will translate placeholders associated with branches into paths in the SVG canvas. We assign our paths the "d" attribute, which describes where they should be positioned and which nodes they should lead. We assign the x and y coordinates to the leaf labels. As a reminder, a detailed instruction on how the module `TreeConstructor` calculates the radial layout can be read in the Subsection 2.1.1. Branches absent in consensus tree two, which exist in intermediate tree two, will be inserted with zero branch length. A reminder of which branches are deleted and added along the transition from one tree to another can be read in Section 2.1.1.

The exit selection (`d3.exit()`) (Fig. 4.10) returns a set of DOM nodes that no longer have corresponding data points. The exit selection is mostly used in combination with the remove method to delete elements (Janert, 2019, Chapter 3). In Phylo-Movies, when we switch from intermediate tree one to consensus tree one, the number of branches in the trees of the data set may decrease. In Phylo-Movies, there are branches represented by paths, which are existent in intermediate tree one but absent in consensus tree one. The exit selection is then used to delete these no longer existing branches. The `d3.data(dataPoints, key)` method itself returns DOM nodes that have been successfully bound to data points by keys. These are for instance branches, which are not deleted or newly added, but their length is adjusted from the previous tree to one of the current trees. In the transition from full tree one to intermediate tree one the branches which are not existent in full tree two will be set to zero in the visualisation. Whereas the branches that are existent in full tree one and full tree two will be adjusted to the average. The successfully bounded branches by the keys (as before discussed are the leaf set of the branches) will be moved along the radial, to depict their relationship to their descendants.

With the `D3 Update Pattern` as it was described here, the pictures would be static, and the animation would correspond to the static exposition of the transition as in Fig. 2.3. But as we described in the previous

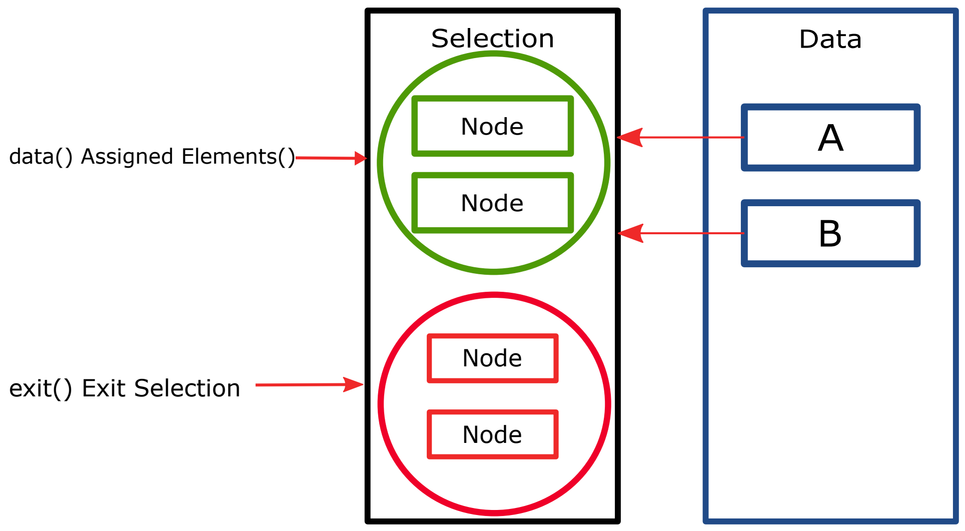


Figure 4.10: Binding data to a selection when the number of data points is not equal as the number of selected `DOM` nodes (Janert, 2019, Chapter 4).

Chapter 2 that we want to have an animation along the sequence of times to reflect the changes in our dataset because we want to depict in a flux changes in trees constantly over time.

Moving Leaves Branches Over a Time Span Smooth transitions visualise development over time and change the appearances of graphical elements (Janert, 2019, Chapter 3). We want our change to occur gradually rather than immediately. The `d3.transition()` function generates animations that visualise change gradually, using the `d3.attr(property, value)` function (Janert, 2019, Chapter 3). The changes are not executed immediately; they are applied over a time span (Janert, 2019, Chapter 4).

The function `d3.transition()` calls an interpolator function (Janert, 2019, Chapter 3). Given a starting value a and an ending value b, an interpolator takes a parameter t in the interval of a unit vector $[0, 1]$ and returns the corresponding interpolated value between a and b. Any intermediate state of a branch or leaf for the time point in the period is calculated. An interpolator will be called for every branch that changes its position, which calculates interpolated positions to the parameter t from the origin position a to its final destination b.

The `d3.transition()` function gives us the ability to visualise the movement when the coordinate of a component changes (Listing 4.10). A value to the `d3.duration(timeSpan)` function specifies how long the transition should last. After invoking the `d3.transition()` function the `d3.duration(timeSpan)` function has to be invoked to define the time span, how long the transformation should take. The duration function must be specified after the `d3.transition()` function, and the duration is specified in milliseconds (Janert, 2019, Chapter 4). So, for example, if 1000 milliseconds are defined for a transition, the animation will last one second (Murray, 2017).

We found a way to access the ”d” attribute of paths. Due to the given circumstance that the attribute

```
d3.select('#leaf-1')
.attr('rx','0')
.attr('ry','0')
.attr('radius','100px')
.transition()
.duration(1000)
.attr('radius', '250px');
```

Listing 4.10: We are changing the radius of a circle representing a leaf from 100px to 250px over the time span of 1 second. So in the animation, a circle would just grow bigger.

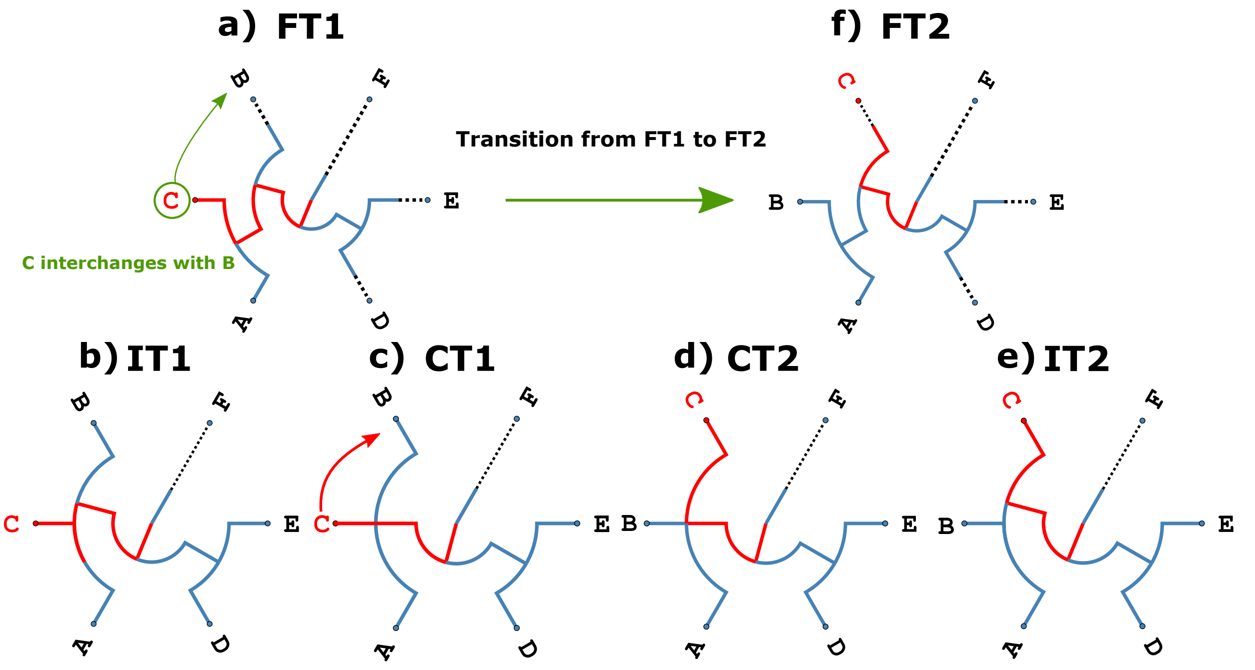
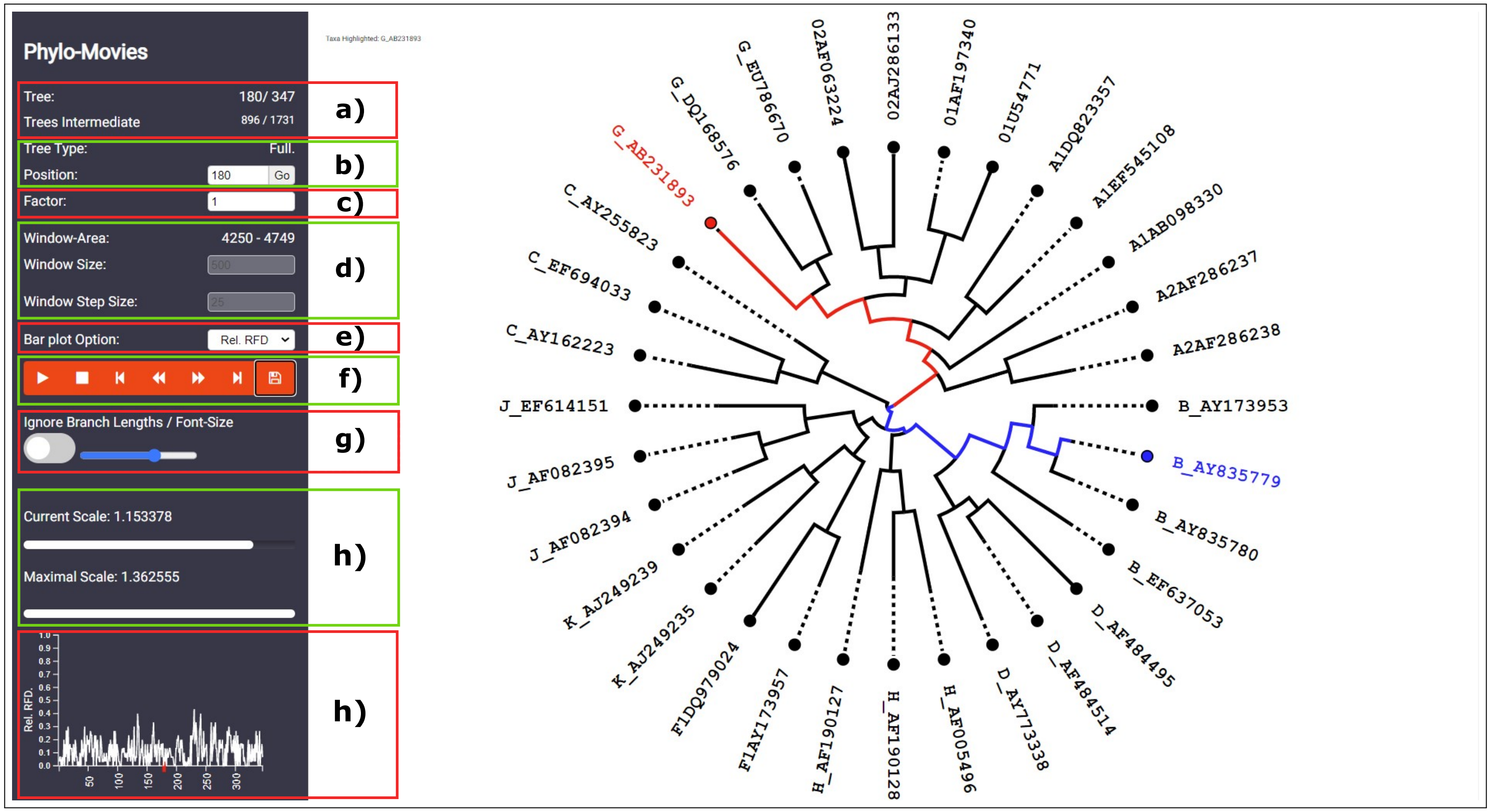


Figure 4.11: The first tree of the tree pair (a) where the transition is supposed to happen. (f) the Second tree of the tree pair. (b) - (e) The transition trees that are constructed by the program to simplify visualisation. This visualisation was generated with Phylo-Movies.

”d” of a path cannot be interpolated along a radial with the standard D3 interpolator as we wanted, we used the `d3.attrTween(property, interpolator)`, which is used to specify a custom interpolator during a transition. Then we can interpolate the values in the instruction in the ”d” attribute, so we interpolate the previous state of the branch and tween the branch along the radial to its new position. This method gives us precise, fluid animations showing how the branches’ path moves from the origin to a new destination. These functions are provided for building up animation; we try to maintain good graphics of trees during the transitions. So that we, as before mentioned, prohibit misunderstandings about the animation, which might disrupt the user’s attention.



Phylo-Movies

Tree: 180/ 347

Trees Intermediate 896 / 1731

Tree Type: Full.

Position: 180 Go

Factor: 1

Window-Area: 4250 - 4749

Window Size: 500

Window Step Size: 25

Bar plot Option: Rel. RFD

Ignore Branch Lengths / Font-Size

Current Scale: 1.153378

Maximal Scale: 1.362555

Taxa Highlighted: G_AB231893

Figure 4.12: A screenshot of Phylo-Movies. The visualised tree is from the HIV-1 dataset, which we will discuss in the following next sections. On the left corner (h) is the Robinson-Foulds chart (e). At (h), we can see the scale of the current tree and the maximum scale. (g) shows a checkbox where we can ignore the given branch lengths topology. Then aside from the checkbox (g), there is a slider for changing the size of the labels. At (f), there is the Button Bar. Then we have deactivated input boxes to see the given window size and step, which the user before defined (d). Furthermore, we see the genome region. Then, an input box is labelled with position (b) for hopping to a specific position in the tree sequence. At (a), we display which kind of intermediate or full tree we are displaying, and we show the position of the trees in the sequence.

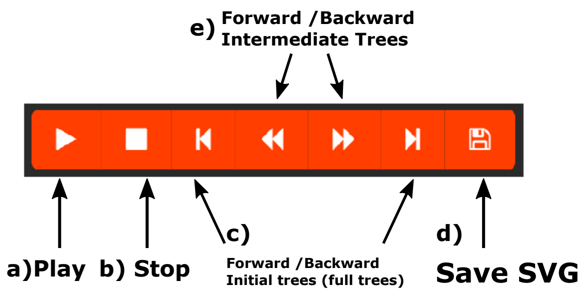

e) Forward /Backward Intermediate Trees
a)Play
b) Stop
c) Forward /Backward Initial trees (full trees)
d) Save SVG

Figure 4.13: Here we can see the application bar with different buttons for stopping and starting the animation, rewinding intermediate trees, and saving the current tree as `SVG` pictures.

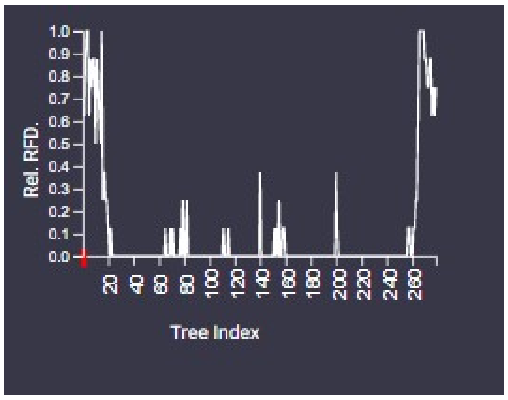


Figure 4.14: A line chart showing the Robinson-Foulds distance and scale so that the user can detect regions in the tree sequence that might be an interesting transition, indicating recombinations. The line chart is in Phylo-Movie in the bottom left corner.

4.8 Realisation of Phylo-Movies

We can provide a continuous animation transition from one tree to another using the mentioned technologies. The user can upload an order file to define the circular order of the trees.

The whole application Phylo-Movies is depicted in Fig. 4.12. The jumping taxa are highlighted with the colour red (see Fig. 4.11). We can see in Fig. 4.11 branches leading to the jumping taxa are also highlighted in red. The user can mark specific taxa and the branches leading to them, marked in blue, as it can be seen Fig. 4.12. Chapter 3 explains the highlighting algorithm. In combination with colouring the possible jumping taxa, the animation help track the change between two trees.

To move to the trees of interest, rewinding and starting the animation, we developed a user interface inspired by the old VHS player's design (see Fig. 4.13). The user can move between the intermediate steps and move between full trees. The rewinding of the intermediate steps allows the user to rewind the animation until one comprehends the difference. So the ability to stop, start and rewind to focus on specific parts and actions allow reinspection of specific transitions in the trees.

We will shortly describe which utilities are built in the bar on the left side, which can be seen in Fig.4.12. We provide a line chart showing the Robinson-Foulds distance and scale so that the user can detect regions in the tree sequence that might indicate recombinations (see Fig. 4.14). On the left corner Fig. 4.12h is the Robinson-Foulds chart, and also one can switch to the scale chart with the option input (Fig. 4.12e). Above (Fig. 4.12h) then, we can see the scale of the current tree and the maximum scale in the sequence of trees.

One level above, we see a checkbox where we can ignore the given branch lengths, to only analyse the tree topology(Fig. 4.12g). Then aside from the checkbox (Fig. 4.12g), we can a see slider for changing the size of the labels. One level above, we can see the Button-Bar (Fig. 4.12f) and as discussed before and explained in Fig. 4.13. Then we have deactivated input boxes to see the given window size and step, which the user before defined (Fig. 4.12d). Furthermore, we see the genome region that a user-generated the tree from. Then there is an input box labelled with the position (Fig. 4.12b). The user will use this input to hop to a specific position in the tree sequence. We display at (Fig. 4.12a) which kind of intermediate or full tree we are displaying, and we show the position of the trees in the sequence.

The next chapter will examine whether the method we have built works on different data sets to find interesting changes in a sequence of trees.

Chapter 5

Analysing Tree Sequences with Phylo-Movies

We have explained the steps of how Phylo-Movies creates and visualises trees. We want to apply the method built on different data sets and see if there are no animation problems or if the highlighting algorithms higlights false candidates. Such flaws could disturb the whole process of watching the animation or using the features such as going backwards or forwards in the animation or taking a snapshot of a coloured highlighted phylogenetic tree. In addition, we want to see if we can identify some known changes in the topologies of phylogenetic trees with different backgrounds. At first, we focus on a simulated data set. Heiko Schmidt provided the simulated data.

Then we proceed to a section where we try to find some indications of recombinations in the HIV-1 data set provided by Abecasis et al., 2007. At last, we are going to focus on a sequence of trees obtained from the heuristic tree search algorithm in IQ-Tree (Nguyen et al., 2014). We visualise and see if the operations such as NNIs are traceable in our visualisation of the generated tree sequences.

It is hard to incorporate animations in a written work, except using flipbooks (see Wells, 2013, Chapter 1), which we did not consider. Hence we describe what is happening in a transition from one tree to another and provide static pictures depicting the full tree before and the transition, with taxa highlighted our topology highlighting algorithm (Chapter 3.3). As a reminder, our algorithm suggests taxa which might have caused the topological changes like the for example taxon X in Fig. 5.3). We will also highlight the branches leading from the root to the jumping taxa for easier detection.

5.1 Simulated Scenario with Overlapping Recombinations

5.1.1 Introduction and Description of the Simulated Dataset

The dataset was a product of the unpublished study, which focused on developing a tool called `Visrec`. Using the information obtained from initial phylogenetic reconstructions from sliding windows to reveal recombinants and recombination breakpoints (Schmidt et al., 2013). The simulated dataset was provided by Heiko A. Schmidt.

The alignment sequences had been simulated using seq-gen (Rambaut and Grassly, 1997). The simulated alignment has the length of 7000 base pairs and eleven genomes ($A1$, $A2$, $B1$, $B2$, $C1$, $C2$, $D1$, $D2$, $O1$, $O2$,

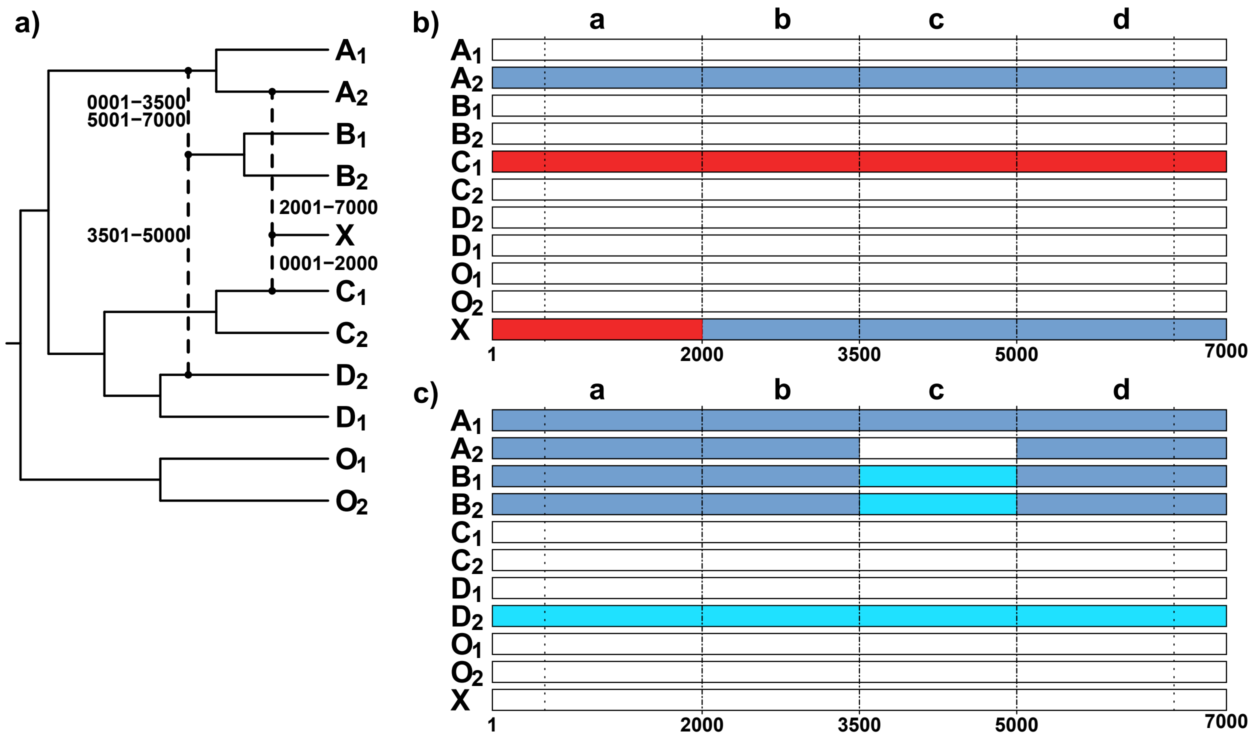


Figure 5.1: Visualisation of the relation of a simulated data set with genomes A1 - X. (a): Tree of the alignment with markings where changes happen (dotted lines). (b) Alignment of the genomes, the blue and red fragments show the relation of X to other genomes. (c) Alignment of the genomes, the blue and cyan fragments show the relation of B1 and B2 to other genomes. *A1 and A2 have switched patterns. (Adapted from Fig. 1.1 that was provided by H.A.Schmidt)

X). Fig. 5.1 shows the structure of the simulation along the alignment (Figs. are taken from the (Schmidt et al., 2013)), while Fig 5.2 shows the detailed evolutionary histories along the sequences. In the first 500 base pair of the sequences, the evolutionary rates are 50-fold increased with respect to the regions 501-6500, while last 500 base pair have a 50-fold reduced evolutionary rate. Due to the high 50 fold mutation rates in the first 500 base pair and the low mutation rates in the last 500 base pair, we ignore these regions for our further analysis (see Fig. 5.7b). Some branch lengths of the phylogenetic trees, which depict evolutionary rate, generated from the first 500 base pair are too long compared to the other branches. Hence, it is impossible to detect differences, whereas, in the last 500 base pair, some of them are about zero, indicating recombinations might lead to wrong conclusions. This also can be seen in the line chart depicting the scale

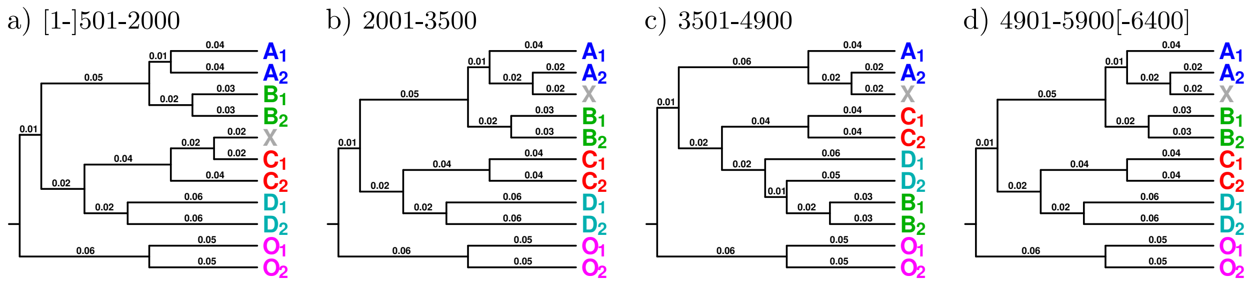


Figure 5.2: Evolutionary history of the simulated data set (courtesy of H.A.Schmidt)

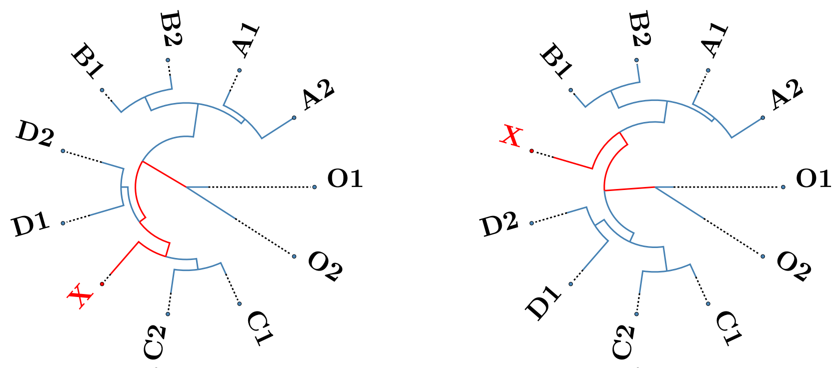


Figure 5.3: The tree pairs are from the simulated dataset. Trees we are from the Tree 79 (region 1825-2124) on the left side; Tree 80 (region 1850 - 2149) on the right side; Highlighted Taxon X jumping into the common ancestors of $\{A1, A2\}$ and $\{B1, B2\}$

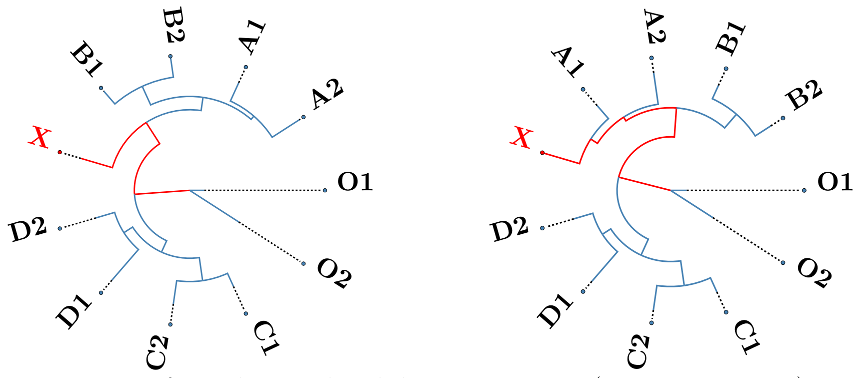


Figure 5.4: The tree pairs are from the simulated dataset. Tree 81 (region 1875-2174) on the left side; Tree 82 (region 1900-2199) on the right side; Highlighted Taxon X jumping into $\{A1\}$

of each tree, which is provided by the Phylo-Movies. The line-chart is in the bottom left corner of the application. Remember that the scale of each tree is the branch length of the longest branch leading from the root to a leaf.

The data set contains with the recombinant sequence X. Moreover we have four groups $\{A1, A2\}$, $\{B1, B2\}$, $\{C1, C2\}$, $\{D1, D2\}$ and an outgroup $\{O1, O2\}$. Group $\{B1, B2\}$ is also recombinant. Taxon X clusters with $C1$ in fragment "a" (1-2000bp) (Fig. 5.1b), while it clusters with $A2$ (2001-7000bp). $\{B1, B2\}$ group shares a common history with $\{A1, A2\}$ in fragment "a", "b", "d" (1-3500bp and 5001-7000bp) and with $D2$ in fragment "c" (3501-5000bp) (Fig. 5.1c).

The trees of the analysed sequence were reconstructed with overlapping sliding windows using IQ-Tree (Nguyen et al., 2014) with a window size of 300 and step size of 25. The file containing 280 trees in Newick file format is referred to as alltrees.tree and provided by Heiko Schmidt (Fig 5.1). The Newick file format is discussed in Section.2.1.1.

We analyse the sequence of trees and look at pairs of trees where the Robinson-Foulds distance is high between a pair of trees (see for example Figs. 5.3-5.6). At first, we are checking the pair of trees where the trees have a high Robinson-Foulds distance, indicating that there are big changes between the trees potentially signalling a recombination event (Fig. 5.7b). Phylo-Movies provides a diagram of the Robinson-Foulds distance for all input trees, which is located in the bottom-left corner on the web interface as mentioned in Section 4.8.

We know that the sequences are grouped based on some prior knowledge $\{A1, A2\}$, $\{B1, B2\}$, $\{C1, C2\}$, $\{D1, D2\}$ and $\{O1, O2\}$ (see Fig 5.1). There are peaks in the Robinson-Foulds distance plot for the trees at

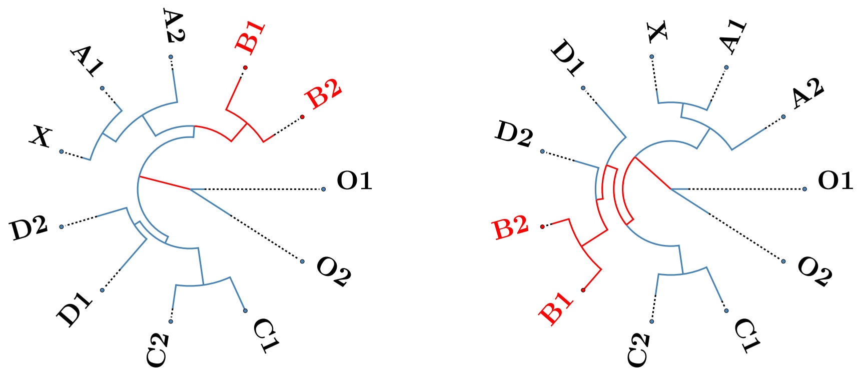


Figure 5.5: The tree pairs are from the simulated dataset. Full Tree 140 (region 3350-3649) on the left side; Tree 141 (region 3375-3674) on the right side; Highlighted Subtree $\{B1, B2\}$ jumping into $D2$

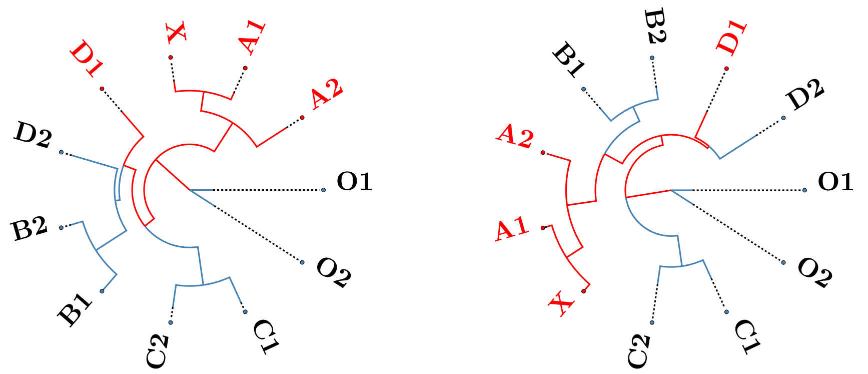


Figure 5.6: The tree pairs are from the simulated dataset. Full Tree 200 (region 4850-5149) on the left side; Tree 201 (region 4875-5174) on the right side; Highlighted Taxa $\{A1, A2, XD1\}$. Here we can that the highlighted taxa are misleading. These highlighted taxa can lead to a false assumption Fig. 5.1 the evolutionary history shows in fragment "d" that $\{B1, B2\}$ are recombination of $\{A1, A2\}$. We assumed that taxa $\{B1, B2\}$ would be highlighted but instead we see here that these are not the ones.

the position 79 (region 1825 - 2124) to 80 (region 1850 - 2149), then 81 (region 1900 - 2199) to 82 (region 1925 - 2224), 140 (region 3350 - 3649) to 141 (region 3375 - 3674). At last we will focus on the the tree 200 (region 4850 - 5149) to 201 (region 4875 - 5174) (see Fig. 5.7a). These candidate positions are further investigated for additional evidence of recombination.

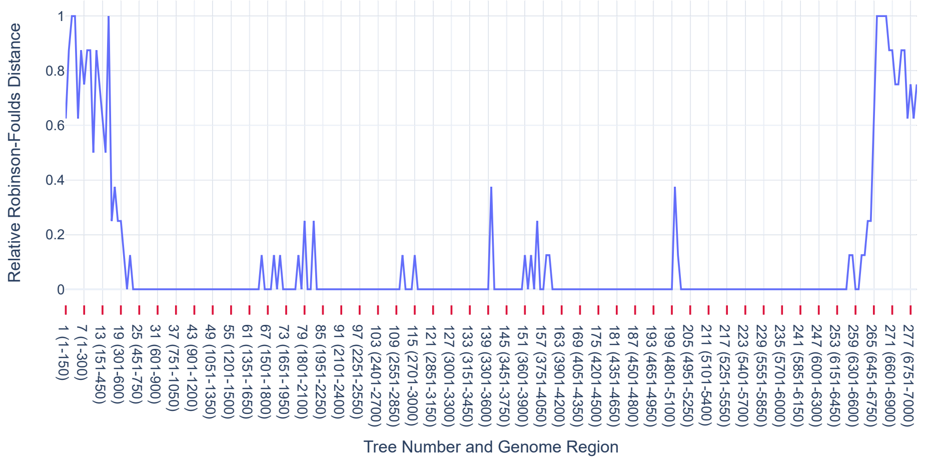
5.1.2 Observations Made in the Simulated Dataset with Phylo-Movies

In Fig 5.3 taxon X is highlighted and clusters with $\{C1, C2\}$ tree 79 (region 1825-2124). From tree 80 to 81 (region 1900-2199) taxon X jumps to $\{B1, B2\}$ and $\{A1, A2\}$. In the transition to tree 82 taxon X (region 1900-2199) forms a subtree with $A1$. This corresponds to taxon X being a recombinant of $A1$.

Next we observe trees 140 (region 3350 - 3649) and 141 (region 3375-3674), taxa $\{B1, B2\}$ jumps from subtree $\{X, A1, A2\}$ to $D2$. In Fig. 5.5 we can see that Phylo-Movies highlighted taxa $\{B1, B2\}$ and the branches leading to them. The detailed representation can be see in Fig. 5.5, where on the left the highlighted red subtree $\{B1, B2\}$ clusters with $\{A1, A2, X\}$ and then on the right it jumps into the group with $\{D1, D2\}$.

Later from tree 200 (region 4850-5149) to 201 (region 4875 - 5174) the subtree $\{B1, B2\}$ jumps from $D1$ to $A2$. This corresponds to $\{B1, B2\}$ being a recombinant of $\{A1, A2, X\}$ and $D1$, while $\{C1, C2\}$ moves toward the root, possibly due to the conflicting signal in the window. In Fig. 5.6 we can see the taxa $\{A1, A2, X, D1\}$ being highlighted. Because the highlighting algorithm indicate that $D1$ jumps into $D2$, repairing the subtree with taxa $\{D1, D2\}$. Then it indicates a second jump, which is that the subtree

(a)



(b)

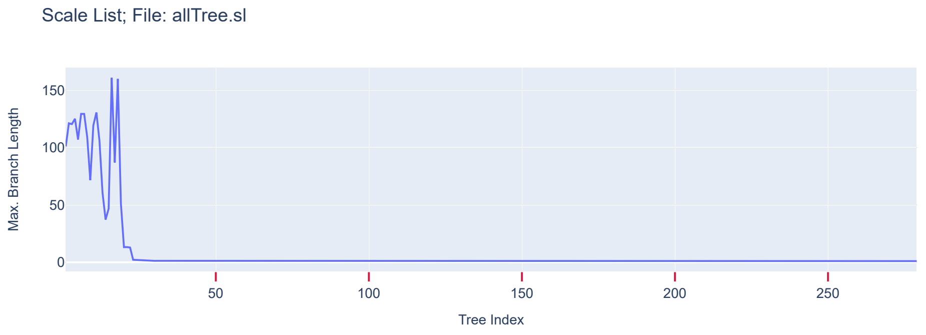


Figure 5.7: The top plot depicts of Robinson-Foulds distances along the sequence of trees. At (a) the x-axis, the first number denotes the position of the tree, the numbers in the bracket denote the genome region. The last (b) represents the scales for each tree. The scales represent for each tree the highest distance between a leaf and the root. There are peaks in the Robinson-Foulds distance plot at the pair of trees at the position 79 (region 1825-2124) to 80 (region 1850-2149), then 81 (region 1875- 2174) to 82 (region 1900-2199), 140 (region 3350-3649) to 141 (region 3375-3674) and the tree at the position 200 (region 4850-5149) to 201 (region 4850-5149). These candidate positions are further investigated for additional evidence of recombination events, which we want to detect.

$\{A1, A2, X\}$ jumps to $\{B1, B2\}$.

5.1.3 Discussing the Observations Made on Simulated Dataset

We have identified three major topological changes of the simulated scenario in Fig 5.2. The first was detected at tree 79 (region 1825-2124)-80 (region 1900-2199) , 81 (region 1900-2199) - 82 (region 1925-2224), 140 (region 3350-3649) - 141 (region 3375-3674) and 200 (region 4850-5149) - 201 (region 4875 - 5174). Phylo-Movies has marked that taxon X as a recombinant of $A1$ and $\{C1, C2\}$ and that $\{B1, B2\}$ are recombinant of $\{D1, D2\}$ and $\{A1, A2\}$.

The recombination events that we detected are coinciding with recombination breakpoints that we can see in Fig. 1.1. In Figs. 5.3 -5.4 coincides with the recombination breakpoint in Fig. 1.1b from fragment ”a” to ”b”. Then we can see that the recombination breakpoints in Fig. 1.1c in fragment ”c” coincides with the recombination events detected with Phylo-Movies depicted at Fig 5.5. Fig. 5.6 depicts the recombination breakpoint in Fig. 1.1c from fragment ”c” to ”d”.

At the transition from tree 200 to 201 the highlighting of the algorithm is counter intuitive. Because in case of changes of topologies there can be several different solutions, reasoning change of topologies. We can see in our depiction that the taxa $\{A1, A2, X, D1\}$ are highlighted. These highlighted taxa can lead to a false assumption Fig. 5.1 the evolutionary history shows in fragment ”d” that $\{B1, B2\}$ are recombination of $\{A1, A2\}$. We assumed that taxa $\{B1, B2\}$ would be highlighted but instead we see in Fig. 5.6 that these are not the ones. But except the case at the trees at the position of 200 and 201 the taxa which are recombinants depicted in the fragments ”a”,”b”,”c” of Fig. 5.1 were highlighted.

Therefore along the sequence from 78 (region 1800-2099) until the region of 82 (region 1925-2224), we saw that potential recombinations could be found in two ways. Either one big jump with a huge spike in the Robinson-Foulds distance chart (Fig. 5.7a) indicates a big jump in which the recombinant taxa directly jump, or we see multiple little jumps. In this case, the recombinant jumps at first to the most recent common ancestor of the taxa that he inherited his traits from and then in the later by moving forward, it jumps to a taxon or taxa. Hence, intermediate states are visible because of the underlying principle of the sliding window approach. This is because the sliding window approach does not accurately detect the breakpoint in the genome where recombination occurs. One of the methods that could help in accurate region detection would be to generate trees with smaller window sizes, resulting in more trees that would be much more time consuming to observe. Thus, an important task in region detection would be to find the optimal size of the sliding window so that the accuracy of the alignment is accurate enough. But when the windows size is too small, the sequence of trees becomes so long that it is feasible to observe all the trees.

As we have seen in in our results chapter 5 the analysis of multiple sequence alignment using a overlapping sliding window, then animating this sequences of trees and highlighting the jumping taxa led to the fact that we could detect most of the recombinations. Except in the case of the transition at tree 200 to 201, here we are seeing misleading candidates because the highlighting algorithm takes other solutions as valid for the reasoning of the topological change.

But all in all, we can say that the animation and the recognition of recombination, in this case, seemed very easy to us. Finding the interesting position of the recombination was also easy with the help of the Robinson-Foulds distance, and the graph showing the maximum branch length for each tree helped us to know which region of trees did not pay off to analyse.

5.2 Analysing a Biological HIV Dataset

Next, we want to analyse a sequence of trees generated along an alignment of HIV-1 genomes. HIV-1 is genetically diverse and is classified mostly in three groups, namely M, N, and O (D. L. Robertson et al., 1999). Group M is responsible for the global HIV-1 pandemic (Van Heuverswyn and Peeters, 2007). This group can be classified into subtypes A, B, C, D, F, G, H, J, and K, then in subsubtypes, and several circulating recombinant forms (CRFs) (D. L. Robertson et al., 1999).

The analysed dataset is based on the sequences used by Abecasis et al., 2007, where they discuss that the subtype G is a circulating recombinant form. Based on the sequence accession numbers the genome alignment was retrieved from the Los Alamos HIV database (Kuiken et al., 2003, https://hiv.lanl.gov).

The dataset used here contained 82 taxa. The tree sequence was generated by Heiko A. Schmidt using RecDetec (Kutmon, 2008) to split the alignment into overlapping windows of size 500 and step size of 25 from these trees which have been reconstructed using IQ-Tree 2 (Minh et al., 2020) with the GTR model

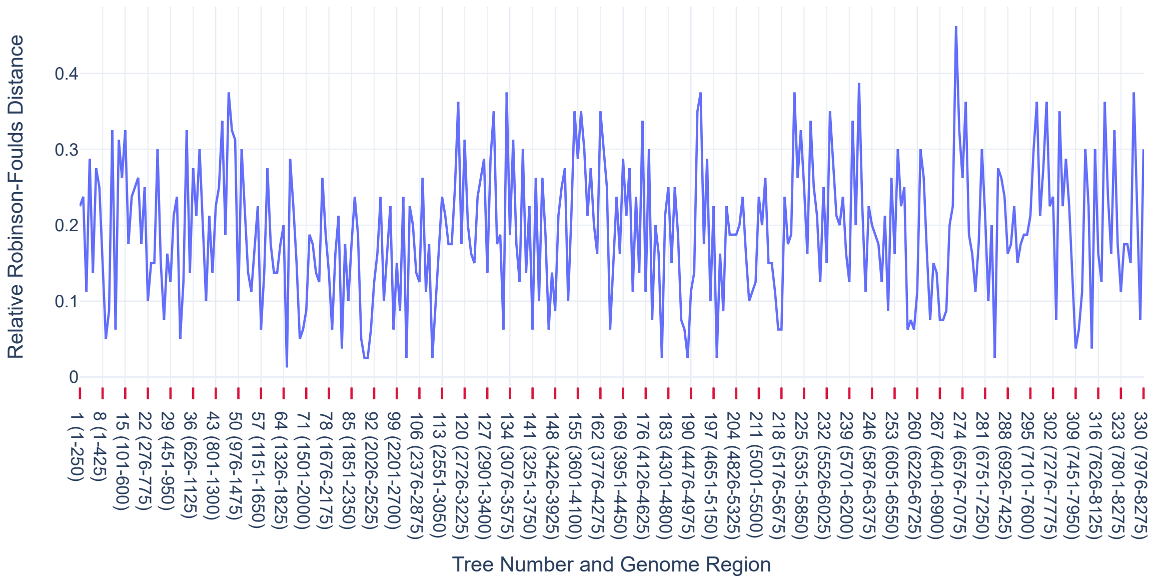


Figure 5.8: The plot shows the Robinson-Foulds distance for the trees generated from the Abecasis et al., 2007 dataset. On the x-axis, the first number denotes the tree position, the numbers in the brackets denote the genome region.

and default parameters.

The leaf labels in the trees show the subtypes *A1*, *A2*, *B*, *C*, *D*, *F1*, *F2*, *G*, *H*, *J*, *K* and *O1* and *O2*. *O1* and *O2* denote *CRF01_01AG* and *CRF01_02AG* respectively.

Abecasis et al., 2007 discuses that two plausible hypotheses could explain the origin of *CRF02_AG* and subtype *G*. The first hypothesis is that subtype *G* is a recombinant of *A*/*J*. The second hypothesis is that subtype *G* is recombinant of $CRF02\_AG/J$, and $CRF02\_AG$ is actually a pure non-recombinant parental strain. We will be focusing on the second hypothesis in on our analysis.

It was shown in the work of Abecasis et al., 2007 that *G* is clustering with the subtype *A* in the regions from 4316 to 5162. *G* clustered significantly with subtype *J*, in the regions from 5577 to 6083. Therefore, we will try to find evidence about the recombination event of about *J* clustering with the subtype *G*. We provide a Robinson-Foulds chart, to see regions where high topological changes are (Fig5.8).

5.2.1 Discussion and Observations on the HIV-1 Dataset with Phylo-Movies

Now we will describe the results of the analysis of the sequence of trees.

We begin our analysis at tree 219. In tree 219 (region 5225-5724) the subtype *J* is not clustering with any other subtype but rather toward the root of the tree Figs. 5.9-5.10. Subtype *G* is clustering with *O1*. Then in the transition from 219 to 220 (region 5250-5749) the subtype *J* moves to and forms a subtree with subtype *G* (Figs. 5.9-5.10), whereas the recombinant forms *O1* are jumping outside of *G*. As it can be seen in Figs 5.9-5.10 the subtype *G* and *J* is highlighted in red by the highlighting algorithm, indicating the changes in the subtrees. Other taxa are also highlighted because of the changes in the topologies underlying the conflicting signals in the alignment and the construction of the trees. Hence we see the same relations

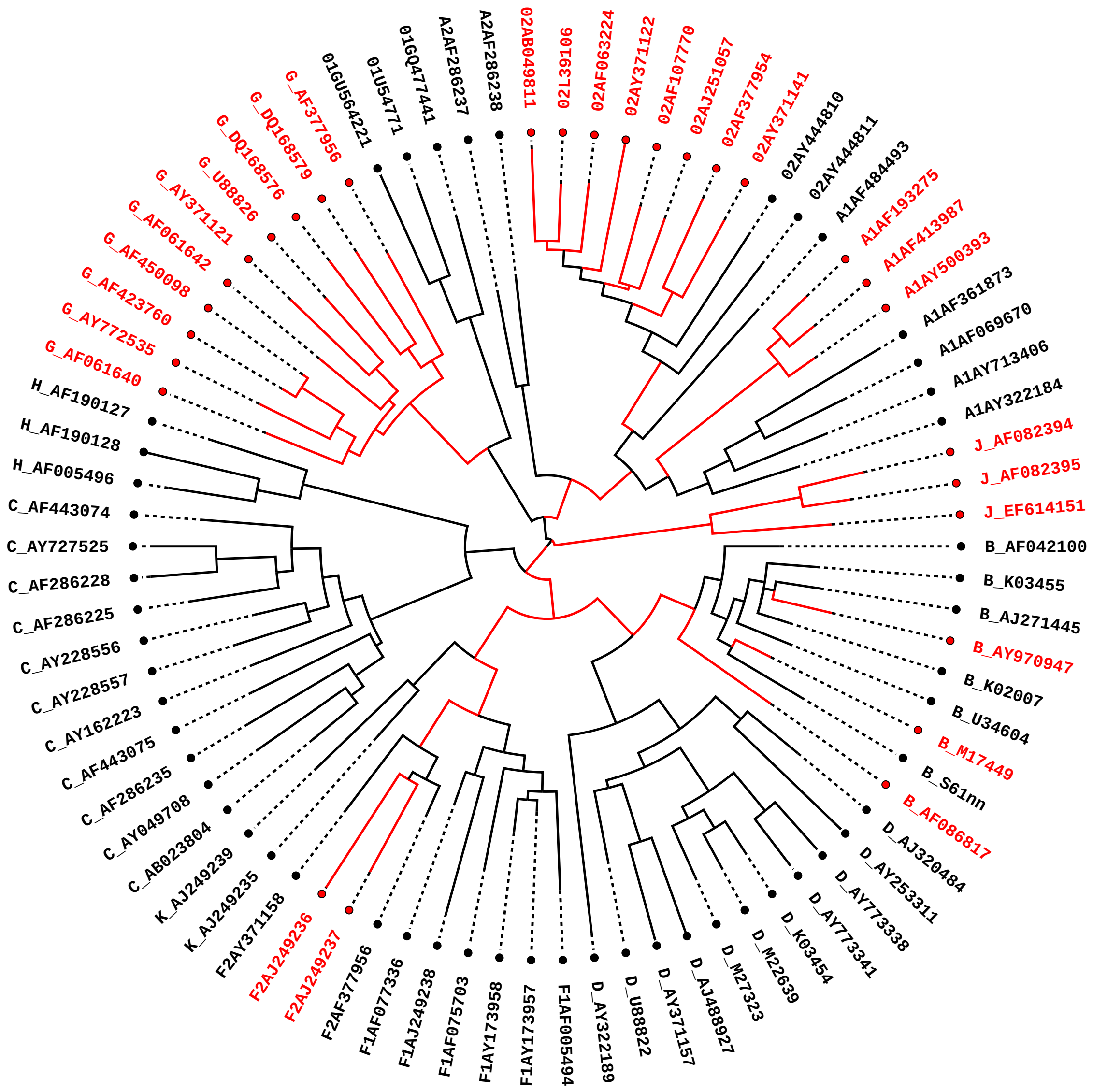


Figure 5.9: Trees from the HIV-1 dataset: Full Tree 219 (region 5225-5724) on the top. The first two letters denote the HIV subtype, followed by the accession number.

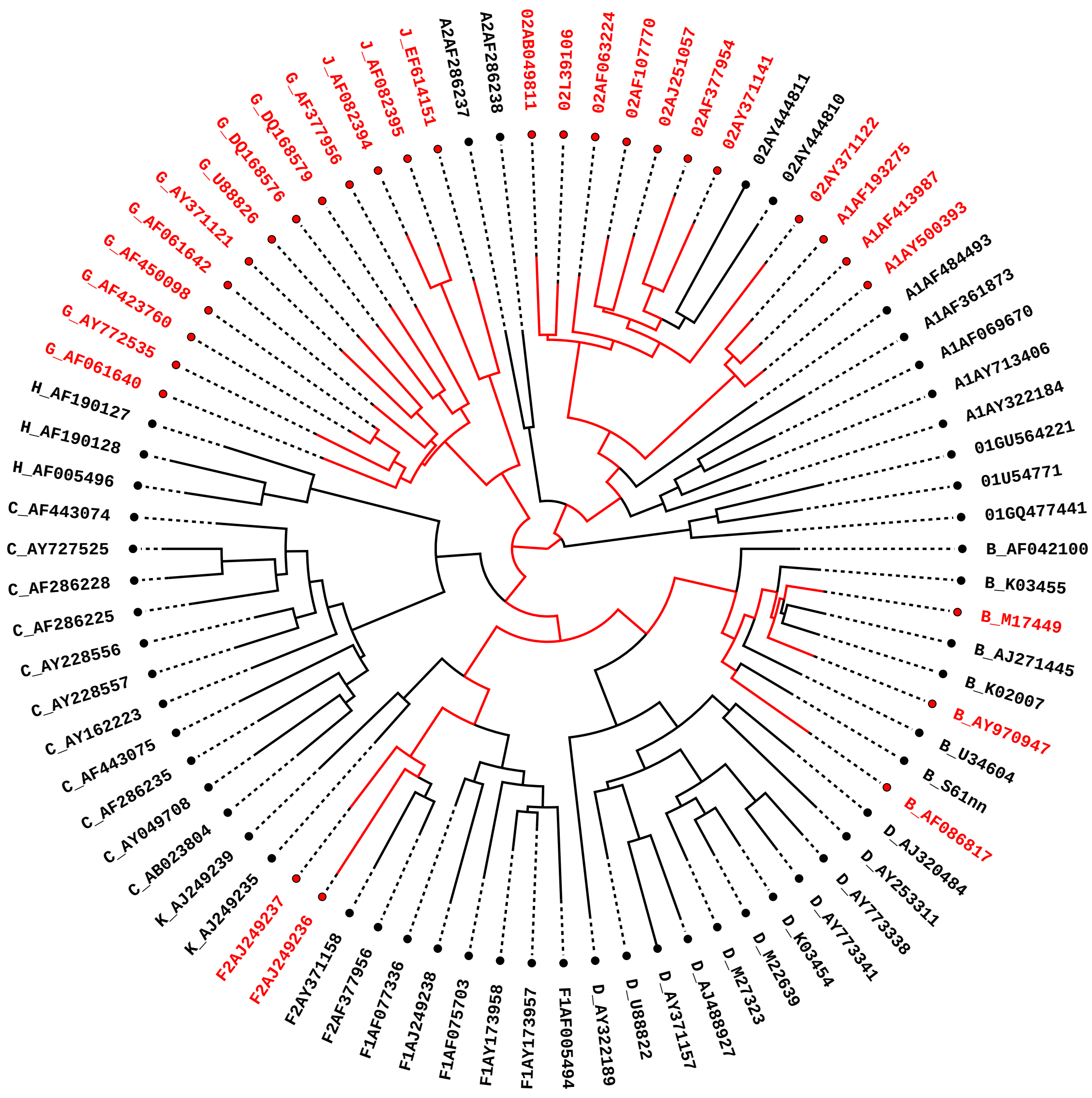


Figure 5.10: Trees from the HIV-1 dataset: Full Tree 220 (region 5250-5749). The first two letters denote the HIV subtype, followed by the accession number.

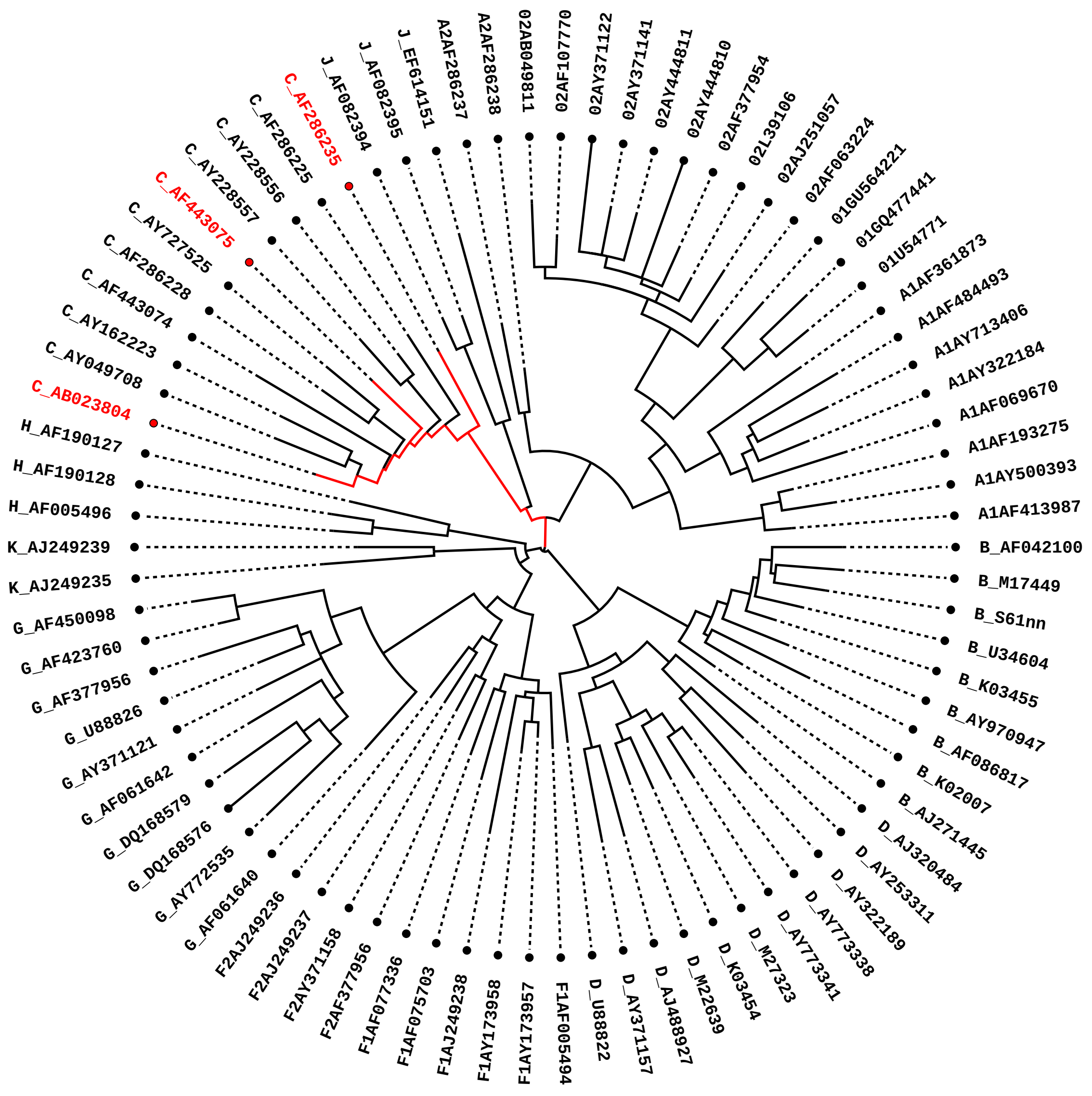


Figure 5.11: Trees from the HIV-1 dataset: Full Tree 32 (region 550-1049) The first two letters denote the HIV subtype, which is followed by the accession number.

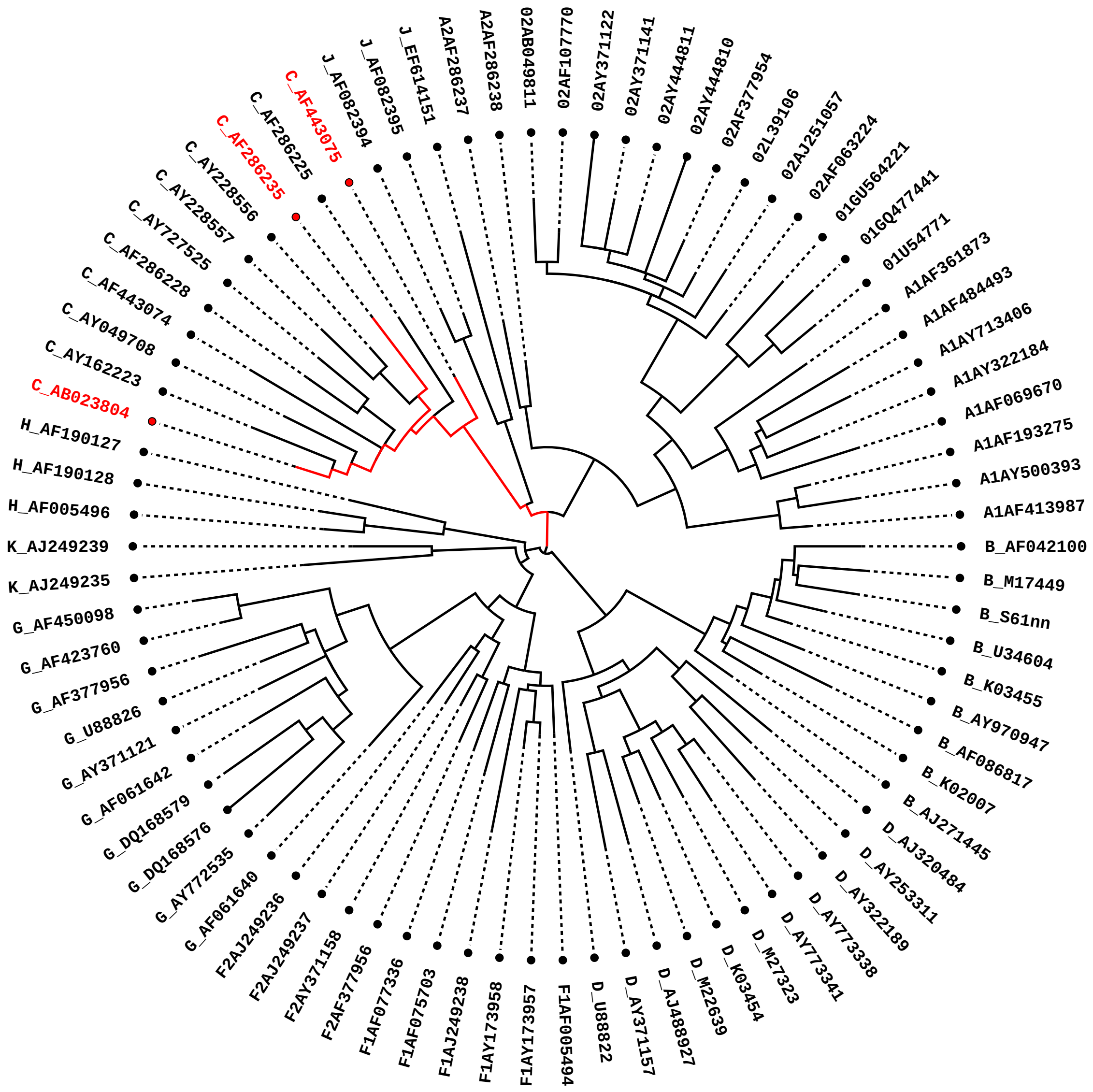


Figure 5.12: Trees from the HIV-1 dataset: Full Tree 33 (region 575-1074) on the top side; The first two letters denote the HIV subtype, which is followed by the accession number.

with G and J seen by Abecasis et al., 2007 at regions 5577 to 6083 but we see the jump of subtype J to G at the region at the position of 5225-5724.

Our strategy was to focus on the regions described in Abecasis et al., 2007 and to see if we could find the same relations that were mentioned. The rewind function, where we can move back and forth between trees, made it easier to see the detailed differences. The analysis of the HIV dataset has been difficult due to a large number of changes in the topology of the trees, along the genomes. However, we were able to find regions of interest that are also described in Abecasis et al., 2007. It is known that there are a lot of recombination and mutations of HIV (Fischer et al., 2021), leading to unstable backbones of the trees due to conflicting and missing signals. These conflicting signals are leading our highlighting algorithm to detect more changes than we would have expected.

Furthermore, we will focus on a pair of trees with subtle changes. As we can see in tree Figs. 5.11-5.12 at tree 32 (region 550-104) the taxa *C_AB023804*, *C_AF443075* and *C_AF286235* is highlighted. From tree 32 (region 550-1049) to 33 (region 575-1074) we can see some rearrangements in the subtypes *C*. We wanted to show that small changes in subtrees can be traced and analysed, which can also play an important role in analysing such sequences of trees.

Thus, Phylo-Movies should be tested more even in real data to see whether the detection of the jumping taxa also lead to truthful statements in these cases.

5.3 Analysing a Sequence Generated by a Tree Search Algorithm

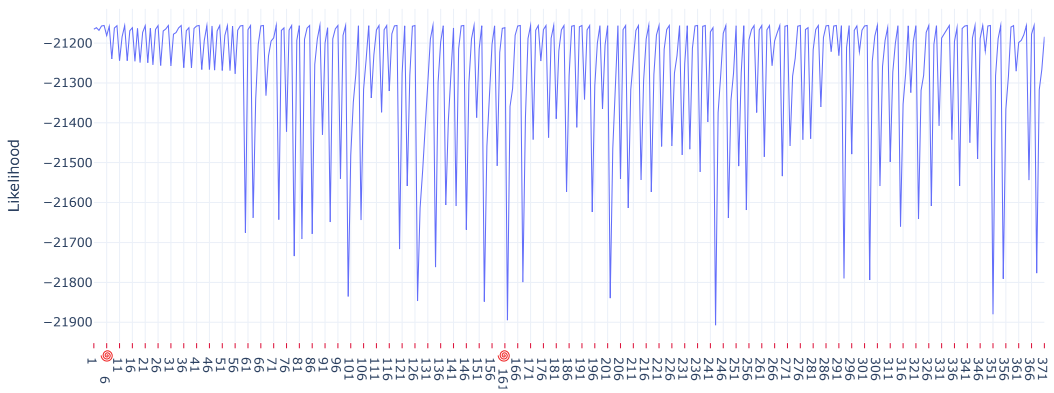
Here we will analyse another source of tree sequences. Generating phylogenetic trees by likelihood (ML) is a well-known tool in bioinformatics (Felsenstein, 2004). In this method, the branch lengths and tree topology will be estimated (Nguyen et al., 2014). Likelihood tree searches apply different types of local tree rearrangements such as Nearest Neighbour Interchange (NNI), subtree pruning and regrafting (SPR), or tree bisection and reconnection (TBR) to improve the current tree (Felsenstein, 2004). IQ-Tree (Nguyen et al., 2014) combines methods such as hill-climbing and stochastic NNIs to find the optimal tree. NNIs is an operation for rearranging two subtrees across an internal branch (Felsenstein, 2004). In addition, IQ-Tree uses Fast NNI, which means it applies several NNI at once to improve the likelihood. But only NNIs will be taken into account, which is not contradicting each other concerning the topology. The hill-climbing algorithm optimises the likelihood by applying NNIs that improve the likelihood.

We plotted the likelihood of a sequence of trees generated by a tree search done by IQ-Tree Fig. 5.13a and the relative Robinson Foulds distance along the sequence of trees in Fig. 5.13b.

Where the likelihood in Fig. 5.13a raises, IQ-Tree used hill-climbing algorithms. The hill-climbing algorithm can get stuck in a local optimum. Hence applying a set of random NNIs on the current best tree candidates allow the escape from local optima. As a consequence, there will be a drop of the likelihood, where afterwards the hill-climbing algorithm will operate on the current trees to generate a new, improved tree topology Fig. 5.13a, which might have better likelihoods than the previous ones.

For analyses of visualising tree search sequences, we used phylogenetic reconstruction from the `mtDNA.phy` provided by Lemey et al., 2009, Chapter 6. The alignment consists of mitochondrial DNA sequences from birds, reptiles, several mammals and three sequences from lungfish sequences. A sequence of trees is generated during the tree search, which Phylo-Movies can analyse. We started the tree search, setting the candidate set to be kept to 1. Along with this tree search, IQ-Tree generated 370 trees. In this analysis of the sequence of trees, we will be looking at pair of trees concerning their likelihood values and the Robinson-Foulds distance.

(a)



(b)

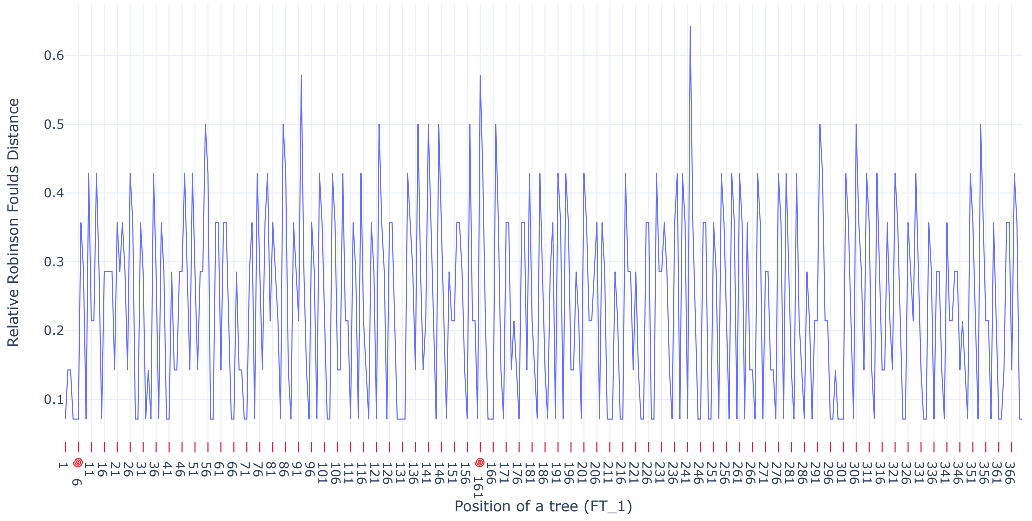


Figure 5.13: Here we can see (a) the Robinson-Foulds distance for each tree pair and (b) the likelihoods of the trees generated during the tree search. In (a) the log likelihoods for each tree is shown, while (b) shows the relative Robinson-Foulds distance for the tree at that position in the tree search and its successor. The likelihoods are obtained from IQ-Tree while the relative Robinson-Foulds distance were calculated by Phylo-Movies. Improvements of trees show increasing likelihoods. The decline in likelihood shows moving to a worse tree caused by the application of random NNIs. The red curls are indicting the position we are focusing on.

At first, we focus on regions where the likelihood of a tree is high, and the likelihood of the following neighbouring tree remains almost the same. In this case, we assume that in this tree pair, the changes in the tree will be small. So we will focus on regions where the Robinson-Foulds distance is low, and the likelihood remains almost the same.

Afterwards, we will focus on a sequence of trees, where we will observe the change in the trees generated from a hill-climbing NNI to a random perturbation NNI. So we focus on positions in the sequence where the likelihood of a tree is high and the likelihood of the successor is low. We assume that the Robinson-Foulds distance correlates with the likelihood values (Fig. 5.13a and Fig. 5.13b). So when the likelihood rises a bit, we assume that a tree's topology will also not be affected by a hill-climbing NNI. Whereas when the likelihood drops caused, for instance, by a random NNI, then the Robinson-Foulds distance will also be high.

5.3.1 Observations Made on the Tree Search Sequence

We will look at the transition from the pair of the tree at the position of 6 (Likelihood: -21181.12125) and 7 (Likelihood: -21158.16664). We also assume that if the likelihood and the corresponding Robinson-Foulds distance is small, it might indicate small topology changes, reasoned by a hill-climbing NNI. Then we focus on the sequence of trees, where we can observe the change from the hill-climbing NNI to a random NNI.

First we focus on pair of trees focusing at positions 6 – 7. This pair, as we can see in Fig. 5.13a, has a slight difference in likelihood, so we assume that the change will be small and that the changing factor was a hill-climbing NNI. So in Fig. 5.14 we can see that the taxon {Sphenodon} is highlighted as clustering with the taxon {Turtle}. So taxon {Sphenodon} jumps out of the cluster changes the position with the subtree {Bird, Crocodile}. In tree at the position 7 {Bird, Crocodile} clusters with {Turtle}.

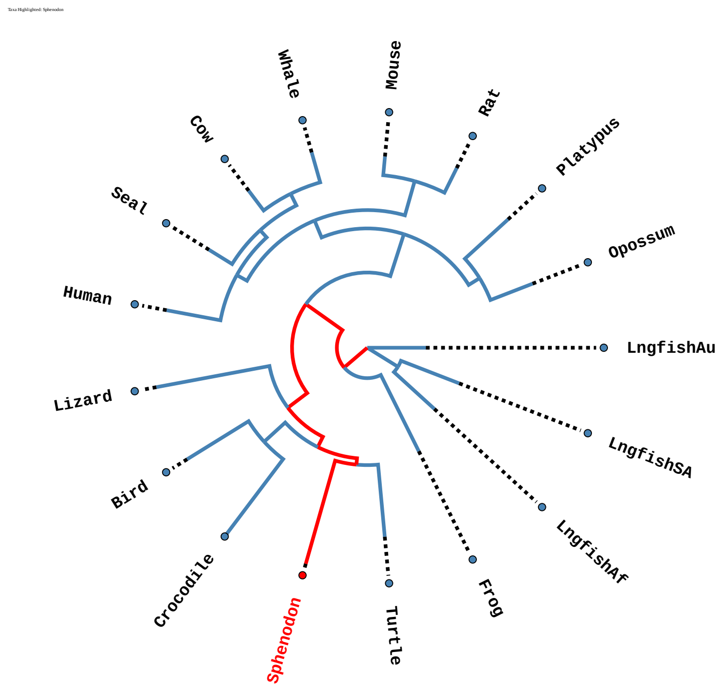
Now we focus on a sequence of trees where a hill-climbing NNI and a random NNI took place. At the trees 160 (Likelihood: -21163.03595) - 161 (Likelihood: -21161.40692), we can in see in Fig. 5.13a a slight rise in the likelihoods happens. At first, we assume that we will be seeing small topological changes because of the low Robinson-Foulds distances. The likelihood values rise slightly because of the hill-climbing NNIs trying to optimise trees while stuck in local optima. Then it is followed by a significant drop of the likelihood values at the trees at the position 161 (Likelihood: -21161.40692) - 162 (Likelihood: -21895.2775). We assume that the topology of the successor will be different from the following tree because of the random NNI. Then we see a rise in the likelihood and at positions 162 (Likelihood: -21895.2775) to 163 (Likelihood: -21358.8747). So we know that here a hill-climbing NNI took place.

The sequence begins at the trees 160-161. We assume that at first a hill-climbing NNIs took place because it seems that IQ-Tree was stuck in a local optimum. After all, the changes are small. Also, our assumption is led by the small differences between likelihoods and the small Robinson-Foulds distance. As we can see in Fig. 5.15, the taxon {Sphenodon} is highlighted. The taxon {Sphenodon} interchanges with {Lizard}. So that {Lizard} clusters with {Bird, Crocodile}. This cannot be seen because of small branch lengths, but when we ignore the branch lengths, we can see the topological changes, which is reasoned of the hill-climbing NNI (Fig. 5.15).

The next step is the transition from 161 to 162 (Fig. 5.16), where can we see that the topological differences between trees are high. When we look at the Robinson-Foulds distance (Fig. 5.13a) we can see that there is a peak between the distance of these trees. Fig. 5.13a, we can see that the likelihood of tree 162 is low.

So from trees 161 to 162, we can see that a random NNI took place. Here we can also say the change cannot be described easily, and we have to compare the trees as in (Fig. 5.16) to see differences. Twelve taxa from seventeen {Turtle, LngfishSA, Frog, Rat, Whale, LngfishAu, Mouse, Human, Crocodile, Seal, Cow, Bird} are highlighted. The highlighting algorithm operates on the non-existent branches, which are a lot because the topologies of trees are so different, as can also be seen in the high Robinson-Foulds distance. Therefore, the algorithm recognises these eleven from seventeen as jumping taxa.

Then in tree transition from tree 162 to 163 (Fig. 5.17), we assume that a hill-climbing NNI took place. We observe that the likelihood of value of tree 162 is in a pit, whereas at position 163, it rises. So here we can say that a hill-climbing NNI took place, optimising the tree.



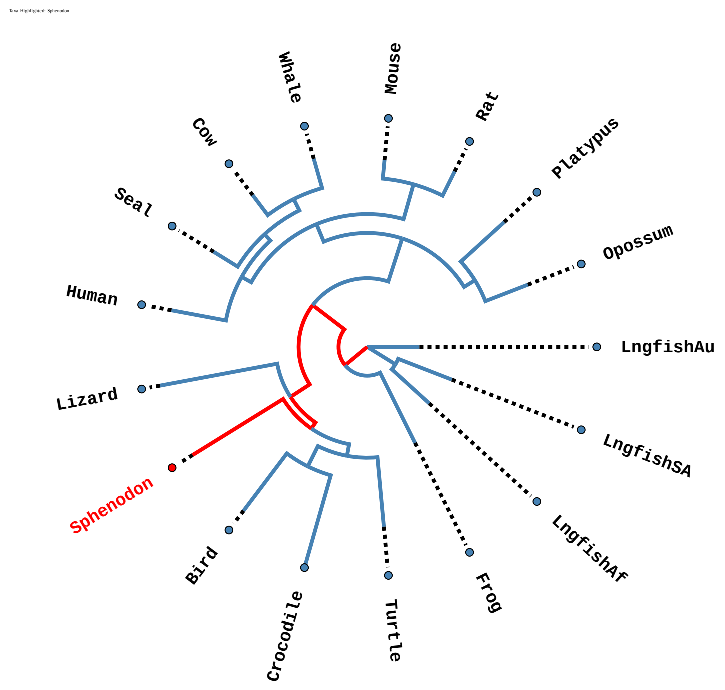
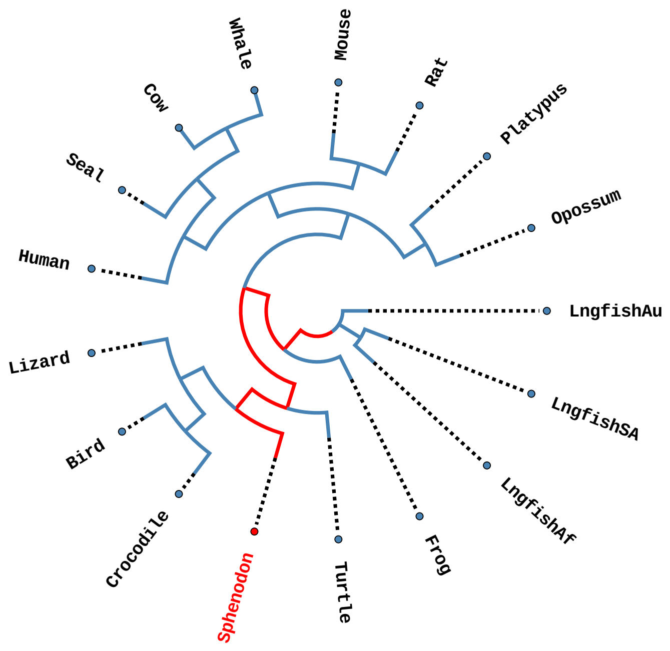


Figure 5.14: Tree Pair from the tree search dataset. Full Tree 6 in the sequence on the top (Likelihood value: -21181.12125); Full Tree in the position 7 on the bottom (Likelihood value: -21158.16664); Highlighted taxon: Sphenodon. The second tree was generated by a hill-climbing NNI. Sphenodon exchanges with {Bird, Crocodile};

Taxa Highlighted: Sphenodon



Taxa Highlighted: Sphenodon

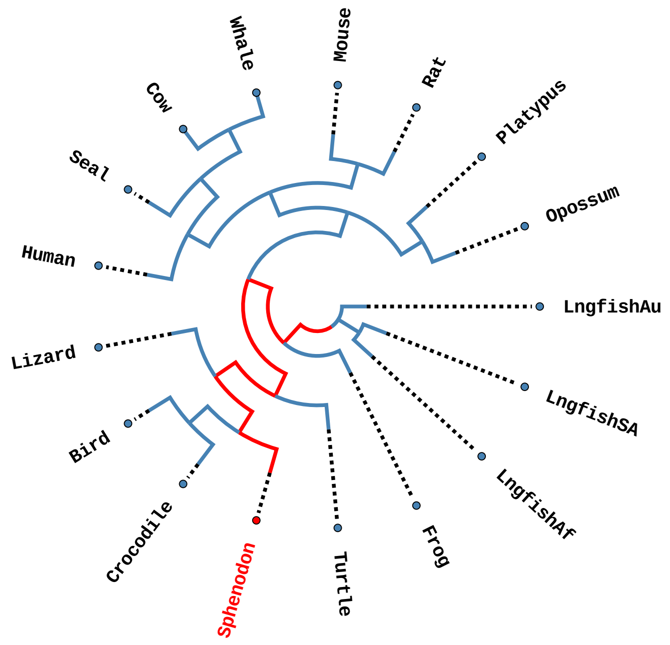
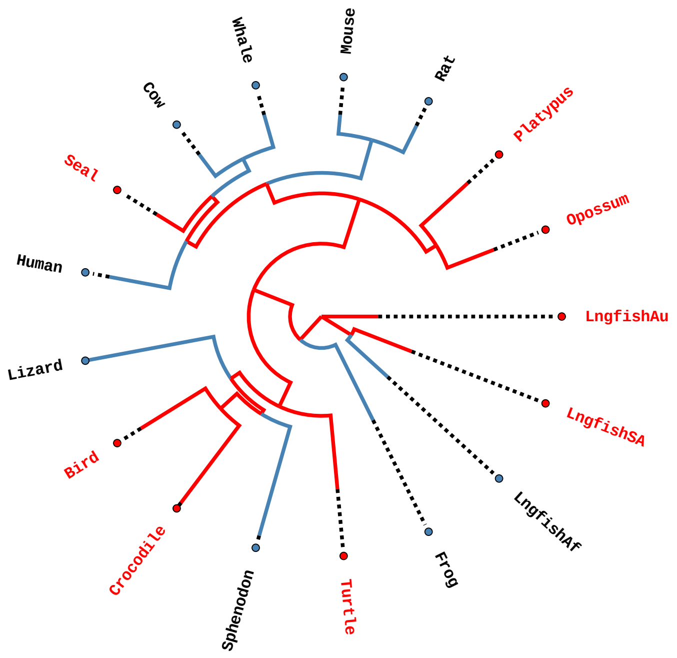


Figure 5.15: Tree Pair from the tree search dataset. Full Tree 160 (Likelihood: -21163.03595) on the top ; Full Tree 161 (Likelihood: -21161.40692) on the bottom. Highlighted taxon: Sphenodon; The second tree was generated by a hill-climbing NNI. Spendon exchanges with Lizard

Taxa Highlighted:



Taxa Highlighted:

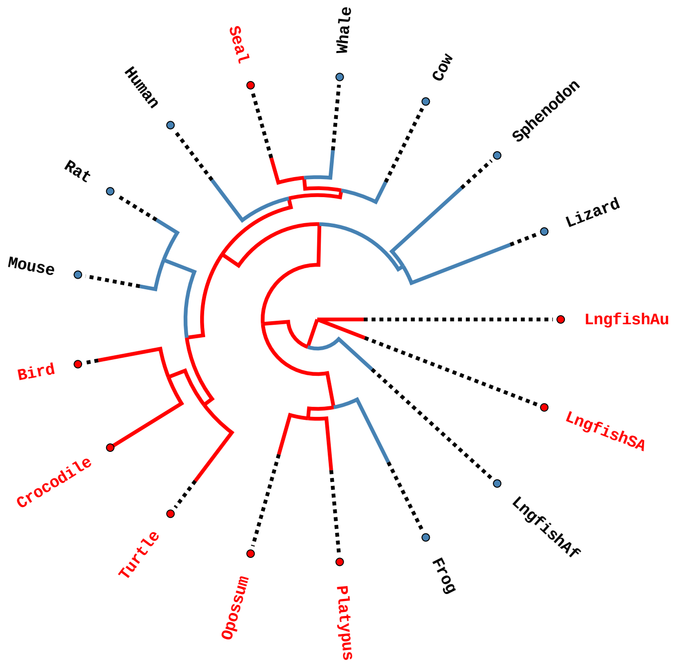
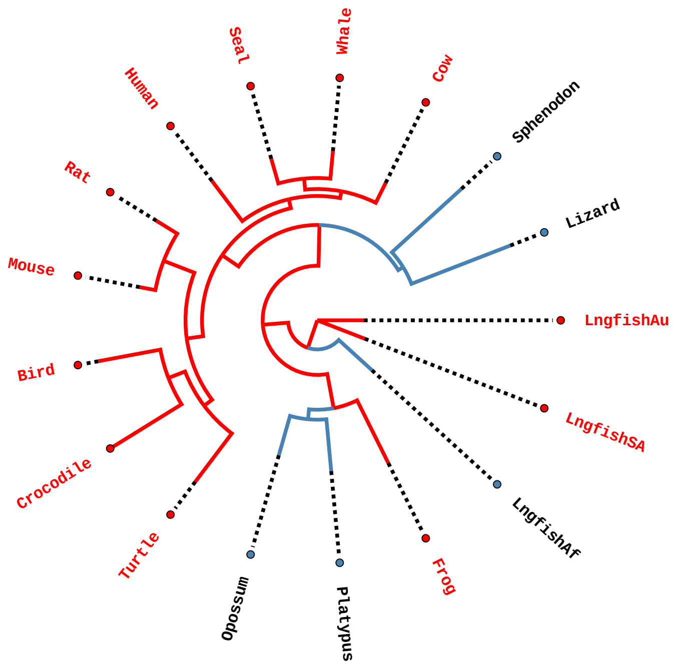


Figure 5.16: Tree Pair from the tree search dataset. Full Tree 161 on the top (Likelihood: -21161.40692); Full Tree 162 (Likelihood: -21895.2775) on the bottom, Highlighted taxa: Opossum, LngfishSA, Turtle, LngfishAu, Crocodile, Platypus, Seal, Bird; The second tree (tree at the position 162) on the bottom was generated by a Random NNI

Taxa Highlighted:



Taxa Highlighted:

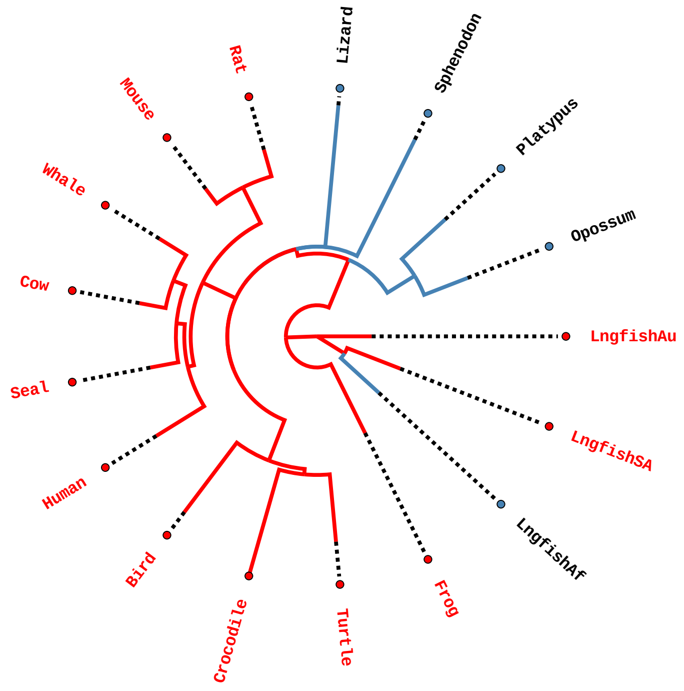


Figure 5.17: Tree Pair from the tree search dataset. Full Tree 162 (Likelihood: -21895.2775) on the top; Full Tree 163 (Likelihood: -21358.8747) on the bottom, Highlighted taxa: Turtle, LngfishSA, Frog, Rat, Whale, LngfishAu, Mouse, Human, Crocodile, Seal, Cow, Bird; The second tree (tree at the position 162) on the bottom was generated by a random NNI

5.3.2 Discussing Observation on the Tree Search Sequence

Our conjecture that the difference in the topology of the two trees is significantly high when the likelihood of these are different has proven to be true. The assumption that when the likelihood of a tree's successor remains almost the same and the Robinson-Foulds distance is small, we will see small hill-climbing NNI is true. The detection of small topology changes, as in the hill-climbing NNI, becomes easy with Phylo-Movies. These small changes can be easily found when the likelihood increases slightly in the predecessor than in the successor. When a random disturbance happens, the difference between the trees is difficult to detect. Also, the difference between the trees is difficult to detect when a hill-climbing NNI has happened where the trees have a small likelihood value, whereas the successor has a higher one. We also discovered that in the cases of the transition as from tree 6 to 7 (Fig. 5.14), we recognise that only one taxon is marked, which makes the recognition easier. However, it would be better if Phylo-Movies highlighted both interchanging taxa and their branches leading to them. Also, in the more extensive transitions, like in 161 to 162, we can see that the highlighting only helps to a limited extent.

Our analyses in the context of the sequence of phylogenetic trees, using the calculation of the absolute distance of likelihood values of trees with the combination of the Robinson-Foulds distance, led us to interesting conclusions. So in the scenario of finding changes in topologies in sequences generated by the tree search of IQ-Tree, we can say that, in retro perspective, also the distance between the likelihood for each pair in the sequence of trees should be taken into account for finding indication where hill-climbing NNI happened and where random perturbations took place.

Chapter 6

Discussion

In our analysis, we can claim that no major disturbances appeared, and the animations ran fluidly. We still have to claim that our tool needs to be tested on more real data to get more insight into animating trees' behaviour to develop more improvements. In the section on further development, we will give more suggestions to make it easier for users.

Some other tools were developed for the comparison of trees. One of them known as `Compare2Trees` (Nye et al., 2005) and `Phylo.io` (O. Robinson et al., 2016). Both of them are not animating the transition of trees, but they were developed to compare them statically. As we did it in our Results section for example Fig. 5.5. `Compare2Trees` can annotate nodes and has as Phylo-Movies a novel approach to compare tree structures. `Compare2Trees`, is as Phylo-Movies a web-based tool, but because of the underlying technology, it can be tedious to use in modern web browsers (O. Robinson et al., 2016). `Phylo.io` is also a web-based software, based as Phylo-Movie on `CSS`, `HTML`, `JavaScript` and `D3`, for side-by-side comparison. `Phylo.io` indicates similarities and differences by applying colour schemes to them. Both of these tools don't allow an analysis of a sequence of trees. If one wanted to compare a sequence of trees, it would be feasible because one would need to upload a pair of trees and then analyse it and redo the process with the next tree and its successor.

But in some cases, it is sometimes easier to detect changes by comparing trees not by animation but rather by depicting the first tree on the right and the second one on the left side. This kind of comparison sometimes gives users more possibilities to see the difference. The two previously mentioned tools make this form of static comparison.

Therefore, it would be a consideration to include this form of comparison in Phylo-Movies to gain a more integrative understanding of the difference of topologies of phylogenetic trees. We can say that Phylo-Movies, allows a user to view a sequence of trees and see their differences. For example, it would be interesting to apply Phylo-Movies to other sequences of phylogenetic tree search algorithms and compare them with each other.

6.1 Further Development

We will now discuss various ideas that could be interesting to further develop and improve Phylo-Movies.

One features missing is that Phylo-Movies can only display rooted trees, limiting the use. An essential extension would be to offer the possibility of visualising the non-rooted trees either by midpoint rooting on by enabling the display of unrooted trees.

Another consideration that has already been discussed is the display in different layouts, such as rectangular phylogenetic trees or phylograms.

In addition, it would be of further relevance that the weighted Robinson-Foulds distance is also displayed in the lower right margin of the application, but other distance function should also be considered. An explanation on different distance functions can be found in Kupczok, 2010 and Felsenstein, 2004, Chapter 3. Currently, only the Relative Robinson-Foulds Distance is displayed.

The individual colourisation of subtypes could lead to a better contribution for having a better overview of the recombinations of several subtypes.

Mike Bostock, the developer of `D3`, has an excellent example of the Tree Of Life (D. Wu et al., 2009), where he divides the group of bacteria, eukaryotes and archaea by using different colours https://observablehq.com/@d3/tree-of-life. The classification into groups could be done by placing a certain substring in front of the different taxa.

If a taxon is marked as jumping several times in several s-edge, it would also be interesting to mark the taxa marked as jumping several times in other colours. The more often a taxon is identified as jumping, the darker it could be coloured, creating a gradient that makes it possible to spot the difference at first glance.

Last but not least, it would be interesting to offer the possibility for the user to record certain parts in video formats as `MP4` or `GIFS` that are played in Phylo-Movies, so that they can be integrated into presentations.

6.2 Conclusion

In this thesis, we have explained how Phylo-Movies can visualise sequences of phylogenetic trees and where one can apply Phylo-Movies. It has been shown that the differences between neighbouring trees are essential, as it has been from sequences generated from reconstructing trees for sliding windows along genomic alignments, e.g., to detect recombination events. Another source might be the sequence of trees found during a tree search during phylogenetic analysis. After obtaining such a sequence of trees, we have shown that it is no longer tedious to manually analyse neighbouring pairs of trees and detect major transitions, changes, or recombination events. Because it is hard to see differences between the two trees, we found ways to highlight changes in the tree topologies so that the user can easily recognise the significant changes. So we developed a highlighting algorithm that detects jumping taxa reasoning the topological changes. With the combination of colourisation and the intermediate steps, it is easier for the eyes to see apparent differences and transitions. The following properties have been achieved: The tools are easy to use, and visualisations are high fidelity. Because we have built Phylo-Movies as a browser application, it is more accessible to the scientific community, especially experimentalists with limited computational resources. Also, an essential requirement for the tool had high animation quality. We have seen that the animation quality provides enough resolution to detect changes in topologies.

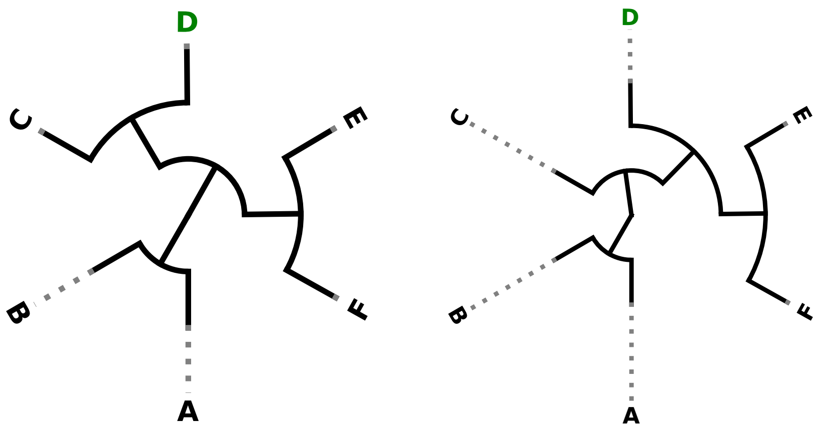
Appendix A

Test Cases for the Highlighting Approaches

First, all the case examples and their results for the Initial Approach are shown, followed by the same examples, except that this time the results of the Final Approach are shown. The pictures were generated with ETE3 (Huerta-Cepas, François Serra, et al., 2016a). All cases are numbered and can be recognised either by their numbering. First, we have examples where a taxon jumps Section A.2, two taxa exchange Section A.1, and a small example where a taxon jumps in a subtree Section A.3. After that, we see a case where two subtrees exchange Section A.4. Then we have a case where two different s-edges appear Section A.5. Then we have a case where a subtree move toward the root Section A.6. Then we have two cases where subtrees dissolves into a multifurication and where the subtree swapping position (Section A.7 and Section A.8). It is a representative subset of cases the approaches where tested on to develop a better understanding.

A.1 Case 1: One Taxon jumps

A.1.1 Result: Initial Approach



Left side Full Tree 1; Right Side Full Tree 2

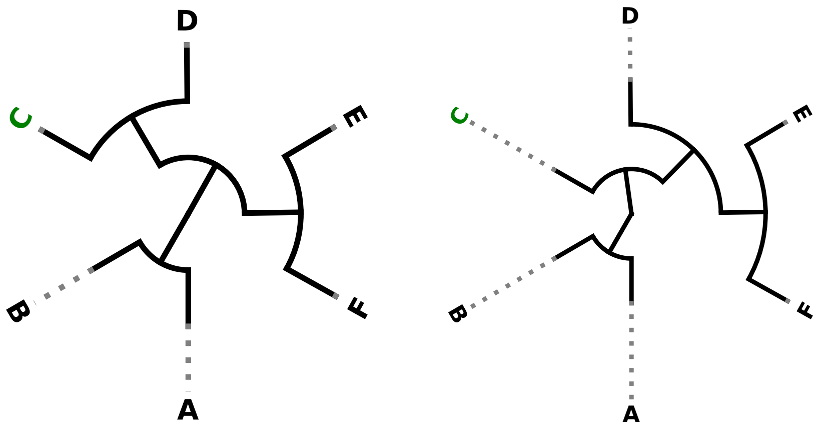
Description: One Taxon Jumps or Two Taxa Jump

Algorithm Worked: True

Possible solutions: {‘D’},{‘C’},{‘F’, ‘E’}

Yielded Solution: {‘D’}

A.1.2 Result for the Final Approach



Left side Full Tree 1; Right Side Full Tree 2

Description: One Taxon Jumps or Two Taxa Jump

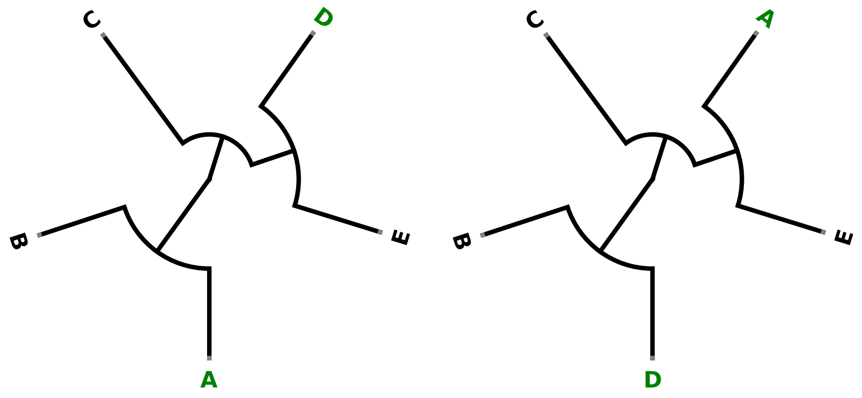
Algorithm Worked: True

Possible solutions: {‘D’},{‘C’},{‘F’, ‘E’}

Yielded Solution: {‘C’}

A.2 Case 2: Small Change in subtree

A.2.1 Result for the Initial Approach



Left side Full Tree 1; Right Side Full Tree 2

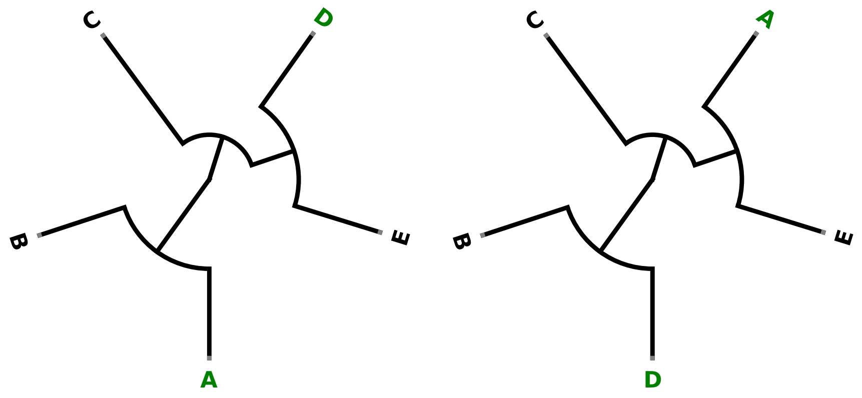
Description: Taxa A and D exchange or Taxa E and B exchange

Algorithm Worked: True

Possible solutions: {‘A’, ‘D’},{‘B’, ‘E’}

Yielded Solution: {‘A’, ‘D’}

A.2.2 Result for the Final Approach



Left side Full Tree 1; Right Side Full Tree 2

Description: Taxa A and D exchange or Taxa E and B exchange

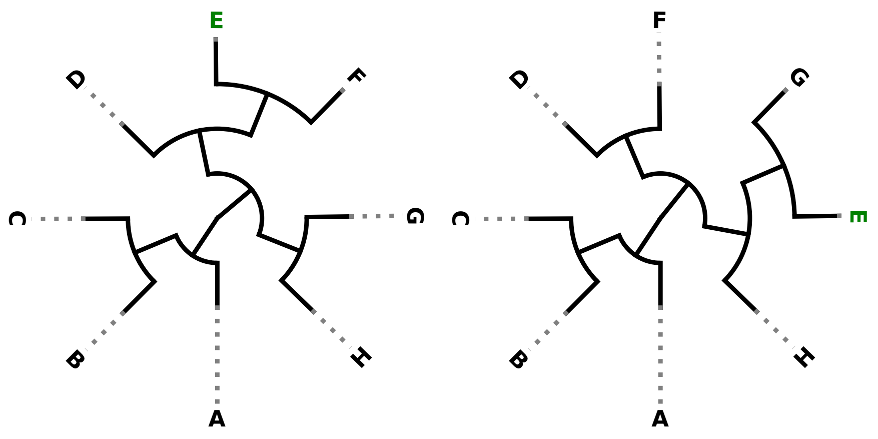
Algorithm Worked: True

Possible solutions: {‘A’, ‘D’},{‘B’, ‘E’}

Yielded Solution: {‘A’, ‘D’}

A.3 Case 3: Small Change in subtree

A.3.1 Result for the Initial Approach



Left side Full Tree 1; Right Side Full Tree 2

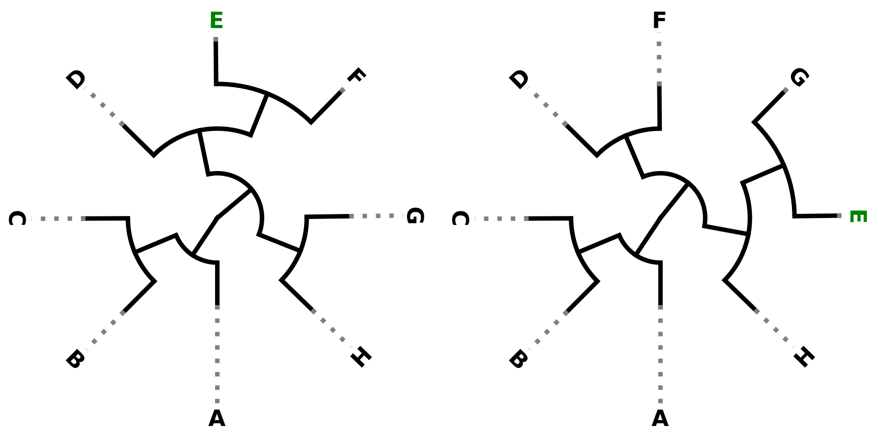
Description: Taxon E jumps into G

Algorithm Worked: True

Possible solutions: {‘E’}

Yielded Solution: {‘E’}

A.3.2 Result for the Final Approach



Left side Full Tree 1; Right Side Full Tree 2

Description: Taxon E jumps into G

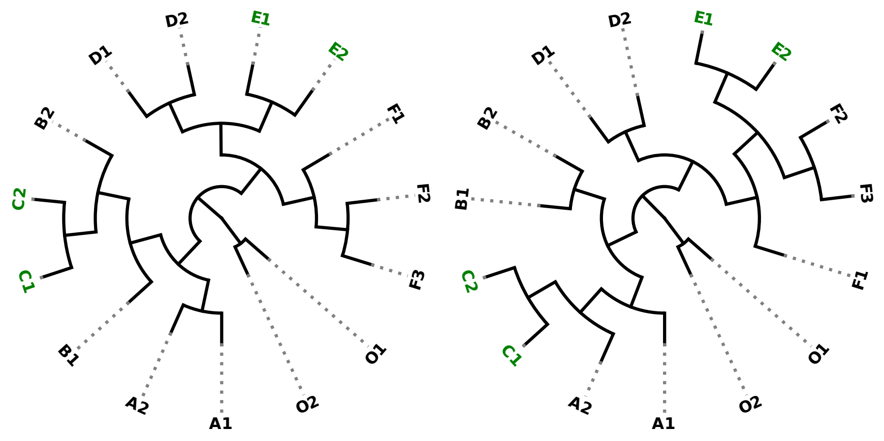
Algorithm Worked: True

Possible solutions: {‘E’}

Yielded Solution: {‘E’}

A.4 Case 4: Two Groups

A.4.1 Result for the Initial Approach



Left side Full Tree 1; Right Side Full Tree 2

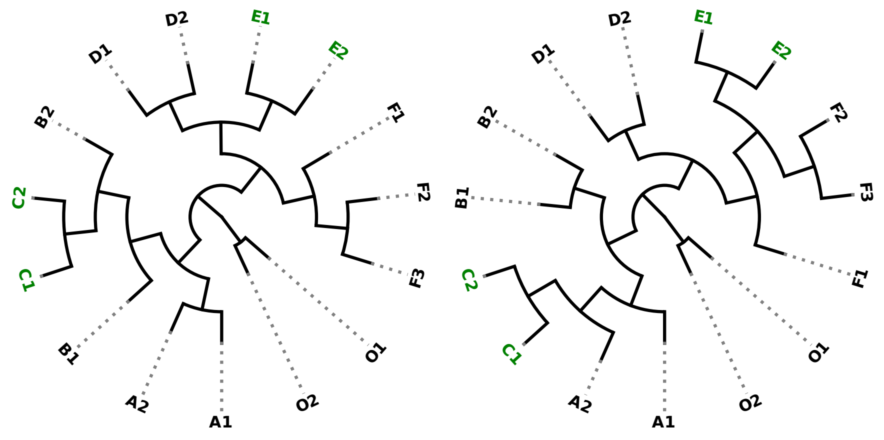
Description: The group {C1 C2} jumps into A2, the group {E1, E2} jumps into the ancestor of {F1, F2}

Algorithm Worked: True

Possible solutions: {‘C1’, ‘C2’, ‘E1’, ‘E2’}

Yielded Solution: {‘C1’, ‘C2’, ‘E1’, ‘E2’}

A.4.2 Result for the Final Approach



Left side Full Tree 1; Right Side Full Tree 2

Description: The group {C1 C2} jumps into A2, the group {E1, E2} jumps into the ancestor of {F1, F2}

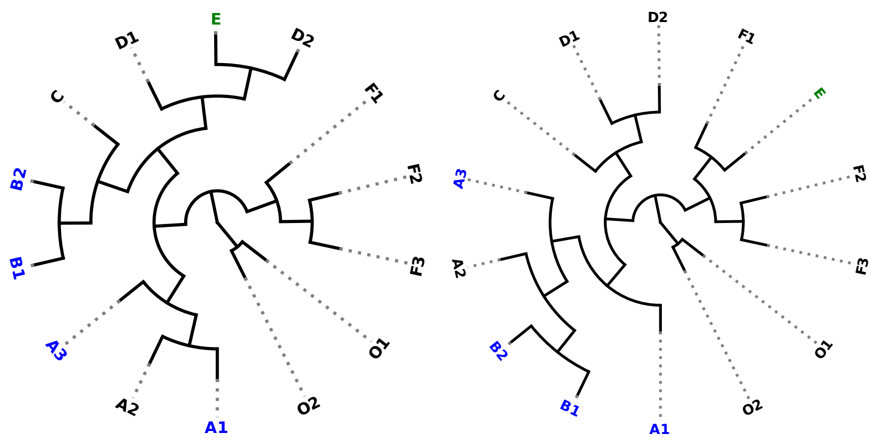
Algorithm Worked: True

Possible solutions: {‘C1’, ‘C2’, ‘E1’, ‘E2’}

Yielded Solution: {‘C1’, ‘C2’, ‘E1’, ‘E2’}

A.5 Case 5: Two s-edges

A.5.1 Result for the Initial Approach



Left side Full Tree 1; Right Side Full Tree 2

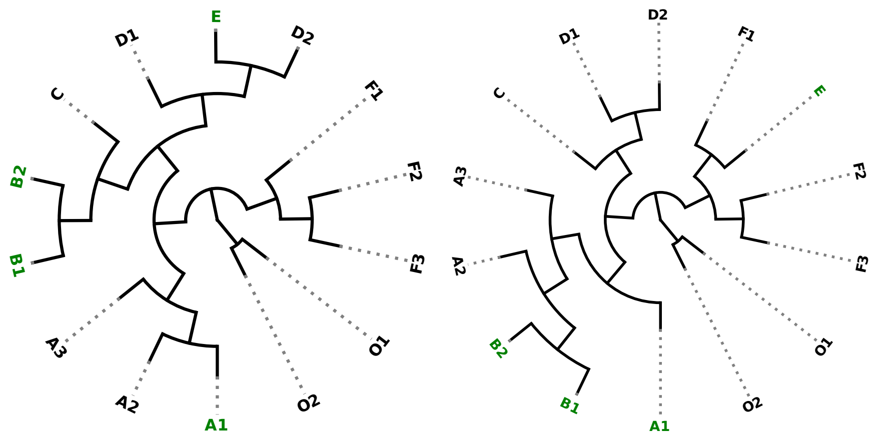
Description: The group {B1, B2} jumps to A2, E jumps into F1

Algorithm Worked: False

Possible solutions: {‘A1’, ‘B2’, ‘E’, ‘B1’},{‘A3’, ‘B2’, ‘E’, ‘B1’}

Yielded Solution: {‘E’}

A.5.2 Result for the Final Approach



Left side Full Tree 1; Right Side Full Tree 2

Description: The group {B1, B2} jumps to A2, E jumps into F1

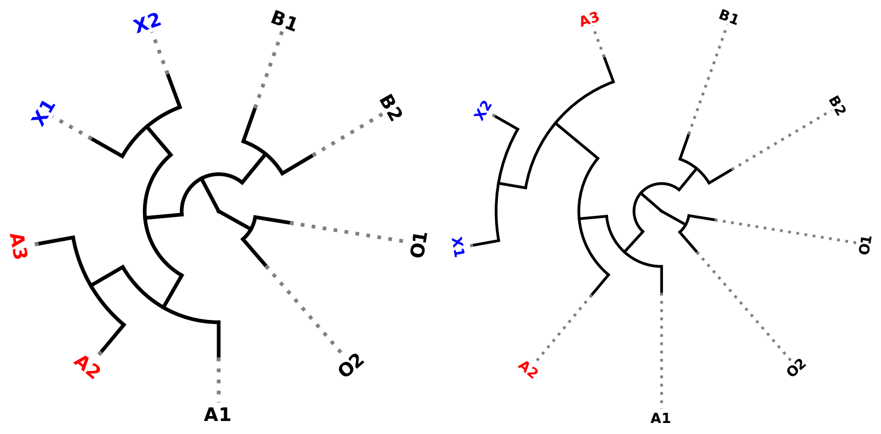
Algorithm Worked: True

Possible solutions: {‘A1’, ‘B2’, ‘E’, ‘B1’},{‘A3’, ‘B2’, ‘E’, ‘B1’}

Yielded Solution: {‘A1’, ‘B2’, ‘E’, ‘B1’}

A.6 Case 6: Subtree moving towards the root

A.6.1 Result for the Initial Approach



Left side Full Tree 1; Right Side Full Tree 2

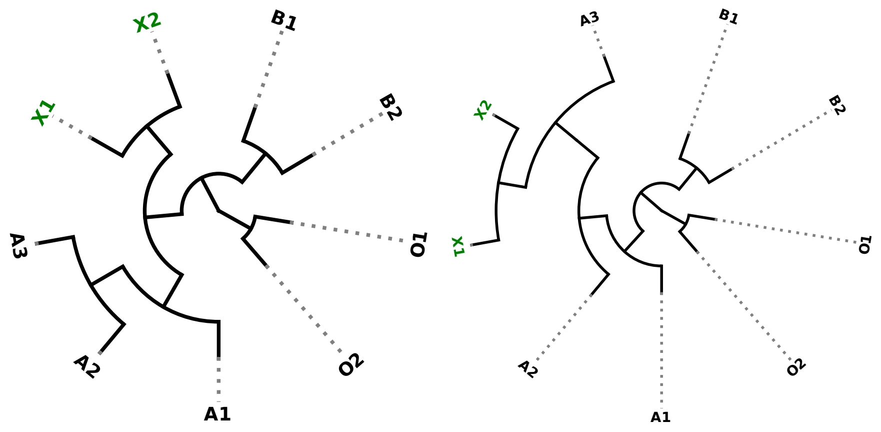
Description: Subtree {X1, X2} is moving is jumping downwards

Algorithm Worked: False

Possible solutions: {'X2', 'X1'}

Yielded Solution: {'A3', 'A2'}

A.6.2 Result for the Final Approach



Left side Full Tree 1; Right Side Full Tree 2

Description: Subtree {X1, X2} is moving is jumping downwards

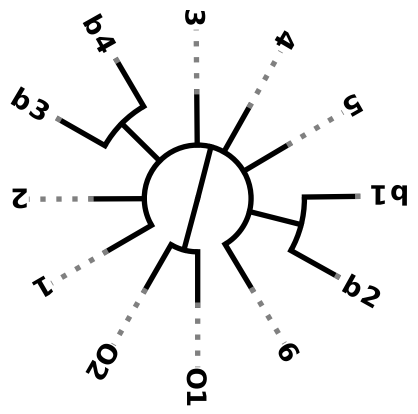
Algorithm Worked: True

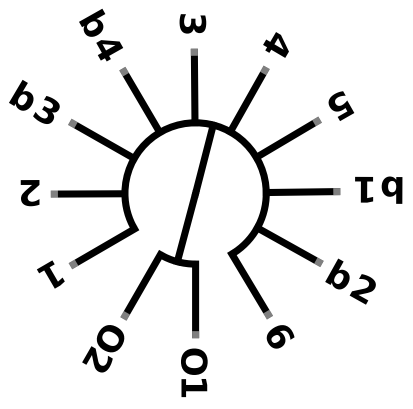
Possible solutions: {'X2', 'X1'}

Yielded Solution: {'X2', 'X1'}

A.7 Case 7: Subtree moving towards the root

A.7.1 Result for the Initial Approach





Left side Full Tree 1; Right Side Full Tree 2

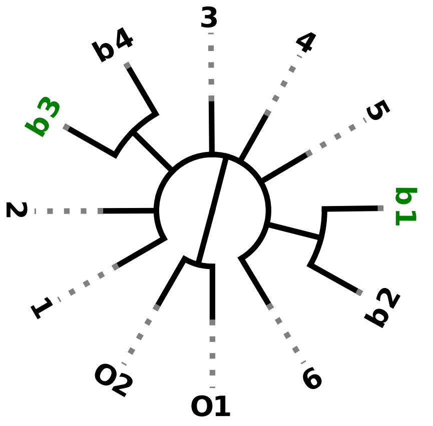
Description: B1 or B2 jumps to his ancestor and B3 or B4 jumps to his ancestor

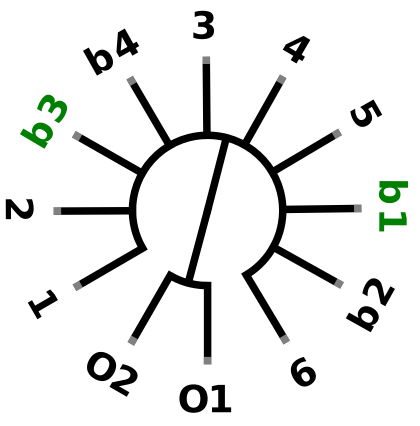
Algorithm Worked: False

Possible solutions: {‘b3’, ‘b1’},{‘b4’, ‘b1’},{‘b2’, ‘b3’},{‘b2’, ‘b4’}

Yielded Solution: set()

A.7.2 Result for the Final Approach





Left side Full Tree 1; Right Side Full Tree 2

Description: B1 or B2 jumps to his ancestor and B3 or B4 jumps to his ancestor

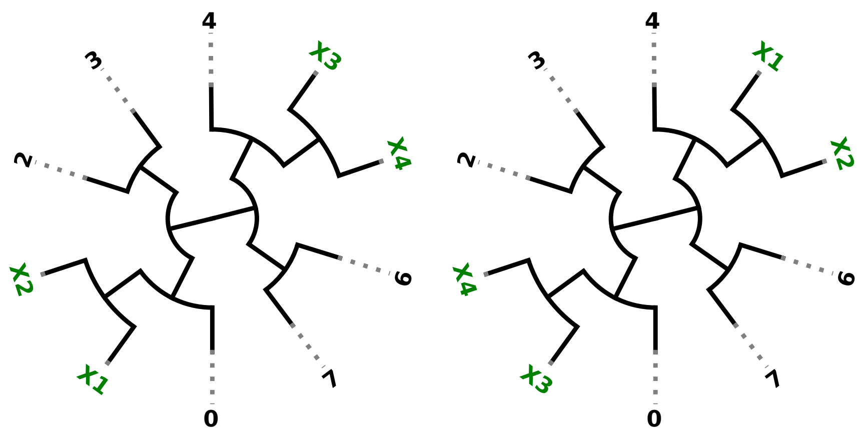
Algorithm Worked: True

Possible solutions: {‘b3’, ‘b1’},{‘b4’, ‘b1’},{‘b2’, ‘b3’},{‘b2’, ‘b4’}

Yielded Solution: {‘b3’, ‘b1’}

A.8 Case 8: Two exchanging subtrees

A.8.1 Result for the Initial Approach



Left side Full Tree 1; Right Side Full Tree 2

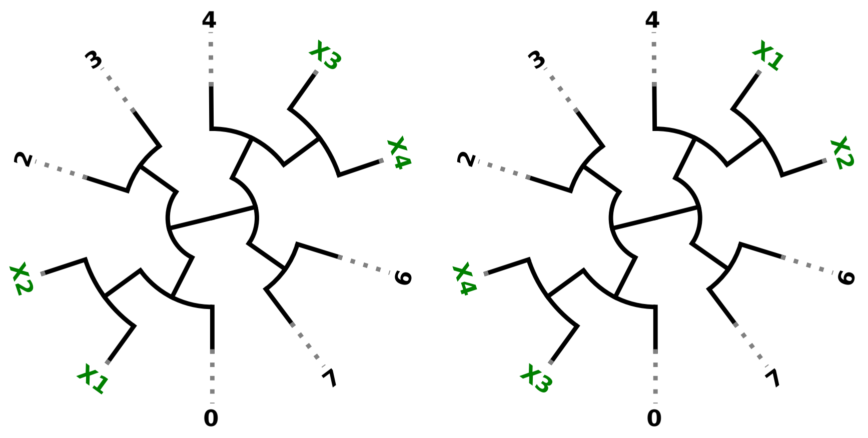
Description: Subtree (X1, X2) and (X3, X4) or taxa 0 and 4 exchange

Algorithm Worked: True

Possible solutions: {'X2', 'X1', 'X3', 'X4'},{'0', '4'}

Yielded Solution: {'X2', 'X1', 'X3', 'X4'}

A.8.2 Result for the Final Approach



Left side Full Tree 1; Right Side Full Tree 2

Description: Subtree (X1, X2) and (X3, X4) or taxa 0 and 4 exchange

Algorithm Worked: True

Possible solutions: {'X2', 'X1', 'X3', 'X4'},{'0', '4'}

Yielded Solution: {'X2', 'X1', 'X3', 'X4'}

Appendix B

Backend Requirements

Here are the python requirements listed for the backend of Phylo-Movies.

package	version	Useage	URL
ete3	3.1.2	Tree Manipulation	http://etetoolkit.org/
Werkzeug	2.0.1	Request Handling	https://werkzeug.palletsprojects.com/en/2.0.x/
Flask	2.0.1	Web Framework	https://flask.palletsprojects.com/en/2.0.x/
gunicorn	20.1.0	WSGI	https://gunicorn.org/
Jinja2	3.0.1	Template Language	https://jinja.palletsprojects.com/en/3.0.x/
numpy	1.19.5	Array Handling	https://numpy.org/
pandas	1.3.0	Table Handling	https://pandas.pydata.org/
biopython	1.79	Nexus Parser	https://biopython.org/

Bibliography

Abecasis, A. B., Lemey, P., Vidal, N., de Oliveira, T., Peeters, M., Camacho, R., Shapiro, B., Rambaut, A., & Vandamme, A.-M. (2007). Recombination confounds the early evolutionary history of human immunodeficiency virus type 1: Subtype g is a circulating recombinant form. *Journal of virology*, *81*(16), 8543–8551.

Alvestrand, H. T. (1998). X.400 Image Body Parts. https://doi.org/10.17487/RFC2158

Bederson, B. B., & Boltman, A. (1999). Does animation help users build mental maps of spatial information? *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis' 99)*, 28–35.

Berners-Lee, T., & Connolly, D. W. (1995). Hypertext Markup Language - 2.0. https://doi.org/10.17487/RFC1866

Berners-Lee, T., Masinter, L. M., & McCahill, M. P. (1994). Uniform Resource Locators (URL). https://doi.org/10.17487/RFC1738

Billera, L. J., Holmes, S. P., & Vogtmann, K. (2001). Geometry of the space of phylogenetic trees. *Adv. Appl. Math.*, *27*, 733–767. https://doi.org/https://doi.org/10.1006/aama.2001.0759

Bos, B., Çelik, T., Hickson, I., & Lie, H. W. (2005). Cascading style sheets level 2 revision 1 (css 2.1) specification. *W3C working draft, W3C, June.*

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE Trans. Vis. Comput. Graph.*, *17*, 2301–2309. https://doi.org/10.1109/TVCG.2011.185

Boutell, T. (1997). PNG (Portable Network Graphics) Specification Version 1.0. https://doi.org/10.17487/RFC2083

Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format. https://doi.org/10.17487/RFC8259

Brownlee, N., & IAB. (2016). SVG Drawings for RFCs: SVG 1.2 RFC. https://doi.org/10.17487/RFC7996

Felsenstein, J. (2004). *Inferring phylogenies* (2nd ed.). Sinauer Associates.

Ferraiolo, J., Jun, F., & Jackson, D. (2000). *Scalable vector graphics (svg) 1.0 specification.* iuniverse Bloomington.

Fischer, W., Giorgi, E. E., Chakraborty, S., Nguyen, K., Bhattacharya, T., Theiler, J., Goloboff, P. A., Yoon, H., Abfalterer, W., Foley, B. T., Tegally, H., San, J. E., de Oliveira, T., Network for Genomic Surveillance in South Africa (NGS-SA), Gnanakaran, S., & Korber, B. (2021). HIV-1 and SARS-CoV-2: Patterns in the evolution of two pandemic pathogens. *Cell Host Microbe*, *29*, 1093–1110. https://doi.org/10.1016/j.chom.2021.05.012

Flanagan, H. (2016). Cascading Style Sheets (CSS) Requirements for RFCs. https://doi.org/10.17487/RFC7993

Gonzales, C. (1996). Does animation in user interfaces improve decision making? in proceedings of the international conference in computer human interaction chi'96(pp. 7-34).

Grinberg, M. (2014). *Flask web development: Developing web applications with python* (1st). O'Reilly Media, Inc.

Grosskurth, A., & Godfrey, M. (2005). A reference architecture for web browsers. *21st IEEE International Conference on Software Maintenance (ICSM'05)*, 661–664.

Heer, J., & Robertson, G. (2007). Animated transitions in statistical data graphics. *IEEE Trans. Vis. Comput. Graph.*, *13*(6), 1240–1247. https://doi.org/10.1109/TVCG.2007.70539

Holmes, E. (2009). *The evolution and emergence of rna viruses*. Oxford University Press.

Holovaty, A., & Kaplan-Moss, J. (2009). *The definitive guide to django: Web development done right*. Apress.

Huerta-Cepas, J., Serra, F. [François], & Bork, P. (2016a). ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. *Mol. Biol. Evol.*, *33*, 1635–1638. https://doi.org/10.1093/molbev/msw046

Huerta-Cepas, J., Serra, F. [François], & Bork, P. (2016b). Ete 3: Reconstruction, analysis, and visualization of phylogenomic data. *Molecular biology and evolution*, *33*(6), 1635–1638.

Janert, P. (2019). *D3 for the impatient: Interactive graphics for programmers and scientists* (1st ed.). O'Reilly Media.

Kaiser, A., Krüger, J., & Evers, D. J. (2007). RNA Movies 2: Sequential animation of RNA secondary structures. *Nucleic Acids Res.*, *35*, W330–W334. https://doi.org/10.1093/nar/gkm309

Khatchikian, D., Orlich, M., & Rott, R. (1989). Increased viral pathogenicity after insertion of a 28s ribosomal rna sequence into the haemagglutinin gene of an influenza virus. *Nature*, *340*, 156–157.

Krause, E. (1973). Taxicab geometry. *The Mathematics Teacher*, *66*(8).

Kuiken, C., Korber, B., & Shafer, R. W. (2003). Hiv sequence databases. *AIDS reviews*, *5*(1), 52.

Kupczok, A. (2010). *Postprocessing phylogenies: Tree distances and supertrees* (Doctoral dissertation). University of Vienna. https://doi.org/10.25365/thesis.9865

Kutmon, M. (2008). *Detection of recombination events in dna sequences* (Practical Bachelor thesis). Fachhochschule Hagenberg.

Lee, M. S., & Palci, A. (2015). Morphological phylogenetics in the genomic age. *Curr. Biol.*, *25*(19), R922–R929. https://doi.org/https://doi.org/10.1016/j.cub.2015.07.009

Lemey, P., Salemi, M., & Vandamme, A.-M. (2009). *The phylogenetic handbook: A practical approach to phylogenetic analysis and hypothesis testing* (2nd ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511819049

Lim, Y., & Singer, D. (2006). MIME Type Registration for MPEG-4. https://doi.org/10.17487/RFC4337

Maddison, D., Swofford, D., & Maddison, W. (1997). NEXUS: an extensible file format for systematic information. *Syst. Biol.*, *46*, 590–621.

Malim, M. H., & Emerman, M. (2001). Hiv-1 sequence variation: Drift, shift, and attenuation. *Cell*, *104*, 469–472.

Minh, B. Q., Schmidt, H. A., Chernomor, O., Schrempf, D., Woodhams, M. D., Von Haeseler, A., & Lanfear, R. (2020). Iq-tree 2: New models and efficient methods for phylogenetic inference in the genomic era. *Molecular biology and evolution*, *37*(5), 1530–1534.

Murray, S. (2017). *Interactive data visualization for the web: An introduction to designing with d3*. "O'Reilly Media, Inc."

Nguyen, L.-T., Schmidt, H. A., von Haeseler, A., & Minh, B. Q. (2014). IQ-TREE: A Fast and Effective Stochastic Algorithm for Estimating Maximum-Likelihood Phylogenies. *Mol. Biol. Evol.*, *32*(1), 268–274. https://doi.org/10.1093/molbev/msu300

Nielsen, H., Mogul, J., Masinter, L. M., Fielding, R. T., Gettys, J., Leach, P. J., & Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. https://doi.org/10.17487/RFC2616

Nora, T., Charpentier, C., Tenaillon, O., Hoede, C., Clavel, F., & Hance, A. J. (2007). Contribution of recombination to the evolution of human immunodeficiency viruses expressing resistance to antiretroviral treatment. *J. Virol.*, *81*, 7620–7628.

Nye, T. M., Liò, P., & Gilks, W. R. (2005). A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. *Bioinformatics*, *22*(1), 117–119. https://doi.org/10.1093/bioinformatics/bti720

Rambaut, A., & Grassly, N. (1997). Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Bioinformatics*, *13*(3), 235–238. https://doi.org/10.1093/bioinformatics/13.3.235

Richard, G., Andrew, M. et al. (1994). Advanced mathematics: Precalculus with discrete mathematics and data analysis. *Boston: Houghton Mifflin.*

Robert, G., & Donald, S. (1967). The elements of real analysis.

Robertson, D. L., Anderson, J. P., Bradac, J. A., Carr, J. K., Foley, B., Funkhouser, R. K., Gao, F., Hahn, B. H., Kalish, M. L., Kuiken, C., Learn, G. H., Leitner, T., McCutchan, F., Osmanov, S., Peeters, M., Pieniazek, D., Salminen, M., Sharp, P. M., Wolinsky, S., & Korber, B. (1999). Hiv-1 nomenclature proposal: A reference guide to hiv-1 classification. In C. L. Kuiken, B. Foley, B. Hahn, B. Korber, F. McCutchan, P. A. Marx, J. W. Mellors, J. I. Mullins, J. Sodroski, & S. Wolinksy (Eds.), *The human retroviruses and aids 1999 compendium* (pp. 492–505). Los Alamos National Laboratories. https://doi.org/10.2172/1186019

Robertson, G. G., Mackinlay, J. D., & Card, S. K. (1991). Cone trees: Animated 3d visualizations of hierarchical information. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 189–194.

Robinson, D., & Foulds, L. (1979). Comparison of weighted labelled trees. *Combinatorial mathematics vi* (pp. 119–126). Springer. https://doi.org/10.1007/bfb0102690

Robinson, D., & Foulds, L. (1981). Comparison of phylogenetic trees. *Math. Biosci.*, *53*(1), 131–147. https://doi.org/https://doi.org/10.1016/0025-5564(81)90043-2

Robinson, O., Dylus, D., & Dessimoz, C. (2016). Phylo.io: Interactive Viewing and Comparison of Large Phylogenetic Trees on the Web. *Mol. Biol. Evol.*, *33*, 2163–2166.

Rose, D. M. T., Hollenbeck, S., & Masinter, L. M. (2003). Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols. https://doi.org/10.17487/RFC3470

Schmidt, H. A. (2003). *Phylogenetic trees from large datasets* (Doctoral dissertation). Universität Düsseldorf. Düsseldorf, Germany. https://doi.org/-

Schmidt, H. A., Vanas, A., & von Haeseler, A. (2013). Visrec - detecting and visualising recombination and phylogenetic information along alignments.

Simon-Lorière, E., & Holmes, E. (2011). Why do rna viruses recombine? *Nat. Rev. Microbiol.*, *9*, 617–626.

Stewart, P., McCanne, S., Fenner, B., Berc, L., & Frederick, R. (1998). RTP Payload Format for JPEG-compressed Video. https://doi.org/10.17487/RFC2435

Tversky, B., Morrison, J. B., & Betrancourt, M. (2002). Animation: Can it facilitate? *Int. J. Hum. Comput. Stud.*, *57*(4), 247–262.

Van Heuverswyn, F., & Peeters, M. (2007). The origins of HIV and implications for the global epidemic. *Curr. Infect. Dis. Rep.*, *9*, 338–346. https://doi.org/10.1007/s11908-007-0052-x

Wells, P. (2013). *Understanding animation*. Routledge.

Wheeler, D., Barrett, T., Benson, D., Bryant, S., Canese, K., Chetvernin, V., Church, D., DiCuccio, M., Edgar, R., Federhen, S. et al. (2007). Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.*, *36*, D13–D21.

Wu, D., Hugenholtz, P., Mavromatis, K., Pukall, R., Dalin, E., Ivanova, N. N., Kunin, V., Goodwin, L., Wu, M., Tindall, B. J. et al. (2009). A phylogeny-driven genomic encyclopaedia of bacteria and archaea. *Nature*, *462*(7276), 1056–1060.