From Trees to Networks and Back

Sarah Bastkowski

Supervisor: Prof. Vincent Moulton
Co-supervisor: Dr. Geoffrey Mckeown

A thesis submitted for the degree of Doctor of Philosophy
at the University of East Anglia

December 2013

School of Computing Science, University of East Anglia
Norwich, United Kingdom

©This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with the author and that no quotation from the thesis, nor any information derived there from, may be published without the author's prior, written consent.

I would like to dedicate this thesis to
Christina and Wolfgang Bastkowski ...

Declaration

No portion of this work referred to in this thesis has been submitted in support of an application for another degree or qualification at this or any other university or other institute of learning.

Acknowledgements

I would like to thank my supervisor Vincent Moulton, who gave me the opportunity to do this PhD, sat through many sometimes long meetings with me and kept me focused and motivated. Andreas Spillner, who first got me interested in phylogenetics, answered many of my questions, corrected and proof read my first attempts made in scientific writing. He also gave me the opportunity of collaborative visits to the University of Greifswald and supported me throughout my PhD. Taoyang Wu helped me to develop the skills that allowed me to discover the absorption into mathematical proofs and the satisfaction of seeing through them. He always had time for a discussion, checked my work very carefully and was patient and encouraging. I would also like to thank my second supervisor Geoffrey Mckeown. He was one of the first people I presented my research to and took away a lot of my fear about it.

The last three and a half years were a challenging time, starting somewhere new, learning a lot of new skills, developing strategies and ideas and defending these was not always easy, but the supportive environment and the people of the School of Computing Science, my colleagues in the Computational Biology lab and people I met through collaboration and conferences made it a very enjoyable and unmissable time for me. I made some very good friends in this time and for their support I am very thankful. Also I would like to thank the School of Computing Science for not just providing an intellectually stimulating environment, but also for funding my PhD.

One friend, I made in these three years, I would like to especially mention my fiance Daniel Mapleson. He did not just proof read my thesis several times, spent many evenings listening to my talks and explained to me good programming practice, but also taught me to be a better scientist and maybe even a better person.

And last, I would like to thank my family. In particular, my parents Christina and Wolfgang Bastkowski, even if far away in Germany, were always reachable and there for me (a big thanks to Skype and the internet here!). Also my grandmother Elfriede Bastkowski was always available for a chat whether it concerned happiness or sorrows. Even if my other grandparents Siegfried Bastkowski, Anneliese and Hans Hoffmann could not be with me any more during the time of my PhD, they contributed greatly to my interest in nature and technology and to my general personal as well as intellectual development. They are not forgotten and are thanked very much for all their efforts and love.

Publications

S. Grünewald, A. Spillner, **S. Bastkowski**, A. Boegershausen, V. Moulton (2013) SuperQ: Computing Supernetworks from Quartets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 1(10): 151-160.

S. Bastkowski, A. Spillner, V. Moulton (2013) Fishing for minimum evolution trees with Neighbor-Nets. *Information Processing Letters* 114(1-2): 13-18.

Statement of Originality

I certify that this thesis, and the research to which it refers, are the product of my own work, and that any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged.

Abstract

The evolutionary history of a set of species is commonly represented by a phylogenetic tree. Often, however, the data contain conflicting signals, which can be better represented by a more general structure, namely a phylogenetic network. Such networks allow the display of several alternative evolutionary scenarios simultaneously but this can come at the price of complex visual representations. Using so-called circular split networks reduces this complexity, because this type of network can always be visualized in the plane without any crossing edges. These circular split networks form the core of this thesis. We construct them, use them as a search space for minimum evolution trees and explore their properties.

More specifically, we present a new method, called SuperQ, to construct a circular split network summarising a collection of phylogenetic trees that have overlapping leaf sets. Then, we explore the set of phylogenetic trees associated with a fixed circular split network, in particular using it as a search space for optimal trees. This set represents just a tiny fraction of the space of all phylogenetic trees, but we still find trees within it that compare quite favourably with those obtained by a leading heuristic, which uses tree edit operations for searching the whole tree space. In the last part, we advance our understanding of the set of phylogenetic trees associated with a circular split network. Specifically, we investigate the size of the so-called circular tree neighbourhood for the three tree edit operations, tree bisection and reconnection (TBR), subtree prune and regraft (SPR) and nearest neighbour interchange (NNI).

Contents

List of Figures

Chapter 1

Introduction

Where do we come from? This is one of the big questions that has been engaging human thought since the first societies were formed. Early answers to this question predominantly involved the invocation of some kind of creator. An alternative, and perhaps more satisfying answer, was built upon work by 17^{th} and 18^{th} century naturalists who started to systematically classify and catalogue species. In the mid 19^{th} century, Charles Darwin undertook a five year journey to survey and catalogue species from around the world. Based on the information from his travels, and his work on selective breeding of plants, he noticed that species might change from one to another and that different species might share a common ancestor.

The idea of common ancestry or branching descent can be visualised with a tree structure. The tree representing the evolution of species is often referred to as the "Tree of Life". This theory enabled us to come up with an alternative answer to the question of where do we come from: we come from apes. Of course, this immediately raises another question: where do apes come from? In fact, it is theorised that all species on the planet today can be traced back to a single common ancestor which lived around 4 billion years ago.

The theory of evolution is now well established in the scientific community but continues to be a commonly discussed topic, as there are several big questions connected to it, such as: How do species evolve? How are species related? What

drives changes in organisms? Are these changes directed or random? And what constitutes a species?

Charles Darwin and other naturalists until the late 20^{th} century primarily based their observations of organisms around their external features such as behaviour, bone structure, skin and fur colouring. These external features are also known as phenotypic traits. Today there are more sophisticated ways to conduct these investigations. With the introduction of genetic sequencing and related technologies, researchers can compare individuals and species at the molecular level. These technologies generate huge amounts of data, creating a challenge to find efficient and accurate means for its analysis.

Whether the data are phenotypic or genotypic, the evolutionary relatedness of groups of organisms (taxa) may be described by a phylogeny [89]. In many cases the evolutionary relationships between taxa can be represented by a graph theoretical object called a phylogenetic tree. A wide range of methods exist to construct phylogenetic trees. Even so, there are some biological processes that lead to non-treelike evolution, such as hybridisation, horizontal gene transfer or recombination. It can also be difficult to distinguish between homology and analogy (see Figure 1.1 for an example of an analogy), which can lead to ambiguous signals in the data.

In these cases, the data might be better represented by a phylogenetic network, a structure that generalizes a tree. Trees just show one of the possible relationships and discard the rest of the signals supported by the data, while networks keep this information. The more general structure of the network enables a representation of ambiguity or a simultaneous representation of a collection of trees.

There are many different kinds of phylogenetic networks and constructing them is an exciting area of current research [50, 65]. Some of these phylogenetic networks try to model specific events, however, this thesis is concerned with networks that give a snapshot of the data and of any existing conflicts. Displaying alternative evolutionary scenarios with networks can come at the price of quite complex

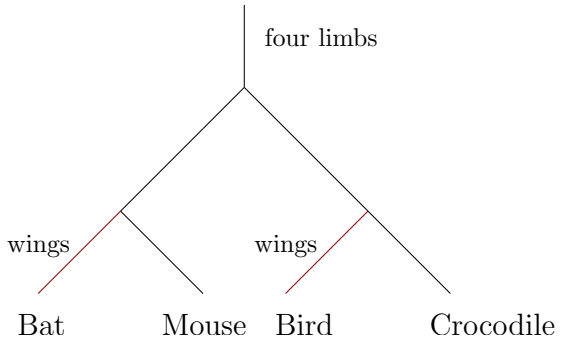


Figure 1.1: An example of an analogy. The four taxa have no common ancestor with wings, but they inherited their forelimbs from a common ancestor.

visual representations, that can contain crossing edges, making their analysis difficult. To avoid this complexity we focus on so-called "circular split networks", which can always be visualised in two dimensions without crossing edges.

A circular split network is a way of summarising a set of trees. On the other hand given a circular split network we can extract a number of trees from this network. In this thesis we will introduce methods to construct circular split networks from trees and, conversely, use them as the search space to find trees that are optimal within this search space according to a certain optimisation criterion.

We now give a brief overview of the contents of this thesis. In **Chapter 2** we give an introduction to the area of phylogenetics, including a brief overview of tree construction principles and an introduction to phylogenetic networks. In addition we explain the basic concepts and definitions essential for the investigations carried out later on.

In **Chapter 3** we introduce *SuperQ*, a new method for constructing circular split networks from collections of trees. It works by first breaking the input trees into quartet trees (trees containing four taxa), and then stitching these together to form a circular split network. Compared with previous super network methods, SuperQ has the advantage of producing a super network that can always be drawn in the plane, in addition to employing a novel approach to incorporating the edge weights from the input trees. As well as presenting the SuperQ method, we com-

pare its performance to existing super network methods, and present an analysis of some published data sets as an illustration of its applicability. Whilst SuperQ was a collaborative effort that was published in [41] my main contribution was the implementation of a new way of using the edge weights from the input trees to calculate weights for the edges in the resulting network. I also designed and performed simulations to assess SuperQ's performance. The implementation of SuperQ is open source and freely available.

In **Chapter 4**, other ways to construct circular split networks are reviewed. More specifically, the NeighborNet algorithm, and a general framework related to it, which can be used to produce circular split networks with interesting properties, are explained. As well as describing these approaches, we present a new way to construct a circular split network by greedily optimising the so-called minimum evolution criterion. I implemented and extended the mentioned methods. A biological dataset consisting of Salmonella sequences is used to illustrate the circular split networks resulting from these methods.

A common approach taken to construct a phylogenetic tree is to search through the space of all possible phylogenetic trees on the set of species so as to find one that optimizes some score function, such as the so-called minimum evolution criterion. However, this is hampered by the fact that the space of phylogenetic trees is extremely large in general. An alternative approach, which has received somewhat less attention in the literature, is to search for trees within some set of bipartitions or splits of the set of species in question. In **Chapter 5** we consider the problem of searching for trees within a set of splits associated to a circular split network. Using simulations and a biological dataset, we compare the performance of our algorithm when applied to the output of the methods introduced in Chapter 4 with that of another leading method for searching for minimum evolution trees in tree space. This approach is based on a preliminary version of this work described in my Diploma thesis [5]. In particular, I extended the method's implementation, reformulated its design, designed and performed new tests to compare and evaluate the method, as well as being involved with an improved description of our algorithm. The work was published in [6].

Another approach to search for an optimal tree within tree space is using tree edit operations. These operations could also be used to search for trees in circular split systems. **Chapter 6** contains answers to several questions, concerning the structure of the subspace of tree space induced by certain tree edit operations, that naturally occurs when investigating this idea. We are interested in special tree edit operations that produce trees that can be found in the same circular split system as the tree that they are applied to. This work is based on a manuscript with Taoyang Wu, Andreas Spillner and Vincent Moulton (in preparation). In this chapter I only present results to which I made major contributions. In particular, we characterised these special tree edit operations and established formulae describing the number of trees that can be found by applying one of these operations to a given tree. This number is dependent on the given tree and therefore we could establish upper and lower bounds for this number, as well as the trees that led to these minimum and maximum numbers.

In **Chapter 7** some conclusions drawn from our investigations are summarised and we give an outlook into future projects and pose some interesting open questions.

Chapter 2

Basic concepts and definitions

2.1 Summary

This chapter explains some relevant concepts used in phylogenetics. We first introduce graphs and trees and then discuss the representation of evolutionary relationships using phylogenetic trees as well as principles for their construction. This is followed by the introduction of the concepts of splits and split systems and their graphical representation as split networks. In the sections concerning networks we mainly follow [53].

2.2 Graphs and trees

Graphs can aid the understanding of large datasets. Here we use them to display evolutionary relatedness of different groups of organisms, which will be referred to as taxa. First we introduce some basic graph theory.

Definition 2.2.1. *(Graph): A graph G is an ordered pair (V, E), consisting of a non-empty set V of* vertices *and a set E of* edges *between the vertices of V, in which every edge $\{x, y\}$ is a 2-element subset of V.*

The vertex set of graph G is called $V(G)$ and the set of edges $E(G)$. We work with finite graphs, that is, both sets $V(G)$ and $E(G)$ are finite. The number of vertices in a graph is its *order* $|V|$; analogously the number of edges is written

as $|E|$ (in general $|Z|$ is used to refer to the number of elements in a set Z). A *cluster* C of a set Z is any non-empty proper subset of Z, denoted $C \subset Z$.

Given a graph G a vertex $x \in V(G)$ is called *incident* to an edge $e \in E(G)$, if $x \in e$. Every edge e has exactly two vertices that are incident to e. An edge $\{x, y\}$ is also written as xy or yx. Two vertices x, y of G are *adjacent* or *neighbours*, if xy is an edge of G. Two distinct edges are *adjacent* if they share a vertex. The *degree of a vertex* x is the number of edges that contain x. If a vertex has degree 1, it is called a *leaf*. The set of leaves of a graph will be written as $L(G)$. If a vertex has a higher degree than 1, it is called an *internal vertex* of G. A vertex with degree 0 is called an *isolated vertex*. An edge is called a *pendant edge* if it is incident to a leaf, and an *interior edge* otherwise.

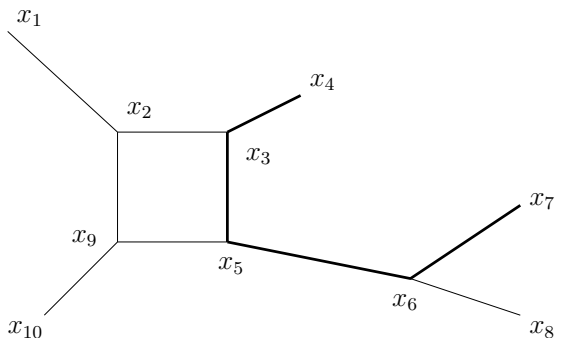


Figure 2.1: A path from v_4 to v_7. The edges in the path are highlighted in bold.

A path P in a graph G is a sequence of pairwise distinct vertices $x_1, x_2, \ldots, x_k$ such that $x_i x_{i+1} \in E(G)$, $i = 1 \ldots k-1$ (see Figure 2.1 for an example). The set of the edges in a path $P = x_1, \ldots, x_k$ is called $E(P) = \{x_1x_2, \ldots, x_{k-1}x_k\}$. A *cycle* is a path $x_1, \ldots, x_k$ with $k > 2$ vertices where x_1 and x_k are connected through an additional edge. A graph G is called *acyclic* if it does not contain any cycles. A graph G is called connected, if every pair of vertices in G is connected by a path. A subgraph of G where all vertices are connected by a path is called a *connected component* of G. Sometimes we weight the edges in a graph G, that is, we have a map $l : E(G) \to \mathbb{R}_{>0}$, which assigns a weight to all edges $e \in E(G)$. The *length of a path* P, denoted by $\ell(P)$, in a graph is the sum of all edge weights

$\omega(e)$, and $e \in E(P)$. If the edges do not have a weight assigned, then the length of a path is the number of its edges. The *distance* $d_G(x, y)$ between two vertices x, y in a graph G is the length of a shortest path from x to y in G, that is the minimum of the lengths of all possible paths from x to y in G. The *total length of a graph* G equals the sum of all of its edge lengths, that is

$$\ell(G) = \sum_{e \in E(G)} \omega(e).$$

We now define a special type of graph that is central to this thesis.

Definition 2.2.2. *(Tree): A tree T is an acyclic connected graph.*

We distinguish between rooted trees and unrooted trees. A rooted tree has a special vertex: the root of the tree. A rooted tree is depicted in Figure 2.2b), where the root is labelled with r. Two trees T_1 and T_2 are *isomorphic* if there exists a bijection $i : V(T_1) \to V(T_2)$ so that $xy \in E(T_1)$ if and only if $i(x)i(y) \in E(T_2)$. Trees that are isomorphic are considered as being equal. A *cherry* of a tree T is a pair of leaves of T that are adjacent to the same interior vertex of T.

2.3 Phylogenetic trees

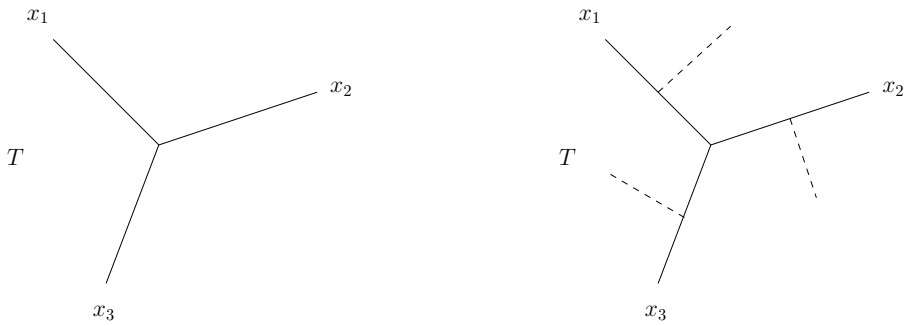
Roughly speaking a phylogenetic tree is a tree where all leaves are labelled uniquely with the elements of some set of taxa. The edges correspond to some event, for example a phenotypic change through mutation. Formally we need to define a more general concept, an *X-tree*. In this section we follow [79]. Let X be a finite non-empty set, called the *label set*. This usually corresponds to some set of taxa or species. An X-tree is defined as an ordered pair (T,ϕ), where $T = (V, E)$ is a tree, and $\phi : X \to V$ is a map from X into the vertices of T such that each vertex $v \in V$ of degree at most two must be contained in $\phi(X)$. It is sometimes also called a *semi-labelled tree*. A *phylogenetic tree* is an X-tree where ϕ is a bijection between X and the leaves of T. Given a subset X' of X, we let $T(X')$ denote the tree consisting of the minimum subtree of T that connects all of the vertices in X'.

a) b)

Figure 2.2: a) An unrooted and b) a rooted binary phylogenetic tree, the root is labelled with r.

An *unrooted phylogenetic tree* has no vertices with degree 2. *Binary unrooted trees* are the trees whose internal vertices all have degree 3 (see Figure 2.2a). A binary tree is also called a *fully resolved* tree. A *rooted phylogenetic tree* has one vertex declared the root and for a binary rooted tree the root has degree 2 while all other internal vertices have degree 3 (see Figure 2.2b). The edges of phylogenetic trees can have edge weights assigned. These weights represent evolutionary distances. In some cases they can be seen as being proportional to the time between different evolutionary states. While in a rooted phylogenetic tree taxa are assumed to have common ancestors, in an unrooted phylogenetic tree this is not made explicit. The tree simply represents the evolutionary relationship between the taxa. In both representations, the inner vertices are unlabelled and are seen as intermediate, unknown or extinct taxa. Note that unless a distinction is made, all phylogenetic trees in this thesis are considered unrooted and binary.

Given an unrooted binary phylogenetic tree T on n taxa, the number of vertices in T is $2n-2$ and the number of edges is $2n-3$. With the help of the last formula the number of all possible unrooted binary phylogenetic trees on n taxa can be determined. If $n = 3$ there is one possible tree (see Figure 2.3a). Adding a new taxon into this tree, results in 3 new possible trees (see Figure 2.3b). These are all possible trees with 4 taxa. Another taxon could be added to all 5 edges of all 3 trees, which results in 3×5 trees. Inductively we can say that the number of possible trees for n taxa is the number of possible trees for $n-1$ taxa times the number of edges in a tree on $n-1$ taxa, the latter of which is $2(n-1)-3 = 2n-5$.



a) b)

Figure 2.3: a) A phylogenetic tree T on 3 taxa and b) the different possibilities to add another taxon into T.

Hence the number of all possible binary trees for n taxa is

$$3 \times 5 \times 7 \times ... \times (2n - 5). \tag{2.1}$$

We have seen that for 5 taxa there are 15 possible trees. Using Equation 2.1 the number of all possible trees for 10 taxa is 2,027,025, and for 30 taxa it is 8.687×10^{36}. The fact that the number of possible binary trees grows so fast with the number of taxa makes tree construction sometimes difficult.

2.3.1 Phylogenetic construction principles and methods

There are two main approaches used to construct phylogenetic trees, which are determined by the kind of input data they use [29]. *Character-based* methods use character data, which in the biological context, are usually either nucleotide or protein sequences but historically were phenotypic characters, typically coded as zeros and ones. *Distance-based* methods use a distance matrix, which are often calculated from sequence alignments, to construct a phylogenetic tree.

2.3.1.1 Character-based methods

For character-based methods the input is character data, commonly nucleotide or protein sequences. The sequences are usually aligned. Given two sequences a and b over some alphabet $\mathcal{A}$ (for example $\mathcal{A} = \{A, T, G, C\}$ for DNA), a *pairwise*

sequence alignment of a and b is one sequence written underneath the other, such that each symbol in one string is opposite to exactly one symbol in the other string. To make both sequences the same length spaces "_" can be inserted. Note that we never allow a space to be aligned to a space to ensure there are only a finite number of possible alignments. In a biological context sequence alignments are a way of arranging the sequences of DNA, RNA, or amino acids to determine differences and similarities that may be a consequence of functional, structural, or evolutionary relationships between the sequences. The construction of a sequence alignment is therefore an optimisation where the two sequences should be aligned such that they differ in as few positions as possible. In a *multiple alignment* spaces are inserted into the elements of a set of sequences such that they all have the same length. All sequences of this set are then written underneath each other in a similar way as to the pairwise alignment (see Figure 2.4).

Scarites	C T T A G A T C G T A C C A A - - - A A T A T T A C
Carenum	C T T A G A T C G T A C C A C A - T A C - T T T A C
Pasimachus	A T T A G A T C G T A C C A C T A T A A G T T T A C
Pheropsophus	C T T A G A T C G T T C C A C - - - A C A T A T A C
Brachinus armiger	A T T A G A T C G T A C C A C - - - A T A T A T T C
Brachinus hirsutus	A T T A G A T C G T A C C A C - - - A T A T A T A C
Aptinus	C T T A G A T C G T A C C A C - - - A C A A T T A C
Pseudomorpha	C T T A G A T C G T A C C - - - - - A C A A A T A C

Figure 2.4: An example of a simple multiple sequence alignment for different genera of beetles in the family Carabidae.

Common construction principles for phylogenetic trees from character data are maximum parsimony [27], maximum likelihood [28] and Bayesian inference [90]. Constructing a phylogeny following the maximum parsimony principle involves choosing the least complex explanation for the data. Finding a phylogenetic tree that explains a given multiple sequence alignment using a minimal number of evolutionary events is called the large maximum parsimony problem and is known to be NP-hard [34], essentially because of the large number of possible trees (see Section 2.1). Finding a labelling for a given tree T and a multiple sequence alignment M that assigns hypothetical ancestor sequences x_i to the in-

ternal vertices such that $\sum_{x_r x_s \in E(T)} \text{diff}(x_r, x_s)$, where $\text{diff}(x_r, x_s)$ is the number of different characters between two sequences x_r and x_s, is minimised, is called the small maximum parsimony problem and can be solved efficiently with the Fitch algorithm [33].

Construction methods that follow the maximum likelihood principle usually try to maximise the likelihood of generating the given multiple sequence alignment M along a given tree T with edge lengths given by ω under a given probability model of sequence evolution $\mathcal{E}$. That is, they try to maximise the probability $P(M|T, \omega, \mathcal{E})$ that M occurs, by carefully selecting T and ω. There are different models of evolution that could be used. A simple one is the Jukes Cantor model of evolution [55], which assumes that all character changes are equally probable. Note that the model $\mathcal{E}$ might itself be the subject of inference. The maximum likelihood approach is quite popular because it tries to take relevant biological processes that generated the data into account. In addition, it is statistically consistent, that is the parameter estimates converge towards their true values with increasing quantities of data. As with the large maximum parsimony problem, the maximum likelihood problem is NP-hard [75].

The aim of Bayesian inference is to estimate the distribution $P(T, \omega, \mathcal{E}|M)$, called the posterior probability, which is the probability of observing tree T (plus branch lengths and other model parameters) given a multiple sequence alignment M. It can be obtained using *Bayes' Theorem*. Usually it is impossible to solve the equation given by Bayes' Theorem analytically and the *Markov chain Monte Carlo* [54] approach is used. This approach constructs a chain of phylogenetic trees. The stationary distribution of these trees then approximates the posterior probability distribution. This approach has become very popular recently as it permits complicated models of evolution to be used.

2.3.1.2 Distance-based methods

The comparison of species through one or more attributes, for example sequences, often leads to a score for measuring their difference. This score can be thought

of as a function $D : X \times X \to \mathbb{R}_{\geq 0}$ that assigns a non-negative real number to each given pair of elements in X, which represents the distance between them. In general two species are seen as closely related if the distance between them is small. The distance between sequences can be found in different ways. For example, the *Hamming distance* defines a simple measure for the difference of two character sequences by counting the number of positions they differ in [43].

The Hamming distance has some properties that also apply to other distance measures. Usually the distance between two sequences is larger than zero and the distance of a sequence to itself is zero, that is $D(x, y) \geq 0$ for all x, y and if $x = y$ then $D(x, y) = 0$. Distance measures are also usually symmetric, that is, $D(x, y) = D(y, x)$ for all $x, y \in X$. If the function also fulfils the triangle inequality ($D(x, z) \leq D(x, y) + D(y, z), x, y, z \in X$) it is called a *metric*. The values of a distance function can also be thought of as a symmetric matrix D, called a distance matrix, having $|X| \times |X|$ entries and a diagonal of zeros.

Given a distance matrix D on X, it is natural to try to find a weighted tree with leaf set X that somehow represents D. As mentioned in section 2.2 the distance between two taxa in a tree is given by the total length of the path between them in the tree. Given a distance matrix D on X, when choosing the edge lengths in a tree T with leaf set X to represent D, it is thus desirable that the distances induced by the tree are close to the distances in D. Let $\ell_{\omega_D}(x, y)$ be the distances between x and y derived from T and D the given distance matrix on X. Then the edge lengths $\omega(e)$ are sometimes chosen such that $\sum_{x,y \in X, x \neq y} \left(\ell_{\omega_D}(x, y) - D(x, y)\right)^2$ is minimal. This is called the *ordinary least squares* method [29].

We can estimate the edge weights according to the ordinary least squares method by using the following formulae (cf. [11, p. 136], [76]). For an internal edge $e = \{u, w\}$, letting v_i, $1 \leq i \leq 4$, denote the four vertices in $V - \{u, w\}$ that are adjacent to u or w, where v_1, v_2 are connected to u and v_3, v_4 are connected to w (see Figure 2.5a). We define the subset $X_i \subseteq X$ consisting of those $x \in X$ for

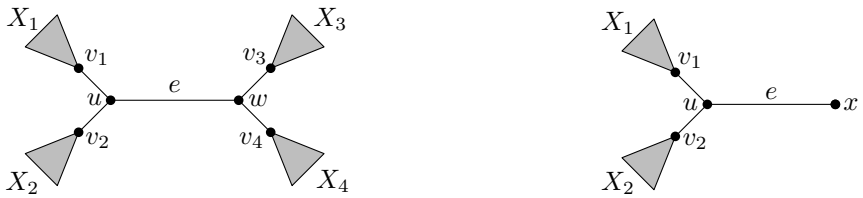
which the unique path from u to x in T contains vertex v_i. Then we have

$$\begin{aligned}\omega_D(e) =& \frac{1}{4(n_1+n_2)(n_3+n_4)}\Big((\frac{n}{n_4}+\frac{n}{n_3}+\frac{n}{n_2}+\frac{n}{n_1}-4)P_0 \\ &+\frac{n_1+n_2}{n_1n_2}((2n_2-n)P_1+(2n_1-n)P_2) \\ &+\frac{n_3+n_4}{n_3n_4}((2n_4-n)P_3+(2n_3-n)P_4)\Big) = \omega_D(X_1,X_2,X_3,X_4),\end{aligned} \tag{2.2}$$

where $P_0 = \sum_{x\in X_1\cup X_2, y\in X-(X_1\cup X_2)} D(x,y)$, $n_i = |X_i|$ and $P_i = \sum_{x\in X_i, y\in X-X_i} D(x,y)$, $1 \leq i \leq 4$. Similarly, for every external edge e (cf. Figure 2.5b) we have

$$\begin{aligned}\omega_D(e) &= \frac{1}{4(n_1n_2)}\Big((1+n_1+n_2)P_0-(1+n_1-n_2)P_1-(1-n_1+n_2)P_2\Big) \\ &= \omega_D(X_1,X_2).\end{aligned} \tag{2.3}$$

Note that in both of these formulae $\omega_D(e)$ only depend on the subsets of X that form the leaf sets of the subtrees incident to the endpoints of e and not on the internal structure of these subtrees.



a) b)

Figure 2.5: Schematic representation of the situation around (a) an internal edge e and (b) an pendant edge e of a phylogenetic tree referred to in the context of Formulae 2.2 and 2.3.

Different criteria can be used to determine how close a given tree based metric is to a given distance [17, 22]. For example, given a tree T and a distance matrix D on X, we can fit weights on all edges using the ordinary least squares method. The *minimum evolution score* for T is the sum of the weights of all edges in T

under ordinary least squares. A tree with the smallest possible score is called a *minimum evolution tree* [17]. Finding a minimum evolution tree in the set of all possible trees for the taxa set, is called the *minimum evolution problem*.

Another criterion is the *balanced minimum evolution criterion*. In contrast to the minimum evolution criterion the edge weights for a given tree T and a distance matrix D on a set X are not estimated using the ordinary least squares method, but a scheme introduced by Pauplin [69]. The weights of external edges can be estimated by

$$\omega(e) = \frac{1}{2}(D(X_1, x) + D(X_2, x) - D(X_1, X_2))$$

and internal edges by

$$\omega(e) = \frac{1}{4}(D(X_1, X_3)+D(X_2, X_4)+D(X_1, X_4)+D(X_2, X_3))-\frac{1}{2}(D(X_1, X_2)+D(X_3, X_4)),$$

where $X_i, \subset X$ denotes the leaf set of the subtrees adjacent to the edge e and x is a single leaf at the end of an external edge (see Figure 2.5). The distances $D(X_i, X_j)$ are the balanced average distances between the sets of taxa X_i to X_j, viz

$$D(X_i, X_j) = D(x, y)$$

if $X_i = \{x\}$ and $X_j = \{y\}$ and

$$D(X_i, X_j) = \frac{1}{2}(D(X_i, X_{j'}) + D(X_i, X_{j''}))$$

if X_j contains two subsets $X_{j'}, X_{j''}$ that are the leaf sets of two distinct subtrees of T_j and $X_{j'} \cup X_{j''} = X_j$. In contrast to the ordinary least squares estimation, where the length of each edge just depends on the leaf sets of the subtrees adjacent to the edges, the internal structure of these subtrees plays an important role in the calculation of the edge length.

In the last part of this section, we briefly review some other distance-based methods for phylogenetic tree construction. In the beginning of phylogenetic tool development, simple algorithms like the UPGMA (unweighted pair group method

using arithmetic averages) algorithm [81] were used. This method works by agglomeration. Another widely used method is the NeighborJoining algorithm. Introduced by N. Saitou and M. Nei [77], it is also an agglomerative method to construct a phylogenetic tree from distance data similar to the UPMGA algorithm. We now review the NeighborJoining algorithm in slightly more detail, because it is relevant later in this thesis.

Given X with $|X| = n$ and distance D on X the algorithm starts with initialising a star tree where the leaves are labelled with the elements of X (see Figure 2.6a) and a set of clusters $C = \{C_1, \ldots, C_n\}$, where each cluster contains one taxon. The distances between the clusters are initialised by the distances between the taxa they contain. Two clusters that minimise a selection criterion, called the *Q-criterion* [13] are chosen and seen as neighbors. The Q-criterion is given as follows:

$$\begin{aligned} Q(C_i, C_j) =& (|C| - 2)D(C_i, C_j) - \sum_{C_t \in C \setminus C_i} D(C_i, C_t) \\ & - \sum_{C_t \in C \setminus C_j} D(C_j, C_t) \end{aligned} \tag{2.4}$$

The two chosen clusters C_{i*} and C_{j*} are thus merged into C_* and the two vertices are pulled out of the star tree to form a cherry. This is illustrated in Figure 2.6 a, where cluster C_2 and C_3 are chosen to be merged. The distances from the newly formed cluster C_* to any other cluster C_t from the set of clusters is given by $D(C_*, C_t) = \frac{1}{2}(D(C_{i*}, C_t) + D(C_{j*}, C_t) - D(C_{i*}, C_{j*}))$. These steps are repeated until the tree is fully resolved, which can be seen in Figure 2.6b - d.

It has been shown that NeighborJoining greedily optimises the balanced minimum evolution criterion [35]. This means, that in each selection the algorithm chooses the local optimum according to this criterion; note that this approach does not always also lead to a global optimum. The resulting tree is called the *NeighborJoining tree*.

In general, the advantage of using distances is that different types of data can

be used as long as a distance measure can be defined. Usually distance based methods are much faster than character based ones, although this can come at a price as the data is usually "compressed" when a distance is computed. The balanced minimum evolution problem is known to be NP-hard, while, to the best of our knowledge, the minimum evolution problem is not proven to be NP-hard, though the structure of the problem suggests that it might be NP-hard.

One heuristic for finding optimal minimum evolution and balanced minimum evolution trees is FastME [21]. It works by modifying a starting tree using tree edit operations, like tree bisection and reconnection (TBR), subtree prune and regraft (SPR) or nearest neighbour interchange (NNI). This local search is repeated until the tree can not be significantly improved any more.

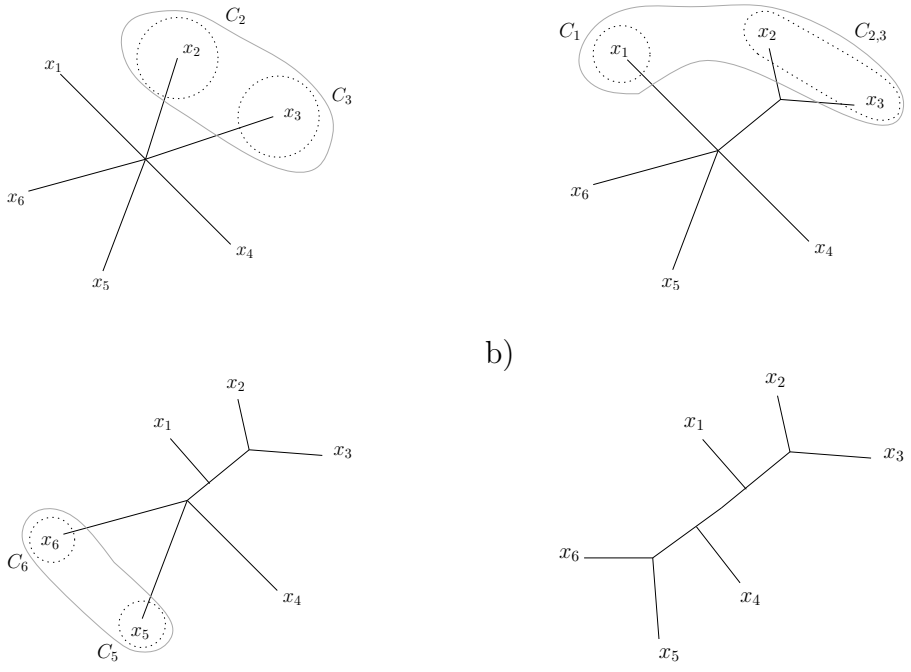


Figure 2.6: An illustration of the NeighborJoining algorithm on $X = \{x_1, \ldots, x_6\}$.

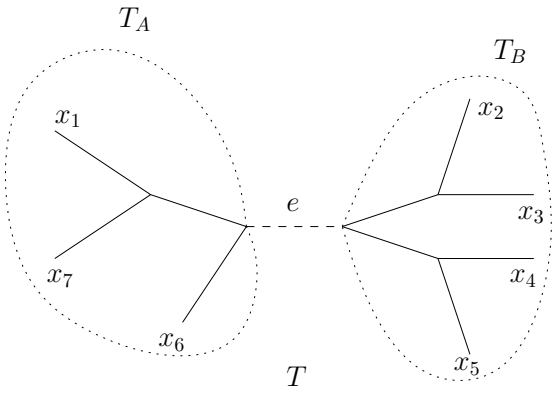


Figure 2.7: A phylogenetic tree T on the set of taxa $X = \{x_1, \ldots, x_7\}$. The edge e corresponds to the split $S_e = \{x_6, x_7, x_1\}|\{x_2, x_3, x_4, x_5\}$. If e is deleted T decomposes into T_A and T_B where $L(T_A) = \{x_6, x_7, x_1\}$ and $L(T_B) = \{x_2, x_3, x_4, x_5\}$. The weight of the split S_e is the length of e.

2.3.2 Characterising phylogenetic trees

Phylogenetic trees can be characterised mathematically in various ways (see [25]). Three ways will be important in this thesis; one based on splits, one on quartets and the other on distances.

2.3.2.1 Splits and split systems

P. Buneman [16] established the fundamental equivalence between X-trees and a special type of collection of bipartitions or *split* of the set X. First, note that each edge in an X-tree induces a bipartition on X (see Figure 2.7). We now formalise this notion.

Definition 2.3.1. *(Split): A split $S = A|B(= B|A)$ is a bipartition of X into two non-empty subsets A and B, which can also be called the two sides of the split.*

The split induced by edge e is denoted S_e. If $|A| = 1$ or $|B| = 1$, then the split $A|B$ is called *trivial*. In trees, trivial splits correspond to pendant edges. A set of splits is called a *split system*. For each edge e in a weighted phylogenetic tree on X, the split corresponding to e can be assigned with the length of the edge. More generally, we will usually assign a non-negative weight $\mu(S)$ to a split S in

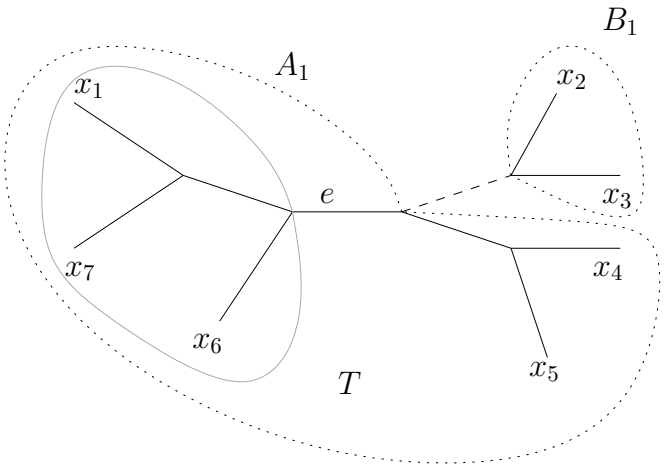


Figure 2.8: A split $A_1|B_1 = \{x_4, x_5, x_6, x_7, x_1\}|\{x_2, x_3\}$ that is compatible with S_e in Figure 2.7, since $T_A \cap B_1 = \emptyset$.

any given split system. Formally, a *weighted split system* is a split system Σ on X and a weight function $\mu : \Sigma \to \mathbb{R}$, that assigns a weight to each split $S \in \Sigma$.

It is well known that every phylogenetic tree T is completely determined by the split system that arises from T (see [16]). Every pair of splits in a split system Σ that is induced by a tree have the following important property [16] (see Figure 2.8).

Definition 2.3.2. *(Compatible) Two splits $S_1 = A_1|B_1$ and $S_2 = A_2|B_2$ are compatible if one of the intersections $A_1 \cap A_2$, $A_1 \cap B_2$, $B_1 \cap A_2$, $B_1 \cap B_2$ is empty.*

On the other hand a split system in which every pair of splits is compatible can always be represented by a necessarily unique X-tree [16]. Thus X-trees and compatible split systems on X are in one-to-one correspondence.

2.3.2.2 Quartets

Another way to determine an X-tree is by a set of quartets.

Definition 2.3.3. *(Quartet tree) Given a phylogenetic tree T on a set X, and any set $X' \subseteq X$ of four taxa, the smallest subtree $T(X')$ of T that connects the four given taxa is called the quartet tree for the given taxa.*

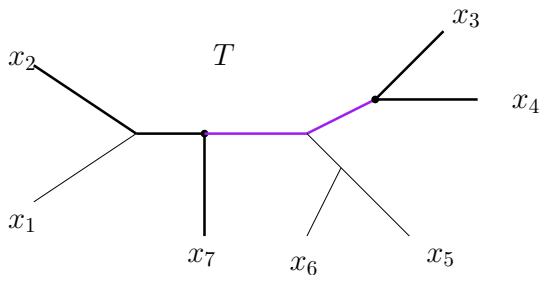


Figure 2.9: The smallest subtree (highlighted with bold edges) of the phylogenetic tree T that connects the taxa in $\{x_2, x_3, x_4, x_7\}$. It gives rise to the quartet $q = x_7x_2|x_3x_4$ whose weight is the total length of the bold purple edges.

In the case where $T(X')$ has precisely two distinct vertices u and v of degree three, the path P connecting u and v in $T(X')$ separates two of the four given taxa, say a and a', from the remaining two taxa, say b and b' (see Figure 2.9). We denote this fact by $aa'|bb'$ and refer to $aa'|bb'$ as the quartet *induced* by the given phylogenetic tree on the taxa a, a', b and b'. The weight $\kappa(q)$ of a quartet q induced by a phylogenetic tree is the total length of the edges on the path P. It is well known that the set of quartets induced by a tree determines the tree (that is no other tree induces the same set of quartets) [79, Chapter 6].

2.3.2.3 Distances

An X-tree can also be characterised by the distances that it induces on X, using the following condition:

Definition 2.3.4. *(4-point condition) Let D be a distance function on a set X. The 4-point condition is fulfilled for D if the following holds for all $w, x, y, z \in X$:*

$$D(w,x) + D(y,z) \leq \max\{D(w,y) + D(x,z), D(w,z) + D(x,y)\}.$$

It is easy to see that the distances induced by a weighted phylogenetic tree satisfy this condition (see Figure 2.10). Conversely, if the 4-point condition is fulfilled there exists an X-tree T and the distances between the leaves induced by T are equal to D on X [16].

a) b)

Figure 2.10: The distances in a tree on the 4 taxa w, x, y, z are highlighted. In a) $D(w, x)$ and $D(y, z)$ are highlighted in green and the distances $D(x, y)$ and $D(w, z)$ in blue. In b) the distances $D(w, y)$ and $D(x, z)$ are highlighted in red. In this example $D(w, x) + D(y, z)$ must be smaller than $D(w, y) + D(x, z)$ and $D(w, z) + D(x, y)$. Using any other labelling for the leaves it is easy to see that the 4-point condition always holds.

2.4 Phylogenetic networks

A phylogenetic network can be thought of as the generalisation of a phylogenetic tree. Evolutionary data often contain conflicting signals. These signals can arise for different reasons; they can be the result of biological conflicts, caused by several biological processes; or through estimation errors, like inappropriate sampling, incorrect data or model mis-specification [65]. Phylogenetic networks are useful for displaying such data conflicts.

In general, a phylogenetic network is any graph used to present evolutionary relationships between a set of taxa that labels some of its vertices (usually the leaves) [53]. In order to reconstruct evolution, networks that explicitly represent an evolutionary event, like hybridisation, horizontal gene transfer, recombination, gene conversion, duplication or loss, are desirable. However, they are not easy to construct [53]. Such networks are often called *explicit or evolutionary phylogenetic networks*. Vertices in these networks can be seen as representing ancestors. Explicit networks are usually rooted, that is just as for rooted trees, they have a special vertex or root representing the most recent common ancestor. Some common rooted phylogenetic networks are, for example, cluster [51], hybridisation [61] and recombination networks [37] (see Figure 2.11 for an example).

In contrast a *data-displaying* or *implicit network* displays conflicting patterns

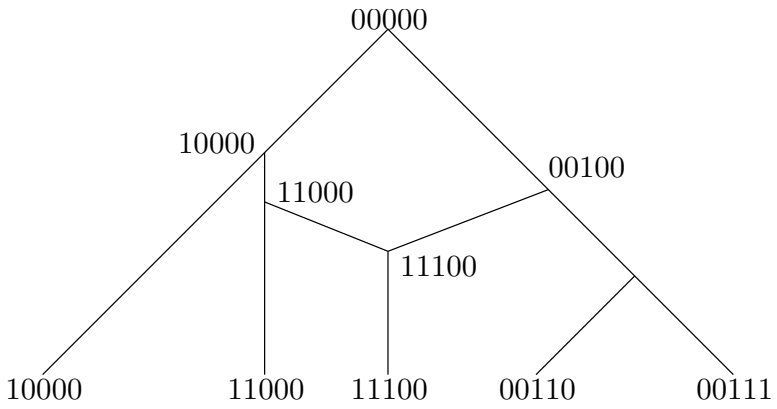


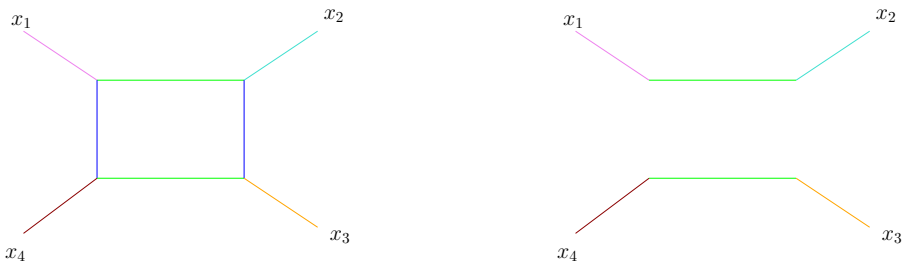
Figure 2.11: A recombination network constructed from binary sequences. Each vertex is labelled with a binary sequence, the sequences of the internal vertices can be seen as ancestor sequences. This network shows one recombination event between sequences 11000 and 00100 that combined and produced one offspring sequence 11100.

in the data; the vertices do not necessarily represent common ancestors. In this thesis the focus is on implicit phylogenetic networks. These are usually not rooted. The main examples of implicit networks are split networks and quasi-median networks [53]. Split networks are explained in more detail in the next section and some new methods for their construction from a set of trees as well as from distance data are discussed in Chapters 3 and 4.

2.4.1 Split networks

Split networks are a generalisation of phylogenetic trees that can be used to visualise split systems that are not necessarily compatible [4, 50]. As already mentioned, compatible split systems can always be represented as an (unrooted) phylogenetic tree. Similarly, an arbitrary set of splits can always be visualised by a split network, in which one or more edges represent a split. A special property of these edges is that if they are all deleted, the split network breaks apart into two connected components [24]. The two sides of the split correspond to the elements of X in the two components, as illustrated in Figure 2.12.

Before we can formally define a split network we have to define some related



a) b)

Figure 2.12: a) A split network on $X = \{x_1, x_2, x_3, x_4\}$. b) The deletion of the blue edges leads to two connected components; one contains the vertices x_1, x_2 the other one x_3, x_4, therefore the blue edges correspond to the split $\{x_1, x_2\}|\{x_3, x_4\}$.

concepts. Let G be a finite, connected graph, C be a finite set of labels, which we call colors, and $\sigma : E(G) \to C$ be a mapping that assigns a color to each edge in G such that no two adjacent edges are given the same color. An undirected path P in G with length $\ell(P) = t$ is called *properly colored*, if all edges along the path have different colors, that is $|\sigma(P)| = t$. If for any two vertices all shortest paths between these vertices are properly colored and use the same set of colors, then σ is called an *isometric coloring*. A *split graph* is a connected graph G together with an isometric and surjective edge coloring $\sigma : E(G) \to C$.

Definition 2.4.1. *(Split network) [53] Let Σ be a set of splits on X. A split network $N = (V, E, \sigma, \lambda)$ that represents Σ is given by a split graph $(G, \sigma : E(G) \to \Sigma)$ and a vertex labelling $\lambda : X \to V(G)$ such that for every split $S = A|B \in \Sigma$, we have*

$$A = \bigcup_{v \in V_S^0} \lambda^{-1}(v) \text{ and } B = \bigcup_{v \in V_S^1} \lambda^{-1}(v)$$

where V_S^0 is the vertex set of one of the two connected components G_c^0 that result if all edges with the same color c are deleted and V_S^1 the vertex set of the other component.

Instead of using colors, we will represent edges that correspond to the same split (or would have the same color) by bands of parallel edges, and the length of these edges are proportional to the weight of the split. Boxes in the network correspond to pairs of splits that are *not* compatible, thus indicating conflict in the

two groupings of the taxa in X. An example of a split network is illustrated in Figure 2.13. If the split system is pairwise compatible, the split network is just the unique tree corresponding to the system. Note that such a representation is always possible, although the networks in general can be very complicated and therefore difficult to visualise.

Restrictions to split systems can be applied to avoid overly complicated networks. One such restriction is the existence of a special ordering of the taxa in X. Formally, a split system Σ on X is *circular* if there is an ordering $\pi = (x_1, \ldots, x_n)$ of the elements of X, such that all splits $S \in \Sigma$ have the form:

$$S := S_{i,j} = \{x_{i+1}, \ldots, x_j\}|\{x_{j+1}, \ldots, x_n, x_1, \ldots, x_i\}$$

for some i, j, $1 \leq i < j \leq n$. If such an ordering exists, it is called a *circular ordering* (relative to Σ). A split that is circular respects a certain circular ordering. Note that every compatible split system is a circular split system [4].

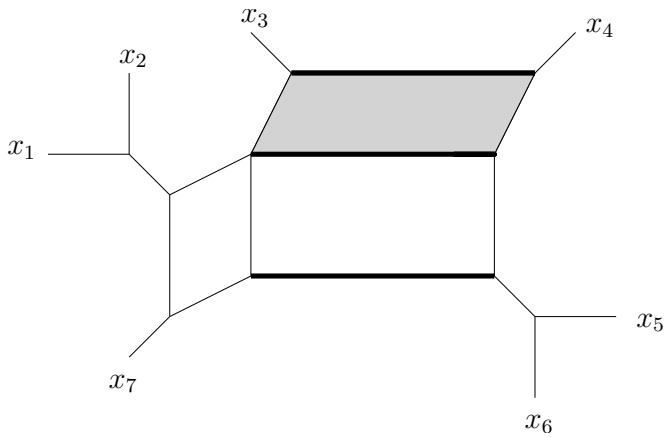
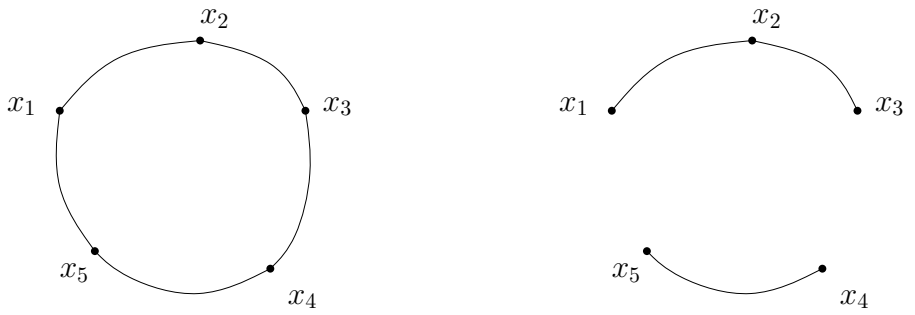


Figure 2.13: An example of a split network on $X = \{x_1, ..., x_7\}$. The band of bold parallel edges represents the split $S_1 = \{x_7, x_1, x_2, x_3\}|\{x_4, x_5, x_6\}$. The shaded box indicates that the splits S_1 and $S_2 = \{x_5, x_6, x_7, x_1, x_2\}|\{x_3, x_4\}$ are incompatible.

To get a better idea of circular split systems, one can draw the taxa on a circle as seen in Figure 2.14a. If two links in this circle are deleted then the circle



a) b)

Figure 2.14: a) A circular split system, b) two links are deleted, which leads to two connected paths. The vertices connected by each path correspond to the elements on the two sides of the split $S = \{x_1, x_2, x_3\}|\{x_4, x_5\}$.

decomposes into two components. This decomposition gives a split of the taxa. In this manner a circular ordering π gives rise to the set of all possible splits that respect π, called the *full circular split system* induced by π.

A circular split system can always be represented as an *outer-labelled planar split network*, that is, a circular split network as seen Figure 2.13 that can be drawn in the plane so that all taxa lie on the outside of the network (cf. Dress and Huson [24] for more details). Such a network is therefore easier to visualise. We will refer to this kind of network as a circular split network.

There are various methods to construct split networks. For example a popular method that constructs a circular split network from distances is the NeighborNet algorithm [14], which extends the NeighborJoining algorithm (see Section 2.3.1.2) and is reviewed in more detail in Chapter 4.

Another popular way of constructing split networks is from a collection of trees. For a set of trees where each tree has the same leaf set, strict consensus or majority consensus networks can be computed by strictly just including all splits that occur in all trees in the input or including all splits that occur in more than 50% of the input trees [53]. A set of partial trees, trees with incomplete leaf sets, can be summarised in a super network. Common methods to construct super

networks from partial trees are Q-imputation [47] and Z-Closure [52, 88]. Z-closure essentially works by converting the input trees into a collection of partial bipartitions where the partial bipartitions correspond to edges in the input trees, and then iteratively applying a set-theoretical rule so as to produce new collections of bipartitions of the total leaf set. This collection is then represented by a split network. Q-imputation, on the other hand, first completes the partial trees to trees on the total leaf set, using quartets, and then constructs a consensus network for these trees.

2.5 Concluding remarks

In this chapter we have introduced some fundamental concepts in phylogenetics which underline the research presented in this thesis. We saw that a phylogenetic tree is a basic and straightforward approach to represent evolutionary relationships between species and that there are different ways to characterise such trees mathematically, in particular, splits, which play an important role in this thesis. A phylogenetic tree might not always be appropriate to capture complex evolutionary histories. In the following chapter we describe a method to summarise information from different trees in a circular split network.

Chapter 3

SuperQ: Computing super networks from quartets

This chapter is an adaptation of the text presented in the paper: S. Grünewald, A. Spillner, S. Bastkowski, A. Bögershausen, V. Moulton. SuperQ: Computing Super networks from Quartets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(1): 151-160, 2013.

3.1 Summary

In this chapter, we introduce *SuperQ*, a new method for constructing split networks from collections of trees. It works by first breaking the input trees into quartet trees, and then stitching these together to form a split network. This stitching process is performed using an adaptation of the QNet method for split network reconstruction [38]. Compared with previous super network methods, SuperQ has the advantage of producing a super network that can always be drawn in the plane, in addition to employing a novel approach to incorporating the edge weights from the input trees. As well as presenting the SuperQ method, we compare its

performance to the Z-closure and Q-imputation super network methods, and also present an analysis of some published data sets as an illustration of its applicability. SuperQ is freely available at `http://www.uea.ac.uk/computing/superq`.

3.2 Background

In phylogenetics it is common practice to summarise a collection of phylogenetic trees as a consensus tree [12] or, in case the trees are on different leaf sets (that is they are partial trees), as a super tree [8]. Even so, such trees can be limited by the fact that conflicting groupings in the input trees cannot be represented in any single tree. In such situations, it can be helpful to also construct a consensus network or a super network as described in the last chapter, since these can permit the visual representation of such conflicts, though at the price that a tree is no longer necessarily reconstructed.

In a similar way to super trees, super networks can be regarded as a special type of phylogenetic network which is used to give a visual representation of a collection of partial phylogenetic trees. Here we shall be only concerned using super networks to represent unrooted phylogenetic trees: there are some variants that can be used for rooted phylogenetic trees (cf. for example Huson et al. [53, Chapter 11]), but methods for their construction are still in quite early stages of development. Super networks have been used for applications such as visualising large collections of trees (arising for example from Bayesian tree inference) [46], analysis of multiple gene trees [52], the study of genome size evolution [62], the study of species radiations and species delimitation [64] and the analysis of congruence in phylogenetic data [59].

In this chapter we describe a new method for constructing super networks from a collection of partial, unrooted phylogenetic trees. As already mentioned Z-Closure and Q-imputation (see Section 2.4.1) are available to construct such super networks. SuperQ works in a different manner. Essentially, each of the input partial trees is first broken down into a collection of quartet trees, after which the

collection is stitched together to form a split network. To perform this stitching process, we adapt the QNet algorithm for constructing split networks from quartet trees [38, 40], and subsequently compute edge lengths for the resulting network by optimising a function of the edge lengths of the input trees. SuperQ has the advantage that, in contrast to Z-closure and Q-imputation, it is guaranteed to generate a circular split network. Such networks have proven useful for analysing various types of phylogenetic data [66], and can be easier to interpret than the sometimes quite complex networks that Z-closure and Q-imputation can produce.

3.3 Method

We first introduce some terminology and notation - more details concerning these and related concepts may be found for example in [25, 79]. We then describe the SuperQ method. Note that a preliminary version of this method is presented in [39] and that the main new contribution described in this thesis are steps 1 and 4, which are described in detail in this section.

3.3.1 Quartets and networks

To build super networks, we will use the quartets induced by the input trees (see Section 2.3.2.2). Any split network on X induces a collection of quartets as follows. Consider the system Σ of weighted splits visualised by the network. A quartet $q = aa'|bb'$ is induced by Σ if there exists a split $S = A|B$ in Σ such that a and a' are contained in A while b and b' are contained in B. We will also say that the split S *extends* the quartet q. The weight $\kappa(q)$ of any quartet q induced by Σ is the *total weight* (that is the sum of the weights) of those splits in Σ that extend q. Note that the quartets induced by a phylogenetic tree T, as described in Section 2.3.2.2 are precisely those induced by the corresponding compatible split system. Moreover, the weight assigned to any quartet $aa'|bb'$ according to this split system is just the sum of the weights of those edges that lie on the path P in the smallest subtree of T spanned by a, a', b, b' as also described before.

3.3.2 Description of the method

We assume that we are given as input a collection $\mathbf{T} = \{T_1, T_2, \ldots, T_t\}$ of phylogenetic trees. Note that these trees may be *partial*, that is, they need not be all on the same set of taxa. We shall let X denote the set consisting of all those taxa that are a leaf of at least one tree in $\mathbf{T}$. In particular, the set of taxa labelling any tree in $\mathbf{T}$ is always a subset of X. SuperQ computes a split network summarising $\mathbf{T}$ using the following five steps:

Step 1: Optionally scale the trees in $\mathbf{T}$.

Step 2: Break up all of the trees in $\mathbf{T}$ into quartets to produce a collection $\mathbf{Q}$ of weighted quartets.

Step 3: Input $\mathbf{Q}$ into the QNet algorithm to obtain a split system Σ.

Step 4: Compute suitable weights for the splits in Σ.

Step 5: Visualise the weighted split system Σ by a circular split network.

We now present a more detailed description for each of these steps.

3.3.2.1 Step 1: Scaling the input trees

As with previous super network (and super tree) methods that use edge length information from the input trees (see for example [9]), our method implicitly assumes that a unit of edge length has the same interpretation for all of the trees in the input set $\mathbf{T}$. Clearly, this might not always be the case. For example, the trees in $\mathbf{T}$ could be the result of different studies or be produced using different methods, and so the trees could have different scales. To cope with this, we first scale each of the input trees T_i, $i = 1, 2, \ldots, t$, by a non-negative factor σ_i. These factors are chosen so that the function

$$h(\sigma_1, \sigma_2, \ldots, \sigma_t) = \sum_{1 \leq j < k \leq t} \sum_{q \in \mathbf{Q}(T_j) \cap \mathbf{Q}(T_k)} (\sigma_j \cdot \kappa_j(q) - \sigma_k \cdot \kappa_k(q))^2 \qquad (3.1)$$

is minimised under the constraint that $\sigma_1 + \sigma_2 + \cdots + \sigma_t = 1$ must hold, where $\mathbf{Q}(T_i)$ denotes the collection of quartets induced by T_i and $\kappa_i(q)$ denotes the

induced weight of the quartet q in $\mathbf{Q}(T_i)$, $1 \leq i \leq t$. Intuitively, (3.1) measures, for each pair of distinct trees $T, T' \in \mathbf{T}$ and each quartet q induced by both T and T', the difference in the induced weight of the quartet q in T and T' and aims to minimise the sum of all these squared differences. Note that minimising (3.1) amounts to solving a so-called quadratic program for the unknown scaling factors σ_i, which can be done efficiently using well-known algorithms (see for example [23]).

3.3.2.2 Step 2: Breaking the input trees into quartets

Let $\mathbf{Q}$ be the collection of quartets that can be formed using elements in X. We assign a weight $\kappa^*(q)$ to each quartet $q = aa'|bb'$ in $\mathbf{Q}$ as follows. Let ℓ denote the number of trees in $\mathbf{T}$ that contain all of the four taxa a, a', b and b'. If $\ell = 0$ we simply put $\kappa^*(q) = 0$. Otherwise we let $\kappa^*(q)$ be the sum of the weights $\kappa_i(q)$ over those trees T_i in $\mathbf{T}$ that induce q divided by ℓ. In other words, q is assigned the average weight that it receives over all those trees in $\mathbf{T}$ that relate all four taxa involved in q.

3.3.2.3 Step 3: Applying QNet

QNet is described in [40], but as it is a key step in our approach we briefly summarise its main features here. It is a heuristic algorithm that computes, for any input collection $\mathbf{Q}$ of weighted quartets (such as the one computed in Step 2), an ordering $x_1, x_2, \ldots, x_n$ of the taxa in X. As described in Chapter 2 such an ordering induces a full circular split system Σ. It has the useful property that it (and any subcollection of splits from this system) can always be represented by an outer-labelled planar split network, which means it is always drawable in the plane (see Section 2.4.1 Figure 2.13 for an example). Essentially, QNet tries to construct an ordering of X such that for the associated split system Σ the total weight of all those quartets in $\mathbf{Q}$ that are extended by some split in Σ is as large as possible. In this way it is hoped that Σ will capture a large proportion of the structural information contained in $\mathbf{Q}$.

3.3.2.4 Step 4: Computing split weights

Suppose that Σ has been computed using QNet in Step 3 using the collection $\mathbf{Q}$ of quartets computed in Step 2. Let $S_1, S_2, \ldots, S_k$ be an arbitrary ordering of the splits in Σ. We now use the weights of the quartets in $\mathbf{Q}$ to compute weights for each of the splits in Σ (which will correspond to edge lengths in the split network representing Σ). Let $\mathbf{Q}^* = \{q_1, q_2, \ldots, q_m\}$ be the set of those quartets q in $\mathbf{Q}$ for which the four taxa present in q appear together in at least one tree in $\mathbf{T}$. Note that, since the set of taxa in any of the trees in $\mathbf{T}$ can be a proper subset of X, $\mathbf{Q}^*$ can be a proper subset of $\mathbf{Q}$. One can think of $\mathbf{Q}^*$ as the set of those quartets q in $\mathbf{Q}$ for which we have relevant information coming from the input trees about the target weight $\kappa^*(q)$ of q.

Computing weights for the splits in Σ amounts to assigning a non-negative real number $\mu(S)$ to each split $S \in \Sigma$ so that the induced vector of quartet weights is as close as possible to the vector of target weights in the least squares sense. More formally, we minimise the following objective function

$$g(\mu(S_1), \mu(S_2), \ldots, \mu(S_k)) = \sum_{q \in \mathbf{Q}^*} (\kappa^*(q) - \sum_{\substack{S \in \Sigma \\ S \text{ extends } q}} \mu(S))^2. \tag{3.2}$$

Note that this optimisation problem does not necessarily have a unique solution. This is partly due to the fact that the weights of the *trivial* splits in Σ, that is, splits of the form $\{x\}|X - \{x\}$ for some element $x \in X$, do not have any impact on the value of the objective function. The reason for this is that quartets do not convey any information about the length of the pendant edges in a phylogenetic tree or, equivalently, the weight of the trivial splits. Therefore, in the following we assume that all trivial splits have been removed from Σ.

Even with all trivial splits removed, (3.2) need not have a unique solution. To shed some light on this issue, it is helpful to consider the $m \times k$-matrix $M = (M_{i,j})$ with entries from the set $\{0, 1\}$ that is defined by putting $M_{i,j} = 1$ if and only if the quartet q_i is extended by the split S_j. Then, arranging the split weights we want to compute as well as the target quartet weights in vectors

$\mu = (\mu(S_1), \mu(S_2), \ldots, \mu(S_k))$ and $\kappa^* = (\kappa^*(q_1), \kappa^*(q_2), \ldots, \kappa^*(q_m))$, respectively, we can write (3.2) as

$$g(\mu) = \|\kappa^* - M \cdot \mu\|_2 \tag{3.3}$$

where $\|w\|_2$ denotes the squared Euclidean length $w_1^2 + w_2^2 + \cdots + w_m^2$ of an m-dimensional vector $w = (w_1, w_2, \ldots, w_m)$ with real-valued entries.

Now, it is well known that (3.3) has a unique optimal solution if and only if the matrix M has full rank, that is, the product $M \cdot \mu$ yields the null vector only in case μ is the null vector. Therefore, in cases where M does not have full rank, we offer the user the option to add a second optimisation step as follows. Suppose we have an arbitrary optimal solution μ_0 of (3.3). Then, guided by the minimum evolution principle used to reconstruct phylogenetic trees (see for example [17]), among all μ with $M \cdot \mu = \mathbf{0}$, we compute one that minimises some suitably chosen objective function $g'(\mu)$ subject to the constraint that the resulting vector of split weights $\mu_0 + \mu$ has non-negative entries.

Below we use a linear objective function for which an optimal solution can be found very efficiently using *linear programming* [63] although, at least theoretically, the solution need not be unique. Note that there is some freedom in the choice of the coefficients in the objective function. In preliminary tests, we found that treating all splits in Σ equally could lead to a bias towards *balanced* splits, that is, splits $S = A|B$ with A and B containing roughly the same number of elements. The reason for this, is that balanced splits display more quartets than unbalanced ones and, therefore, have a bigger impact on the resulting quartet weights. To address this bias, in the following we shall use the objective function

$$g'(\mu) = \sum_{S=A|B \in \Sigma} |A| \cdot (|A| - 1) \cdot |B| \cdot (|B| - 1) \cdot \mu(S), \tag{3.4}$$

that is, the coefficient for the split $S = A|B$ corresponds to the number of quartets $aa'|bb'$ that can be formed by selecting a and a' from A and b and b' from B.

3.3.2.5 Step 5: Computing a split network for Σ

We use the algorithm presented in [24] and implemented in the SplitsTree software package [50] to compute a split network visualising the split system Σ computed in Step 4. As noted above, since Σ is circular, this algorithm is guaranteed to produce a circular split network. To ease the readability of the network, at this stage we put the trivial splits back into Σ. Each trivial split is assigned the same weight, namely the average weight of the non-trivial splits in Σ.

3.3.3 Implementation and properties of SuperQ

SuperQ has been implemented using the Java programming language. It includes the option to switch off the scaling of the input trees and also provides alternatives to objective function (3.4). The method for filtering the resulting split system described in [38] has also been integrated into SuperQ: The user can provide a real-valued threshold t, $0 < t \leq 1$, to remove from Σ any split S for which there exists some other split S' in Σ such that S and S' are not compatible and the weight of S is less than a fraction t of the weight of S'. The implementation of SuperQ is freely available under the GNU general public license 3.0 at `http://www.uea.ac.uk/computing/superq`. We used this implementation in the computational experiments described below (using the scaling of input trees, objective function (3.4) and a threshold $t = 0.1$), which were performed on a PC with an Intel Core 2 Quad 2.4 GHz CPU, with 8 GB of main memory using the operating system Ubuntu (version 11.04).

The run time of our implementation is superpolynomial in the worst case. This is due to the fact that Steps 1 and 4, where the input trees are scaled and suitable weights for the splits in Σ are computed, involve solving quadratic and linear programs. This is done in the current implementation using the algorithms available through the Gurobi Optimiser, version 4.5 (`www.gurobi.com`). The other steps of SuperQ, in particular computing the split system Σ, can be done in $O(t \cdot n^4)$ time where n is the number of taxa in $|X|$ and t is the number of trees in $\mathbf{T}$.

3.4 Results

In this section we present the results of our investigation on the performance of SuperQ for simulated data sets and biological data sets.

3.4.1 Simulations

In view of the fact that SuperQ is a heuristic for finding a solution for a generalisation of the super tree problem, which is known to be NP-hard in general [83], there is no guarantee that it will always produce a split system that induces all quartets in the input trees in case such a split system exists. This is related to the question as to what extent SuperQ reconstructs the dominant splits of the input trees. To shed more light on this, and to assess the performance of SuperQ as compared with previous super tree/super network methods, we performed a simulation following the approach presented in Holland et al. [47] (cf. also [52] for a similar approach).

More specifically, we first generated a random phylogenetic tree T according to the Yule-Harding model [44, 91] with n leaves, also called the *underlying tree*. Since previous methods such as Q-imputation do not take edge length information into account, we assigned each edge in that tree length 1. Then we generated a collection $\mathbf{T}$ of t partial trees by randomly selecting and removing k leaves from T, for some fixed k, repeating this t times in total (if necessary we repeated this process to ensure that every leaf of T was contained in at least one of the partial trees in $\mathbf{T}$). Similarly, to understand the impact of conflict in the input trees, we generated a collection $\mathbf{T}$ of t partial trees by randomly applying k' SPR moves to T, for some fixed k', again repeating this t times in total. We took $n = 16, 32$, $k, k' = 1, 2, 3, 4, 5, 6$, and $t = 2, 4, 8, 16, 32$.

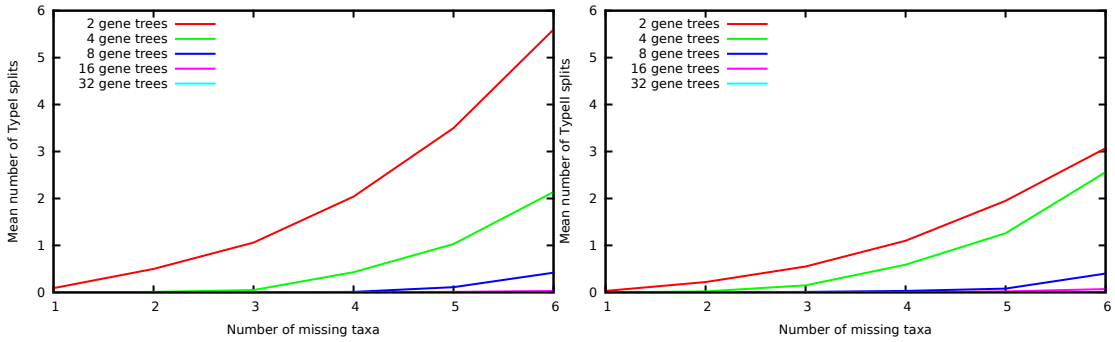
We then measured the deviation of the output split system Σ to the split system corresponding to the underlying tree T, by measuring the number of *Type I* and *Type II* splits which are given as follows. A Type I split is a split in Σ that does not arise from T, while a Type II split arises from T but is not contained in Σ. We generated 100 collections $\mathbf{T}$ for each combination of n, k/k' and t; the

number of Type I and Type II splits presented is an average for each of these collections. Note that we count the number of Type I/Type II splits for the purpose of comparison with methods such as Q-imputation as these do not generally assign weights to the splits in the output.

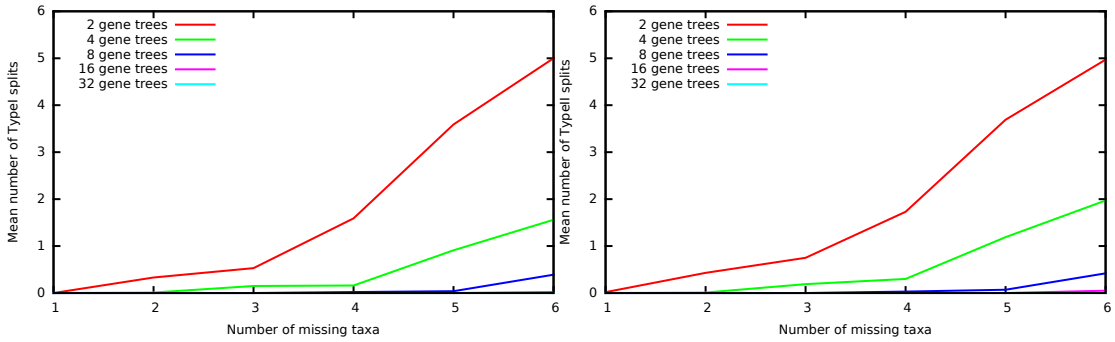
The simulation study in Holland et al. [47] included Q-imputation, Z-closure and the widely used super tree method *matrix representation with parsimony (MRP)* [7, 73]. It was found in Holland et al. [47] that for all three methods the number of Type I splits grows with the number of missing taxa for a fixed number of partial trees in $\mathbf{T}$. The rate of growth found was similar for Q-imputation and MRP and, for these methods, the number of Type I splits decreased with increasing number of trees in $\mathbf{T}$. This is illustrated in Figure 3.1a for Q-imputation and, as can be seen, SuperQ exhibits a similar behaviour (Figure. 3.1c). In contrast, it was found in Holland et al. [47] that for Z-closure the number of Type I splits increased with an increasing number of trees in $\mathbf{T}$.

For Type II splits the picture is more clear-cut: it was found in Holland et al. [47] that for all methods the number of Type II splits increases, for a fixed number of partial trees in $\mathbf{T}$, with the number of missing taxa, and it decreases with an increasing number of trees in $\mathbf{T}$. In Figures 3.1b and d, we present the simulation results for Type II splits for Q-imputation and SuperQ which illustrates that SuperQ shares this behaviour, a fact that was also confirmed in the simulations for $n = 32$ (plots included in the Appendix A material).

The simulations for different numbers of SPR moves in Holland et al. [47] suggest that again it is the behaviour with respect to Type I splits that is more variable for the different methods. In particular, it was found in Holland et al. [47] that (i) for MRP the number of Type I splits increases, for a fixed number of trees in $\mathbf{T}$, with the number of SPR moves, and it decreases with the number of trees in $\mathbf{T}$, (ii) for Q-imputation the number of Type I splits, for a fixed number of trees in $\mathbf{T}$, levels off or even decreases slightly with the number of SPR moves and it increases with the number of trees in $\mathbf{T}$, and (iii) for Z-closure the number of Type I splits increases both with the number of SPR moves and with the number



a) b)



c) d)

Figure 3.1: a) Type I splits and b) Type II splits for Q-imputation and c) Type I splits and d)Type II splits for SuperQ for gene trees with missing taxa generated from a random tree with 16 taxa.

of trees in $\mathbf{T}$ (see Figure 2 in the Appendix A material for the for the latter two methods). Our simulations for SuperQ (see Figure 3 in the Appendix A material) indicate that it has a behaviour that is quite similar to that of Q-imputation in that the number of Type I splits levels off with an increasing number of SPR moves. However, in contrast, the number of these splits for SuperQ is much smaller than for Q-imputation or for Z-closure.

3.4.2 Biological data sets

To illustrate the applicability of SuperQ we present its application to three data sets. These datasets were chosen as they have a range of sizes and properties, and have also, in part, been used to test previous super network methods.

The first data set that we consider was recently presented in Tepe et al. [86] and consists of 10 gene trees (available from TreeBASE) relating 15 species from the genus *Solanum*. For some of these species multiple accessions were included in the study, resulting in a set of 24 taxa. In Figure 3.2 we present the super networks constructed using Q-imputation, Z-closure and SuperQ. The colored taxa are from section *Herpystichum* of *Solanum*. They are divided into the group of ground-trailing vines (pink) and climbing vines (green). The taxa in black represent other sections of *Solanum*.

As can be seen, all three methods confirm the finding in Tepe et al. [86] that there exists a major split that separates the climbing taxa of section *Herpystichum* from the ground-trailing taxa of the same section and the taxa representing other sections of *Solanum*. The degree of conflict in the input trees referred to in Tepe et al. [86] is reflected by the non-treelikeness of the Z-closure and the SuperQ networks. The Q-imputation network is a tree and only for a threshold of 0.16 or less do non-treelike parts start to appear in the consensus network. According to Tepe et al. [86] most of the conflict in the trees is related to grouping the ground-trailing taxa of section *Herpystichum*. This is represented more clearly in the SuperQ network than in the other networks. In all three networks there is no clear split that separates the taxa of Section *Herpystichum* from the other taxa. In Tepe et al. [86] some methods gave strong support for such a split while others gave only weak support. In particular the relationship between section *Pteroidea* (represented by the two *S. anceps* taxa) and section *Herpystichum* cannot be completely resolved.

An attractive feature of the super network analysis for this example is that it gives a visual impression of the different possible groupings discussed in Tepe

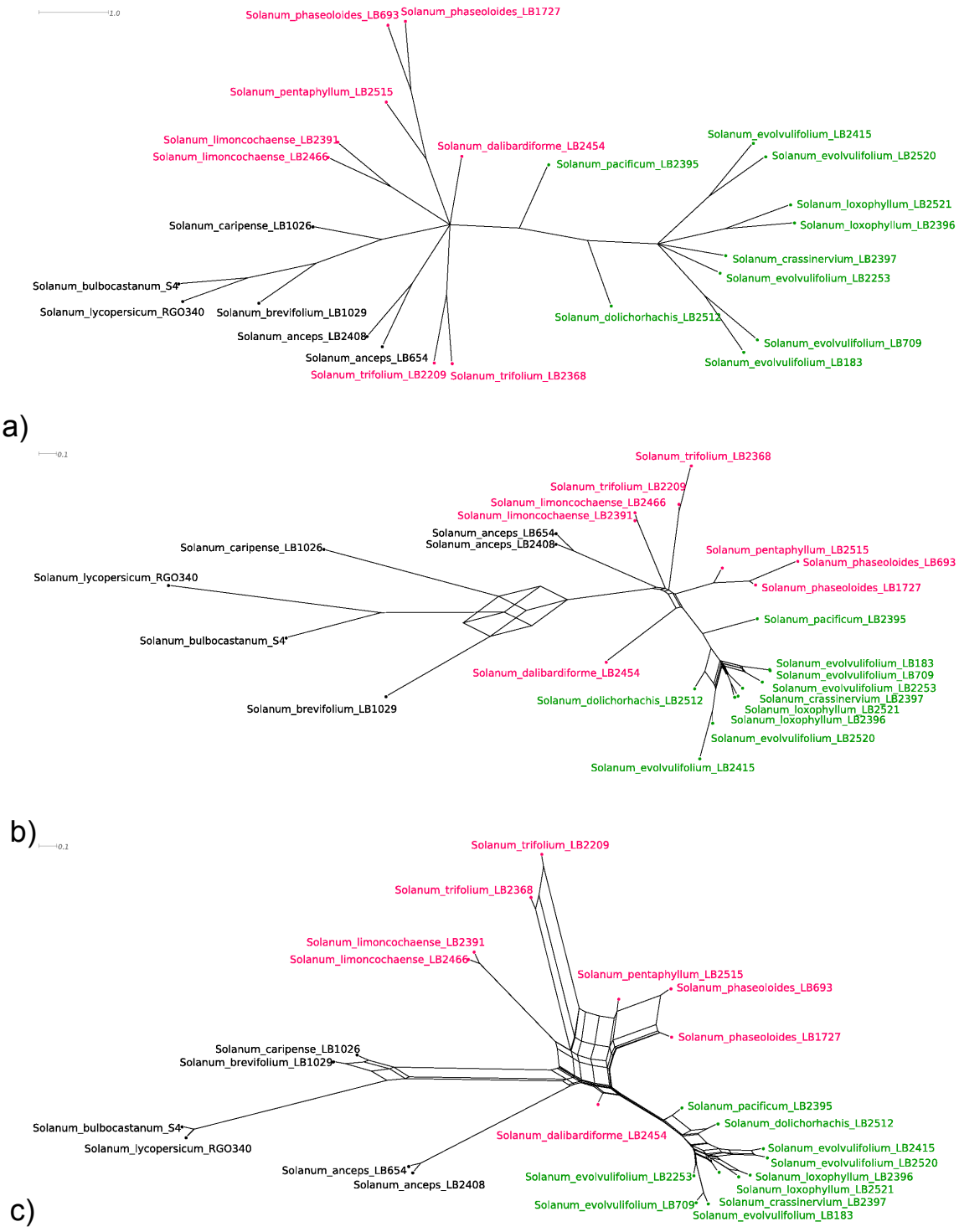


Figure 3.2: Super networks for the *Solanum* data set. a) Q-imputation consensus network (threshold 0.33). b) Z-closure network (filtered super network with standard options except MinNumberTrees set to 3). c) SuperQ network.

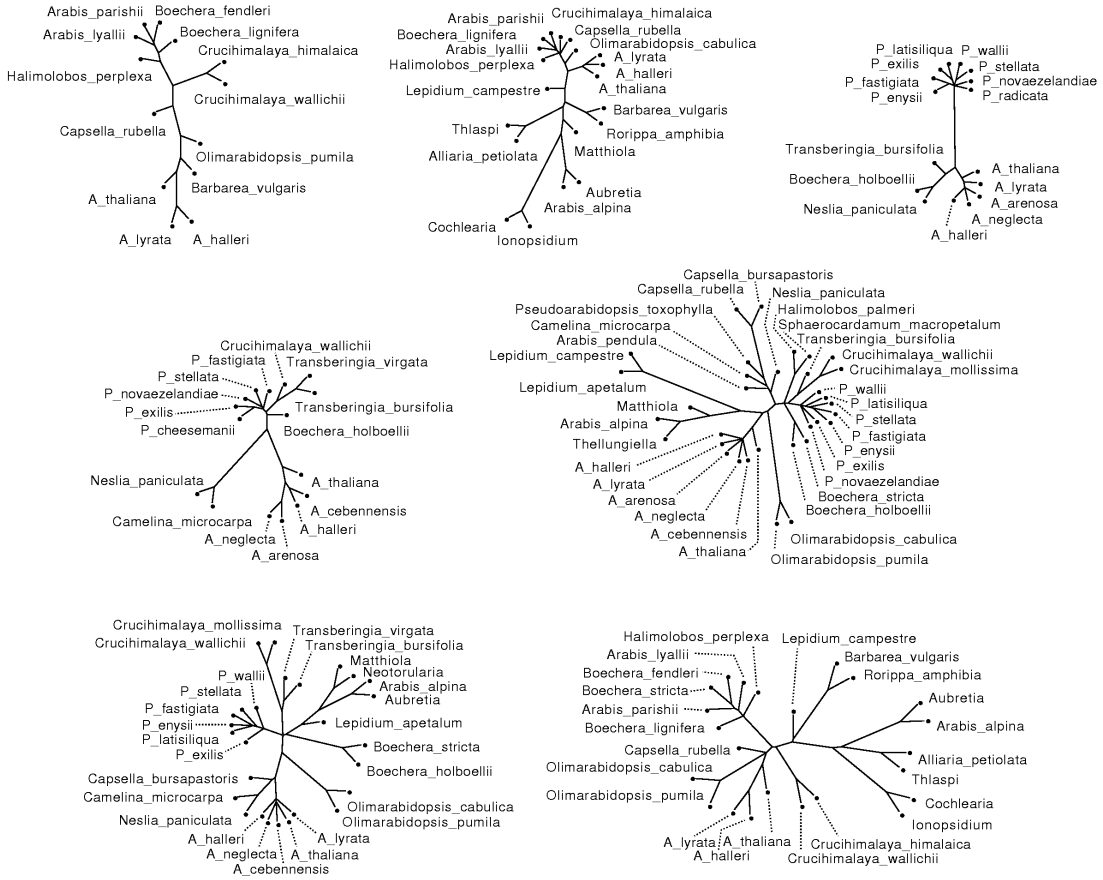


Figure 3.3: Seven partial gene trees showing 50 species of *Brassicaceae*.

et al. [86] at a single glance. For example, some support exists for a split that groups *S. lycopersicum* and *S. bulbocastanum* together with the two *S. trifolium* taxa. Also note in the Z-closure network the relationship between the non-colored taxa is represented so that edges of the network cross each other. In contrast, the SuperQ network is always guaranteed to be without any crossing edges which can help improve the readability of more complex networks.

The second data set consists of 7 maximum likelihood trees estimated from independent genome loci of flowering plants from the family *Brassicaceae*. The gene trees, which are depicted in Figure 3.3, were reconstructed using nuclear and chloroplast nucleotide sequences obtained from GenBank, and others deter-

mined as part of a study on phylogenetic relationships among close relatives of *Arabidopsis* (McBreen et al., unpublished). The total number of taxa is 50. Note that in the trees shown in Figure 3.3 all pendent edges have the same weight. As noted above, these weights do not influence the quartet-weight function κ^* and we only included these edges to make the trees easier to read.

In contrast to the previous data set, the matrix M in objective function (3.3) does not have full rank and so this example illustrates the impact of the choice of the objective function g'. The network constructed by SuperQ using objective function (3.4) is depicted in Figure 3.4a. This network contains a split S that separates all species of the genus *Pachycladon* (abbreviated by P) from the other taxa. Note that, as this split is not balanced, it is lost from the output if we do not take into account the bias towards balanced splits (see Fig. 3.4b). In particular, instead of S, we obtain splits that separate some or all of the *Pachycladon* species together with other species, such as *Rorippa amphibia* and *Barbarea vulgaris*, from the remaining taxa. One of the reasons for this is that the input trees do not give much information on the grouping of the *Pachycladon* species relative to some other species. For example, none of the input trees contains a *Pachycladon* species as well as *Rorippa amphibia* or *Barbarea vulgaris*.

The third data set consists of five gene trees for fungal species (see Figure 3.5) which was published in Pryor and Bigelow [71], Pryor and Gilbertson [72] and used as an example for Z-closure in Huson et al. [52]. The total number of taxa is 64. The network obtained by SuperQ for the fungi data set is depicted in Figure 3.6. Here again the matrix M did not have full rank. When we compare this super network with the output of Z-closure presented in Huson et al. [52, p. 156] (see also Figure 3.7) we find that both networks agree on many of the major splits. The Z-closure network, however, contains many crossing edges in the central part of the network. The SuperQ network is planar and, therefore, the conflicts depicted might be somewhat easier to grasp. Moreover, SuperQ resolves parts of the Z-closure network such as, for example, the grouping of the *Alternaria* taxa marked red in Figure 3.6. We suspect that this is due to the fact that SuperQ takes into account edge lengths in the input trees whereas Z-closure

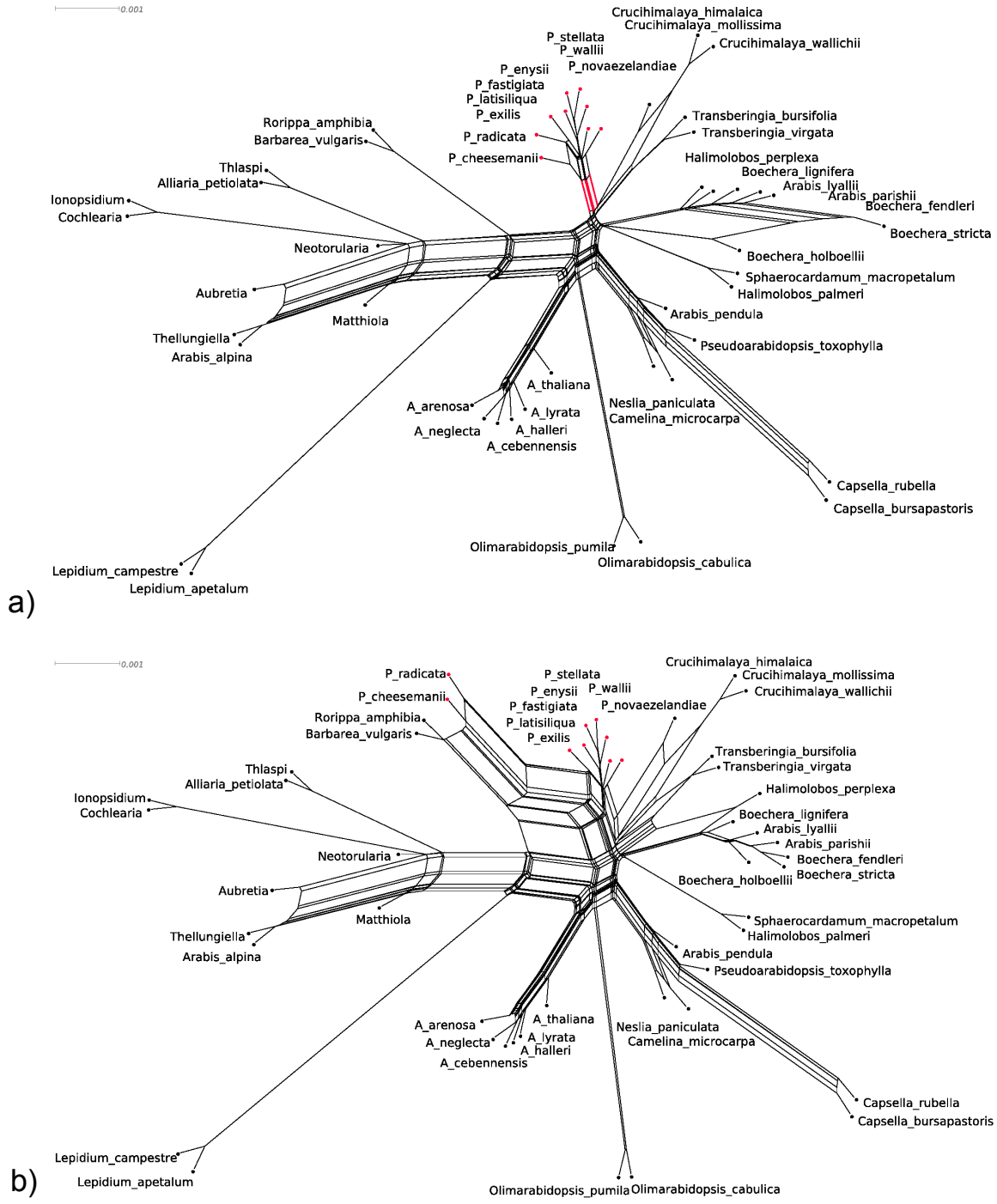
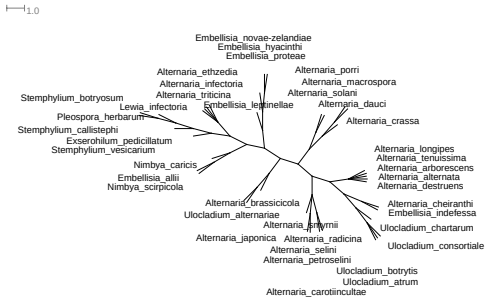
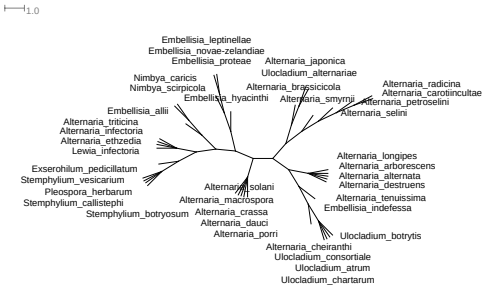


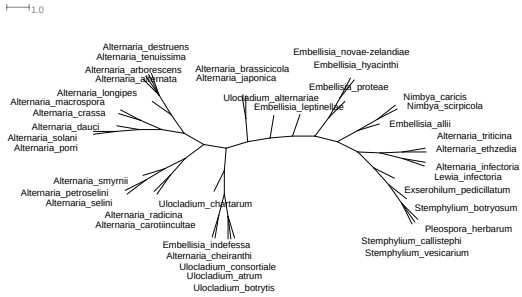
Figure 3.4: The super network constructed by SuperQ for the *Brassicaceae* data set using a) the objective function (3.4), and b) a linear objective function that does not take into account the bias towards balanced splits. Edges marked in red in a) correspond to the split separating the *Pachycladon* species from the other taxa, which does not appear in network b).



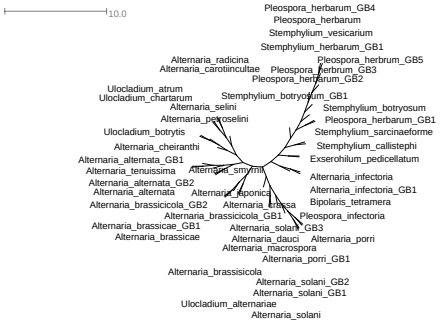
a)



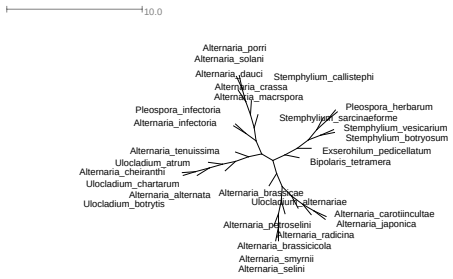
b)



c)



d)



e)

Figure 3.5: The five gene trees for fungal species published in Pryor and Bigelow [71], Pryor and Gilbertson [72].

does not, leading, potentially, to a loss of information.

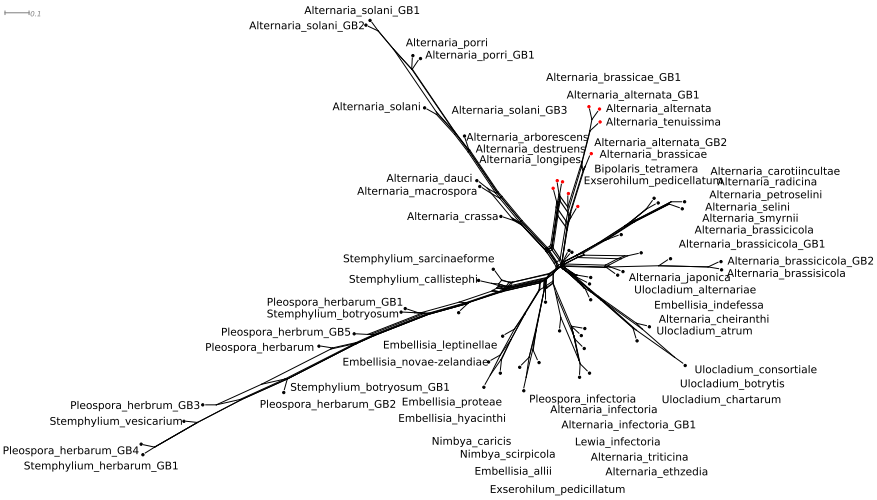


Figure 3.6: The super network constructed by SuperQ for the five partial gene trees from Pryor and Bigelow [71], Pryor and Gilbertson [72] showing 64 species of fungi.

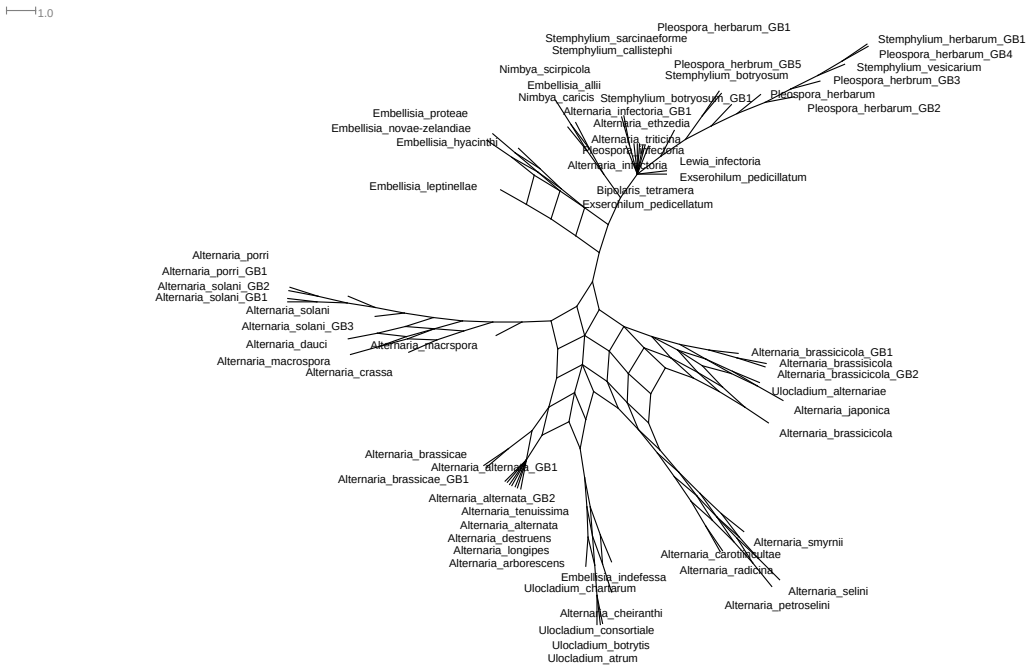


Figure 3.7: The super network constructed by Z-closure for the five partial gene trees from Pryor and Bigelow [71], Pryor and Gilbertson [72] showing 64 species of fungi.

3.5 Discussion

In this chapter we have presented SuperQ, a new method to construct super networks from partial trees. The simulation and illustrative examples indicate that the method could be useful in practice. In particular, SuperQ tends to generate results that are in some agreement with Z-closure and Q-imputation, but with the advantage that a circular split network is always produced.

There are some important differences between SuperQ and Z-closure/Q-imputation that are worth highlighting. For example, in contrast to Z-closure, SuperQ does not depend on the order in which the input data is processed (the output of Z-closure can potentially depend on the order in which splits from the input trees are processed). We should note, however, that for SuperQ ties might occur in the score function used in the QNet algorithm, although such ties will probably occur rarely in practice for real-valued quartet weights. In addition, in contrast to Q-imputation, the input to SuperQ is not restricted to partial trees but, as with Z-closure, SuperQ could clearly be applied more generally to any collection of partial split networks. This could be a useful feature as the rules used in Z-closure were designed under the assumption input data do not contain too much conflict.

Concerning the SuperQ algorithm, while the objective function (3.4) used in the optional second optimisation step seems to work well in practice, there are other variants that might be worth considering. For example, minimising the squared Euclidean length of the resulting weighting vector seems very natural and, in addition, it has the (at least theoretically) appealing property that it always yields a unique optimal solution. In practice, however, we found that the resulting split system tended to contain many splits with small positive weight which led to uninformative networks. Another natural choice, yielding only a lower bound on the weights that should be assigned to the splits in Σ, minimises, for each split S in Σ, the weight $\mu(S)$ that is assigned over all μ with $M \cdot \mu = \mathbf{0}$ and $\mu_0 + \mu$ non-negative. This amounts to solving a family of k linear programs, adding an increasing computational burden compared to minimising (3.4). Even so, it could

potentially allow a more detailed investigation of individual splits that are not assigned a unique weight in all optimal solutions of (3.2).

For the simulations and data sets presented above we found that the computational time needed to process a data set with the current implementation of SuperQ was similar to that of Z-closure and Q-imputation. More generally, we found that the average run time for SuperQ applied to randomly generated input collections with 10 trees involving 50 taxa was just over 2 minutes, whilst for 100 taxa the average run time increased to 54 minutes (see Table 1 in Appendix A for more data). We also found that for the biological data sets the run time was slightly less than expected from these experiments: it took approximately 22 seconds and 63 seconds to construct the networks for the *Brassicaceae* and fungi data sets, respectively. This could be because these data sets have less conflict than randomly generated ones. Even so, the main drawback of SuperQ, in common with most quartet-based methods (such as for example QNet), is the high memory consumption arising from storing and handling the collection of quartets derived from the input trees, which grows asymptotically with n^4 for n taxa. Reducing the memory consumption involved in our approach could be an interesting direction for future research.

In conclusion there are various methods available for computing phylogenetic super networks. There is great deal of scope for further developing such methods, especially for rooted phylogenetic trees (cf. for example [53, Chapter 11]), and using them it should be possible to gain a greater understanding of the complexities hidden within collections of partial trees. As the number of fully sequenced genomes continues to grow and, correspondingly, the need for analyzing collections of trees, these tools should continue to become increasingly important.

3.6 Concluding remarks

In this chapter we presented SuperQ, a new method to construct super networks from partial trees and compared it to two other leading super network methods. SuperQ is a novel approach for incorporating the edge weights of the input trees

in the construction of a circular split system, which can be represented as a circular split network. It offers a scaling method for the input trees and the input is not restricted to phylogenetic trees but could also be split networks. In the next chapter we look at approaches of constructing circular split networks from distances rather than quartets and trees.

Chapter 4

Construction of circular split systems from distances

4.1 Summary

In this chapter, we review some ways to construct circular split systems from distances. This will be useful for the next chapter, where we will describe a method to search for trees in such split systems. The NeighborNet algorithm, introduced by D. Bryant and V. Moulton [14], is a popular tool for constructing a weighted circular split system that can be represented by a circular split network. In [60] D. Levy and L. Pachter propose a framework for constructing circular split systems based on NeighborNet, and pointed out that, depending on the adjustment of this framework, it can be used to produce circular split systems with interesting properties. For example, constructing a circular split system in a certain way guaranties that it contains a compatible split system corresponding to the NeighborJoining tree. As well as reviewing both of these approaches we use Levy and Pachter's framework to describe a new way to construct a circular split system by greedily optimising the minimum evolution criterion.

All of these methods operate on distance data and output a circular ordering, which gives rise to a full circular split system. Weights for the splits can then be subsequently calculated in another step using the method of ordinary least

squares (often used with non-negative constraints), which we shall also review. A by-product of all circular ordering constructions is thus a circular split network. In order to use these methods in the investigations carried out in the next chapter we have also implemented them in Java. We shall illustrate the circular split networks resulting from the various methods with a biological dataset consisting of Salmonella sequences.

4.2 Original NeighborNet

The NeigborNet algorithm [14] constructs a collection of weighted circular splits from a distance matrix. This collection of weighted circular splits can be represented by a circular split network. NeighborNet is widely used to provide a snapshot of the input data and is closely related to the NeighborJoining algorithm [77] that we briefly described in Section 2.3.1.2.

The construction of the collection of weighted circular splits can be broken up into two main steps: the derivation of a circular ordering of the taxa that determines a full circular split system and the calculation of the weights of the splits. Note that for displaying splits with a positive weight in a split network the network construction algorithm in [53] can be used.

4.2.1 Construction of the circular ordering

The input for the NeighborNet algorithm is a distance matrix D on a set of taxa X with $|X| = n$. A graph $G = (V, E)$ where $V(G) = X$ and $E(G) = \emptyset$ is initialised. Recall that a connected component of G is a maximal connected subgraph of G, therefore initially we have n connected components, all isolated vertices, in G. A distance D_C between connected components in G is also computed which depends on the distance D_V between vertices in $V(G)$ (see below for details). Initially, both distances $D_V = D_C = D$. An example for this initial state is given in Figure 4.1 1.

We now give an overview of the algorithm (see Figure 4.1). NeighborNet itera-

tively updates the graph G by selecting two connected components and connecting them by inserting an edge between one vertex of one selected component and one vertex of the other selected component. The vertices that are considered to be connected by the inserted edge must be either isolated vertices or end vertices of a path, so they have either degree 0 or 1 before the connection. Since the connected components in G change as the algorithm progresses the distance D_C is also updated. In this manner connected components of G, consisting of paths that connect vertices in G, are built. The graph G is updated again if a path reaches a length of at least two edges. In this case the three connected vertices are replaced by two new vertices that are connected by an edge. Since the vertex set of G changes it is then also necessary to update D_V. All these steps are repeated until G consists of one connected component. This path gives rise to a circular ordering of the elements of X by back-substituting the vertices that have been replaced before. We will now clarify this agglomerative process by detailing the steps of NeighborNet. Note that we also provide the pseudo code in the Appendix B.

First selection step

At any stage in the algorithm, let G consist of components $C_1, C_2, \ldots, C_m$. In order to update G two components C_{r*} and C_{s*} of G are chosen, that minimise the Q-criterion,

$$\begin{aligned} Q(C_r, C_s) =& (m-2)D_C(C_r, C_s) - \sum_{t=1\ldots m, t\neq r} D_C(C_r, C_t) \\ & - \sum_{t=1\ldots m, t\neq s} D_C(C_s, C_t) \end{aligned} \tag{4.1}$$

where

$$D_C(C_r, C_s) = \frac{1}{|C_r||C_s|} \sum_{x\in C_r} \sum_{y\in C_s} D_V(x, y).$$

In the first selection step there are three different scenarios of which connected components can be selected (see Figure 4.2):

- two isolated vertices, are selected (see Figure 4.2a),

- an isolated vertex and two connected vertices are selected (see Figure 4.2b), or
- two connected components of G consisting of two connected vertices are selected (see Figure 4.2c).

Note that this first step is equivalent to the selection step in the NeighborJoining algorithm. If we keep track of the selected components, we can reconstruct a tree in the same way as in the NeighborJoining algorithm. Initially there is a star tree T with its leaves labelled with the taxa in X. In each iteration T is updated. Choosing two components can be seen as making a cherry in T using the selected components. This process is illustrated in parallel to building a circular ordering in Figure 4.1.

Second selection step

Let C_{r*} and C_{s*} be the components of G chosen to optimise the Q-criterion in the first selection step. As illustrated in Figure 4.2 there are several possibilities for how to connect the vertices in the selected components. One of these possibilities is selected in a second selection step. More specifically, the vertices $x_{i*} \in C_{r*}$ and $x_{j*} \in C_{s*}$ are chosen to be connected by an edge if they minimise the score

$$\begin{aligned}\hat{Q}(x_i, x_j) =&(\hat{m} - 2)D_V(x_i, x_j) - \sum_{t \neq r*, s*} D_C(x_i, C_t) - \sum_{t \neq r*, s*} D_C(x_j, C_t) \\ &- \sum_{k \neq i} D_V(x_i, x_k) - \sum_{k \neq j} D_V(x_j, x_k)\end{aligned} \tag{4.2}$$

where $\hat{m} = m + |C_{r*}| + |C_{s*}| - 2$, because here the vertices in the connected components are treated as individual vertices.

Reduction

The insertion of an edge between the vertices chosen in the second selection step can result in one of three possible scenarios:

- The resulting path consists of one edge and therefore no reduction is performed.

- The resulting path consists of two edges and therefore one reduction must be performed. In particular, the three connected vertices are replaced by two connected vertices. An example of this reduction process is shown in Figure 4.3. Note that this process makes it necessary to calculate the distances between the two new vertices and all other vertices in G, that is update D_V and D_C.

- The resulting path consists of three edges and therefore two reductions must be performed. First, two adjacent edges are replaced by one edge and then the remaining two edges are reduced again to one edge. An example of this reduction process is shown in Figure 4.4. Note that this process makes it necessary to update the distances D_V and D_C twice.

Updating the distances

Since in the reduction step three vertices are replaced by two, the distances from these two new vertices to all other vertices in G need to be updated. Let x, y, z be three vertices in G connected by xy and yz, that are replaced by the two vertices u, v and that a is any other vertex in G. Then the distances from u to a and v to a as well as from u to v are calculated using the following formulae:

$$D_V(u,a) = (\alpha+\beta)D_V(x,a) + \gamma D_V(y,a),$$

$$D_V(v,a) = \alpha D_V(y,a) + (\beta+\gamma)D_V(z,a),$$

$$D_V(u,v) = \alpha D_V(x,y) + \beta D_V(x,z) + \gamma D_V(y,z),$$

where α, β, γ are non-negative real numbers and $\alpha+\beta+\gamma = 1$. NeighborNet uses $\alpha = \beta = \gamma = \frac{1}{3}$ by default.

By keeping track of the insertion of edges into G and the reductions, a circular ordering π can be constructed once G consists of one connected component. In each step two vertices are connected and therefore a path through the vertices is constructed and this path gives rise to π (see Figure 4.1 7).

4.2.2 Calculation of the split weights

The constructed circular ordering π of the elements of X gives rise to a full circular split system $\Sigma(\pi)$ which consists of all possible splits $\{S_1, \ldots, S_k\}$ induced by π. NeighborNet uses the least squares framework, as mentioned in Chapter 2, to compute weights for these splits. The input distance matrix D on X has $m = \frac{n(n-1)}{2}$ unique entries and is represented as the m-dimensional vector

$$D = \begin{pmatrix} D_{1,2} \\ D_{1,3} \\ \vdots \\ D_{(n-1),n} \end{pmatrix}$$

where $D_{i,j}$ corresponds to the distance between x_i and x_j. Let A be the matrix with m rows representing pairs of taxa and k columns representing all splits in $\Sigma(\pi)$, given by

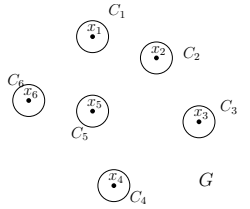
$$A_{\{i,j\},t} = \begin{cases} 1 & \text{if } x_i \text{ and } x_j \text{ are on different sides of } S_t \\ 0 & \text{else.} \end{cases}$$

In order to compute split weights we solve the equation $D = Ab$ which represents a system of linear equations. The vector b gives the split weights. We want to choose each split weight such that the sum of the weights of all splits where x_i, x_j are on different sides of the split is equal to the distance between these two elements of X in D. Since a full split system is considered, the matrix A has full rank [3] and therefore the system of linear equations has a solution. In particular, a simple rearrangement of the equation yields the formula

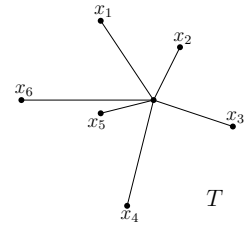
$$b = (A^T A)^{-1} A^T D.$$

Note that this formula can result in negative weights for splits, which is not suitable for the representation as a network. Therefore the ordinary least squares estimation with a non-negative constraint is used, which means that entries of b are constrained to be non-negative. This can be solved using an algorithm by

Lawson and Hanson [58]. This non-negative least squares version has no closed formula and leads to an increase in run time.

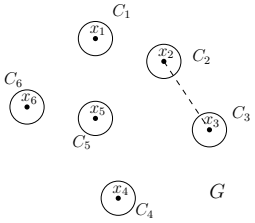


1a)

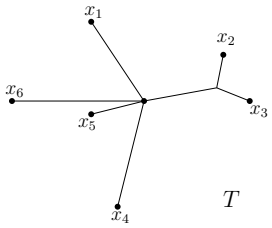


1b)

1a) Initialisation of NeighborNet, each component of G consists of one vertex. 1b) The initialisation of a star tree T on 6 vertices.

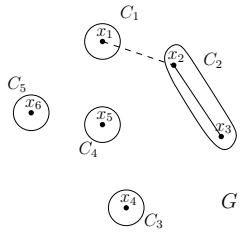


2a)

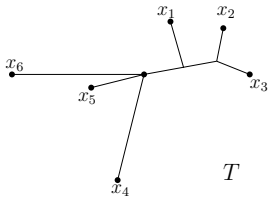


2b)

2a) Selection of component C_2 and C_3, an edge is inserted between x_2 and x_3. 2b) The selection of the components corresponds to making a cherry in T, here vertices x_2 and x_3.

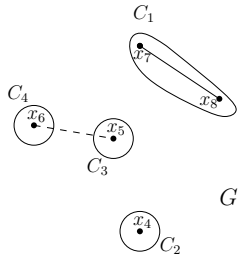


3a)

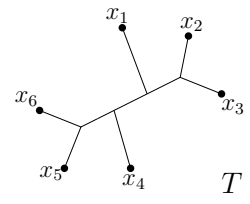


3b)

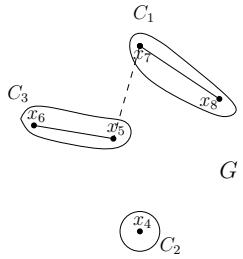
3a) Selection of component C_1 and C_2, an edge is inserted between x_1 and x_2. 3b) The component C_2 and vertex x_1 form a new cherry in T. 4a) After the reduction the component C_1 consists of 2 vertices x_7, x_8. The components C_3 and C_4 are then selected and an edge is inserted between x_5 and x_6. 4b) A cherry with vertices x_5 and x_6 is made.



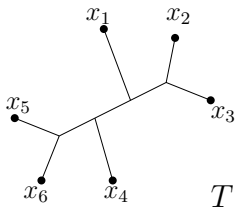
4a)



4b)

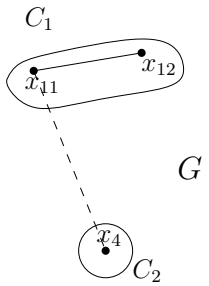


5a)



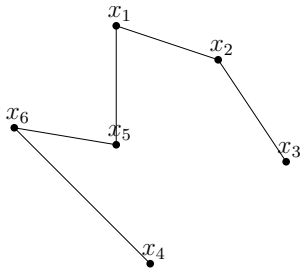
5b)

5a) Component C_1 and C_3 are selected, an edge is inserted between x_5 and x_7. 5b) The tree is the same as in 4b, but the ordering of the leaves is changed (considering the tree embedded in the plane), x_5 and x_6 are flipped.



6)

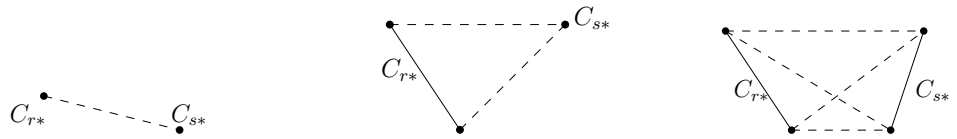
6) After two reductions the component C_1 is formed. Component C_1 and C_2 are selected and an edge is inserted between x_{11} and x_4. The path induces the ordered set (x_4, x_{11}, x_{12}).



7)

7) We can now substitute two adjacent elements of this ordered set back. Here x_{11}, x_{12} are substituted by x_6, x_5, x_7, x_8, which gives $(x_4, x_6, x_5, x_7, x_8)$. Within this ordered set we can now substitute x_7, x_8 by x_1, x_2, x_3 which gives us the final circular ordering $\pi = (x_4, x_6, x_5, x_1, x_2, x_3)$.

Figure 4.1: An example for the NeighborNet algorithm on $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$.

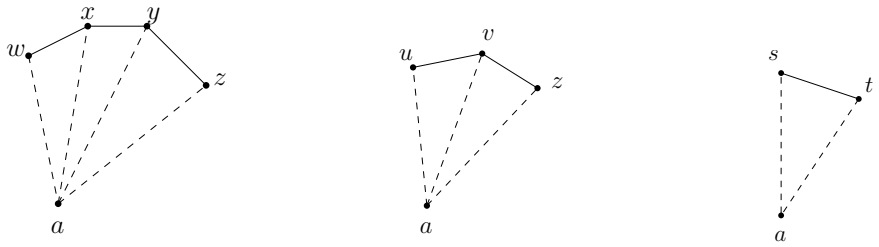


a) b) c)

Figure 4.2: The possibilities to connect two connected components in one NeighborNet iteration: a) no component contains an edge, b) one component contains an edge, c) both components contain an edge. Here the dashed lines stand for candidate edges to be added.

a) b)

Figure 4.3: a) Before the reduction: a connected component of G consisting of three vertices x, y, z. a is another vertex of G outside the connected component. b) After the reduction the connected component in a) is replaced by a connected component consisting of two vertices w, v. The dashed lines represent the distances between the vertices in the component and a.



a) b) c)

Figure 4.4: a) Before the first reduction: a connected component consisting of four vertices w, x, y, z and b) after the first reduction the connected component in a) is replaced by a connected component consisting of three vertices u, v, z. c) A second reduction results in a connected component with two vertices s, t. The dashed lines represent the distances between the vertices in the component and another vertex a outside the component.

4.3 Variants of NeighborNet

In [60] D. Levy and L. Pachter suggest a framework, which includes variants of NeighborNet, that can produce alternative circular split systems to the ones produced by the original NeighborNet. In this section we describe two of their suggested variants of NeighborNet and introduce a new variant that is a greedy heuristic for finding a circular split system of minimal length (corresponding to the minimum evolution criterion).

Levy and Pachter's framework is very similar to the original NeighborNet. The input for the algorithm is a distance matrix D on a set of taxa X with $|X| = n$. It also operates on a graph G consisting of connected components $C_1, \ldots, C_m$, starting with $m = n$ components each consisting of an element in X. As well as in the original NeighborNet, each C_i will consist of one or more vertices that are connected by a path. In the first and second selection steps, two components C_{r*} and C_{s*} are selected and then it is decided which ends of the selected components to join. Instead of the reduction process, the two selected components are then connected through an edge which is inserted between two selected vertices (see Figure 4.5 for an example). Note that for this connection only vertices with degree zero or one can be considered, since they are either isolated vertices or the end vertices of a connected component. We denote the subset of vertices of a component C_i in G with degree less than 2 by $\hat{C}_i$. In the second selection step two vertices $x_{i*} \in \hat{C}_{r*}$ and $x_{j*} \in \hat{C}_{s*}$ are chosen that minimise Equation 4.2. By iteratively connecting the vertices of G, paths are built up, until a path through all vertices of G is created, which gives the circular ordering of the elements of X.

Since there is no reduction performed the vertex set of G is X in every step and the distances between vertices are not updated. However the distances between the components, D_C, are updated. To do this a *weighting*, which determines the way to update distances is introduced. In the next section we explain the updating process and weightings in more detail. For clarity we also provide pseudo code in the Appendix B. As for the original NeighborNet the output of the algorithm

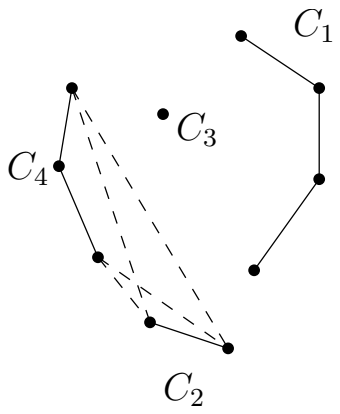


Figure 4.5: Let C_2 and C_4 be the selected components, then the second selection step decides which of the end vertices of each component are connected by an edge. The dashed lines indicate all 4 possible edges.

is a circular ordering and a compatible split system. The weights for the splits in the circular ordering are calculated as described in Section 4.2.2.

4.3.1 Weightings

Definition 4.3.1. *(Weighting) Let C be the set of connected components in G with $|C| = m$. A weighting for C is a function $\mu : X \to \mathbb{R}$ such that $\mu(x_i) \geq 0$ for all $x_i \in X$ and, for each $r = 1, \ldots, m$, $\sum_{x_i \in C_r} \mu(x_i) = 1$ and $\mu(x_i) > 0$ for all $x_i \in \hat{C}_r$.*

The distances in D_C are updated using the following formulae

$$D_C(C_r, C_s) = \sum_{x_i \in C_r, x_j \in C_s} \mu(x_i)\mu(x_j)D(x_i, x_j)$$

and

$$D_C(x, C_r) = \sum_{x_i \in C_r} \mu(x_i)D(x, x_i).$$

Note that in the initial state where $|C| = |X|$, $\mu(x_i) = 1$ for all x_i since every component of C consists of a single vertex. The weighting μ can be chosen in different ways; we present two that are of interest. Note that these two weightings lead to variants of NeighborNet that are distinctively different to the original NeighborNet.

Travelling salesman problem (TSP) weighting

Given a set of vertices (which represents a set of cities), and the distance between them the travelling salesman problem is to find the shortest circular tour (that is a circular ordering) through the vertices. In the NeighborNet algorithm we also calculate a circular tour through the elements of X. Here we describe a weighting that leads to a greedy heuristic for the travelling salesman tour through the vertices $x_i \in X$, because in every step an edge is inserted between vertices such that it leads to the shortest path between the vertices in $\hat{C}_r$ and $\hat{C}_s$. Therefore this weighting is called TSP (Travelling Salesmen Problem) weighting.

Definition 4.3.2. *(TSP weighting) A weighting $\mu : X \to \mathbb{R}$ is a TSP weighting if $\mu(x_i) = 0$ for all $x_i \in C_r \backslash \hat{C}_r$ for each r.*

Definition 4.3.3. *(balanced TSP weighting) A balanced TSP weighting is a TSP weighting where*

$$\mu(x_i) = \begin{cases} \frac{1}{2} & \text{if } x_i \in \hat{C}_r, |\hat{C}_r| = 2 \\ 1 & \text{if } x_i \in \hat{C}_r, |\hat{C}_r| = 1. \end{cases}$$

This weighting gives all vertices in $\hat{C}_r$ (one or two vertices) the same influence in the distance update.

Tree weighting

As for the original NeighborNet algorithm the first selection step in Levy and Pachter's framework yields a tree as described in Section 4.2. Another weighting that is introduced in [60] is called tree weighting, because it influences what kind of tree is produced in the algorithm.

Definition 4.3.4. *(Tree weighting) Let μ be a weighting for C in the k^{th} iteration and consider a new weighting μ' for the $(k+1)^{th}$ iteration of the variant of NeighborNet. Then μ' is a tree weighting if it satisfies*

$$\mu' = \begin{cases} \alpha\mu(x_i) & \text{if } x_i \in C_{r*}, \\ (1-\alpha)\mu(x_i) & \text{if } x_i \in C_{s*}, \end{cases}$$

where C_{r*} and C_{s*} are the two components being merged.

The important observation made in [60] is that if $\alpha = 0.5$ then the tree that is produced through the components selection in the first selection step is the NeighborJoining tree. Therefore, for this weighting the circular split system will be guaranteed to contain the splits of this tree.

4.3.2 Greedy minimum evolution variant of NeighborNet

In the next chapter we will approach the problem of approximating minimum evolution trees by restricting the search space to trees that can be found in a circular split system. An interesting question that arises from this idea is how to construct a circular split system so that it captures relevant information for finding a good approximation of a minimum evolution tree. In [60] hybrid weightings are described. For example, we could use a tree weighting to update the distances between two components which are used in the first selection step and a balanced TSP weighting to update the distances between vertices and components used in the second selection step.

We now describe an approach called GreedyME along these lines that in the first selection step greedily optimises a tree according to the minimum evolution criterion, and in the second selection step it uses the balanced TSP weighting to compute a shortest ordering that is respected by this tree.

The input to GreedyME is a distance matrix D on X. The graph $G = (V, E)$ is initialised by setting $V = X$ and $E = \emptyset$. As before, C is the set of connected components in G and the distance matrix D_C is initialised by putting $D_C = D$. Note that initially C consists of $|X| = n$ components, where each is an isolated vertex labelled with an element of X. Additionally a split system $\Sigma(T)$ that represents a star tree T is initialised. The idea is to choose two components C_{r*}, C_{s*} that form a cherry in T such that the total length of the resulting tree is minimised using weights calculated by the method of ordinary least squares. We implemented this idea using an algorithm by D. Bryant to calculate the edge lengths in the tree [11].

The following steps are repeated until $|C| = 1$. In the first selection step two components C_{r*}, C_{s*} are selected that minimise

$$CalculateEdges(\Sigma(T'), D)$$

where $\Sigma(T') = \Sigma(T) \cup ((C_r \cup C_s)|X - (C_r \cup C_s))$ for all $C_r, C_s \in C$ and $r \neq s$. The split system induced by the tree with minimal length is denoted by $\Sigma(T^*)$. The algorithm *CalculateEdges* uses the method of ordinary least squares for the edge weight calculation as described in Chapter 2 and can be found in Chapter 5, page 139 of [11].

In the second selecting step two vertices x_{i*} and x_{j*} are selected that minimise the same criterion (Equation (4.2)) as for the original NeighborNet and its variants. The edge $x_{i*}x_{j*}$ is added to G and the components set C updated. The split system $\Sigma(T) := \Sigma(T^*)$ and the distances are also updated according to the weighting μ that is used in the second selection step. We used the balanced TSP weighting. The output of GreedyME is a circular ordering π for X and a compatible split system $\Sigma(T^*) \subseteq \Sigma(\pi)$. The split weights are then calculated with the method explained in Section 4.2.2. We also provide the pseudo code for GreedyME in the Appendix B.

4.4 Networks from a Salmonella dataset

To illustrate the variants of NeighborNet that we have described above, we shall present their output for the Salmonella dataset that was used in [14], where the original NeighborNet algorithm was introduced. The dataset consists of 33 MLSTmanB (multilocus sequencing typing) sequences. There it illustrated how NeighborNet could be used for investigations to reconstruct the phylogenetic relationship between the different organisms in this dataset.

In Figure 4.6 we see the network constructed by NeighborNet. It contains several boxes, which indicate conflicts in the data. This is probably caused by recombination [14]. The networks constructed using balanced TSP and Tree weightings

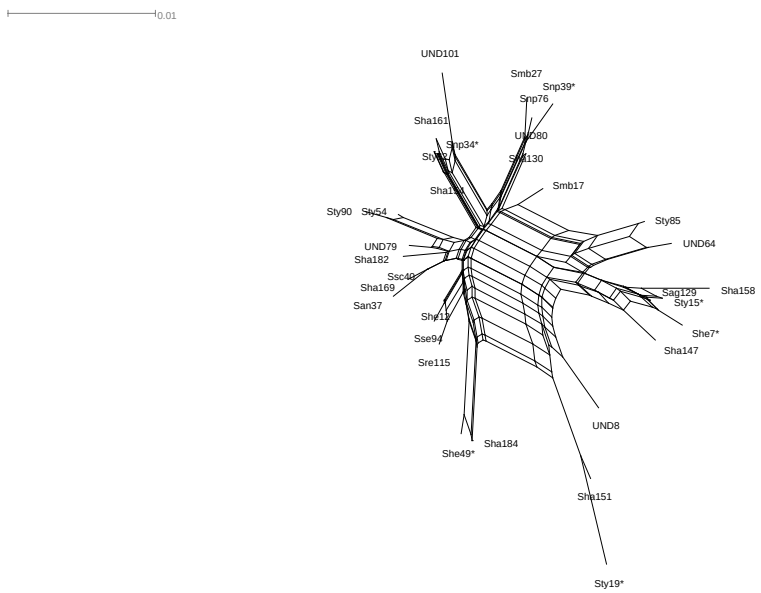


Figure 4.6: The circular split network constructed for 33 manB sequences of the Salmonella dataset using the original NeighborNet algorithm.

are presented in Figure 4.7 and Figure 4.8 respectively. These networks appear to contain fewer boxes and are therefore more treelike. In Figure 4.9 the network constructed by GreedyME is depicted. It shows more similarity to the network constructed by the original NeighborNet, indicating more conflicts by several boxes in the network than the TSP and Tree variants. In particular, this example illustrates that the choice of ordering can be quite critical for the the network that is produced.

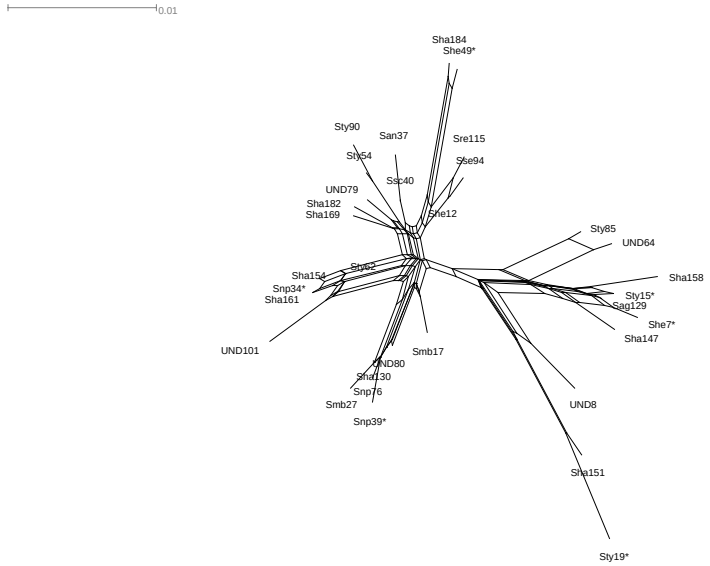


Figure 4.7: The circular split network constructed for 33 manB sequences of the Salmonella dataset using the balanced TSP weighting.

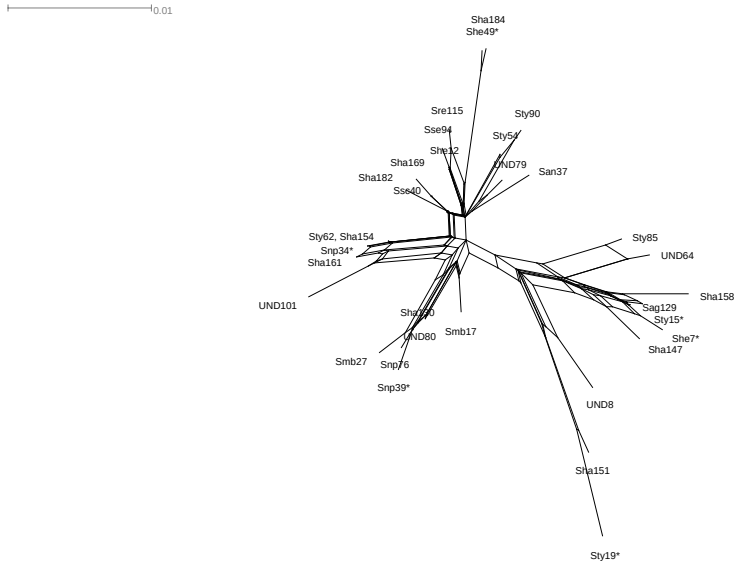


Figure 4.8: The circular split network constructed for 33 manB sequences of the Salmonella dataset using tree weighting.

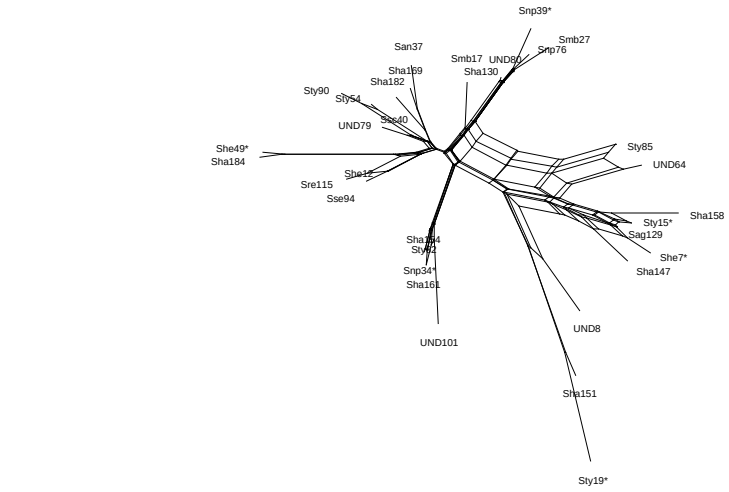


Figure 4.9: The circular split network constructed for 33 manB sequences of the Salmonella dataset using the GreedyME version of NeighborNet and TSP weighting.

4.5 Concluding remarks

We have reviewed the NeighborNet algorithm and three different variants of it (one of which we developed), and showed that they can produce quite different results on a biological data set. All of these methods construct a circular ordering, which gives rise to a full split system from a distance matrix. In the next chapter we explain an algorithm that uses full circular split systems produced by the described methods in order to search for trees.

Chapter 5

NetME: Fishing for minimum evolution trees with NeighborNets

This is an adaption of: S. Bastkowski, A. Spillner, V. Moulton. Fishing for Minimum Evolution Trees with NeighborNets. *Information Processing Letters*, 2013.

5.1 Summary

A common approach taken to construct a phylogenetic tree is to search through the space of all possible phylogenetic trees on the set of species so as to find one that optimizes some score function, such as the minimum evolution criterion. However, this is hampered by the fact that the space of phylogenetic trees is extremely large in general. An alternative approach, which has received somewhat less attention in the literature, is to search for trees within some set of splits of the set of species in question.

In this chapter we consider the problem of searching through a set of splits that is circular. More specifically we present NetME, an $O(n^4)$ algorithm for finding an optimal minimum evolution tree in a circular set of splits on a species set

of size n. In the last chapter we introduced methods such as the NeighborNet algorithm to construct circular split systems. Using simulations and a biological dataset, we compare the performance of our algorithm when applied to the output of these methods with that of FastME [21], a leading method for searching for minimum evolution trees in tree space. We find that, even though a circular set of splits represents just a tiny fraction of the total number of possible splits of a set, the trees obtained from circular sets can compare quite favourably with those obtained with FastME, suggesting that the approach could warrant further investigation.

5.2 Background

As mentioned in Chapter 2, a common approach to construct phylogenetic trees is to search through the space of phylogenetic trees, trying to find a tree (or trees) that optimize some score such as the minimum evolution criterion [76]. However, this is hampered by the fact that the space of phylogenetic trees on X grows exponentially in $n = |X|$ and, indeed, it has been shown that finding an optimal tree is NP-hard for many of the popular optimization criteria (see for example [17, 18]).

Interestingly, there is an alternative approach to searching through tree space, which was studied quite early on in the development of phylogenetics (see for example [20, 70]), and more recently in [10], but that has received somewhat less attention in the literature. In particular, instead of searching through the set of all possible trees on the set X, we look for trees within a collection of splits of X. The rationale behind this approach is that any phylogenetic tree induces a set of splits of X in which every split corresponds to an edge of the tree, and that this set of splits uniquely determines the tree (cf. [79]).

Intriguingly, in [11] a dynamic programming framework is developed to search for trees in a given collection of splits of X. Although still requiring exponential time in general, this approach has the advantage that it can yield polynomial time algorithms when restricted to split systems having size that is polynomial

in $n = |X|$. It is therefore of interest to develop efficient algorithms to search for trees in special classes of split systems, as well as ways to generate split systems which capture salient information.

In this vein, here we develop an algorithm to search for a tree that optimizes the minimum evolution criterion, as described in Chapter 2, by searching in a circular split system. A preliminary version of this algorithm was presented in [5]. In particular, we show that for a circular split system there is an $O(n^4)$ time algorithm for computing an optimal minimum evolution tree, which improves on the run time of $O(n^7)$ for the more general minimum evolution algorithm presented by D. Bryant in [11, Section 5.5]. We also present some simulations which indicate that minimum evolution trees in circular split systems generated by NeighborNet can compare favourably with those obtained by searching through the whole of tree space.

Before continuing, we note that the problem of searching for trees in split systems is related to the problem of finding trees in phylogenetic networks (cf. [87] and [56] for some recent results on finding trees in networks). However, this is a different problem in general since, for example, the minimum evolution tree in a circular split system generated by NeighborNet is not necessarily a subtree of the associated network.

We recall some relevant background material on the minimum evolution problem (cf. also [79]). As described in Chapter 2 given a distance matrix D on X, and a phylogenetic tree $T = (V, E)$ on X, the edge lengths $\omega_D(e)$, $e \in E$, can be computed using the method of ordinary least squares. Recall that here the minimum evolution score $\sigma_D(T)$ is defined to be the total length of T where the edge weights are calculated using the method of ordinary least squares and that the minimum evolution problem is to find a phylogenetic tree T on X with smallest score $\sigma_D(T)$ for a given distance matrix D. Note that, it is possible to compute $\sigma_D(T)$ for given D and T in $O(n^2)$ time [11, p. 137] using the Formulae (2.2) and (2.3) presented in Chapter 2. To the best of our knowledge, the computational complexity of the minimum evolution problem under ordinary

least squares is still open, although the complexity of many related problems [17] suggests that it is NP-complete.

5.3 Method

In this section, we recall Bryant's dynamic programming algorithm for finding minimum evolution trees within a split system and then describe our algorithm in Section 5.3.2. We are interested in the problem of searching for trees in split systems. More specifically, given a distance matrix D and a split system Σ on X, the *restricted minimum evolution problem* requires us to find the minimum $\sigma_{(D,\Sigma)}$ of $\sigma_D(T)$ over all binary phylogenetic trees T on X with $\Sigma(T) \subseteq \Sigma$, where $\Sigma(T)$ is the split system consisting of the splits associated to the edges of T. Any phylogenetic tree T on X with $\Sigma(T) \subseteq \Sigma$ and $\sigma_D(T) = \sigma_{(D,\Sigma)}$ is called a *restricted minimum evolution tree* for D and Σ. Note that the original minimum evolution problem corresponds to the restricted version with $\Sigma = \Sigma(X)$ where $\Sigma(X)$ contains all possible splits for X.

5.3.1 Bryant's algorithm

In [11] Bryant presented an algorithm for solving the restricted minimum evolution problem with run time $O(n^2k + nk^3)$, where $n = |X|$ and $k = |\Sigma|$. In this section, we present a self-contained version of this algorithm, also describing it in such a way that it can be easily adapted to our specific needs in the next section.

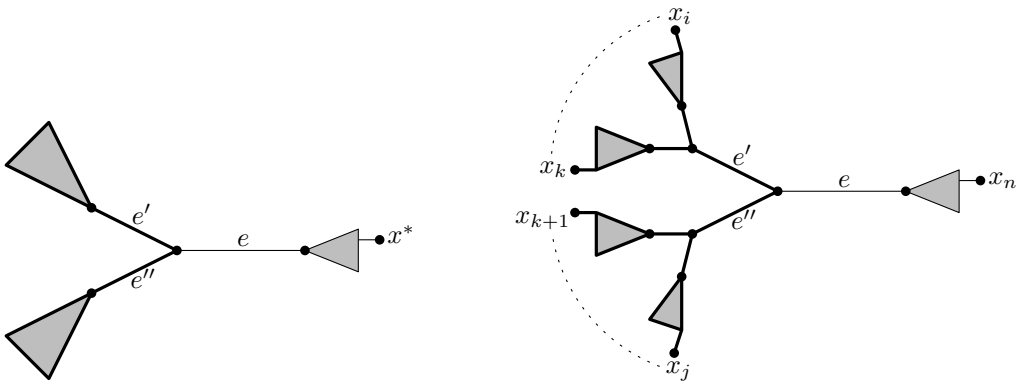
As above, let D denote the given distance matrix on X and $\Sigma \subseteq \Sigma(X)$ denote the given split system. For any split $S = A|B$ of X and any $x \in X$, we denote by $S(x)$ that set, A or B, that contains x and by $\overline{S}(x)$ the other set. To be able to apply and evaluate the Formulae (2.2) and (2.3) in Chapter 2 in constant time in the course of the algorithm, we compute, as a preprocessing step, for every split $S = A|B \in \Sigma$, the cardinalities $|A|$ and $|B|$ as well as the value $P_S = \sum_{x\in A, y\in B} D(x, y)$. This preprocessing can clearly be done in $O(n^2k)$. Moreover, we store the splits in Σ in a suitable data structure $\mathcal{D}$ (for example in a multidimensional dictionary [36]) that allows to check in $O(n)$ time whether a

given split is contained in Σ.

Bryant's algorithm uses a dynamic programming scheme to compute $\sigma_{(D,\Sigma)}$ [19, ch. 15]. Each subproblem arising in this scheme corresponds to a particular triple $(S = S_e, S' = S_{e'}, S'' = S_{e''})$ of splits that correspond to edges e, e' and e'' in the resulting phylogenetic tree as indicated in Figure 5.1a. To describe this more precisely, we introduce some further notation. Fix an arbitrary element $x^* \in X$. We call an ordered triple (S, S', S'') of splits $S, S', S'' \in \Sigma$ *relevant* if $\overline{S'}(x^*) \cup \overline{S''}(x^*) = \overline{S}(x^*)$ and $\overline{S'}(x^*) \cap \overline{S''}(x^*) = \emptyset$ hold, and denote the set of relevant triples of splits in Σ by $\mathrm{rel}(\Sigma)$. In addition, for any $(S, S', S'') \in \mathrm{rel}(\Sigma)$, we let $\mathscr{T}(S, S', S'')$ denote the set of binary phylogenetic trees T on X with $\{S, S', S''\} \subseteq \Sigma(T) \subseteq \Sigma$ and define

$$\sigma_D(S, S', S'') = \min_{T=(V,E)\in\mathscr{T}(S,S',S'')} \left(\sum_{e\in E, \overline{S_e}(x^*)\subsetneq\overline{S}(x^*)} \omega_D(e) \right). \tag{5.1}$$

Note that, as indicated in Figure 5.1a, only the lengths of the edges in the bold part of any tree $T \in \mathscr{T}(S, S', S'')$ are taken into account in the sum in Formula (5.1).



a) b)

Figure 5.1: a) The overall structure of a phylogenetic tree $T \in \mathscr{T}(S, S', S'')$ for some relevant triple $(S = S_e, S' = S_{e'}, S'' = S_{e''})$ of splits. b) In the case where the split system Σ is circular, we can use the special structure of Σ to pin down the relevant triples: $S_e = S_{i,j}$, $S_{e'} = S_{i,k}$ and $S_{e''} = S_{k+1,j}$ for some $1 \leq i \leq k < j < n$.

In view of the structure of the trees in $\mathscr{T}(S, S', S'')$ for any $(S, S', S'') \in \mathrm{rel}(\Sigma)$, it is not hard to derive the following recursive formula for $\sigma_D(S, S', S'')$:

$$\sigma_D(S, S', S'') = \alpha(S, S', S'') + \beta(S, S', S'') \quad \text{with}$$

$$\alpha(S, S', S'') = \begin{cases} \omega_D(S(x^*), \overline{S''}(x^*)) & \text{if } |\overline{S'}(x^*)| = 1 \\ \min\limits_{(S', S_1, S_2) \in \mathrm{rel}(\Sigma)} \big(\sigma_D(S', S_1, S_2) + \\ \omega_D(\overline{S_1}(x^*), \overline{S_2}(x^*), \overline{S''}(x^*), S(x^*))\big) & \text{otherwise,} \end{cases}$$

$$\beta(S, S', S'') = \begin{cases} \omega_D(S(x^*), \overline{S'}(x^*)) & \text{if } |\overline{S''}(x^*)| = 1 \\ \min\limits_{(S'', S_1, S_2) \in \mathrm{rel}(\Sigma)} \big(\sigma_D(S'', S_1, S_2) + \\ \omega_D(\overline{S_1}(x^*), \overline{S_2}(x^*), \overline{S'}(x^*), S(x^*))\big) & \text{otherwise,} \end{cases}$$

where, in case the minimum is taken over the empty set, we assume that the value $+\infty$ is obtained.

Note that the formulae for $\alpha(S, S', S'')$ and $\beta(S, S', S'')$ only involve (i) values $\omega_D(\cdot)$ which can be computed in constant time in view of the preprocessing mentioned above and (ii) values $\sigma_D(S', \cdot, \cdot)$ and $\sigma_D(S'', \cdot, \cdot)$ for which, by definition, $|\overline{S'}(x^*)| < |\overline{S}(x^*)|$ and $|\overline{S''}(x^*)| < |\overline{S}(x^*)|$ hold. Also note that the number of relevant triples of the form (S', S_1, S_2) and (S'', S_1, S_2), respectively, is in $O(k)$ and that, given S' and S_1 (or, similarly, S'' and S_1), with the help of the data structure $\mathcal{D}$ we can check in $O(n)$ time whether there exists a suitable split $S_2 \in \Sigma$ to form a relevant triple (S', S_1, S_2) (or (S'', S_1, S_2)). Hence, within the dynamic programming scheme each value $\sigma_D(S, S', S'')$ can be computed in $O(nk)$ time. Since there are $O(k^2)$ triples in $\mathrm{rel}(\Sigma)$, the overall run time is therefore in $O(nk^3)$.

To conclude the description of the algorithm, note that, for any $(S, S', S'') \in \mathrm{rel}(\Sigma)$ and any $T \in \mathscr{T}(S, S', S'')$, the sum in Formula (5.1) never includes the length of the edge e^* of T that is incident to x^* and corresponds to the split $S^* = S_{e^*} = \{x^*\}|X - \{x^*\}$. Therefore, to obtain the minimum of the total length

of the resulting tree, in the last step we compute

$$\sigma_{(D,\Sigma)} = \min_{(S^*,S_1,S_2)\in \mathrm{rel}(\Sigma)} \big(\sigma_D(S^*,S_1,S_2) + \omega_D(\overline{S_1}(x^*),\overline{S_2}(x^*))\big),$$

which can clearly be done in $O(nk)$ time. So, the overall run time is indeed $O(nk^3)$ and, by tracing back the computation of $\sigma_{(D,\Sigma)}$ through the dynamic programming scheme, we can easily obtain a restricted minimum evolution tree for D and Σ in case $\sigma_{(D,\Sigma)} \neq +\infty$. Otherwise there is no binary phylogenetic tree T on X with $\Sigma(T) \subseteq \Sigma$.

5.3.2 Computing minimum evolution trees in circular split systems

We now focus on the restricted minimum evolution problem for a circular split system. This is a special type of split system that can be generated, for example, from a distance matrix D using the NeighborNet algorithm [14] or one of the variants introduced in Chapter 4. Recall that a split system $\Sigma \subseteq \Sigma(X)$ is circular [3] if there exists an ordering $x_1, x_2, \ldots, x_n$ of the elements in X such that, for every split $A|B \in \Sigma$, there exist $1 \leq i < j \leq n$ with $A = \{x_{i+1}, \ldots, x_j\}$ or $B = \{x_{j+1}, \ldots, x_i\}$. If such an ordering of X exists it can be computed in $O(nk)$, $k = |\Sigma|$, [24] and we say that Σ respects that ordering. Note that the maximum possible number of splits in a circular split system Σ on X is $\binom{n}{2}$ [4]. Thus, Bryant's algorithm runs in $O(n^7)$ on a circular split system. We now show how this can be improved to $O(n^4)$.

Assume that Σ is a circular split system and let $x_1, x_2, \ldots, x_n$ be an ordering of X which Σ respects. We put $x^* = x_n$ and, for any $1 \leq i < j \leq n$, we define the split $S_{i,j} = \{x_{i+1}, \ldots, x_j\}|X - \{x_{i+1}, \ldots, x_j\}$. Note that $\Sigma \subseteq \{S_{i,j} : 1 \leq i < j \leq n\}$ holds, that is, every split in Σ corresponds to a unique pair of indices i and j. As an immediate consequence it follows that the preprocessing outlined above can be done in $O(n^2)$ time (for computing the values P_S see for example [68] for an $O(n^2)$ time algorithm in a more general context).

The key observation, however, is that every relevant triple $(S, S', S'') \in \mathrm{rel}(\Sigma)$ must be such that $S = S_{i,j}$ and either $S' = S_{i,k}$ and $S'' = S_{k,j}$ or $S' = S_{k,j}$ and $S'' = S_{i,k}$ for some $1 \leq i \leq k < j \leq n$ (cf. Figure 5.1b). Hence, for any splits $S', S'' \in \Sigma$, there are only $O(n)$ triples (S', S_1, S_2) and (S'', S_1, S_2) in $\mathrm{rel}(\Sigma)$. Moreover, for the data structure $\mathcal{D}$ we can simply use a two-dimensional array in which we mark the presence/absence of the split $S_{i,j}$ in Σ for each pair $1 \leq i < j \leq n$. Then we can easily check whether a split is contained in Σ in constant time. As a consequence, within the dynamic programming scheme each value $\sigma_D(S, S', S'')$ can be computed much faster, namely in $O(n)$ time. Together with the fact that there are only $O(n^3)$ triples in $\mathrm{rel}(\Sigma)$, this implies that the overall run time is $O(n^4)$.

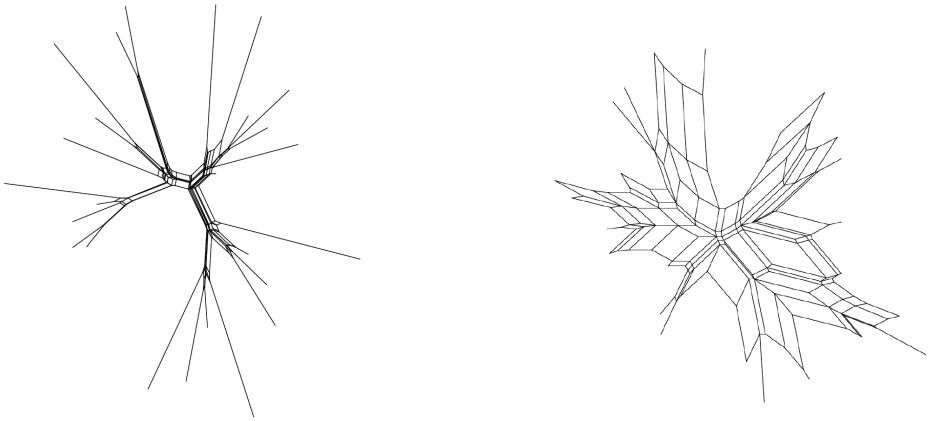
5.4 Results

To give some idea of how well our approach fares in practice, we implemented it and generated some simulated data sets to compare its performance with FastME [21], one of the leading methods to construct an approximation of a minimum evolution tree. Note that FastME performs a local search in tree space using a neighborhood based on certain types of tree edit operations. In FastME we chose the NeighborJoining-tree option as the start topology for the local search together with nearest neighbour interchange tree edit operations, and ordinary least squares for searching the neighborhood of a tree.

5.4.1 Simulations

In our investigation we conducted two different experiments. In the first one we generated 1000 approximately treelike and 1000 random, non-treelike distance matrices D on a set X with $|X| = 25$. To simulate approximately treelike matrices, we followed the procedure described in [45]. In particular, we evolved 25 molecular sequences of length 1000 along a tree (with probability r of recombination set to 0), and computed a distance matrix from the resulting sequence alignment using the so-called Kimura 2 parameter model. To simulate random distance matrices we created symmetric matrices with zero's on the diagonal and

uniformly distributed random values between 0 and 1 in the remaining entries. To provide a visual impression of the structure of approximately treelike versus non-treelike distance matrices we depict typical examples of the phylogenetic networks produced by NeighborNet from these two types of distance matrix (constructed using SplitsTree4 [50]) in Figure 5.2. As can be seen, the visual appearance of the network on the left is quite similar to a tree whereas the one on the right is not.



a) b)

Figure 5.2: Phylogenetic networks produced by NeighborNet representing an approximately treelike (a) and non-treelike (b) distance matrix.

To generate circular split systems we tried various approaches. In particular, for each distance matrix we used NeighborNet (NNet) as well as its traveling salesman (TSP) and GreedyME (Greedy) variants, as described in Chapter 4, to produce an ordering of the elements in X. For comparison purposes we also generated random orderings of X. Once an ordering of X was obtained, we computed a restricted minimum evolution tree for the circular split system consisting of all possible splits that respect the ordering, as described in Section 5.3.2.

In Figure 5.4 we present the results of our experiments. The shown results indicate that NeighborNet and its variants all seem to be capturing relevant splits for both approximately treelike and for non-treelike data, although NeighborNet tends to perform slightly better. This is supported by the average minimum

evolution scores of (i) the trees generated using the NeighborNet ordering, (ii) the trees generated using the random ordering, and (iii) the trees generated by FastME which were (i) 0.92537, (ii) 1.23472 and (iii) 0.92535, respectively, for the approximately treelike distance matrices and, (i) 2.92332, (ii) 4.78671 and (iii) 2.93992 for the non-treelike matrices.

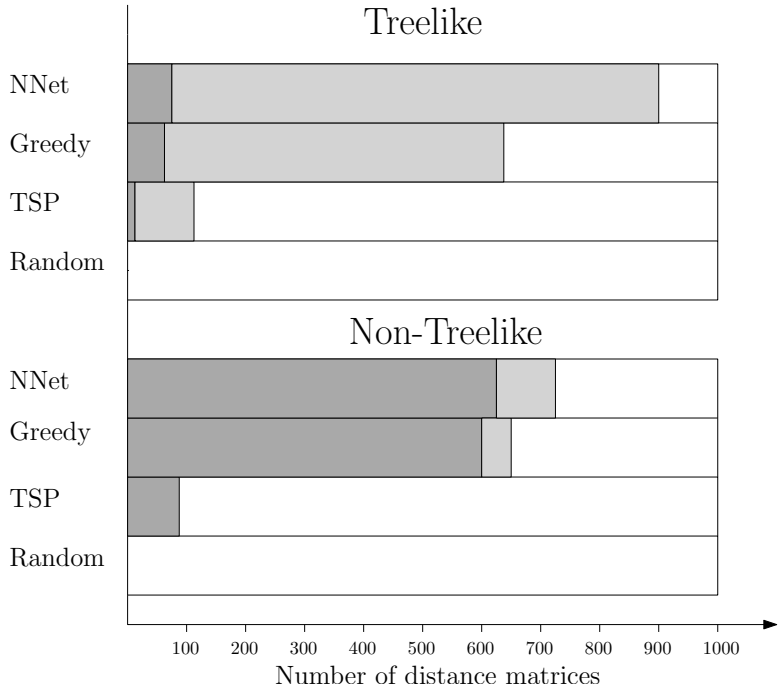


Figure 5.3: The stacked bar chart shows the number of distance matrices (out of 1000) for which the minimum evolution score of the phylogenetic tree produced by solving an instance of the restricted minimum evolution problem was equal (light gray), smaller (dark gray) or larger (white) than the minimum evolution score of the tree produced by FastME.

In our second experiment, we considered sets X with $n = 25$, 50, 100, 200, 400 and 800 taxa and generated 100 approximately treelike as well as 100 random, non-treelike distance matrices on each of them. We simulated the approximately treelike and non-treelike datasets in the same way as for the first experiment. To generate circular split systems we used NeighborNet as well as its traveling salesman variant (TSP) [60] and again we also generated random orderings of X for comparison. As in the first experiment we computed a restricted minimum evolution tree for the full circular split system induced by the ordering. The run

times of NetME and FastMe for the two different experiments are shown in Tables 2 and 3 in the Appendix C.

In Figure 5.4 we present the results of our experiments using NeighborNet to construct the ordering of X. As in the first experiment our results suggest that NeighborNet seems to be capturing relevant splits for both approximately tree-like and non-treelike data. For the tree-like data, FastME appears to perform somewhat better for larger numbers of taxa, but this trend is reversed for the non-treelike data. We also found that random orderings and orderings produced by TSP tended to be a lot worse than those produced by NeighborNet, especially for larger numbers of taxa (not shown in Figure 5.4). The average minimum evolution scores of the trees generated using NeighborNet, FastME, TSP and random orderings are shown in Table 5.1. Interestingly, for approximately treelike data, the average scores of trees generated using NeighborNet and FastME coincide on the first 4 digits. In contrast, for non-treelike data, there is a noticeable difference in the average scores for these two methods. It appears that for non-treelike datasets tree scores generated using NeighborNet are noticeably smaller than for FastME. This indicates that for non-treelike data NeighborNet captures a more informative part of tree space and that the combination of NeighborNet and NetME might be a preferable approach compared to a local search.

5.4.2 Biological data sets

In Chapter 4 we constructed circular split systems from a Salmonella dataset using different versions of NeighborNet. Here we use these circular split systems to search for a minimum evolution tree and compare this tree to the tree produced by FastME. The tree constructed by FastME has a total length of 0.136328 and can be seen in Figure 5.5. In Figure 5.6 the minimum evolution tree found within the full split system given by the original NeighborNet is depicted. The length of this tree is 0.136661 and therefore slightly longer than the length of the tree produced by FastME.

The tree in Figure 5.7 is the minimum evolution tree found within the split system

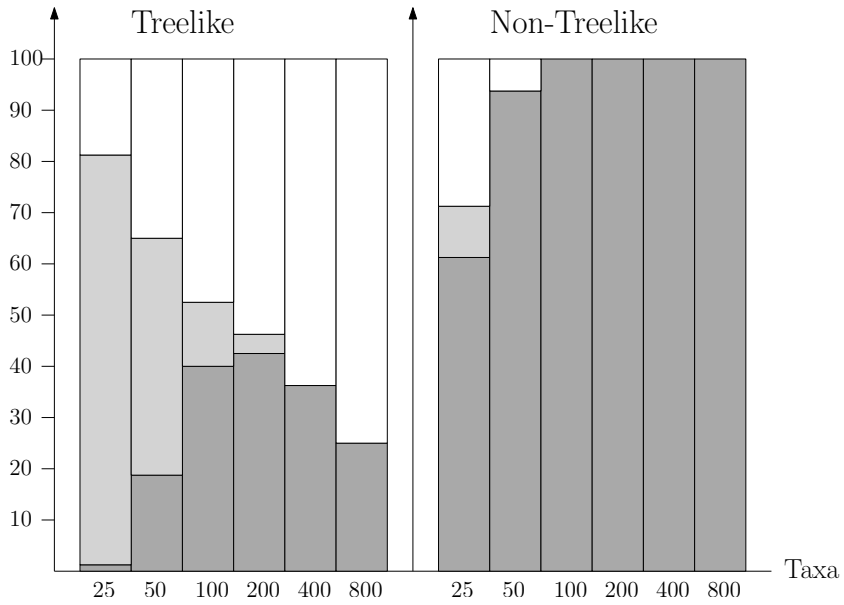


Figure 5.4: The stacked bar chart shows the number of distance matrices (out of 100) for which the minimum evolution score of the phylogenetic tree produced for NeighborNet was equal (light gray), smaller (dark gray) or larger (white) than the minimum evolution score of the tree produced by FastME.

derived from the TSP variant of NeighborNet. It has a length of 0.138431. This supports the impressions given by the results of our experiments on simulated data. The split systems derived from the TSP variant of NeighborNet does not seem to capture relevant splits as well as the original or the GreedyME versions of NeighborNet. In Figure 5.7 the grouping that differs from the NeighborNet derived tree is highlighted. Sha 169 and Sha 182 are grouped together in the TSP minimum evolution tree while in the NeighborNet minimum evolution tree these two are grouped together with Ssc 40 and San 37.

The minimum evolution tree that resulted from using the GreedyME version of NeighborNet is the same as that produced by FastME. Both trees agree with the original NeighborNet minimum evolution tree in the grouping of Sha 169, Sha 182, Ssc 40 and San 37, but differ in a split that is highlighted in Figure 5.5.

Taxa	NNet	FastME	TSP	Random
	Treelike			
25	0.925	0.925	0.931	1.234
50	1.689	1.689	1.701	2.574
100	3.063	3.063	3.084	5.493
200	5.532	5.532	5.574	11.63
400	9.720	9.720	9.786	24.32
800	16.56	16.56	16.66	51.20
	Non-Treelike			
25	2.923	2.940	3.083	4.787
50	4.844	5.177	5.367	9.332
100	8.439	9.108	9.862	18.60
200	15.09	16.33	18.57	36.92
400	27.61	29.79	35.32	73.65
800	51.28	55.07	67.42	147.6

Table 5.1: The average minimum evolution scores of the trees obtained with NeighborNet, FastME, TSP and random orderings for approximately treelike and non-treelike input data.

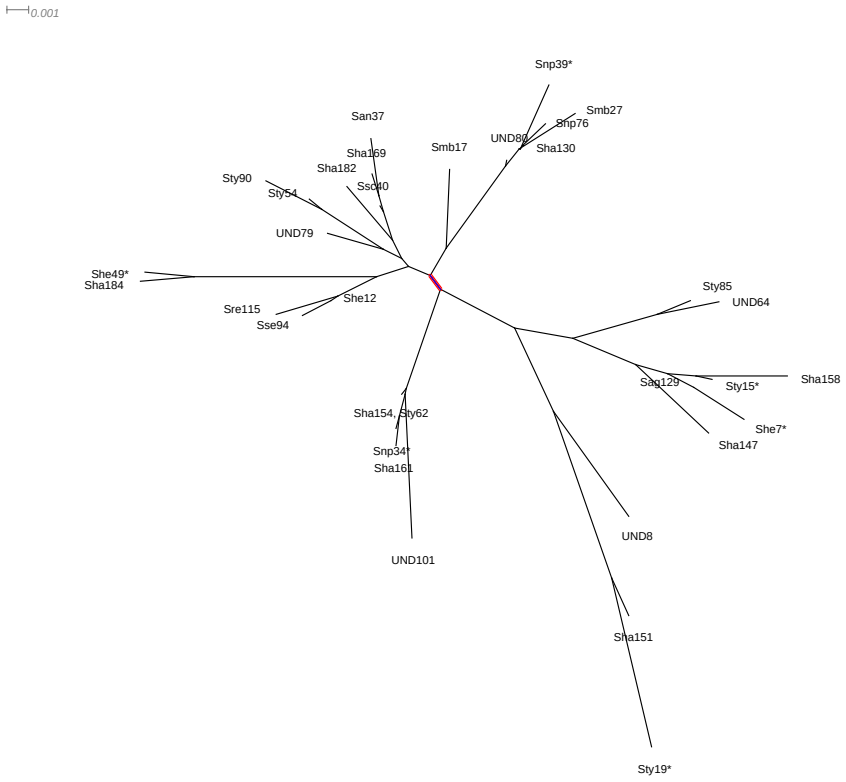


Figure 5.5: The minimum evolution tree found by NetME within a split system constructed by the GreedyME variant of NeighborNet and by FastME.

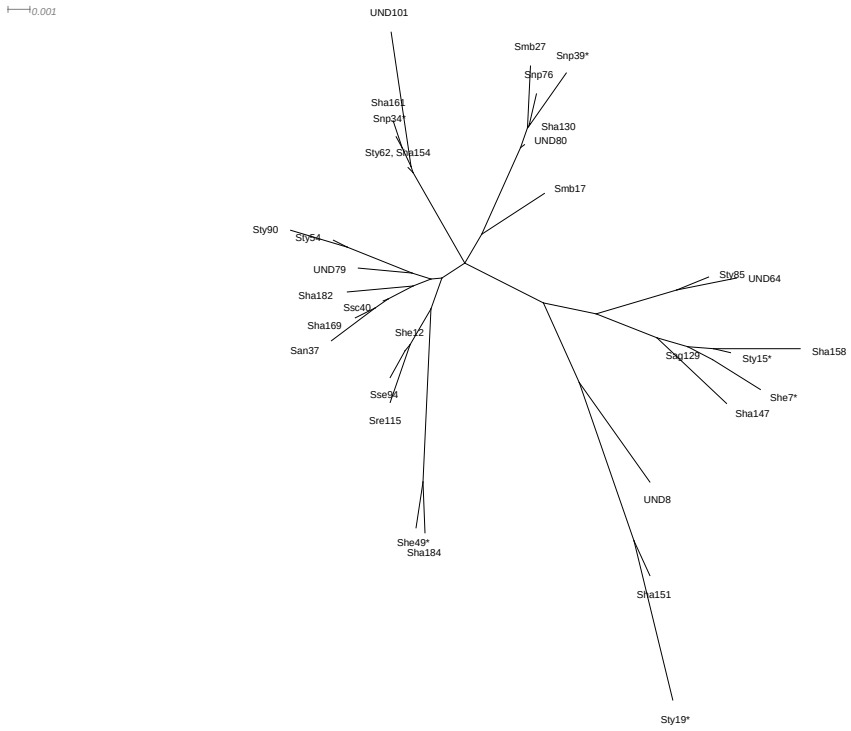


Figure 5.6: The minimum evolution tree found by NetME within a split system constructed by NeighborNet.

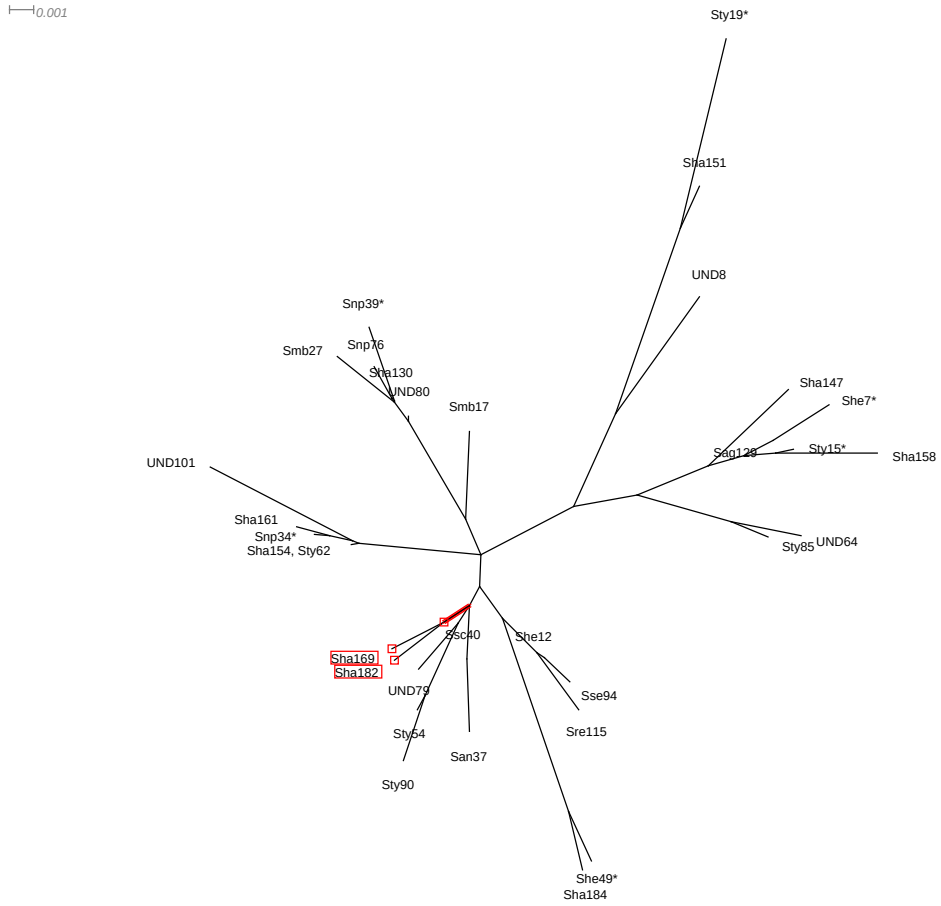


Figure 5.7: The minimum evolution tree found by NetME within a split system constructed by the TSP variant of NeighborNet. The split grouping Sha169 and Sha182 together is highlighted in red.

5.5 Discussion

We conclude with a discussion of some possible future directions. We have presented an efficient algorithm for finding a restricted minimum evolution tree in a circular split system which improves on the run time of a more general algorithm presented in [11]. We have also seen that the restricted minimum evolution trees obtained in split systems generated by NeighborNet compare favorably with the ones produced by FastME. This is of some interest since the split systems generated by NeighborNet only represent a tiny fraction of the total number of all possible splits ($\binom{n}{2}$ vs. $2^{n-1}-1$ on a set of size n). In addition, our computational experiments indicate that NeighborNet produces an ordering that is better at capturing splits relevant to building minimum evolution trees compared with other methods (such as the TSP ordering). It could be of interest to see if there may be other ways to generate even better orderings.

In phylogenetics there are criteria other than minimum evolution that are commonly used to construct phylogenetic trees. For example, the balanced minimum evolution criterion (as described in Chapter 2) is also used for constructing trees from distances (cf. [17]). It would be interesting to know whether or not a restricted balanced minimum evolution tree can be efficiently constructed for a circular split system, and whether a similar type of approach might work for other criteria such as likelihood [18]. In this regards, it might be useful to also consider searching in different types of split systems (such as weakly compatible split systems [4]), and also to consider different ways to generate such split systems.

Finally, note that we have only considered unrooted trees, and so it could be of interest to develop restricted approaches for rooted trees. In this case it would be appropriate to consider special classes of cluster systems rather than split systems (cf. [10]).

5.6 Concluding remarks

In this chapter we have discussed the idea of searching a full circular split system for a minimum evolution tree as an alternative to an exhaustive search of the tree space. We introduced NetME an efficient algorithm to investigate this idea. Simulated and biological data sets were used to construct full circular split systems by NeighborNet and its variants, which are described in Chapter 4. The resulting trees from NetME on these split systems were compared to the tree constructed by FastME. The results for the simulations and the biological dataset suggest that our approach could be of some interest and its constructed trees compare equally or even favourably in many cases. In the next chapter we will investigate some properties of circular split systems further. In particular we are interested in the trees that can be found in a circular split system using tree edit operations, such as the ones FastME uses.

Chapter 6

Trees in circular orderings

The content of this chapter forms part of the material in a paper that is in preparation, in collaboration with V. Moulton, A. Spillner and T. Wu.

6.1 Summary

In the previous chapter we presented an approach to reduce the search space for finding an optimal tree by using a circular split system. We found that Neighbor-Net seems to perform well in capturing relevant information in order to construct a tree satisfying the minimum evolution criterion. Another approach to search for an optimal tree within circular split systems could be to use tree edit operations, such as NNI (nearest neighbour interchange), SPR (subtree prune and regraft) and TBR (tree bisection and reconnection), which we define below.

There are several questions that naturally occur when investigating this idea which we shall focus on in this chapter. For example, given a tree T that is contained in a circular ordering, how many trees are there in the same ordering that can be found by applying one edit operation to T? We call this set of trees the circular neighbourhood of the given tree for the applied tree edit operation. It is known that for NNI operations the size of this neighbourhood just depends on the number of leaves in the given tree. However, we shall show that the size of the circular tree neighbourhood for SPR and TBR operations depends on the

structure of the given tree. We shall also establish formulae that describe the size as well as upper and lower bounds of the size of these neighbourhoods as well as characterise the type of trees that lead to minimum and maximum size neighbourhoods.

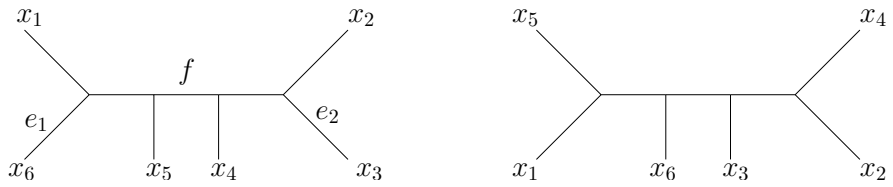
6.2 Background

We begin by defining some key concepts.

6.2.1 Tree operations

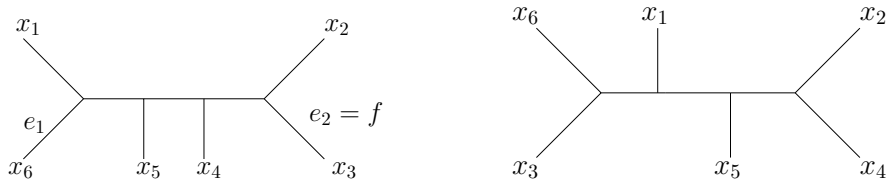
There are three operations on trees that commonly appear in the literature [1]. We begin by recalling tree bisection and reconnection (TBR), the most general of the three. Let $\mathcal{O}_{\text{TBR}}(T)$ be the set of triplets $(e_1, e_2; f)$ of edges in T, in which (i) f is an edge on the path $P(e_1, e_2)$, (ii) e_1 and e_2 share no common ends, and (iii) $|f \cap e_i| \in \{0, 2\}$ for $i = 1, 2$, that is e_1 and e_2 can not be edges adjacent to f. Here e_1 and e_2 are interchangeable, that is, $(e_1, e_2; f)$ and $(e_2, e_1; f)$ are regarded as the same triplet in $\mathcal{O}_{\text{TBR}}(T)$ and are therefore not seen as two distinct triplets. Each triplet $\theta = (e_1, e_2; f)$ in $\mathcal{O}_{\text{TBR}}(T)$ is associated with a binary tree $\theta(T)$ obtained from T by inserting a new edge between e_1 and e_2 (reconnection), and subsequently removing edge f (bisection). We require $\theta(T)$ to be binary and therefore it is necessary to subdivide both e_1 and e_2 before inserting the new edge. Note that if $f = e_1$ then we denote the two edges resulting from subdividing e_1 by e_1' and e_1'', so that $E(P(e_1', e_2)) \subset E(P(e_1'', e_2))$ and remove e_1' to form $\theta(T)$. Since e_1 and e_2 are interchangeable a similar connection applies to the case $f = e_2$. Our definition is equivalent to the one in [1], but here we use a triplet of edges to characterise a TBR operation. Note that the third condition in the definition is introduced to be consistent with [1], that is, each TBR operation corresponds to precisely one triplet in $\mathcal{O}_{\text{TBR}}(T)$.

Subtree prune and regraft (SPR) is a special case of TBR in which there is less freedom for the choice of f in the triplet, that is, $\theta = (e_1, e_2; f) \in \mathcal{O}_{\text{TBR}}(T)$ is an SPR *operation* for T if and only if $f \in \{e_1, e_2\}$. *Nearest neighbour interchange*



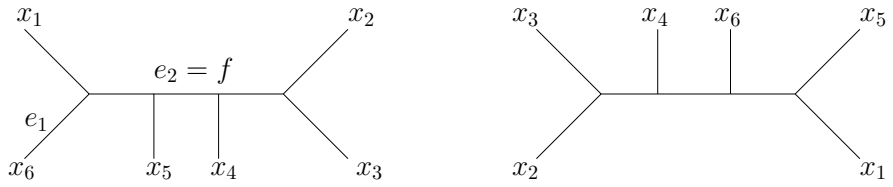
a) b)

Figure 6.1: a) A tree T on 6 taxa b) $\theta(T)$ where $\theta = (e_1, e_2; f)$ encodes a TBR operation.



a) b)

Figure 6.2: a) A tree T on 6 taxa b) $\theta(T)$ where $\theta = (e_1, e_2; f)$ and $f = e_2$ encodes a SPR operation.



a) b)

Figure 6.3: a) A tree T on 6 taxa b) $\theta(T)$ where $\theta = (e_1, e_2; f)$, $f = e_2$ and $|E(P(e_1, e_2))| = 3$ encodes a NNI operation.

*(*NNI*) operations* are SPR (and hence also TBR) operations in which the triplets satisfy an even more restrictive condition; $P(e_1, e_2)$ contains exactly three edges and $f \in \{e_1, e_2\}$. An example of a TBR operation is given in Figure 6.1. Figure 6.2 shows an example of a SPR operation and a NNI operation is depicted in Figure 6.3.

Note that the following clearly holds,

$$\mathcal{O}_{\text{NNI}}(T) \subseteq \mathcal{O}_{\text{SPR}}(T) \subseteq \mathcal{O}_{\text{TBR}}(T)$$

for any tree T. Given two phylogenetic trees T and T' with the same leaf set, the TBR *distance* $d_{\text{TBR}}(T, T')$ between T and T' is defined as the minimum number of TBR operations that is required to be applied one-by-one to change T into T'. The SPR *distance* $d_{\text{SPR}}(T, T')$ and NNI *distance* $d_{\text{NNI}}(T, T')$ are defined in a similar manner. Note that $d_{\text{NNI}}(T, T') \geq d_{\text{SPR}}(T, T') \geq d_{\text{TBR}}(T, T')$ all clearly hold.

6.2.2 Circular orderings

We now recall some notations related to circular orderings and trees, mainly following the ones used in [80]. Let $\pi = (x_1, x_2, \ldots, x_n)$ be a circular ordering as defined in Chapter 2.4.1. Note that $(x_1, x_2, \ldots, x_n)$ and $(x_i, x_{i+1} \ldots, x_n, x_1, \ldots, x_{i-1})$ represent the same ordering. For a non-empty subset Y of X, let $\pi(Y)$ denote the circular ordering of Y obtained by restricting π to Y. Given a circular ordering π the full split system induced by π is denoted by $\Sigma^o(\pi) = \{A_{ij} \,|\, (X - A_{ij}) : 1 \leq i \leq j \leq n-1\}$. Note that π is a circular ordering for T if $\Sigma(T) \subseteq \Sigma^o(\pi)$. Let (T, π) be a pair consisting of a phylogenetic tree T on X and a circular ordering π on X such that π is a circular ordering of T. The set of phylogenetic trees, for which π is a circular ordering, is denoted by $\mathscr{T}_\pi$. By [80, Proposition 3.1], we know the number of trees in $\mathscr{T}_\pi$ for $n \geq 3$ is given by the (Catalan) number

$$\frac{2^{n-2}(2n-5)!!}{(n-1)!} = \frac{1}{n-1}\binom{2n-4}{n-2}.$$

The following result was established by Semple and Steel [80]. It is important since it allows us to use subsets of X of size four to characterise the circular orderings of a phylogenetic tree on X.

Theorem 6.2.1. *[80, Theorem 3.4] Let $\pi = (x_1, \ldots, x_n)$ be a circular ordering of X and let T be a phylogenetic tree on X. Then π is a circular ordering for T, that is, $T \in \mathscr{T}_\pi$, if and only if for all subsets Y of X of size four, $\pi(Y)$ is a circular ordering for $T(Y)$, that is $\Sigma(T(Y)) \subseteq \Sigma^o(\pi(Y))$.*

Let $P_i = P_{i,\pi}$ be the *canonical path* of (T, π), that is the path in T from x_i to x_{i+1} for all $1 \leq i \leq n$. Fix a circular ordering π and a tree $T \in \mathscr{T}_\pi$. To each edge e in

T we associate a pair of indices (i, j) with $1 \leq i < j \leq n$, which will be referred to as the *canonical index pair* for e, such that e is contained in P_i and P_j. Note also that by [80, Theorem 3.2], each edge e of T occurs in exactly two canonical paths of (T, π).

In addition, let $P^+_{e,\pi}$ be the path in T from x_i to x_{j+1}, and $P^-_{e,\pi}$ be the path in T from x_{i+1} to x_j (see Example below). Note that the split induced by e is $A_{i+1,j}|(X - A_{i+1,j})$.

Example:
Let $X = \{x_1, \ldots, x_7\}$ and $\pi = (x_1, \ldots, x_7)$. Consider the tree T depicted in Fig. 6.4. The edge e corresponds to the split $S_e = \{x_2, x_3, x_4, x_5, x_6\}|\{x_7, x_1\}$. The canonical index pair for edge e is $(1, 6)$, because it is contained in the canonical paths P_1 and P_6 as shown in Fig. 6.4. The path $P^+_{e,\pi}$ is the path from x_1 to x_7 and $P^-_{e,\pi}$ is the path from x_2 to x_6.

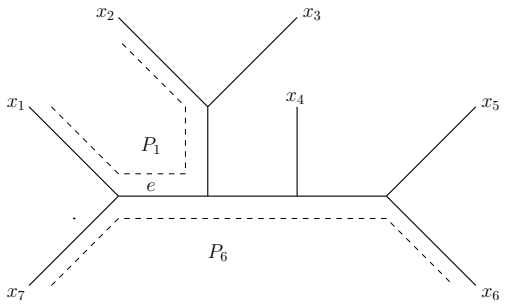
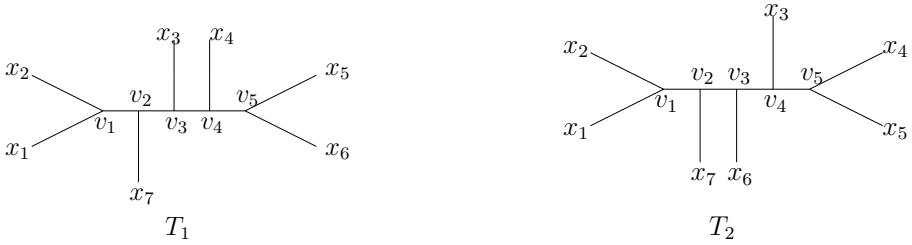


Figure 6.4: Given is the tree T on $X = \{x_1, ..., x_7\}$, the dashed lines show the canonical paths P_1 and P_6, which both contain the edge e (see Example).

6.2.3 Plane binary trees

Given an interior vertex v in a binary tree T, let $N(v) = \{v_1, v_2, v_3\}$ be the triplet of neighbours (adjacent vertices) of v and $\kappa(v)$ the orientation of $N(v)$ induced by T, that is either $\kappa(v) = (v_1, v_2, v_3) = (v_2, v_3, v_1) = (v_3, v_1, v_2)$ or $(v_3, v_2, v_1) = (v_2, v_1, v_3) = (v_1, v_3, v_2)$. A *plane binary tree* is a triplet (V, E, κ) consisting of the tree $T = (V, E)$ together with κ, which embeds the tree in the plane. Two plane trees $\hat{T}_1 = (V_1, E_1, \kappa_1)$ and $\hat{T}_2 = (V_2, E_2, \kappa_2)$ are isomorphic if there is a graph isomorphism $f : (V_1, E_1) \to (V_2, E_2)$ such that $\kappa_1(v) = (v_i, v_j, v_k)$ if and only if $\kappa_2(f(v)) = (f(v_i), f(v_j), f(v_k))$. Table 6.1 shows the number of non-isomorphic plane trees for up to eight leaves.



a) b)

Figure 6.5: An example of two isomorphic plane trees on 7 leaves.

Example In Figure 6.5 two isomorphic plane trees are shown. The cyclic permutation induced clockwise by the interior vertex v_1 in T_1 is $\kappa(v_1) = (x_1, x_2, v_2)$. In T_2, $f(v_1) = v_5$ and the cyclic permutation induced clockwise is $\kappa(v_5) = (x_4, x_5, v_4)$ where $x_4 = f(x_1), x_5 = f(x_2), v_4 = f(v_2)$. In Figure 6.6 the 6 possible plane trees for 7 taxa are shown.

Given a tree $T = (V, E)$, there is a one-to-one correspondence between the set of plane trees $\hat{T} = (T, \kappa)$ and the set of circular orderings π such that $T \in \mathscr{T}_\pi$. This means if two trees are isomorphic as plane trees, they have the same circular ordering.

n	4	5	6	7	8
Number of plane trees	1	1	4	6	19

Table 6.1: Number of possible plane trees for n leaves.

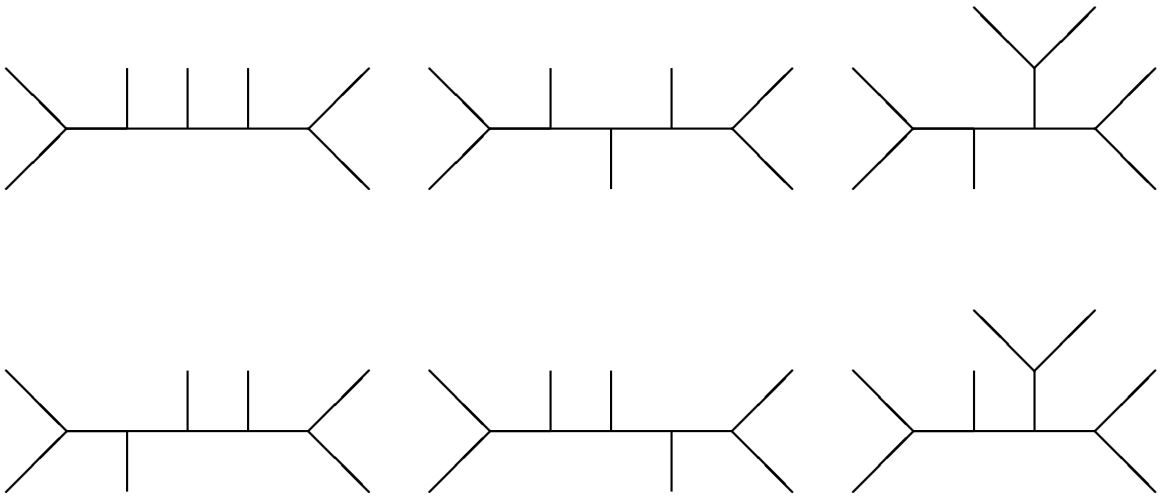


Figure 6.6: The six plane trees on seven leaves and a circular ordering π.

6.3 Circular tree operations

Given a circular ordering $\pi = (x_1, \ldots, x_n)$ and $T \in \mathscr{T}_\pi$, as mentioned before, in some applications it is desirable to consider the tree rearrangement operations θ that preserve the circular ordering π, that is, operations θ such that $\theta(T) \in \mathscr{T}_\pi$. Let

$$\mathcal{O}^\pi_{\text{TBR}}(T) := \{\theta \in \mathcal{O}_{\text{TBR}}(T) \,:\, \theta(T) \in \mathscr{T}_\pi\}$$

be the set of TBR operations preserving π. The members of $\mathcal{O}^\pi_{\text{TBR}}(T)$ will be referred to as circular TBR operations (with respect to π). Similarly, we can define the set of circular SPR operations $\mathcal{O}^\pi_{\text{SPR}}(T)$ and the set of circular NNI operations $\mathcal{O}^\pi_{\text{NNI}}(T)$ on a tree T.

The next theorem is a key technical result, which describes a structural characterisation of the set of TBR operations preserving a given circular ordering.

Theorem 6.3.1. *Given a circular ordering π, a tree $T \in \mathscr{T}_\pi$ and an operation $\theta = (e_1, e_2; f) \in \mathcal{O}_{\text{TBR}}(T)$, the following assertions hold:*
(i) *If $f \notin \{e_1, e_2\}$, then $\theta \in \mathcal{O}^\pi_{\text{TBR}}(T)$ if and only if either $e_1 \in E(P^+_{f,\pi})$ and $e_2 \in E(P^-_{f,\pi})$ or $e_1 \in E(P^-_{f,\pi})$ and $e_2 \in E(P^+_{f,\pi})$.*
(ii) *If $f \in \{e_1, e_2\}$, then $\theta \in \mathcal{O}^\pi_{\text{TBR}}(T)$ if and only if either $e_1 = f$ and $e_2 \in$*

$E(P^+_{f,\pi}) \cup E(P^-_{f,\pi})$, *or* $e_2 = f$ *and* $e_1 \in E(P^+_{f,\pi}) \cup E(P^-_{f,\pi})$.

Proof. (i) "$\Leftarrow$" Given a circular ordering π we fix $T \in \mathscr{T}_\pi$. Let $\theta = (e_1, e_2; f) \in \mathcal{O}_{\text{TBR}}(T)$ where $f \notin \{e_1, e_2\}$ and since $(e_1, e_2; f)$ and $(e_2, e_1; f)$ are interchangeable, we can assume $e_1 \in E(P^-_{f,\pi})$ and $e_2 \in E(P^+_{f,\pi})$. The edge f induces the split $S_f = A_{i+1,j}|(X - A_{i+1,j})$ and the canonical pair index of f is (i, j) so $P^-_{f,\pi}$ is the path from x_j to x_{i+1} and $P^+_{f,\pi}$ is the path from x_i to x_{j+1} (see Figure 6.7a).

We want to show that applying θ to T yields a tree $\theta(T) = T'$ which preserves the circular ordering π, that is $T' \in \mathscr{T}_\pi$. This is equivalent to showing that $\Sigma(T') \subseteq \Sigma^o(\pi)$. Since $\Sigma(T) \subseteq \Sigma^o(\pi)$ it suffices to show that

$$(\Sigma(T') - \Sigma(T)) \subseteq \Sigma^o(\pi).$$

We have to determine the splits contained in $(\Sigma(T') - \Sigma(T))$ and show that these splits are also in $\Sigma^o(\pi)$. There are five cases to consider: the split induced by the newly inserted edge f' (case 1); what happens to the edges adjacent to the edge f that is removed (case 2); and the edges that are formed by splitting e_1 and e_2 when inserting f' (case 3); splits that are induced by the other edges along the paths $P(e_1, f)$, $P(e_2, f)$ that are not covered by the second and third case (case 4); and all other edges in T (case 5).

Case 1: In T, f induces split S_f. The tree T' results from T by deleting f and inserting a new edge f' between e_1 and e_2. The edge f' induces exactly the same split as f, so $S_{f'} = S_f \in \Sigma^o(\pi)$.

Let T_A be the subtree in T that is adjacent to f such that $x_j \in V(T_A)$ and T_B is the other subtree of T that is adjacent to f. For the following four cases we consider edges in T_A, but we can similarly establish the results for the edges in T_B.
Case 2: Let $f_1, f_2 \in E(P^-_{f,\pi})$ be adjacent to f in T_A. In particular let $f_2 \in P(e_1, f)$ (see Figure 6.7a). By removing edge f the two edges f_1 and f_2 are collapsed into one edge f'_1. Therefore the split $S_{f_2} \notin \Sigma(T')$ and $S_{f'_1} = S_{f_1}$.

In the remaining cases we assume the edge e_1 induces the split $S_{e_1} = A_{k,j}|(X - A_{k,j})$ where $i + 1 < k \leq j$.
Case 3: By inserting f', e_1 is split into two edges e'_1 and e''_1. Let e'_1 be on the path $P(e''_1, f'_1)$ and $e''_1 \notin E(P(e'_1, f'_1))$ (see Figure 6.7b) then $S_{e''_1} = S_{e_1}$ and $S_{e'_1} = A_{k,j} \cup (X - A_{i+1,j})|(A_{i+1,j} - A_{k,j}) = \{x_k, \ldots, x_j, x_{j+1}, \ldots, x_i\}|\{x_{i+1}, \ldots, x_{k-1}\} \in \Sigma^o(\pi)$.

Case 4: For each split $S_e = A_{l,j}|(X - A_{l,j})$ induced by an edge e along the path $P(e_1, f_1)$ in T and $e \notin \{e_1, f_1\}$ there is an edge e' on the path $P(e'_1, f'_1)$, $e' \notin \{f'_1, e'_1\}$ in T' such that

$$S_{e'} = (A_{l,j} \cup A_{j+1,i})|A_{i+1,l-1} = \{x_l, \ldots, x_j, x_{j+1}, \ldots, x_i\}|\{x_{i+1} \ldots, x_{l-1}\} \in \Sigma^o(\pi)$$

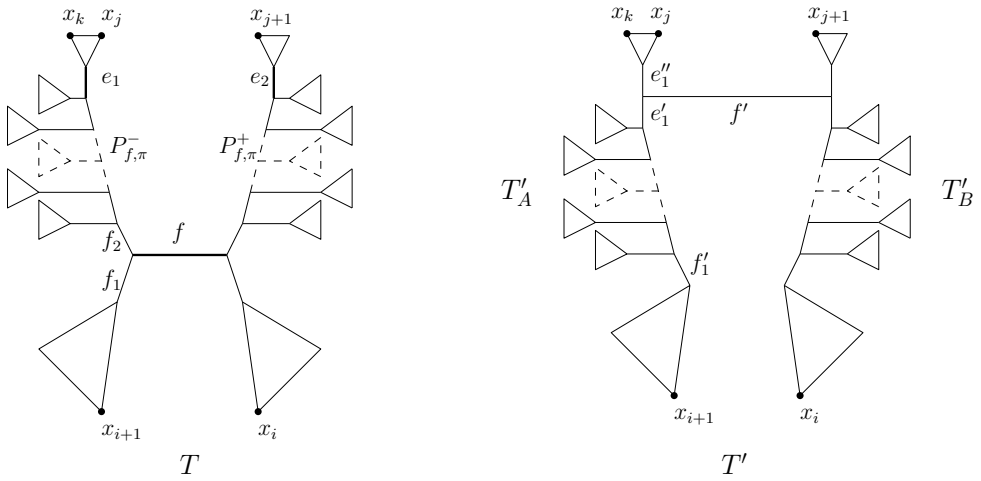
. These last two results can be similarly established for $S_{e_1} = A_{i+1,k}|(X - A_{i+1,k})$.

Case 5: All other splits induced by edges in subtrees adjacent to $P(e'_1, f'_1)$ in T' are exactly the same as the splits induced by edges in the subtrees adjacent to $P^-_{f,\pi}$ (see Figure 6.7a and b).

"$\Rightarrow$" We shall establish this direction by contradiction. To obtain a contradiction, we assume $\theta = (e_1, e_2; f) \in \mathcal{O}^{\pi}_{\text{TBR}}(T)$, e_1 and e_2 are interchangeable and $e_1 \notin E(P^+_{f,\pi}) \cup E(P^-_{f,\pi})$. Let (i, j) be the canonical index pair associated with f; then we have $S_f = A_{i+1,j}|(X - A_{i+1,j})$. Since S_f and S_{e_1} are compatible and $S_f \neq S_{e_1}$, we know there is a unique side of S_{e_1}, denoted by A_1, that is a proper subset of a side of S_f. We further assume $A_1 \subset A_{i+1,j}$; the other case $A_1 \subset (X - A_{i+1,j})$ can be established similarly.

Now, fix an element $y_1 \in A_1$. Since $e_1 \notin P^-_{f,\pi} = P(x_{i+1}, x_j)$, we know $\{x_{i+1}, x_j\} \cap A_1 = \emptyset$. On the other hand, as f is contained in the path $P(e_1, e_2)$ with $f \notin \{e_1, e_2\}$, we know that S_{e_2} contains a unique side, denoted by A_2, such that it is a proper subset of $(X - A_{i+1,j})$. An example of this case is illustrated in Figure 6.8a. Consider an element $y_2 \in A_2$. Then clearly $Y := \{y_1, y_2, x_j, x_{i+1}\}$ consists of four distinct elements. Between the two edges in $\theta(T)$ obtained from dividing e_1, we denote one edge by e'_1 and the other one by e''_1 such that $P(x_j, e'_1) \subset$

$P(x_j, e_1'')$. Then the split restricted on Y of $\Sigma(\theta(T(Y)))$ induced by e_1' is $S_{e_1'}(Y) = \{y_1, y_2\}|\{x_j, x_{i+1}\}$ (see Figure 6.8b). Since the circular ordering on Y was $\pi(Y) = (y_1, x_j, y_2, x_{i+1})$, we know $S_{e_1'}(Y) \notin \Sigma^o(\pi(Y))$. Together with Theorem 6.2.1, this implies $\theta(T) \notin \mathscr{T}_\pi$, a contradiction as required.
(ii) The proof of (ii) is similar to that of (i). □

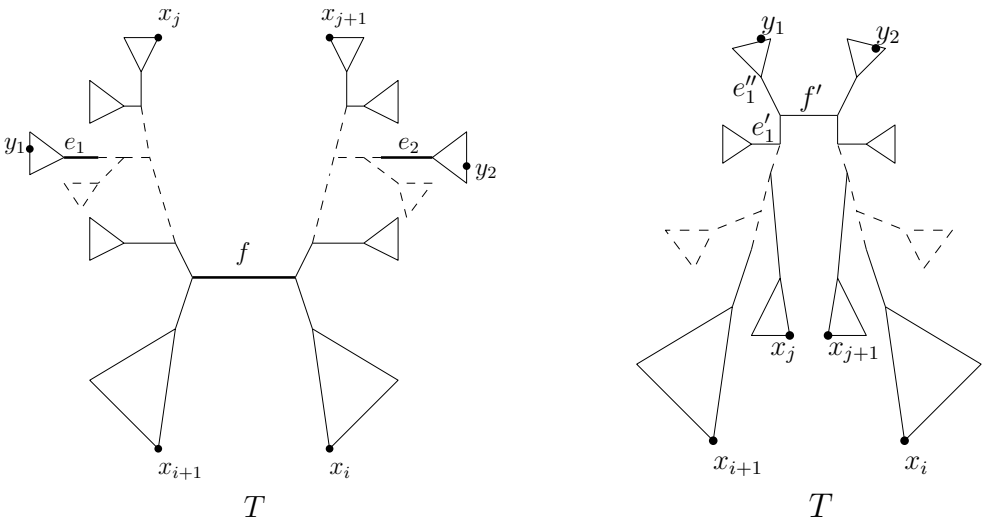


a) b)

Figure 6.7: a) Given the tree T, the edge f induces the split $S_f = A_{i+1,j}|(X - A_{i+1,j})$ and $P^-_{f,\pi}$ is the path from x_j to x_{i+1} and $P^+_{f,\pi}$ the path from x_{j+1} to x_i. We consider the TBR operation $\theta = (e_1, e_2; f)$ where $e_1 \in E(P^-_{f,\pi})$ and $e_2 \in E(P^+_{f,\pi})$. b) In $\theta(T) = T'$ the edge f' induces the split $S_{f'} = S_f$, f_1' induces the split $S_{f_1'} = S_{f_1}$, and the edge e_1 is split into two edges e_1' and e_1''. The splits induced by edges in the subtrees adjacent to $P(f_1', e_1'')$ in T' are the same splits as in T.

Corollary 6.3.2. *Given a circular ordering π, a tree $T \in \mathscr{T}_\pi$ and an operation $\theta = (e_1, e_2; f) \in \mathcal{O}_{\text{TBR}}(T)$ then $\theta \in \mathcal{O}^\pi_{\text{SPR}}(T)$ if and only if either $e_1 = f$ and $e_2 \in E(P^+_{f,\pi}) \cup E(P^-_{f,\pi})$, or $e_2 = f$ and $e_1 \in E(P^+_{f,\pi}) \cup E(P^-_{f,\pi})$.*

Proof. This is a direct consequence of Theorem 6.3.1(ii). □



a) b)

Figure 6.8: Given the tree T, the edge f induces the split $S_f = A_{i+1,j}|(X - A_{i+1,j})$ and $P^-_{f,\pi}$ is the path from x_j to x_{i+1} and $P^+_{f,\pi}$ the path from x_{j+1} to x_i. We consider the TBR operation $\theta = (e_1, e_2; f)$ where $e_1, e_2 \notin E(P^+_{f,\pi}) \cup E(P^-_{f,\pi})$. The edge e_1 induces the split $S_{e_1} = A_1|(X - A_1)$ where $y_1 \in A_1$ and the edge e_2 induces the split $S_{e_2} = A_2|(X - A_2)$ where $y_2 \in A_2$. The ordering of the four element set $Y = \{x_j, x_{i+1}, y_1, y_2\}$ is $\pi(Y) = (y_1, x_j, y_2, x_{i+1})$. b) In $\theta(T) = T'$ the ordering of the elements of Y is $\pi(Y) = (y_1, y_2, x_j, x_{i+1})$.

6.4 Tree neighbourhoods

The TBR neighbourhood of T is the set

$$N_{\text{TBR}}(T) = \{\theta(T) : \theta \in \mathcal{O}_{\text{TBR}}(T)\}$$

consisting of all trees that are precisely one TBR operation from T. The NNI neighbourhood $N_{\text{NNI}}(T)$ and the SPR neighbourhood $N_{\text{SPR}}(T)$ are defined similarly. Clearly, we have

$$N_{\text{NNI}}(T) \subseteq N_{\text{SPR}}(T) \subseteq N_{\text{TBR}}(T).$$

For a tree $T \in \mathscr{T}_n$, where $n \geq 4$, Robinson [74] showed $|N_{\text{NNI}}(T)| = 2n - 6$, while Allen and Steel [1] proved that $|N_{\text{SPR}}(T)| = 2(n-3)(2n-7)$. On the other hand,

$|N_{\text{TBR}}(T)|$ depends on the topology of T, and in a recent study [48], it was shown

$$|N_{\text{TBR}}(T)| = -(4n-2)(n-3) + 4 \sum_{A|B \in \Sigma(T)} |A| \times |B|,$$

where the sum is taken over all non-trivial splits $A|B$ of T. One important step used to establish this result on $N_{\text{TBR}}(T)$ is the following lemma, which generalises an observation by Allen and Steel [1, Proof of Theorem 2.1] on SPR operations to TBR operations, and will be useful for us.

Lemma 6.4.1. *[48] Let $\theta, \theta' \in \mathcal{O}_{\text{TBR}}(T)$ be two distinct* TBR *operations. If $\theta(T) = \theta'(T)$, then $\theta, \theta' \in \mathcal{O}_{\text{NNI}}(T)$.*

Let $T \in \mathscr{T}_\pi$. The circular TBR neighbourhood of T with respect to π is the set

$$N^\pi_{\text{TBR}}(T) := N_{\text{TBR}}(T) \cap \mathscr{T}_\pi$$

consisting of all trees in $\mathscr{T}_\pi$ that are precisely one TBR rearrangement operation from T. The circular NNI neighbourhood $N^\pi_{\text{NNI}}(T)$ and the circular SPR neighbourhood $N^\pi_{\text{SPR}}(T)$ are defined accordingly.

In this section, we shall investigate the size of the SPR and TBR circular neighbourhoods. One key tool we will use is the following lemma, which is a consequence of Lemma 6.4.1.

Lemma 6.4.2. *Let $\theta, \theta' \in \mathcal{O}^\pi_{\text{TBR}}(T)$ be distinct* TBR *operations. If $\theta(T) = \theta'(T)$, then $\theta, \theta' \in \mathcal{O}^\pi_{\text{NNI}}(T)$.*

By the above lemma, we have

Lemma 6.4.3. *For a tree $T \in \mathscr{T}_\pi$ with π a circular ordering on $n \geq 4$ elements, we have*

$$|N^\pi_{\text{SPR}}(T)| = |\mathcal{O}^\pi_{\text{SPR}}(T)| - 3|N^\pi_{\text{NNI}}(T)| = |\mathcal{O}^\pi_{\text{SPR}}(T)| - 3(n-3),$$

and

$$|N^\pi_{\text{TBR}}(T)| = |\mathcal{O}^\pi_{\text{TBR}}(T)| - 3|N^\pi_{\text{NNI}}(T)| = |\mathcal{O}^\pi_{\text{TBR}}(T)| - 3(n-3).$$

Proof. Note that for each $T' \in N^{\pi}_{\text{NNI}}(T)$, there are exactly four distinct operations $\theta \in \mathcal{O}^{\pi}_{\text{NNI}}(T)$ with $\theta(T) = T'$. This is illustrated in Figure 6.9. Together with Lemma 6.4.2 and since $|N^{\pi}_{\text{NNI}}(T)| = n - 3$, this implies the lemma. Recall that $|N_{\text{NNI}}(T)| = 2(n-3)$. Let us consider the tree T in Figure 6.9b), then we see that in order to produce triplets $\theta \in \mathcal{O}^{\pi}_{\text{NNI}}(T)$ we do not allow reconnection such as between e_1 adjacent to A and e_2 adjacent to C, therefore $|N^{\pi}_{\text{NNI}}(T)|$ is half the size of $|N_{\text{NNI}}(T)|$. □

a) b)

Figure 6.9: a) The tree $T' \in N^{\pi}_{\text{NNI}}$ and in b) all four operations $\theta \in \mathcal{O}^{\pi}_{\text{NNI}}(T)$ that result in T'. Recall that the operation $\theta = (e_1, e_2; f)$ where $f = e_1$ yields the same tree as θ with $f = e_2$.

Lemma 6.4.3 forms the basis of calculating the size of circular SPR and circular TBR neighbourhoods because both the number of distinct circular SPR operations and the number of distinct circular TBR operations for any given tree and circular ordering can be found relatively easily.

For this purpose we introduce

$$\alpha(T, \pi) = \sum_{e \in E(T)} (|E(P^{+}_{e,\pi})| + |E(P^{-}_{e,\pi})|)$$

and

$$\beta(T, \pi) = \sum_{e \in E(T)} (|E(P^{+}_{e,\pi})| - 2) \times (|E(P^{-}_{e,\pi})| - 2).$$

We will see that the formula for $\alpha(T, \pi)$ can be simplified using the canonical paths of (T, π). Note that plane binary trees that are isomorphic to each other also have the same α and β regardless of the labelling of the leaves. We proceed

with investigating the size of circular SPR neighbourhoods first.

Theorem 6.4.4. *For a tree $T \in \mathscr{T}_\pi$ with π a circular ordering on $n \geq 5$ elements, we have*

$$|N^\pi_{\text{SPR}}(T)| = \alpha(T, \pi) - 9n + 21.$$

Proof. By Lemma 6.4.3, it suffices to show

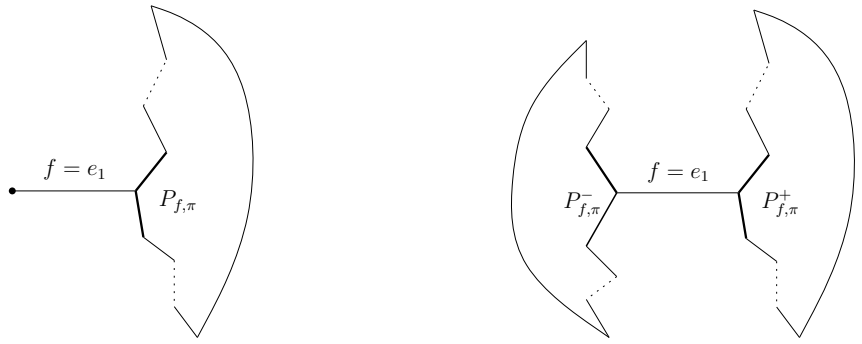
$$|\mathcal{O}^\pi_{\text{SPR}}(T)| = \alpha(T, \pi) - 6n + 12. \tag{6.1}$$

Since for SPR $f \in \{e_1, e_2\}$ we may assume that $e_1 = f$ holds for all operations $(e_1, e_2; f)$ in $\mathcal{O}^\pi_{\text{SPR}}(T)$ because by definition $(e_1, e_2; f)$ and $(e_2, e_1; f)$ are regarded as the same operation. The operations $(e_1, e_2; f)$ in $\mathcal{O}^\pi_{\text{SPR}}(T)$ can be divided into two types, according to whether $f = e_1$ is a pendant edge. We start by counting the number of operations whose last coordinate is a pendant edge. Note that for each pendant edge f, exactly one of the two paths $P^+_{f,\pi}$ and $P^-_{f,\pi}$ contains no edges and the other one at least one non-trivial edge. We denote the path containing at least one non-trivial edge by $P_{f,\pi}$.

By Theorem 6.3.1 we know that $(e_1, e_2; f) \in \mathcal{O}^\pi_{\text{SPR}}(T)$ if and only if e_2 is contained in $P_{f,\pi}$ and e_2 shares no common ends with $e_1 = f$. Since there are exactly $|E(P_{f,\pi})| - 2$ edges contained in $P_{f,\pi}$ that do not incident with f, we know the number of operations in $\mathcal{O}^\pi_{\text{SPR}}(T)$ whose last coordinate is f is $|E(P_{f,\pi})| - 2$, which equals to $|E(P^+_{e,\pi})| + |E(P^-_{e,\pi})| - 2$ (See Figure 6.10a).

For the case in which the last coordinate f is an interior edge, the second coordinate e_2 could be any edge contained in $P^+_{f,\pi}$ or $P^-_{f,\pi}$ that is not incident with f. Clearly, the number of the edges satisfying this condition is $|E(P^+_{f,\pi})| + |E(P^-_{f,\pi})| - 4$. Therefore, the number of operations in $\mathcal{O}^\pi_{\text{SPR}}(T)$ whose last coordinate is f is $|E(P^+_{f,\pi})| + |E(P^-_{f,\pi})| - 4$ (see Figure 6.10b). Summing this up for all $n-3$ interior edges and n pendant edges yields Equation (6.1). □

We now consider TBR operations.



a) b)

Figure 6.10: Given a tree T and $\theta(e_1, e_2; f)$ where $f = e_1$ a) The situation if f is a pendant edge and b) if f is an interior edge. The edges incident to f are highlighted bold.

Theorem 6.4.5. *For a tree $T \in \mathscr{T}_\pi$ with a circular ordering π on $n \geq 4$ elements, we have*

$$|N^\pi_{\text{TBR}}(T)| = \alpha(T, \pi) + \beta(T, \pi) - 9n + 21.$$

Proof. By Lemma 6.4.3, it suffices to show

$$|\mathcal{O}^\pi_{\text{TBR}}(T)| = \alpha(T, \pi) + \beta(T, \pi) - 6n + 12.$$

In view of Equation (6.1) established in the proof of Theorem 6.4.4, it remains to prove that the number of operations in $\mathcal{O}^\pi_{\text{TBR}}(T) - \mathcal{O}^\pi_{\text{SPR}}(T)$ is $\beta(T, \pi)$. To this end, note that for each operation $\theta = (e_1, e_2; f) \in \mathcal{O}^\pi_{\text{TBR}}(T) - \mathcal{O}^\pi_{\text{SPR}}(T)$, f must be an interior edge of $P(e_1, e_2)$ as otherwise $f \in E(P(e_1, e_2))$ implies $f \in \{e_1, e_2\}$, a contradiction to $\theta \notin \mathcal{O}^\pi_{\text{SPR}}(T)$. For each interior edge f in T, by Theorem 6.3.1(i) and switching e_1 and e_2 if necessary, we know $(e_1, e_2; f) \in \mathcal{O}^\pi_{\text{TBR}}(T)$ if and only if $e_1 \in P^+_{f,\pi}$, $e_2 \in P^-_{f,\pi}$, and e_1 and e_2 are not adjacent to f. Clearly, the number of the pairs of edges satisfying this condition is $(|E(P^+_{f,\pi})| - 2) \times (|E(P^-_{f,\pi})| - 2)$. Summing this up for all choices of f, which is $n - 3$, we can conclude $|\mathcal{O}^\pi_{\text{TBR}}(T) - \mathcal{O}^\pi_{\text{SPR}}(T)| = \sum_{f \in E(T)} (|E(P^+_{f,\pi})| - 2) \times (|E(P^-_{f,\pi})| - 2) = \beta(T, \pi)$, as required. □

In the remainder of this section we shall establish upper and lower bounds for

$|N^{\pi}_{\mathrm{SPR}}|$ and $|N^{\pi}_{\mathrm{TBR}}|$ and present a characterisation of the trees (T, π) with the maximum size of N^{π}_{SPR} and N^{π}_{TBR} and the trees (T, π) with the minimum size N^{π}_{SPR} and N^{π}_{TBR}. To this end, we begin with an alternative way of computing $\alpha(T, \pi)$.

Lemma 6.4.6. *Given a circular ordering $\pi = (x_1, \cdots, x_n)$ and a tree $T \in \mathscr{T}_\pi$, we have*

$$\alpha(T, \pi) = \sum_{i=1}^{n} |E(P_i)|(|E(P_i)| - 1),$$

where $P_i = P_{i,\pi}$ $(1 \leq i \leq n)$ is the canonical path in T from x_i to x_{i+1}.

Proof. Let $\{e_1, .., e_m\}$ be the edge set of T, where $m = 2n - 3$. Consider the incident index δ between edges and the canonical paths defined by

$$\delta_{i,k} = \begin{cases} 1, & \text{if } e_k \in E(P_i), \\ 0, & \text{otherwise,} \end{cases}$$

for $1 \leq i \leq n$ and $1 \leq k \leq m$. Since each edge e occurs in exactly two of the canonical paths in $\{P_1, P_2, \ldots, P_n\}$, we have $\sum_{i=1}^{n} \delta_{i,k} = 2$ for each k. On the other hand, $\sum_{k=1}^{m} \delta_{i,k} = |E(P_i)|$ clearly holds for $1 \leq i \leq n$.
Now, fix an index $k \in \{1, \ldots, m\}$ and let (j, j') be the canonical index pair for e_k. Then we have

$$|E(P^{+}_{e_k,\pi})| + |E(P^{-}_{e_k,\pi})| = |E(P_j)| - 1 + |E(P_{j'})| - 1 = \sum_{i=1}^{n} \delta_{i,k}(|E(P_i)| - 1).$$

Here the first equality follows from $E(P^{+}_{e_k,\pi}) \cup E(P^{-}_{e_k,\pi}) = E(P_j) \cup E(P_{j'}) \backslash \{e_k\}$ (see Figure 6.11), and the second one from $\delta_{i,k} = 1$ if and only if $i \in \{j, j'\}$. Therefore, we can conclude that

$$\alpha(T, \pi) = \sum_{k=1}^{m} \sum_{i=1}^{n} \delta_{i,k}(|E(P_i)|-1) = \sum_{i=1}^{n} \sum_{k=1}^{m} \delta_{i,k}(|E(P_i)|-1) = \sum_{i=1}^{n} |E(P_i)|(|E(P_i)|-1).$$

□

Fixing a circular ordering $\pi = (x_1, \ldots, x_n)$ for $n \geq 4$, we introduce two families of trees that are important in this study: (T, π) is a *skew caterpillar* if it contains a

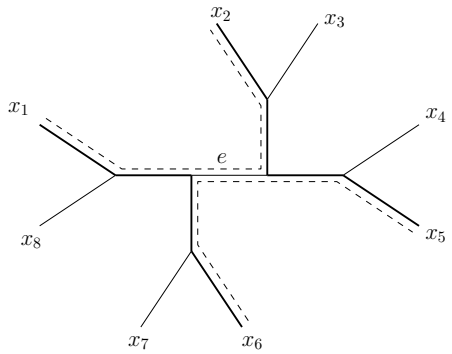


Figure 6.11: A tree T on $X = \{x_1, \ldots, x_8\}$ and a circular ordering $\pi = (x_1, \ldots, x_8)$, the edge e has the canonical index $(1, 5)$ the path $P^-_{e,\pi}$ from x_2 to x_5 and the path $P^+_{e,\pi}$ from x_6 to x_1 are highlighted in bold. The canonical paths P_5 from x_5 to x_6 and P_1 from x_1 to x_2 are highlighted with dashed lines.

a) b)

Figure 6.12: Two caterpillars on $\mathscr{T}_\pi$ with $\pi = (x_1, \ldots, x_n)$: a) a skew caterpillar and b) a centipede (where $k = \lfloor n/2 \rfloor$).

(necessarily unique) canonical path of size $n-1$; (T, π) is a *centipede* if it contains $n - 4$ canonical paths of size 4. See Figure 6.12a for an illustration on a skew caterpillar and b for a centipede. Note that for $n > 4$ a skew caterpillar contains exactly two canonical paths of size 2 and $n - 3$ canonical paths of size 3, while a centipede contains precisely two canonical paths of size 2 and two canonical paths of size 3. For $n = 4$ the skewed caterpillar and the centipede are the same tree with two canonical paths of size 2 and two of size 3.

Theorem 6.4.7. *Suppose that* $|X| = n \geq 4$. *Given a circular ordering* $\pi = (x_1, ..., x_n)$ *and* $T \in \mathscr{T}_\pi$, *then we have*

$$3n - 11 \leq |N^\pi_{\mathrm{SPR}}(T)| \leq (n-1)^2 - 4n + 8, \tag{6.2}$$

where the minimum is achieved if (T, π) *is a centipede and the maximum is achieved if* (T, π) *is a skew caterpillar. Moreover for* $n \geq 7$ *the minimum score*

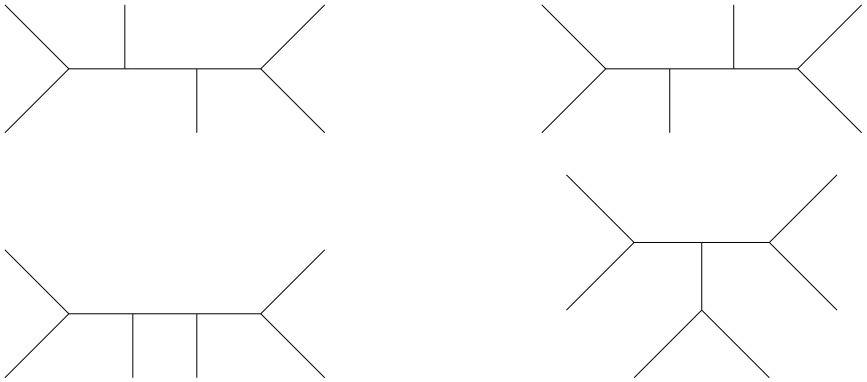


Figure 6.13: The four possible trees (ignoring the leaf labelling) on 6 leaves.

is achieved if and only if (T, π) *is a centipede and the maximum score is achieved if and only if* (T, π) *is a skew caterpillar.*

Proof. For simplicity, let $\alpha^*(T, \pi) = \sum_{i=1}^{n} |E(P_i)|^2$, where $P_1, \ldots, P_n$ is the set of canonical paths of (T, π). Then by Lemma 6.4.6 and Theorem 6.4.4, we have

$$|N^{\pi}_{\text{SPR}}(T)| = \alpha^*(T, \pi) - 13n + 27.$$

Therefore it suffices to show that

$$26 + 16(n-4) \leq \alpha^*(T, \pi) \leq 8 + 9(n-3) + (n-1)^2, \tag{6.3}$$

and the minimum is achieved precisely when (T, π) is a centipede, and the maximum is achieved precisely when (T, π) is a skew caterpillar. We will prove this claim by induction on $n = |X|$.

The reader can easily check that the result holds for the base case $n = 4$, where $\alpha^*(T, \pi) = 26$. There is one possible plane tree for $n = 5$ and it is easy to check that $\alpha^*(T, \pi) = 42$. The four possible plane trees for $n = 6$ are shown in Figure 6.13. The two centipedes have an $\alpha^*(T, \pi) = 58$ and the skewed caterpillar has $\alpha^*(T, \pi) = 60$, the fourth tree has the same α^* as the skewed caterpillar, so we see Equation (6.3) also holds for this case. If $n = 7$ there are 6 plane trees (see Figure 6.6) and it is straightforward to check that $74 \leq \alpha^*(T, \pi) \leq 80$, where the

minimum is achieved precisely when (T, π) is a centipede, and the maximum is achieved precisely when (T, π) is a skew caterpillar.

Now assume that $n > 7$ and the result holds for $n-1$. Without loss of generality, we may assume that $\{x_1, x_n\}$ is a cherry of T, that is, $|E(P_n)| = 2$. Let $X' = X - \{x_n\}$, $\pi' = (x_1, \ldots, x_{n-1})$ and T' be the tree obtained from T by removing leaf x_n and contracting the resulting degree two vertex. Let e'_i be the pendant edge incident to x_i in T'. Denoting the canonical paths of (T', π') by $P'_1, \ldots, P'_{n-1}$, then we have $|E(P'_i)| = |E(P_i)| - 1$ for $i \in \{1, n-1\}$ and $|E(P'_i)| = |E(P_i)|$ for $1 < i < n-1$. This implies

$$\alpha^*(T, \pi) = \alpha^*(T', \pi') + 2|E(P'_1)| + 2|E(P'_{n-1})| + 6. \tag{6.4}$$

Noting that P'_1 is the path between x_1 and x_2 in T', and P'_{n-1} the path between x_{n-1} and x_1, we have

$$5 \leq |E(P'_1)| + |E(P'_{n-1})| \leq n. \tag{6.5}$$

To see the first inequality holds, note that $\{x_{n-1}, x_1\}$ and $\{x_1, x_2\}$ cannot be both cherries in T', and hence either P'_1 or P'_{n-1} contains at least three edges while the other one contains at least two edges. An example is illustrated in Figure 6.14a.

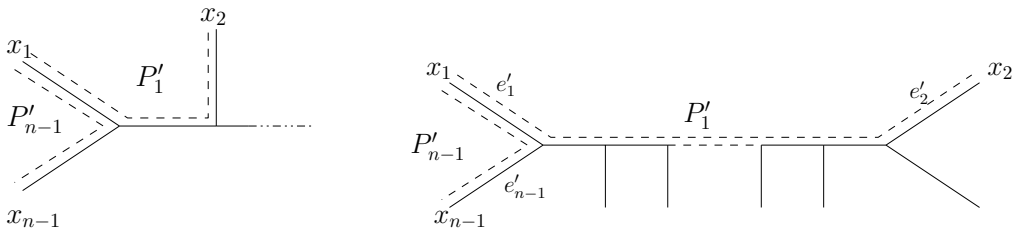
Now, let $E^o(T)$ be the set of the interior edges of T. To see the second inequality holds, note that $E(P'_1) \cap E(P'_{n-1})$ contains exactly the pendant edge e'_1, and $(E(P'_1) \cup E(P'_{n-1})) \subseteq (\{e'_1, e'_2, e'_{n-1}\} \cup E^o(T'))$, where $|E^o(T')| = n-4$. Together with Equation (6.4), this implies that Equation (6.3) also holds for n. An example is illustrated in Figure 6.14b.

Clearly, if (T, π) is a centipede, then we have $\alpha^*(T, \pi) = 26 + 16(n-4)$. Conversely, if $\alpha^*(T, \pi) = 26 + 16(n-4)$, then by Equation (6.4) and Equation (6.5) we know that the minimum change is $|E(P'_1)| + |E(P'_{n-1})| = 5$. This results in $\alpha^*(T', \pi') = 26 + 16(n-4) - (2 \times 5 + 6) = 26 + 16(n-5)$. Together with the induction assumption, this implies that (T', π') is a centipede, and hence (T, π)

contains exactly two canonical paths of size 2, two canonical paths of size 3, and $n-4$ paths of size 4. In other words, we can conclude that (T, π) is a centipede, as required.

On the other hand, if (T, π) is a skew caterpillar, then $\alpha^*(T, \pi) = 8 + 9(n-3) + (n-1)^2$. Conversely, if $\alpha^*(T, \pi) = 8+9(n-3)+(n-1)^2$, then by Equation (6.4) and Equation (6.5) we know that the maximum change is $|E(P'_1)|+|E(P'_{n-1})| = n$ and this results in $\alpha^*(T', \pi') = 8 + 9(n-3) + (n-1)^2 - (2n+6) = 8 + 9(n-4) + (n-2)^2$. Together with the induction assumption, this implies that (T', π') is a skew caterpillar, and hence (T, π) contains a canonical path of size $n-1$. We can conclude that (T, π) is a skew caterpillar, which completes the induction step, and hence also the proof of the theorem. □

We established a lower and upper bound for the size of $N^{\pi}_{\text{SPR}}(T)$ and the lower bound is achieved if T is a centipede and the upper bound is achieved if T is a skewed caterpillar and $n \geq 4$. In the last part of this chapter we will establish upper and lower bounds for $|N^{\pi}_{\text{TBR}}(T)|$ and we characterise the trees that lead to the minimum or maximum size $N^{\pi}_{\text{TBR}}(T)$.



(a) (b)

Figure 6.14: a) A centipede with $|E(P'_{n-1})| = 2$ and $|E(P'_1)| = 3$. b) A skew caterpillar where $|E(P'_{n-1})| = |\{e'_{n-1}, e'_1\}| = 2$ and $|E(P'_1)| = |\{e'_1, e'_2\} \cup E^o(T')| = n-2$.

Lemma 6.4.8. *Suppose that $|X| = n \geq 7$. Given a circular ordering $\pi = (x_1, ..., x_n)$ and $T \in \mathscr{T}_\pi$, then we have*

$$(n-5) \leq \beta(T, \pi) \leq \frac{(n-4)(n-5)(n-3)}{6}, \tag{6.6}$$

where if $n \geq 8$ the minimum is achieved if and only if (T, π) is a centipede and the maximum is achieved if and only if (T, π) is a skew caterpillar.

Remark: For $n = 5$ there is one possible topology [57] and $\beta(T, \pi) = 0$, so the first part of the lemma holds, but for $n = 6$ there are four possible trees with $\beta(T, \pi) = 1$ for the two centipedes and the caterpillar and $\beta(T, \pi) = 0$ for the fourth possible tree.

Proof. We shall establish the lemma by induction on $n = |X|$. For $n = 7$ $\beta(T, \pi) = 2$ for the centipede and $\beta(T, \pi) = 4$ for the skew caterpillar, but the score for the other possible trees are also either 2 or 4. For $n = 8$, $\beta(T, \pi) = 3$ holds if and only if T is a centipede, which is the smallest possible value for $\beta(T, \pi)$, and $\beta(T, \pi) = 10$ if and only if T is a skew caterpillar which is the highest possible value for $\beta(T, \pi)$.

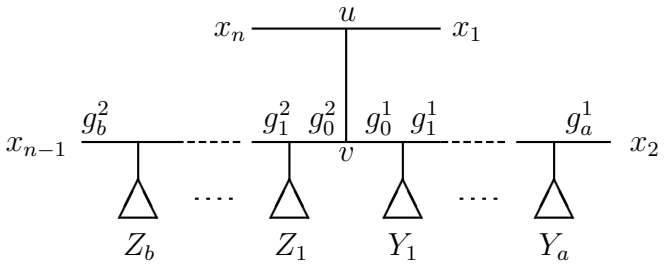


Figure 6.15: The tree T in the proof of Lemma 6.4.8.

Now assume that $n > 8$ and the result holds for $n-1$. Without loss of generality, we may assume that $\{x_1, x_n\}$ is a cherry of T. Let u be the interior vertex incident to both x_1 and x_n, and $\{u, v\}$ the edge inducing the split $\{x_1, x_n\}|X - \{x_1, x_n\}$. In addition, denote the edges in the path from v to x_2 consecutively by g_0^1 to g_a^1, and those in the path from v to x_{n-1} by g_0^2 to g_b^2, (see Fig. 6.15 for an illustration). Without loss of generality, we may further assume that $a \geq b$, which implies $a \geq 1$. On the other hand, we have $a + b \leq n - 4$ as T contains at most $n - 3$ interior edges.

Consider the path P^* between x_2 to x_{n-1}, that is, $E(P^*) = \{g_a^1, \ldots, g_0^1, g_0^2, \ldots, g_b^2\}$. Let $E^o(P)$ be the set of the interior edges of $P(x, y)$, that is, the edges in $E(P(x, y))$ that are incident to neither x nor y. Then for each edge $e \in E^o(P^*)$, let P_e^o be the unique path in $\{P_{e,\pi}^+, P_{e,\pi}^-\}$ that does not contain the edge $\{u, v\}$ and P_e^1 the other one. In addition, for any edge e in $E^o(T) - (E^o(P^*) \cup \{\{u, v\}\})$, neither $P_{e,\pi}^+$ nor $P_{e,\pi}^-$ contains $\{u, v\}$. Let $X' = X - \{x_n\}$, $\pi' = (x_1, \ldots, x_{n-1})$ and T' be the tree obtained from T by removing leaf x_n and contracting the resulting degree two vertex. While for all $e \in E(P^*)$ in T' the path $P_e^{0'}$ is the same as in T, the path $P_e^{1'}$ in T' has one edge less than P_e^1, that is $|E(P_e^{1'})| = |E(P_e^1)| - 1$. Therefore

$$\begin{aligned} \beta(T, \pi) - \beta(T', \pi') &= \sum_{e \in E^o(P^*)} (|E(P_e^o)| - 2)(|E(P_e^1)| - 2) \\ &- \sum_{e \in E^o(P^*)} (|E(P_e^o)| - 2)(|E(P_e^1)| - 3) \\ &= \sum_{e \in E^o(P^*)} (|E(P_e^o)| - 2). \end{aligned} \tag{6.7}$$

We can have maximal two interior edges e of P^* with $|E(P_e^o)| = 2$ (if e is adjacent to a cherry), but for $n > 6$ at least one $e \in E^o(P^*)$ must have $|E(P_e^o)| = 3$. Therefore we have $\beta(T, \pi) - \beta(T', \pi') \geq 1$, and it is straightforward to check that the equality holds when (T, π) is a centipede. Together with the induction assumption, if (T, π) is a centipede, we have $\beta(T, \pi) = (n - 5)$. Conversely if $\beta(T, \pi) = (n-5)$ then the minimum change is $\sum_{e \in E^o(P^*)}(|E(P_e^o)| - 2) = 1$, which leads to $\beta(T', \pi') = (n-6)$. This implies that (T, π) is a centipede with two edges adjacent to the two cherries where $(|E(P_e^+)| - 2) \times (|E(P_e^+)| - 2) = 0$ and $(n-5)$ edges where $(|E(P_e^+)| - 2) \times (|E(P_e^+)| - 2) = 1$.

Now we will investigate the tree topology with the maximum size of $\beta(T, \pi)$. Let $\varphi(i, j) = (1 + \ldots + i) + (1 + \ldots + j)$ for $0 \leq i \leq a, 0 \leq j \leq b$. As established above the difference between $\beta(T, \pi)$ and $\beta(T', \pi')$ is

$$\sum_{e \in E^o(P^*)} (|E(P_e^o)| - 2) = \sum_{e \in E^o(P^*)} |E(P_e^o)| - 2(a + b).$$

Note that each edge g_i^1 contributes i to $\sum_{e\in E^o(P^*)} |E(P_e^o)|$ and g_j^2 contributes j. Let $F = E(T) - \{\{\{u, x_1\}, \{u, x_n\}, \{u, v\}\} \cup E(P^*)\}$. For each edge $f \in F$, there exists at most one $e \in E^o(P^*)$ with $f \in E(P_e^o)$ and $|F|$ can be $(2n-3)-3-|E(P^*)|$ at most, so we can conclude that

$$\begin{aligned}
\beta(T,\pi) - \beta(T',\pi') &\leq \varphi(a,b) + (2n-6) - |E(P^*)| - 2(a+b) \qquad (6.8)\\
&= \varphi(a,b) + (2n-6) - (a+b) - 2 - 2(a+b)\\
&\leq (1 + \cdots + (a+b)) + (2n-6) - 3(a+b) - 2\\
&\leq (1 + \cdots + (n-4)) + (2n-6) - 3(n-4) - 2\\
&= (n-4)^2 + (n-4)/2 - n + 4\\
&= (n^2 - 9n + 20)/2.
\end{aligned}$$

Here one necessary and sufficient condition for the above equalities holding is $b = 0$ and $a + b = n - 4$, that is, T is a skew caterpillar. Together with the induction assumption, we know that $\beta(T,\pi) \leq \frac{(n-4)(n-5)(n-3)}{6}$, in which case the equality holds if and only if (T,π) is a skew caterpillar. This completes the proof of the induction step, and hence also the lemma. □

Theorem 6.4.9. *Suppose that $|X| = n \geq 7$. Given a circular ordering $\pi = (x_1, ..., x_n)$ and $T \in \mathscr{T}_\pi$, then we have*

$$4(n-4) \leq |N_{\text{TBR}}^\pi(T)| \leq (n^3 - 6n^2 + 11n - 6)/6, \qquad (6.9)$$

where if $n \geq 8$ the minimum is achieved if and only if (T,π) is a centipede and the maximum is achieved if and only if (T,π) is a skew caterpillar.

Proof. By Theorem 6.4.4 and Theorem 6.4.5, we know that

$$|N_{\text{TBR}}^\pi(T)| = |N_{\text{SPR}}^\pi(T)| + \beta(T,\pi).$$

Together with Theorem 6.4.7 and Lemma 6.4.8, this implies the theorem, as required. □

6.5 Concluding remarks

In this chapter we established formulae that describe the size of NNI, SPR and TBR neighbourhoods of a tree where the circular ordering of the leaves is preserved. We found upper and lower bounds for the size of these neighbourhoods and characterised the type of trees that have minimum or maximum size neighbourhoods. In further investigations the size of the split neighbourhoods of circular tree edit operations could also be established [78, in preparation]. In the next chapter we shall conclude with an outlook of some possible future directions.

Chapter 7

Discussion and future work

7.1 Conclusions and future work

The use of networks in phylogenetic studies has become increasingly popular in recent years. This is mainly due to the fact that there are cases where evolutionary processes cannot be represented by a tree, such as hybridisation in plants or horizontal gene transfer in micro-organisms. Even if tree-like evolution is expected, methodological errors can lead to conflicting signals in the data that must be identified and potentially removed [65].

Data-displaying networks give a snapshot of the data and enable us to display conflicting signals, even if the cause of these conflicts is not known. One of the main examples of these kind of networks are split networks, which are based on split systems. In this thesis we investigated circular split systems, which have the property of being representable in two dimensions without crossing edges as an outer-labelled planar split network. This is desirable because these networks can be more easily analysed than higher dimensional networks.

More specifically, we developed methods for constructing circular split systems from a set of trees and from distance data, used them to find approximations for minimum evolution trees and established some of their mathematical properties. In Chapter 3 we introduced SuperQ, a method to construct a weighted circular

split network from a set of partial input trees. SuperQ is a novel approach which incorporates the edge weights of the input trees. While it produces results that are in good agreement with former methods, it has some additional advantages, for example it could be used on partial networks and the resulting networks are independent of the order in which the input is processed. A general problem of using quartet methods, such as SuperQ, is that memory consumption for storing quartets grows in $O(n^4)$ where n is the number of taxa. SuperQ would benefit from a more efficient way to store quartets, so this would be a direction for improvement. The publication of the SuperQ paper generated a lot of interest and was discussed by several researchers well known in the area of phylogenetics (see for example [67]). It also appears that the implementation of SuperQ is being downloaded and used by researchers and so we hope to see several citations for the paper over the coming years.

In Chapter 4 we reviewed the NeighborNet algorithm and used the framework by Levy and Pachter to introduce a greedy minimum evolution adaptation of NeighborNet. We then investigated the problem of finding a minimum evolution tree within circular split systems. The methods discussed in Chapter 4 are compared according to their ability to capture relevant information for the construction of minimum evolution trees. Based on the comparison to FastME, we concluded that circular split systems do capture this information well. NeighborNet in particular generated good results for both treelike and non-treelike data, but in general the trend was that our method performs particularly well in comparison to FastME when applied to data which contains conflicting signals. These results are interesting and quite surprising because a circular split system captures just a tiny fraction of the whole search space for trees.

Although we worked with the minimum evolution criterion to construct phylogenies, the balanced minimum evolution criterion is also a commonly used alternative. FastME makes use of this criterion for example. It might be possible to adapt our approach in Chapter 5 to find the balanced minimum evolution tree within a circular split system. Recall that we used a dynamic programming algorithm to find the minimum evolution tree within a circular split system. We

started to investigate this idea for the balanced minimum evolution criterion (as described in Chapter 2), but could not work out how to built up solutions for a problem from the solutions of its sub-problems, since in the balanced minimum evolution framework each edge length in a tree depends on the internal structure of its adjacent subtrees and not just on the number of leaves in these subtrees.

It could also be interesting to adapt our approach to consider different kinds of split systems, such as flat split systems [82]. FlatNJ is a method to produce a flat split system from quartets [2]. This type of split system can also be represented in a two dimensional split network, but not always by an outer-labelled graph. Instead flat split systems are representable by a planar graph in which internal vertices can be labelled with taxa as well. Therefore these networks can display more information than circular split networks. Another future direction could be to develop restricted approaches for rooted trees, where special classes of cluster systems could be considered as search spaces.

After finding such interesting results in Chapter 5 we wanted to investigate circular split systems in depth and from a more theoretical point of view. Since FastME uses tree edit operations to search tree space we were intrigued what properties tree edit operations that preserve a circular ordering of a tree might have. In Chapter 6 we established the sizes of circular tree neighbourhoods for the different tree edit operations as well as their upper and lower bounds. We also characterised the trees that have the minimum and maximum size neighbourhoods. This would also be interesting to investigate in a similar manner for other types of split systems, like flat split systems.

Split networks are used to visualise conflicts. A lot of current work in the area of phylogenetic networks is focusing on developing methods that not only display conflicts but also identify their cause. This is often difficult and leads to less efficient methods. However, it seems like fast methods for constructing networks are in demand as researchers are realising the potential of using phylogenetic networks, rather than trees, in certain situations.

As networks become more frequently used it is important to establish criteria for their construction. For example, recent attempts were made to define a parsimony score for networks and the associated optimisation problems were proven to be NP-hard [15, 32, 42]. It would be interesting to see if the minimum evolution score can be extended to networks.

In [17], Catanzaro describes several versions of the minimum evolution problem and their hardness. To the best of our knowledge the hardness of the minimum evolution problem for trees as defined in this thesis is still open. The balanced minimum evolution problem on the other hand has been shown to be NP-hard [31]. As mentioned above, it would be interesting to see if this can also be shown for the balanced minimum evolution problem restricted to a circular split system.

The way information is displayed in terms of networks and whether improvements are possible could also be the subject of future investigations. For example, it is not really possible to highlight trees within split networks, because all parallel edges that correspond to a split must be highlighted which leads quickly to a very confusing picture.

7.1.1 SPECTRE: Suite of Phylogenetic Tools for Reticulate Evolution

In the course of conducting research described in this thesis we implemented several useful tools, data structures and algorithms. We would like to distribute these implementations, together with an implementation of FlatNJ and a tool that draws two dimensional split networks in a single package that is freely available to the public.

Such a software package is currently work in progress in a collaboration with Daniel Mapleson, Andreas Spillner and Monika Balvociute. The aim is to provide an open source software package that contains a number of reusable and extendible tools for creating phylogenetic trees and networks. The tools should

work well both at the command line, so that they can be easily integrated into bioinformatics pipelines, and, where applicable, should have graphical front ends, so that they are easy to use for those who are not familiar with running tools from the command line. In addition, we will provide common methods and functionality as a library so that other software developers, for example PhD students and PostDocs can easily reuse and extend the toolkit, and speed up the development of their own tools.

There are several other software packages bundling phylogenetic tools, such as Phylogenetics Analysis Library (PAL) [26], the Phylogeny Inference Package (Phylip) [30], Splitstree [49], Phylogenetic Analysis Using Parsimony (PAUP) [84] and Molecular Evolutionary Genetics Analysis (MEGA) [85].

PAL, like our toolkit, is an open source library of commonly used data structures and algorithms for phylogenetics analysis implemented in Java. However, the last changes to PAL were made in 2002 and since then many developments have occurred in both the area of phylogenetics and methods for creating and distributing software.

Phylip, has had an impressively long life (created in 1980) and is still very much alive (last update was mid 2013). It does not have much overlap with our toolkit in terms of functionality. The design goal for Phylip is to infer evolutionary trees from distance matrices and sequence alignments, so there is not much functionality related to creating and manipulating networks, which is the focus of our package. Nevertheless, Phylip was inspriring in terms of its design philosophy as it is a collection of stand alone tools that have a consistent interface. This is useful in bioinformatics pipelines, where users may want to chain tools together. However, it does not provide shared libraries which other software developers can easily reuse.

Splitstree shares the most functionality with our toolkit, specifically with regards to computing unrooted phylogenetic networks. However, it does not provide the same methods. It is also written in Java and is freely available. However, it is

not open source which restricts other developers from improving the software and reusing its functionality in other projects. In addition, due to dependencies on graphical subsystems (at least in its current version Splitstree4), Splitstree does not run well in high performance computing environments.

PAUP, commercial software, and MEGA (free for academic research use) are packages for inferring phylogenies using parsimony and maximum likelihood methods that have large user bases. They however, do not use networks and therefore do not overlap with our phylogenetics toolkit.

As previously mentioned, we have two core aims for our toolkit. The first is to provide open source implementations of the methods mentioned in this thesis as a set of stand alone tools. The second, is to capture reusable data structures and algorithms in an open source software library, which can be readily integrated into other projects. We will achieve this by hosting the library on a publicly accessible Maven repository. To make the development process transparent, all the source code, along with its version history, will be visible online via a web-based hosting service called GitHub. GitHub also makes it easy for the community to participate in extending and improving the software.

Our other objective for the toolkit is to make the interfaces, both command line and graphical, consistent and logically designed throughout the project. In addition, we will provide a manual which will contain detailed information about each executable tool and the shared functionality in the library.

The Phylogenetic toolkit will contain the following ready-to-use tools:

- SuperQ, as described in Chapter 3, constructs a weighted circular split system from a set of partial input trees.
- QNet, a method to construct a weighted circular split system from a set of weighted quartets [38].
- NetMake, allows the user to construct a weighted circular split system from

distances by using different versions of NeighborNet as for example the ones described in Chapter 4.

- FlatNJ, implemented by Monika Balvociute [2], is a method to construct a weighted flat split system from a set of weighted quartets.
- NetME, as described in Chapter 4, finds a minimum evolution tree within a full circular split system.
- NetView, implemented by Monika Balvociute [2], is a tool that draws two dimensional split networks from circular and flat split systems.

All the tools reuse common data structures and algorithms where possible. These will be stored in the *Core* package, which will also be available for download as a separate library via Maven. This package contains sub groups based on their specific kind of functionality. For example, a sub package called *data structures* will provide commonly used phylogenetic data structures relating to concepts such as: splits, trees, networks, distances, quartets, quadruples, alignments, etc. Another sub package called *input and output* will handle loading and saving common phylogenetic file formats, like Nexus and Phylip. The *Maths* sub package will handle common mathematical data structures and algorithms such as basic statistics and matrix algebra.

Several methods in our toolkit require solving linear and/or quadratic optimisation problems. Originally they made use of a third-party solver called Gurobi. However, in order to provide free open source alternatives, we will create another library, which again will be available via Maven. This library has a mechanism for defining linear and quadratic programming problems and translating those into definitions used by solvers. We plan to support both free-ware and commercial solvers, such as Gurobi, JOptimizer, Apache commons math and GLPK. By using this library the software developer can pose an optimisation problem once, and then let the user chose the solver. The subsystem then translates the common problem definition into a solver-specific one. This is beneficial in several ways. First, it allows researchers to compare solvers in terms of results and runtime performance. Secondly, it can be used as a mechanism to provide an alternative,

free, and built-in, solver when a commercial one, such as Gurobi, is not available.

Finally, miscellaneous tools and utilities that might be useful for the user or developer will be made available with a command line interface. These additional tools will include: Qmaker, which breaks down trees into quartets and is used in SuperQ and QNet; GenQS, which takes sequences, splits or geographical coordinates and outputs a set of quadruples; a random distance generator tool, which creates Phylip or Nexus files with a randomly generated distance matrix and a random sequence alignment generator.

We believe that our toolkit will help to increase the openness and transparency of phylogenetic software and thereby allow other researchers to build their own tools faster by leveraging resources that we make available and by using our implementations as a template or protocol for their own projects. In addition, by making our methods and tools freely available and easy to use we hope to enable other researchers to generate new knowledge and make useful biological discoveries.

7.2 Final remarks

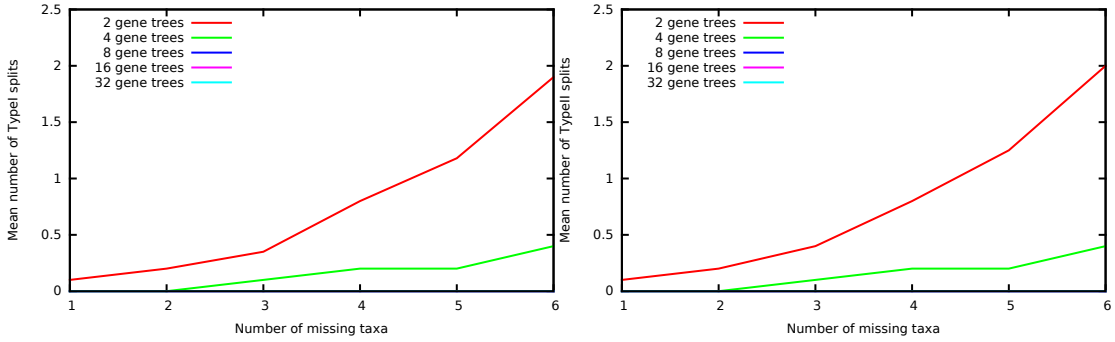
Circular split systems have gained in popularity because of their representation as a planar network. This thesis looked at different aspects and ideas connected to circular split systems. Our research included results that lead to a better understanding of these split systems and enables further research that will improve our knowledge and methodology for producing relevant representations of evolutionary relationships. In the future we are hoping to continue to investigate properties of split systems, improve the construction of networks and extend our ability to identify and characterise the cause of conflicts in phylogenetic data. Especially with the phylogenetic toolkit as a compendium of the work presented in this thesis, we are hoping to start something that will be extended and grow over the years and add many valuable phylogenetic construction methods to be used in the research community.

Appendix A

SuperQ: supplementary materials

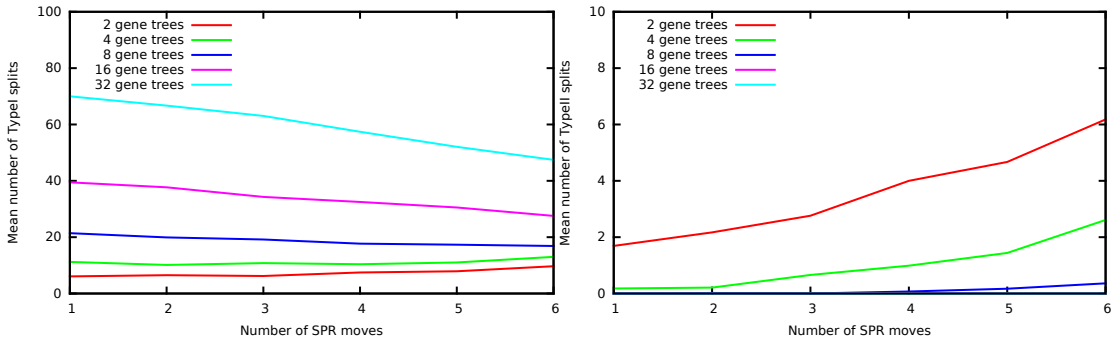
	Number of taxa									
Input trees	10	20	30	40	50	60	70	80	90	100
2	0.5	2.1	4.8	12.7	32.9	79.6	180.7	427.6	719.0	1422.1
4	0.8	2.7	7.0	19.3	47.9	116.3	254.5	572.7	934.1	1749.5
6	1.1	3.4	9.7	27.1	68.0	160.6	345.5	743.0	1204.7	2144.1
8	1.4	4.3	13.1	36.4	91.8	217.3	457.4	935.5	1516.3	2656.4
10	1.7	5.2	16.9	48.0	121.1	281.5	587.7	1165.4	1881.0	3253.5

Table 1: Run times for SuperQ in seconds for selected numbers of taxa (columns) and numbers of input trees (rows). Each run time is averaged over 10 inputs.

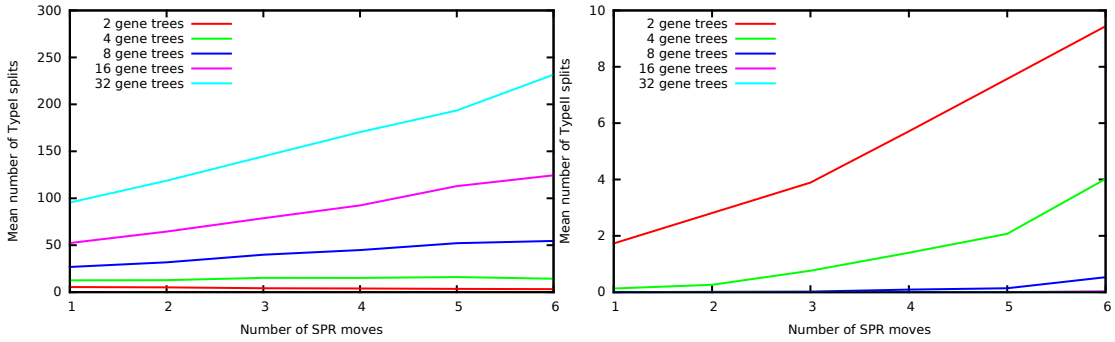


a) b)

Figure 1: a) Type I and b) Type II splits for SuperQ for gene trees with missing taxa generated from a random tree with 32 taxa.

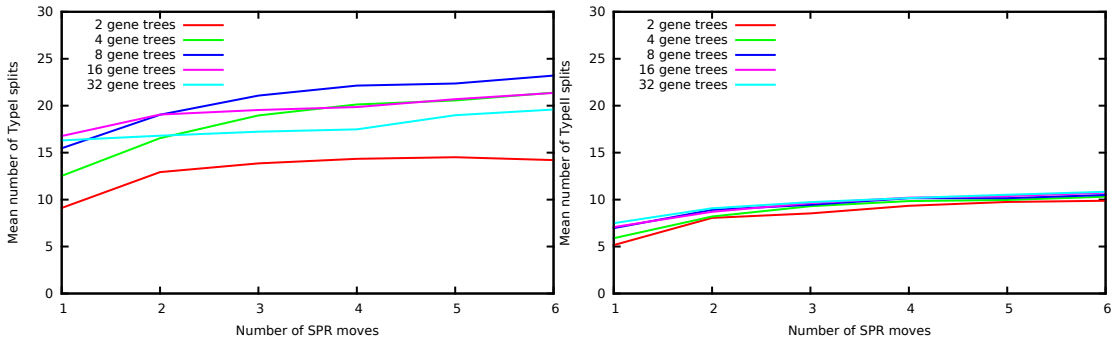


a) b)

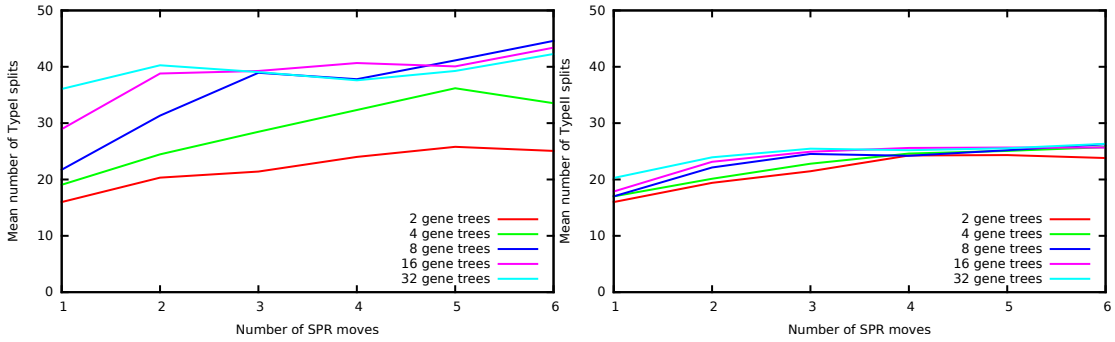


c) d)

Figure 2: a) Type I and b) Type II splits for Q-imputation and c) Type I and d) Type II splits for Z-closure for gene trees with 16 taxa generated by performing random SPR moves.



a) b)



c) d)

Figure 3: a) Type I and b) Type II splits for SuperQ for gene trees with 16 taxa and c) Type I and d) Type II splits for SuperQ gene trees with 32 taxa generated by performing random SPR moves.

Appendix B

The original NeighborNet algorithm

NeighborNet(X, D, α, β)

Initiate $G = \{X, \emptyset\}$
$C = C_1, \ldots, C_{|X|}$ where $C_i = \{x_i\}$ and $x_i \in X$
$D_C = D$
$D_V = D$
$\Sigma(T) = \{S_1, \ldots, S_{|X|}\}$ where $S_i = \{x_i\}|(X \backslash x_i)$
while $|C| > 1$ **do**
  // First Selection Step
  Choose a pair $C_{r*}, C_{s*} \in C$ that minimise the following formula:
  $Q_D(C_r, C_s) = (|C| - 2)D_C(C_r, C_s) - \sum_{t \neq r} D_C(C_r, C_s) - \sum_{t \neq s} D_C(C_r, C_s)$
  // Second Selection Step
  Choose the pair $x_{i*} \in C_{r*}, x_{j*} \in C_{s*}$ that minimise the following formula:
  $\hat{Q}_D(x_i, x_j) = (|C| - 4 + |C_{r*}| + |C_{s*}|)D_V(x_i, x_j) - \sum_{t \neq r*, s*} D_C(x_i, C_t) - \sum_{t \neq r*, s*} D_C(x_j, C_t) - \sum_{x_k \in (C_{r*} \cup C_{s*}) \backslash \{x_i\}} D_V(x_i, x_k) - \sum_{k \in (C_{r*} \cup C_{s*}) \backslash \{x_j\}} D_V(x_j, x_k)$
  // Merge Step
  Update C by adding C_{s*} to C_{r*}
  Add $x_{i*}x_{j*}$ to $E(G)$
  Add $\{C_{r*} \cup C_{s*}\}|(V(G) \backslash (C_{r*} \cup C_{s*}))$ to $\Sigma(T)$
  // Reduction and update of D_C
  while $|C_l| > 2$ **do**
    Reduce$(x_i, x_j, x_k, G, C_l, D_V, \alpha, \beta)$
  end while
  // Update of D_C
  $D_C(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} D_V(x, y)$
end while
Backsubstitution of elements in C_1 and splits in $\Sigma(T)$

OUTPUT: C_1 (circular ordering), $\Sigma(T)$ (tree)

Reduce$(a, b, c, G, C_l, D_V, \alpha, \beta)$

for $i \in V(G) \backslash \{a, b, c\}$ **do**
  $D_V(u, i) = (\alpha + \beta)D_V(a, i) + (1 - \alpha - \beta)D_V(b, i)$
  $D_V(v, i) = \alpha D_V(b, i) + (1 - \alpha)D_V(c, i)$
  $D_V(u, v) = \alpha D_V(a, b) + \beta D_V(a, c) + (1 - \alpha - \beta)D_V(b, c)$
  remove a, b, c from $V(G)$ and C_l and add u, v to $V(G)$ and C_l
  remove edges ab, bc from $E(G)$ and add edge uv to $E(G)$
end for

OUTPUT: G, C_l, D_V

Levy and Pachter's NeighborNet algorithm

NeighborNet(X, D, μ)

1: $G = (X, \emptyset)$
2: C is the set of paths in G where initially:
3: $C = C_1, \ldots, C_{|X|}$ $C_i = x_i$ and $x_i \in X$
4: $D_C = D$
5: $\Sigma(T) = \{S_1, \ldots, S_{|X|}\}$ where $S_i = \{x_i\}|(X \backslash x_i)$
6: **while** $|C| > 1$ **do**
7: // First Selection Step
8: Choose a pair $C_{r*}, C_{s*} \in C$ that minimise the following formula:
9: $Q_D(C_r, C_s) = (|C| - 2)D_C(C_r, C_s) - \sum_{t \neq r} D_C(C_r, C_t) - \sum_{t \neq s} D_C(C_t, C_s)$
10: // Second Selection Step
11: $x_i \in \hat{C}_k$ if $x_i \in C_k$ and x_i is a degree one vertex
12: Choose the pair $x_{i*} \in \hat{C}_{r*}, x_{j*} \in \hat{C}_{s*}$ that minimise the following formula:
13: $\hat{Q}_D(x_i, x_j) = (|C| - 4 + |\hat{C_{r*}}| + |\hat{C_{s*}}|)D(x_i, x_j) - \sum_{t \neq r*, s*} D_C(x_i, C_t) - \sum_{t \neq r*, s*} D_C(x_j, C_t) - \sum_{x_k \in (C_{r*} \cup C_{s*}) \backslash \{x_i\}} D(x_i, x_k) - \sum_{x_k \in (C_{r*} \cup C_{s*}) \backslash \{x_j\}} D(x_j, x_k)$
14: // Merge Step
15: Add the edge $x_{i*}x_{j*}$ to $E(G)$, so C is updated such that C_{s*} is added to C_{r*}
16: // Tree construction step
17: Add $(C_{r*} \cup C_{s*})|(X \backslash (C_{r*} \cup C_{s*}))$ to $\Sigma(T)$
18: // Update D_C
19: $D_C(C_r, C_s) := \sum_{x_i \in C_r, x_j \in C_s} \mu(x_i)\mu(x_j)D(x_i, x_j)$ and
20: $D_C(x, C_r) := \sum_{x_i \in C_r} \mu(x_i)D(x, x_i)$
21: Update μ
22: **end while**

OUTPUT: C_1 (circular ordering), $\Sigma(T)$ (tree)

GreedyME algorithm

GreedyME(X, D, μ)

1: $G = (X, \emptyset)$
2: C is the set of paths in G where initially:
3: $C = C_1, \ldots, C_{|X|}$ $C_i = x_i$ and $x_i \in X$
4: $D_C = D$
5: $\Sigma(T) = \{S_1, \ldots, S_{|X|}\}$ where $S_i = \{x_i\}|(X \backslash x_i)$
6: **while** $|C| > 1$ **do**
7: // First Selection Step
8: Choose a pair $C_{r*}, C_{s*} \in C$ that minimise:
9: $CalculateEdges(\Sigma(T) \cup ((C_r \cup C_s)|X - (C_r \cup C_s))$
10: // Second Selection Step
11: $x_i \in \hat{C}_k$ if $x_i \in C_k$ and x_i is a degree one vertex
12: Choose the pair $x_{i*} \in \hat{C}_{r*}, x_{j*} \in \hat{C}_{s*}$ that minimise the following formula:
13: $\hat{Q}_D(x_i, x_j) = (|C| - 4 + |\hat{C_{r*}}| + |\hat{C_{s*}}|)D(x_i, x_j) - \sum_{t \neq r*,s*} D_C(x_i, C_t) - \sum_{t \neq r*,s*} D_C(x_j, C_t) - \sum_{x_k \in (C_{r*} \cup C_{s*}) \backslash \{x_i\}} D(x_i, x_k) - \sum_{x_k \in (C_{r*} \cup C_{s*}) \backslash \{x_j\}} D(x_j, x_k)$
14: // Merge Step
15: Add the edge $x_i^* x_j^*$ to $E(G)$, so C is updated such that C_{s*} is added to C_{r*}
16: // Tree construction step
17: Add $(C_{r*} \cup C_{s*})|(X \backslash (C_{r*} \cup C_{s*}))$ to $\Sigma(T)$
18: // Update D_C
19: $D_C(x, C_r) := \sum_{x_i \in C_r} \mu(x_i) D(x, x_i)$
20: Update μ
21: **end while**

OUTPUT: C_1 (circular ordering), $\Sigma(T)$ (tree)

Appendix C

NetME: supplementary materials

Number of taxa	50	100	200	400	800
NetME	0.18	1.74	23.24	379.44	6150.23
FastME	0.18	0.18	0.15	0.18	1.22

Table 2: Run times for NetME and FastME in seconds on approximately treelike data for selected numbers of taxa (columns). Each run time is averaged over 100 inputs.

Number of taxa	50	100	200	400	800
NetME	0.18	1.77	25.20	406.40	6101.12
FastME	0.17	0.19	0.20	0.37	1.02

Table 3: Run times for NetME and FastME in seconds on non-treelike data for selected numbers of taxa (columns). Each run time is averaged over 100 inputs.

References

[1] B.L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5:1–15, 2001. 86, 95, 96

[2] M. Balvociute, A. Spillner, and V. Moulton. FlatNJ: A novel network-based approach to visualize evolutionary and biogeographal relationships. *Systematic Biology*, 63(3):383–396, 2014. 111, 115

[3] H.-J. Bandelt and A. Dress. A canonical decomposition theory for metrics on a finite set. *Advances in Mathematics*, 92:47–105, 1992. 53, 73

[4] H.-J. Bandelt and A. Dress. Split decomposition: a new and useful approach to phylogenetic analysis of distance data. *Molecular Phylogenetics and Evolution*, 1:242–252, 1992. 22, 24, 73, 83

[5] S. Bastkowski. Algorithmen zum Finden von phylogenetischen Bäumen in Neighbor-Net Netzwerken. Master's thesis, Ernst-Moritz-Arndt-University Greifswald, 2010. 4, 69

[6] S. Bastkowski, A. Spillner, and V. Moulton. Fishing for minimum evolution trees with NeighborNets. *Information Processing Letters*, 2013. 4

[7] B. Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41:3–10, 1992. 36

[8] O. R. P. Bininda-Emonds, editor. *Phylogenetic supertrees: Combining information to reveal the tree of life*. Kluwer Academic Publishers, 2004. 28

[9] M. Brinkmeyer, T. Griebel, and S. Böcker. Polynomial supertree methods revisited. *Advances in Bioinformatics*, 2011. article ID 524182. 30

[10] D. Bryant. Hunting for trees in binary character sets: Efficient algorithms for extraction, enumeration and optimization. *Journal of Computational Biology*, 3:275–288, 1996. 68, 83

[11] D. Bryant. *Building trees, hunting for trees and comparing trees. Theory and methods in phylogenetic analysis.* PhD thesis, University of Canterbury, 1997. 13, 61, 62, 68, 69, 70, 83

[12] D. Bryant. A classification of consensus methods for phylogenies. In M. Janowitz, F.-J. Lapointe, F. McMorris, B. Mirkin, and F. Roberts, editors, *BioConsensus*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. American Mathematical Society, 2003. 28

[13] D. Bryant. On the uniqueness of the selection criterion in neighbor-joining. *Journal of Classification*, 22:3–15, 2005. 16

[14] D. Bryant and V. Moulton. Neighbor-net: An agglomerative method for the construction of phylogenetic networks. *Molecular Biology and Evolution*, 21(2):255–265, 2004. 25, 48, 49, 62, 73

[15] D. Bryant and M. Steel. Bureaucratic set systems, and their role in phylogenetics. *Applied Mathematics Letters*, 25(8):1148–1152, 2012. 112

[16] P. Buneman, F. R. Hodson, D. G. Kendall, and P. T. Tautu. The recovery of trees from measures of dissimilarity. *Mathematics in the Archaeological and Historical Sciences, Edinburgh University Press*, 1:387–395, 1971. 18, 19, 20

[17] D. Catanzaro. The minimum evolution problem: overview and classification. *Networks*, 53:112–125, 2009. 14, 15, 33, 68, 70, 83, 112

[18] B. Chor and T. Tuller. Finding a maximum likelihood tree is hard. *Journal of ACM*, 53:722–744, 2006. 68, 83

[19] Th. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. 2001. 71

[20] W. Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Biology*, 35:224–229, 1986. 68

[21] R. Desper and O. Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of Computational Biology*, 9(5):687–705, 2002. 17, 68, 74

[22] R. Desper and O. Gascuel. Theoretical foundation of the balanced minimum evolution method of phylogenetic inference and its relationship to weighted least-squares tree fitting. *Molecular Biology and Evolution*, 21(3):587–598, 2004. 14

[23] Z. Dostál. *Optimal quadratic programming algorithms*. Springer, 2009. 31

[24] A. Dress and D. Huson. Constructing split graphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1:109–115, 2004. 22, 25, 34, 73

[25] A. Dress, K. Huber, J. Koolen, V. Moulton, and A. Spillner. *Basic phylogenetic combinatorics*. Cambridge University Press, 2012. 18, 29

[26] A. Drummond and K. Strimmer. PAL: An object-oriented programming library for molecular evolution and phylogenetics. *Bioinformatics*, 17:662–663, 2001. 113

[27] A. W. F. Edwards and L. L. Cavalli-Sforza. The reconstruction of evolution. *Heredity*, 18, 1963. 11

[28] J. Felsenstein. Evolutionary trees from gene frequencies and quantitative characters: Finding maximum likelihood estimates. *Evolution*, 35:1229–1242, 1981. ISSN 00143820. 11

[29] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2 edition, September 2003. 10, 13

[30] J. Felsenstein. *PHYLIP (Phylogeny Inference Package) version 3.6.* Department of Genome Sciences, University of Washington, Seattle, 2005. 113

[31] S. Fiorini and G. Joret. Approximating the balanced minimum evolution problem. *Operations Research Letters*, 40(1):31 – 35, 2012. 112

[32] M. Fischer, L. van Iersel, S. Kelk, and C. Scornavacca. On computing the maximum parsimony score of a phylogenetic network. *arXiv preprint arXiv:1302.2430*, 2013. 112

[33] W. M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971. 12

[34] L.R. Foulds and R.L. Graham. The steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3(1):43 – 49, 1982. 11

[35] O. Gascuel and M. Steel. Neighbor-joining revealed. *Molecular Biology and Evolution*, 23(11):1997–2000, 2006. 16

[36] T. Gonzalez. Simple algorithms for the on-line multidimensional dictionary and related problems. *Algorithmica*, 28:255–267, 2000. 70

[37] R.C. Griffiths and P. Marjoram. Ancestral inference from samples of dna sequences with recombination. *Journal of Computational Biology*, 3(4):479–502., 1996. 21

[38] S. Grünewald, K. Forslund, A. Dress, and V. Moulton. QNet: An agglomerative method for the construction of phylogenetic networks from weighted quartets. *Molecular Bioliology and Evolution*, 24:532–538, 2007. 27, 29, 34, 114

[39] S. Grünewald, A. Spillner, K. Forslund, and V. Moulton. Constructing phylogenetic supernetworks from quartets. In *Workshop on Algorithms in Bioinformatics (WABI)*, volume 5251 of *LNBI*, pages 284–295. Springer, 2008. 29

[40] S. Grünewald, V. Moulton, and A. Spillner. Consistency of the QNet algorithm for generating planar split graphs from weighted quartets. *Discrete Applied Mathematics*, 157:2325–2334, 2009. 29, 31

[41] S. Grünewald, A. Spillner, S. Bastkowski, A. Bögershausen, and V. Moulton. SuperQ: Computing supernetworks from quartets. *Transactions on Computational Biology and Bioinformatics*, 10 (1):151–160, 2013. 4

[42] J. Guohua, L. Nakhleh, S. Snir, and T. Tuller. Parsimony score of phylogenetic networks: Hardness results and a linear-time heuristic. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(3):495–505, 2009. 112

[43] R.W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950. 13

[44] E. Harding. The probabilities of rooted tree-shapes generated by random bifurcation. *Advances in Applied Probability*, 3:44–77, 1971. 35

[45] B. Holland, K. Huber, A. Dress, and V. Moulton. Delta plots: a tool for analyzing phylogenetic distance data. *Molecular Biology and Evolution*, 19: 2051–2059, 2002. 74

[46] B. Holland, F. Delsuc, and V. Moulton. Visualizing conflicting evolutionary hypotheses in large collections of trees: Using consensus networks to study the origins of placentals and hexapods. *Systematic Biology*, 54:66–76, 2005. 28

[47] B. Holland, G. Conner, K. Huber, and V. Moulton. Imputing supertrees and supernetworks from quartets. *Systematic Biology*, 56:57–67, 2007. 26, 35, 36

[48] P. J. Humphries and T. Wu. On the neighborhood of trees. *arXiv:1202.2203*, 2012. 96

[49] D. Huson. SplitsTree: analyzing and visualizing evolutionary data. *Bioinformatics*, 14:68–73, 1998. 113

[50] D. Huson and D. Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, 23:254–267, 2006. 2, 22, 34, 75

[51] D. Huson and R. Rupp. Summarizing multiple gene trees using cluster networks. In Keith Crandall and Jens Lagergren, editors, *Algorithms in Bioinformatics*, volume 5251 of *Lecture Notes in Computer Science*, pages 296–305. Springer Berlin / Heidelberg, 2008. 21

[52] D. Huson, T. Dezulian, T. Klöpper, and M. Steel. Phylogenetic supernetworks from partial trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1:151–158, 2004. 26, 28, 35, 41

[53] D. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic Networks*. Cambridge University Press, 2010. 6, 21, 22, 23, 25, 28, 46, 49

[54] S. Jackmann. Estimation and inference via Bayesian simulation: An introduction to Markov chain Monte Carlo. *American Journal of Political Science*, 44(2):pp. 375–404, 2000. ISSN 00925853. 12

[55] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In M. N. Munro, editor, *Mammalian protein metabolism*, volume III, pages 21–132. Academic Press, N. Y., 1969. 12

[56] I. Kanj, L. Nakhleh, C. Than, and G. Xia. Seeing the trees and their branches in the network is hard. *Theoretical Computer Science*, 401:153–164, 2008. 69

[57] G. Labelle, C. Lamathe, and P. Leroux. A classification of plane and planar 2-trees. *Theoretical Computing Science*, 307:337–363, 2008. 105

[58] C. L. Lawson and R. J. Hanson. *Solving least squares problems*. Prentice-Hall, 1974. 54

[59] J. Leigh, K. Schliep, P. Lopez, and E. Bapteste. Let them fall where they may: congruence analysis in massive phylogenetically messy data sets. *Molecular Biology and Evolution*, 28:2773–2785, 2011. 28

[60] A. Levy and L. Pachter. The neighbor-net algorithm. *Advances in Applied Mathematics*, 2007. 48, 58, 60, 61, 76

[61] C. Linder and L. H. Randal-Rieseberg. Reconstructing patterns of reticulate evolution in plants. *American Journal of Botany*, 91(10):1700–1708, 2004. 21

[62] M. Lysak, M. Koch, J. Beaulieu, A. Meister, and I. Leitch. The dynamic ups and downs of genome size evolution in *Brassicaceae*. *Molecular Biology and Evolution*, 26:85–98, 2009. 28

[63] J. Matoušek and B. Gärtner. *Understanding and using linear programming*. Springer, 2007. 33

[64] H. Meudt, P. Lockhart, and D. Bryant. Species delimitation and phylogeny of New Zealand plant species. *BMC Evolutionary Biology*, 9, 2009. 28

[65] D. Morrison. Phylogenetic networks in systematic biology (and elsewhere). *Research Advances in Systematic Biology*, 1:1–48, 2009. 2, 21, 109

[66] D. Morrison. Using data-display networks for exploratory data analysis in phylogenetic studies. *Molecular Biology and Evolution*, 27:1044–1057, 2010. 29

[67] D. Morrison, L. van Iersel, S. Kelk, and M. Charleston. Blog: The genealogical world of phylogenetic networks. http://phylonetworks.blogspot.co.uk. 110

[68] V. Moulton and A. Spillner. Optimal algorithms for computing edge weights in planar split networks. *Journal of Applied Mathematics and Computing*, 39:1–13, 2012. 73

[69] Y. Pauplin. Direct calculation of a tree length using a distance matrix. *Journal of Molecular Evolution*, 51(1):41–47, 2000. 15

[70] D. Penny and M. Hendy. Turbotree: a fast algorithm for minimal trees. *Computer Applications in the Biosciences*, 3:183–187, 1987. 68

[71] B. M. Pryor and D. M. Bigelow. Molecular characterization of embellisia and nimbya species and their relationship to alternaria, ulocladium and stemphylium. *Mycologia*, 95:1141–1154, 2003. xiv, 41, 43, 44

[72] B. M. Pryor and R. L. Gilbertson. Phylogenetic relationships among alternaria and related funghi based upon analysis of nuclear internal transcribed sequences and mitochondrial small subunit ribosomal DNA sequences. *Mycological Research*, 104:1312–1321, 2000. xiv, 41, 43, 44

[73] M. Ragan. Phylogenetic inference based on matrix representation of trees. *Molecular Phylogenetics and Evolution*, 1:53–58, 1992. 36

[74] D.F. Robinson. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory*, 11:105–119, 1971. 95

[75] S. Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(1):92–, January 2006. 12

[76] A. Rzhetsky and M. Nei. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular Biology and Evolution*, 10:1073–1095, 1993. 13, 68

[77] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4): 406–425, 1987. 16, 49

[78] A. Spillner V. Moulton S.Bastkowski, T. Wu. Trees in circular orderings. *In preparation*, 2013. 108

[79] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, 2003. 8, 20, 29, 68, 69

[80] C. Semple and M. Steel. Circular permutations and evolutionary trees. *Advances in Applied Mathematics*, 32:669–680, 2004. 88, 89

[81] R.R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958. 16

[82] A. Spillner, B. T. Nguyen, and V. Moulton. Constructing and drawing regular planar split networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(2):395–407, 2012. 111

[83] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992. 35

[84] D. L. Swofford. Paup*. phylogenetic analysis using parsimony (*and other methods). *Sinauer Associates, Sunderland, Massachusetts*, Version 4, 2003. 113

[85] K. Tamura, D. Peterson, N. Peterson, G. Stecher, M. Nei, and S. Kumar. Mega5: Molecular evolutionary genetics analysis using maximum likelihood, evolutionary distance, and maximum parsimony methods. *Molecular Biology and Evolution*, 28:2731–2739, 2011. 113

[86] E. Tepe, F. Farruggia, and L. Bohs. A 10-gene phylogeny of *Solanum* section *Herpystichum* (Solanaceae) and a comparison of phylogenetic methods. *American Journal of Botany*, 98:1356–1365, 2011. 38, 40

[87] L. van Iersel, C. Semple, and M. Steel. Locating a tree in a phylogenetic network. *Information Processing Letters*, 110:1037–1043, 2010. 69

[88] J. Whitfield, S. Cameron, D. Huson, and M. Steel. Filtered Z-closure supernetworks for extracting and visualizing recurrent signal from incongruent gene trees. *Systematic Biology*, 57:939–947, 2008. 26

[89] D. M. Williams and M. C. Ebach. Ernst Haeckel and Systematische Phylogenie. In *Foundations of Systematics and Biogeography*, pages 37–52. Springer US, 2008. ISBN 978-0-387-72728-8. 2

[90] Z. Yang and B. Rannala. Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte Carlo method. *Molecular Biology and Evolution*, 14(7):717–724, 1997. 11

[91] G. Yule. A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S. *Philosophical Transactions of the Royal Society of London. Series B*, 213:21–87, 1925. 35