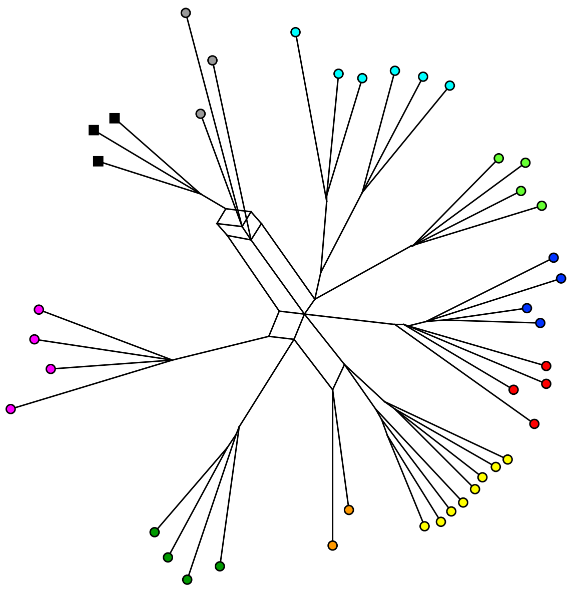
Flat Embeddings of Genetic and Distance Data

Monika Balvočiūtė



a thesis submitted for the degree of

Doctor of Philosophy

at the University of Otago, Dunedin,

New Zealand.

June 30, 2015

Bobutei

To my grandmother Teklė,
who always valued education
and reached for it as much as she could.

Abstract

The idea of displaying data in the plane is very attractive in many different fields of research. This thesis will focus on distance-based phylogenetics and multidimensional scaling (MDS). Both types of method can be viewed as a high-dimensional data reduction to pairwise distances and visualization of the data based on these distances. The difference between phylogenetics and multidimensional scaling is that the first one aims at finding a network or a tree structure that fits the distances, whereas MDS does not fix any structure and objects are simply placed in a low-dimensional space so that distances in the solution fit distances in the input as good as possible.

Chapter 1 provides an introduction to the phylogenetics and multidimensional scaling. Chapter 2 focuses on the theoretical background of flat split systems (planar split networks). We prove equivalences between flat split systems, planar split networks and loop-free acyclic oriented matroids of rank three. The latter is a convenient mathematical structure that we used to design the algorithm for computing planar split networks that is described in Chapter 3. We base our approach on the well established agglomerative algorithms Neighbor-Joining and Neighbor-Net. In Chapter 4 we introduce multidimensional scaling and propose a new method for computing MDS plots that is based on the agglomerative approach and spring embeddings. Chapter 5 presents several case studies that we use to compare both of our methods and some classical agglomerative approaches in the distance-based phylogenetics.

Acknowledgments

First of all I would like to thank my wonderful supervisors. Prof. Andreas Spillner, with whom I started my PhD, for introducing me to split networks and for being such an amazing teacher whose talent to explain difficult things in an easy way I will always admire; also for being such a caring and patient supervisor and for referring me to Prof. David Bryant at University of Otago in New Zealand to finish my PhD. David's passion for mathematics has been very inspiring and extremely motivating. I must thank him for his gentle push towards mathematics and for believing that I can make it; and most importantly for his help whenever I needed it and for proofreading my drafts probably more times than I could manage myself.

I would also like thank Prof. Vincent Moulton from for his help on publishing our paper on the FlatNJ method. Also I say thanks to the very friendly and always ready to help staff of the Maths and Stats Department at the University of Otago. Also, I would like to thank to Dr. Vilius Stakėnas from Vilnius University in Lithuania who helped me to start this PhD journey in the first place.

My work has been supported through a scholarship funded by the federal state of Mecklenburg-Western Pomerania in Germany, University of Otago scholarship and Allan Wilson Centre. I would like to thank Prof. David Bryant for supporting my participation in multiple conferences over the last two years.

Thanks to Sarah for making me feel like at home for the first few months in New Zealand, for all the coffee breaks since then and for sharing her family over the Christmas. Thanks to my friends in Lithuania for their support and long hours of "skyping" that helped with to cope with the homesickness. I must also thank my lovely flatmates for not minding my messiness for the last few months.

I would like to give my special thanks to my parents, also my sisters Ignė and Gabija, to Tetutė and to my cousin Dalia for their support and for believing in me. Especially to my mother for not objecting to me going to the other side of the world, I know it has been hard for you. And finally thanks to Rüdi for being "on the other side" of the screen whenever I needed you there.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Humans are visual creatures: Evolution has shaped our brains to"absorb, manipulate, and react to visual information in an increasingly effective way" (Balaram and Kaas, 2014). Naturally, visualization should be just as important in science as it is in every day life. Whenever someone asks me "So, what are you working on?", initially I try to come up with some easy definition, but only when I draw a picture of a phylogenetic tree or a network, I see that they get a rough idea. This is nicely summarized by Ottino (2003):"Seeing and representing are inextricably linked to understanding".

Advances in computer graphics in the past couple of decades have made the visualization of data easily accessible and more appealing. Emergence of so-called 'big data' made it even more important. A good visualization helps to see trends and patterns in the huge amounts of complex data. It should be clean, clear and intuitive. Unarguably visualization is also an art, but there is a lot of technical merit to it.

In this thesis we focus on two data visualization techniques that have a lot more in common than one might think. We started out with planar split networks, i.e. data-display networks for visualizing evolutionary relationships between sets of biological sequences. For the first two years of my PhD I worked together with Andreas Spillner in cooperation with Prof. Vincent Moulton from the University of East Anglia on a method for computing planar split networks (FlatNJ). The other two years I spent at the University of Otago working together with Prof. David Bryant. First we worked out theoretical aspects of planar split systems. Afterwards we decided to see whether a technique that has been widely used in phylogenetics would fit a somewhat different problem of a multidimensional scaling.

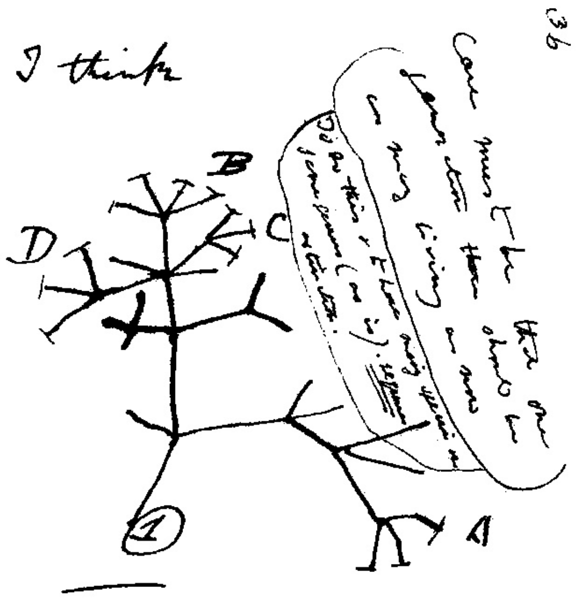


Figure 1.1: The first phylogenetic tree by Charles Darwin.

1.1 Phylogenetics

The origin of phylogenetics dates back to 1837 when Charles Darwin drew a sketch of a first phylogenetic tree (see Figure 1.1) in his *First notebook on transmutation of species*. However it was not until methods for sequencing genes and proteins became available that phylogenetics really started to take off and gain attention of the scientific community. Over the last few decades phylogenetics has become an integral part of evolutionary biology. More than half of the papers published in the latest issues of *Molecular Biology and Evolution* and *Systematic Biology* journals contain phylogenetic trees.

Biological sequences are comprised of bases or amino acids which from a mathematical point of view, are two different alphabets. DNA is coded by four bases: A, C, G and T whereas proteins are composed of twenty different amino acids. Phylogenies are mostly computed for the sequences of the same gene or corresponding DNA regions. To estimate a phylogeny from a set of sequences they must first be aligned so that homologous (evolutionary related) "letters" are placed in same columns. See Figure 1.2 for an example of a *multiple sequence alignment* and a phylogenetic tree computed from it. Differences between sequences arise due to the evolutionary processes shaping DNA sequences which then result in the changes in protein sequences too. These events include, for example, base substitutions when a single base is replaced by another base. Deletions and insertions are another class of events that change sequences even more drastically as they correspond to a few consecutive bases being deleted from or inserted to an existing sequence (Clark and Pazdernik, 2013).

(*a*)

```
s1   ATTTATAA--CTCGGTA--ACTGC
s2   ATAAATAA--CTC-----AACTGC
s3   ATTTA--AAACTCGCTA--ACTGA
s4   AATTATAA--GGCGGTATAACTGC
s5   ATTTATTC--CTCGGA--AACAGC
s6   A-T-AAAAAACTCGGTT--TC--T
s7   AATTATAAGGCTCGGTT--TC--T
```

(*b*)

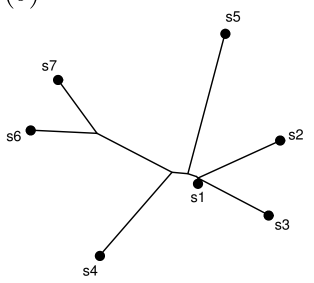


Figure 1.2: (a) A multiple sequence alignment of five DNA sequence fragments and (b) a corresponding phylogenetic tree. Gaps in the alignment do not belong to the DNA alphabet; they are inserted into sequences so that homologous bases could be placed in same columns. Gaps correspond to the deletion and insertion events as mentioned in the text.

One can easily see how these three events would result in a tree-like evolution. However, these are only a few possibilities. More complicated mutations can result in histories that can no longer be explained by trees. For example events such as horizontal gene transfer, i.e., DNA transfer from one existing species into another or recombination, i.e., exchange of parts of DNA between two organisms (Clark and Pazdernik, 2013) result in a reticulate evolution which can only be visualized with *phylogenetic networks* (Huson *et al.*, 2010). As trees are just a special case of networks, from now on whenever we talk about networks, we also mean trees unless stated otherwise.

Events shaping evolutionary histories do not happen completely at random as certain genes or even parts of individual genes are more susceptible to mutations than others. Hence, it is a common practice in phylogenetics to assume some model of evolution and then compute networks under the assumptions of the model. As a result, model based networks explicitly emphasize features of the model. For a good overview on phylogenetic networks please refer to Huson *et al.* (2010).

We focus on the methods for networks that do not assume any model and belong to a class of methods known as *distance-based* phylogenetics. Networks that we discuss in the following chapters are called data-display networks; they do not imply any specific scenario of evolution and instead display conflicting signals in data which can then be interpreted according to whichever hypothesis one has.

1.1.1 Trees and split networks

In distance-based phylogenetics, sequence alignments are reduced to pairwise distance matrices, where the distance between any two taxa is computed as, for example, the

Hamming distance between the two corresponding sequences. Following this approach, the problem of computing a phylogenetic tree for a set of sequences boils down to a very nice mathematical problem of reconstructing a tree from pairwise distances between its leaves.

Let $\mathcal{X}$ be a set of taxa, that is, a set of organisms (species) that we are interested in. A *phylogenetic tree* on $\mathcal{X}$ is a graph $T = (V_{in} \cup V_{ex}, E)$ with a set of external vertices V_{ex} labeled with elements in $\mathcal{X}$ and a set of internal vertices V_{in} that correspond to hypothetical ancestors and a set of edges E. The distance between two external nodes is proportional to the similarity between the corresponding sequences. The closer two taxa are on a tree, the more evolutionary related they are.

Removing some edge $e \in E$ from a tree results in splitting T into two connected components with non-overlapping sets of taxa $A \subset \mathcal{X}$ and $B \subset \mathcal{X}$ such that $A \cup B = \mathcal{X}$ and $A \cap B = \emptyset$. We call the bipartition of $\mathcal{X}$ induced by a branch e a *split* of $\mathcal{X}$ and denote it by $A|B$. A set of splits induced by the edges $e \in T$ gives a *split system* $\mathcal{S}(T)$ (Semple and Steel, 2003, p. 43).

Split networks are a generalization of phylogenetic trees, as they also display splits. Historically, split networks originated from a generalization of Buneman's construction of trees to splits (media) graphs by Barthelemy (1989). The theory on media graphs has been developed independently of split networks, see e.g., Eppstein *et al.* (2008).

A split network $\mathcal{N} = (V_l \cup V_{un}, \{E_i\})$ consists of a set of classes E_i of edges associated with some split i, a set V_l of labeled vertices corresponding to a set of taxa and a set V_{un} of unlabeled vertices that do not play a significant role in the split networks. Split networks are drawn in such a way that edges that belong to the same split class are always parallel and of the same length. Boxes or convex $2n$-gons ($n \in \mathbb{N}$) represent sets of conflicting splits that cannot be visualized by a single phylogenetic tree. The distance between a pair of taxa $a, b \in \mathcal{X}$ in a split network equals the length of a shortest path between a and b. Note that all shortest paths between some two nodes cross edges that belong to the same set of splits (Bryant and Moulton, 2004).

Within the class of split networks there are a few different types, see Figure 1.3. The first and most common type is phylogenetic trees. Circular or outer-planar split networks are closest to the trees in a sense that all labeled vertices are on the outside of the network. A more general case is planar split networks which allow for labeling of the internal vertices as well, but require for the network to be drawn in the plane. Note that all these types are "nested", that is $\{$phylogenetic trees$\} \subset \{$circular networks$\} \subset \{$planar networks$\} \subset \{$split networks$\}$.

See Huson and Bryant (2006) and Huson *et al.* (2010) for more details on split networks, their construction and interpretation.

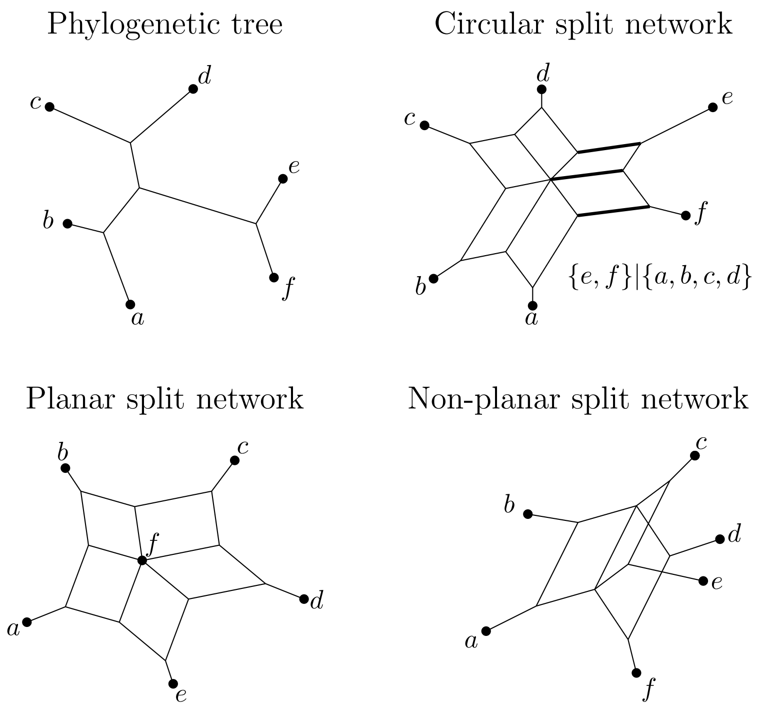


Figure 1.3: Different types of split networks. The class of bold edges in the circular network induces the split $\{e, f\}|\{a, b, c, d\}$.

1.1.2 Split systems

Let $\mathcal{N}$ be a split network on a set of taxa $\mathcal{X}$ and let $\mathcal{S}(\mathcal{N})$ be the set of splits on $\mathcal{X}$ visualized by $\mathcal{N}$. The set $\mathcal{S}(\mathcal{N})$ is a split system. Different types of networks induce different types of split systems with their specific characteristics. Below we review different kinds of split systems and their corresponding types of split networks.

By definition a split is a bipartition of a set of taxa $\mathcal{X}$, hence if we map $\mathcal{X}$ to a set of points in the plane, then we can draw splits as separating curves. To introduce different types of split systems, we use this kind of representation. In Chapter 2 we prove the equivalence of both representations.

Compatible splits

Probably the most common type of split systems are *compatible* split system. Let $S = A|B$ and $S' = A'|B'$ be two splits on $\mathcal{X}$. We say that S and S' are compatible if at least one of the four intersections $A \cap A'$, $A \cap B'$, $B \cap A'$ $B \cap B'$ is empty (Semple and Steel, 2003, Def. 3.1.3), see Figure 1.4. A split system $\mathcal{S}$ is compatible if all pairs of splits $S, S' \in \mathcal{S}$ are pairwise compatible. Such split systems can be visualized as phylogenetic trees (Buneman, 1971; Semple and Steel, 2003, Theorem 3.1.4).

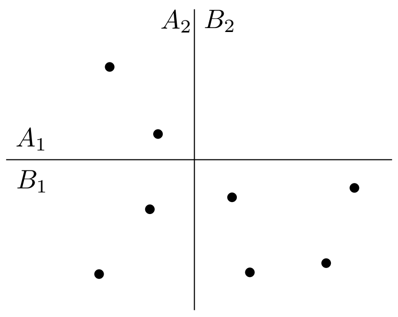


Figure 1.4: A set of points that correspond to elements in $\mathcal{X}$ and two lines inducing to two compatible splits $A_1|B_1$ and $A_2|B_2$ (intersection $A_1 \cap B_2 = \emptyset$).

Weakly compatible splits

Weakly compatible split systems were introduced by Bandelt and Dress (1992a) and have the following simple characterization due to Bandelt and Dress (1993). A split system $\mathcal{S}$ is weakly compatible if for any three splits $S_1 = A_1|B_1$, $S_2 = A_2|B_2$ and $S_3 = A_3|B_3$ in $\mathcal{S}$ at least one of the four intersections $A_1 \cap A_2 \cap A_3$, $B_1 \cap A_2 \cap B_3$, $B_1 \cap B_2 \cap A_3$ and $A_1 \cap B_2 \cap B_3$ is empty, see Figure 1.5. Note that these splits may be incompatible (Huson, 1998, p. 69).

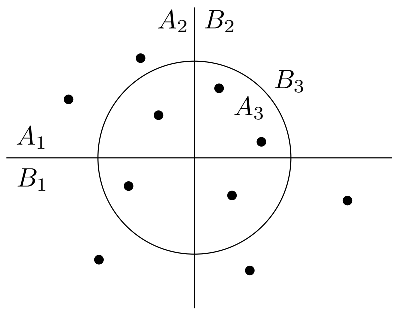


Figure 1.5: Three splits $A_1|B_1$, $A_2|B_2$ and $A_3|B_3$ that are weakly compatible as intersection $A_1 \cap B_2 \cap B_3$ is empty.

There are weakly compatible split systems which cannot be displayed in a planar split network (Bandelt and Dress, 1992a).

Circular split systems

Circular split systems are a special case of weakly compatible split systems (Huson, 1998, p. 69) that can be represented by planar outer labeled split networks, see Figure 1.6.

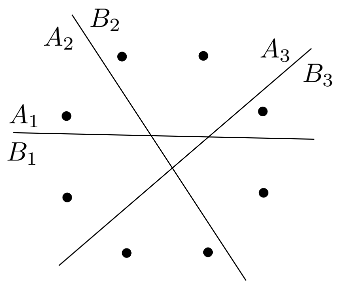


Figure 1.6: A circular split system. Three splits $A_1|B_1$, $A_2|B_2$ and $A_3|B_3$ are weakly compatible because for example $B_1 \cap B_2 \cap A_3$ is empty.

Planar splits

Bryant and Dress (2007) and Spillner *et al.* (2012) introduced flat split systems that are not constrained to be compatible or weakly compatible. Hence they constitute the most general type of split systems that can be visualized by a planar split network. Flat splits can be visualized as arrangements of curves in the plane such that any pair of curves intersect at most once, see Figure 1.7. See Section 2.1 for more details.

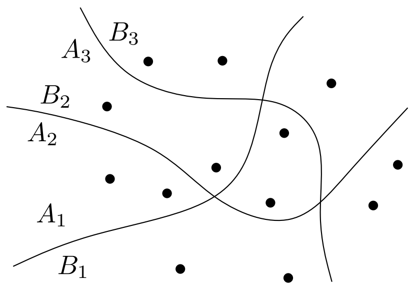


Figure 1.7: Flat split system. Splits are induced by an arrangement of curves such that any pair of curves intersect no more than once.

Affine splits

Affine splits are a special case of planar splits with splits induced by arrangements of *lines* instead of curves. For example, circular split systems are also affine (Bryant and Dress, 2007).

A Venn diagram summarizing relations between different kinds of split system is given in Figure 1.8.

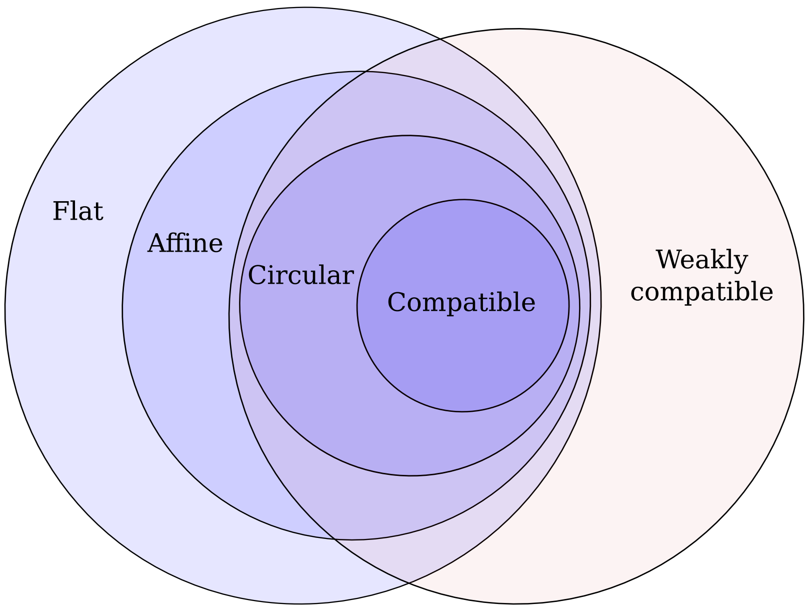


Figure 1.8: Relations between different types of split systems.

1.1.3 Methods for computing split networks from data

Here we review some of the key methods in distance-based phylogenetics for computing split networks.

Neighbor-Joining

Neighbor-Joining (NJ), by Saitou and Nei (1987), is the most frequently used tree reconstruction method in distance-based phylogenetics (Gascuel, 1997). To date it has been cited more than 31,500 times (Web of Science, June 2015). The NJ algorithm is simple to implement and a number of independent studies have proven its efficiency in recovering correct tree topologies from sequence data (Saitou and Nei, 1987; Nei, 1991; Charleston *et al.*, 1994; Kuhner and Felsenstein, 1994). As an agglomerative method for tree reconstruction, NJ falls into the same class of agglomerative algorithms as single linkage (Florek *et al.*, 1951; McQuitty, 1957; Sneath, 1957), average linkage or UPGMA (Sokal and Michener, 1958), and complete linkage (Sørensen, 1948), all of which are very well known methods for hierarchical clustering.

The basic idea behind agglomerative approaches is quite simple. Take a set of objects and a matrix of their pairwise distances. Assign all objects into individual clusters. Select two of the clusters and join them into one, reduce the distance matrix by replacing the selected two clusters with one that contains both. Recursively repeat the steps.

Neighbor-Joining is characterized by three formulas (Saitou and Nei, 1987):

1. The neighbor selection criterion,
2. The distance matrix reduction formula, and
3. The branch length computation formula.

The characterization of neighbors comes from the structure of a binary tree. Let $\mathcal{X}$ be a set of taxa and $T(\mathcal{X})$ an unrooted bifurcating tree such that all external vertices (leaves) of $T(\mathcal{X})$ are labeled with elements in $\mathcal{X}$. Then neighbors are defined as follows.

Definition 1.1.1. (Saitou and Nei, 1987) Two taxa $x', x'' \in \mathcal{X}$ are neighbors in $T(\mathcal{X})$ if they are connected through a single interior node.

Any bifurcating tree always has at least two pairs of neighbors (Saitou and Nei, 1987, p. 407). For example, tree (a) in Figure 1.9 has three pairs of neighbors: (x_1, x_2), (x_4, x_5) and (x_7, x_8). If we pick one pair, say (x_1, x_2), and join them together, we get a reduced tree T' as in Figure 1.9b. After agglomeration, the pair (x_1, x_2) in T' can be treated as a single element. Note that the topology of the tree is then defined by a series of agglomerations (Saitou and Nei, 1987).

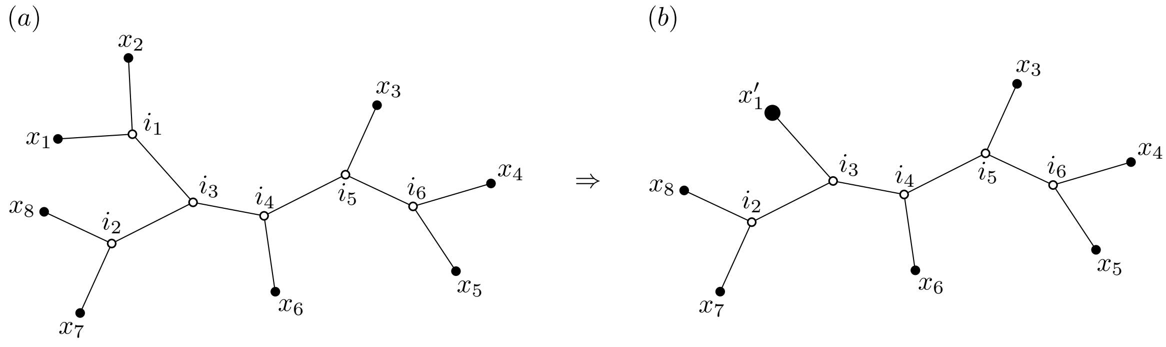


Figure 1.9: (a) A bifurcating tree with three pairs of neighbors: (x_1, x_2) joined via internal vertex i_1, (x_4, x_5) via i_6 and (x_7, x_8) via i_2. (b) A pair of neighbors (x_1, x_2) is agglomerated into one taxon x_1'.

The selection criterion is designed to identify neighbors from pairwise distances, namely the pair x' and x'' minimizing

$$\sigma_{nj}(x', x'') = (n-2)\delta_{x'x''} - \sum_y \delta_{x'y} - \sum_y \delta_{x''y}, \tag{1.1}$$

is identified as neighbors (Studier and Kepler, 1988). The Neighbor-Joining algorithm has been studied many times. Kumar and Gadagkar (2000); Eickmeyer *et al.* (2008) and Haws *et al.* (2011) examined its efficiency and optimality; Vach and Degens (1991); Charleston *et al.* (1993) and Bryant (2005b) showed the selection criterion used by NJ is in fact unique, DeBry (1992) showed that NJ is *consistent*, i.e., if the distance matrix comes from a tree, then the algorithm returns exactly the same tree.

Neighbor-Net

Neighbor-Net is a method by Bryant and Moulton (2004) for estimating phylogenetic networks as circular split systems. It is designed as a generalization of Neighbor-Joining from trees to planar outer-labeled split networks. Among all currently available tools for estimating phylogenetic networks Neighbor-Net is probably the most popular.

The definition of neighbors for a circular split network used by Neighbor-Net is a more general version of Definition 1.1.1. Let $\mathcal{N}_{\mathcal{C}}$ be a circular split network with a circular ordering of taxa $\mathcal{C}$ which indicates the order in which taxa appear around the network $\mathcal{N}_{\mathcal{C}}$. Neighbors are defined as follows:

Definition 1.1.2. Two taxa $x', x'' \in \mathcal{X}$ are neighbors in $\mathcal{N}_{\mathcal{C}}$ if x' and x'' are next to each other in the circular ordering $\mathcal{C}$.

For example pairs of neighbors in the circular split network in Figure 1.3 are (a, b), (b, c), (c, d), (d, e), (e, f) and (f, a).

Neighbor-Net constructs a circular ordering of taxa in a series of iterative neighbor identification and agglomeration procedures. At first each taxon x_i is assigned to its own cluster C_i as shown in Figure 1.10a. Then using the NJ selection Formula (1.1) applied to the average distances between clusters, two taxa are identified as neighbors and linked together (Figure 1.10b). Proximity between two clusters i and j equals

$$\delta(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} \delta_{xy}$$

Then as each cluster may contain two taxa, we must decide which elements from within the clusters must be joined. We do this by selecting $x_i \in C_i$ and $x_j \in C_j$, $i \neq j$ which minimize the following criterion:

$$\sigma_{nnet}(x_i, x_j) = (\hat{m} - 2)\delta_{x_i x_j} - \sum_{k=1, k \neq i}^{\hat{m}} \delta(x_i, C_k) - \sum_{k=1, k \neq j}^{\hat{m}} \delta(x_j, C_k)$$

Here $\hat{m} = m + |C_i| + |C_j| - 2$ and m is the current number of clusters. Agglomeration is delayed until more than two taxa are linked in a chain. Once some chain contains more than two, we agglomerate three or four taxa into two (Figure 1.10c). The ordering of the taxa that we get by joining C_i and C_j gives an ordering of the taxa contained in $C_i \cup C_j$ in the final ordering. The distance matrix is updated on the agglomeration. The procedure is repeated until there are only two (or three) clusters left as there is unique cyclic ordering of two (or three) elements. Agglomeration is then reversed and a circular ordering is obtained by expanding clusters in the reverse order as shown in Figure 1.11. Figure 2.6 on page 22 shows how splits are derived from a circular ordering.

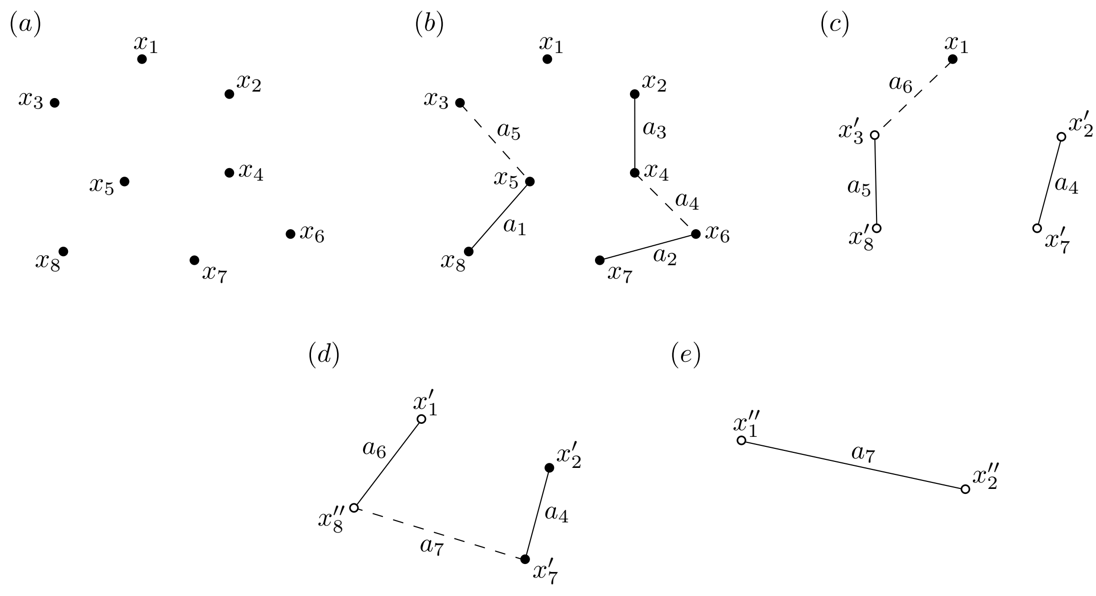


Figure 1.10: Agglomeration in Neighbor-Net. (a) A set of taxa $\{x_1, x_2, \ldots, x_8\}$. (b) Three pairs of taxa (x_2, x_4), (x_6, x_7) and (x_5, x_8) are joined (solid line segments), but not yet agglomerated. Joining (x_4, x_6) and (x_3, x_5) (dashed) forms chains of more than two taxa thus (c) it is followed by an agglomeration. (d),(e) Continuing agglomeration until two taxa are left.

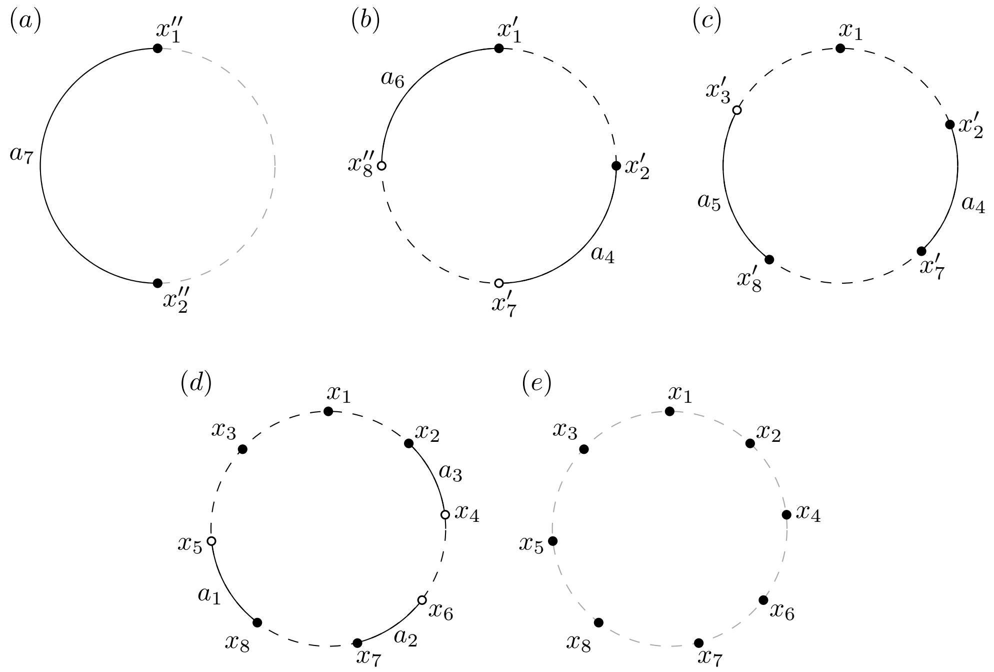


Figure 1.11: Reversing the agglomeration in Neighbor-Net. (a) Two taxa that were left after the last agglomeration in Figure 1.10 are placed on a circle. (b) – (d) Reversing agglomerations. (e) A circular ordering.

Bryant *et al.* (2007) proved that Neighbor-Net is consistent on circular distances, that is, if distances come from a circular split system, then the method returns exactly the same circular split system with phylogenetic trees being a special case of circular split systems.

Quartet-Net

Quartet-Net (QNet) by Grünewald *et al.* (2007) is another method that produces circular split networks. Like Neighbor-Net it is also based on an agglomerative approach, but instead of working on distances it takes *quartets* as input. A quartet is a split on a set of four taxa such that both sides of the split contain exactly two taxa. For any set of four taxa there are exactly three possible quartets. For example, if we have a set of four taxa $Y = \{a, b, c, d\}$, then the three possible quartets on Y are $\{a, b\}|\{c, d\}$, $\{a, c\}|\{b, d\}$ and $\{a, d\}|\{b, c\}$. If quartets are assigned non-negative weights they are called weighted. QNet uses weighted quartets to determine the agglomeration order (Grünewald *et al.*, 2007). Instead of using clusters, QNet assigns each taxon x_i to a path P_i and then joins paths to obtain the circular ordering. Unlike Neighbor-Net, it does not remove elements on agglomeration. QNet constructs the circular ordering by preserving as much of the initial quartet weight as possible. It uses the following criterion to decide which paths should be joined at each step:

$$\sigma_{qnet}(P_i, P_j) = \sum_{\substack{P_k \neq P_i, P_j \\ P_l \neq P_i, P_j}} \sum_{\substack{x_i \in P_i, x_j \in P_j \\ x_k \in P_k, x_l \in P_l}} \frac{w(\{x_i, x_j\}|\{x_k, x_l\})}{|P_i||P_j||P_k||P_l|}$$

here $w(\{x_i, x_j\}|\{x_k, x_l\})$ is the weight of a quartet $\{x_i, x_j\}|\{x_k, x_l\}$ and $|P_y|$ is a number of elements in the path P_y. Paths P_i and P_j are joined together by connecting their ends in a way that maximizes the represented quartet weight. Like Neighbor-Net, QNet is consistent on circular distances, more specifically, if quartets come from a circular split network or a tree, then QNet returns exactly the same network or a tree (Grünewald *et al.*, 2009).

Split Decomposition

Split Decomposition by Bandelt and Dress (1992b) was the first method for estimating split systems that do not have to be compatible. It estimates splits directly from the distance matrix $D = \{d_{ij}\}$. For any split $S = A|B$ they define the isolation index

$$\iota_{A|B} = \frac{1}{2} \min_{\substack{a', a'' \in A \\ b', b'' \in B}} (\max\{d_{a'a''} + d_{b'b''}, d_{a'b'} + d_{a''b''}, d_{a'b''} + d_{a''b'}\} - d_{a'a''} - d_{b'b''}).$$

They showed that the set $\{S : \iota_S \geq 0\}$ contains at most $\binom{n}{2}$ splits and could be computed in polynomial time. The algorithm is consistent on phylogenetic trees as the isolation index for some split $S = A|B$ given a tree-like distance matrix D equals the length of the branch that corresponds to S (Bandelt and Dress, 1992a).

1.2 Multidimensional scaling

Phylogenetic trees and networks may be the most popular way to visualize similarities between biological sequences, but they most surely are not the only ones. Multidimensional scaling has been used for visualizing data where spatial structure is of interest, especially in ecology. Figure 1.12b shows an MDS plot for a set of seven taxa based on the pairwise distances between sequences in the alignment (Figure 1.12a). Sequences 's2'–'s7' were generated as variations of 's1', thus no surprise that as an "ancestral" sequence, 's1' is placed in the middle of the plot.

(a)

```
s1   ATTTATAA--CTCGGTA--ACTGC
s2   ATAAATAA--CTC-----AACTGC
s3   ATTTA--AAACTCGCTA--ACTGA
s4   AATTATAA--GGCGGTATAACTGC
s5   ATTTATTC--CTCGGA--AACAGC
s6   A-T-AAAAAACTCGGTT--TC--T
s7   AATTATAAGGCTCGGTT--TC--T
```

(b)

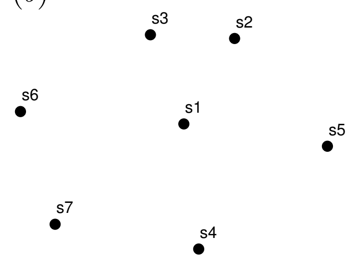


Figure 1.12: (a) A multiple sequence alignment as in Figure 1.2 and (b) its corresponding multidimensional scaling plot.

Multidimensional scaling is a technique that has been well established and has a much wider scope of applications than phylogenetic networks. In multidimensional scaling plots, data is displayed as points in the plane and their similarity is reflected by the Euclidean distances between the points.

There exist various methods for computing MDS plots from distance or vector data. Methods are often designed to emphasize certain features of the data, for example, neighborhoods and small distances or overall structure and large distances. They also differ by the methodology used. For example, classical scaling by Torgerson (1958) finds embeddings analytically, whereas force directed algorithms such as spring system embeddings (Chalmers, 1996; Morrison *et al.*, 2003) compute MDS plots iteratively. We provide a more detailed overview on multidimensional scaling and algorithms in Chapter 4. For a good review on MDS please refer to France and Carroll (2011).

Despite different backgrounds, distance-based phylogenetics and multidimensional scaling are very similar problems as both work with visualizing distance data. Since we focus on planar split networks, we will also mainly discuss MDS applications for scaling data onto two dimensions.

Trees (networks) and multidimensional scaling

When comparing split networks and MDS plots for biological data sets (Chapter 5) we noticed an interesting trend. Some data sets that fit very well on trees and networks were not so well visualized by MDS and vice versa. After looking into the literature we found that is not a new observation: Trees have been proposed as a multidimensional scaling tool before.

Gower (1967b) discussed representations of high-dimensional data using hierarchical trees. Cuadros *et al.* (2007) applied the Neighbor-Joining method to compute trees for documents based on their pairwise content similarity. They found that trees can reflect similarity relationships more accurately than multidimensional scaling plots. Later Paiva *et al.* (2011) used Neighbor-Joining trees for image data. Engel *et al.* (2011) designed a method for high-dimensional data visualization with structural decomposition trees. Gupta (2000) showed that trees with n leaves can be embedded into a d-dimensional spaces with $O(n^{1/(d-1)})$ distortion. Hence the higher the dimension the better the embedding of a tree.

We illustrate this with an example. Consider a so-called worst case scenario data for multidimensional scaling, that is vertices of an n-simplex with pairwise distances all equal. It is obvious that unless $n = 3$ is it not possible to find an exact embedding into two dimensions. Applying greedy force based algorithm as in Eades (1984) for a simplex with 20 vertices, we get a plot as shown in Figure 1.13a. We estimate the least squares fit as in SplitsTree (Formula 9 in Winkworth *et al.* (2005)) and got 85.794% which does not imply a good fit. Processing the same data set with the Neighbor-Joining algorithm we got a perfect 100% fit, see Figure 1.13b

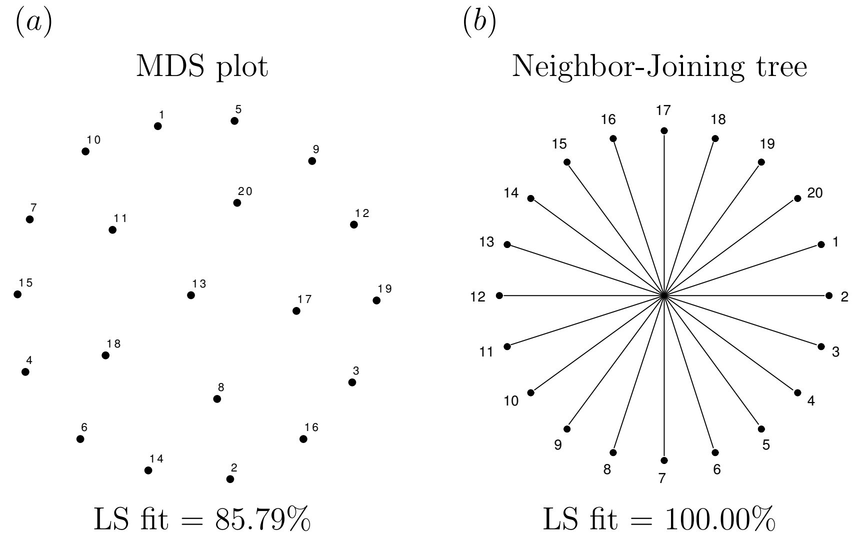


Figure 1.13: A multidimensional scaling plot and a Neighbor-Joining tree for the 20-simplex. LS fit corresponds to the least squares fit.

Chapter 2

Flat splits and related structures

In this chapter we discuss planar split networks (introduced in Section 2.2) and oriented matroids (Section 2.3) as two different representations of flat split systems (Section 2.1) and prove their equivalence. Intuitively, a split system is *flat* if we can map its elements to a set of points in the plane and the splits to an arrangement of curves splitting the points as shown in Figure 2.1a, see Definition 2.1.2. We show that exactly these split systems can be visualized as planar split networks (Figure 2.1c), and correspond to topes of loop-free, acyclic, rank three oriented matroids (Figure 2.1b).

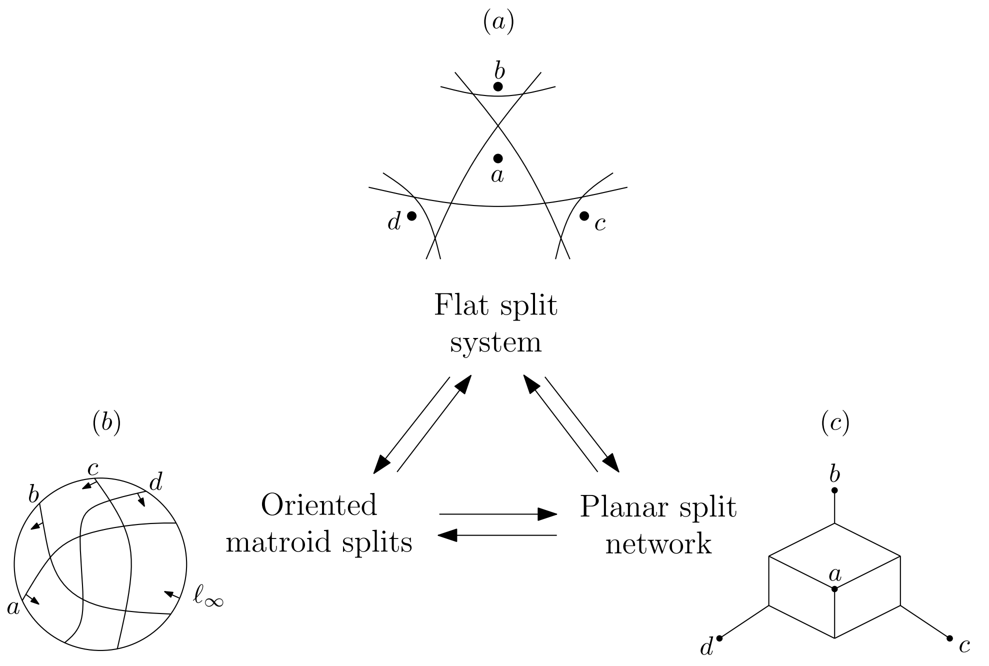


Figure 2.1: (a) A flat split system on four taxa $\{a, b, c, d\}$ and its representation as (b) an oriented matroid and, (c) a planar split network.

2.1 Flat splits

Let $\mathcal{A}$ be a collection of lines in the plane, and let X be a set of points in the plane not lying on any of these lines (Figure 2.2a). Each line $\ell \in \mathcal{A}$ partitions the plane, and therefore X, into at most two parts. The collection of *splits* (bipartitions) of X determined by the lines $\mathcal{A}$ clearly has a great deal of structure, and it is this structure which is of interest.

Actually we consider a slightly more general situation by allowing wobbly lines. A *pseudoline* is the image of a line under a homeomorphism of a plane, i.e., it is a simple curve in the plane that extends to infinity in both directions (Shor, 1991). A *pseudoline arrangement* is a finite collection of pseudolines with the property that each pair of pseudolines intersect in exactly one point, and when they do intersect, they cross. A *weak* arrangement of pseudolines is defined in the same way, except not every pair of pseudolines needs to intersect. Let X be a set of points in the plane not lying on any pseudoline in $\mathcal{A}$ (Figure 2.2b). Each pseudoline $\ell \in \mathcal{A}$ partitions X into at most two parts and, as in the straight line case, induces a split of X.

Definition 2.1.1. A *split* $A|B$ is a bipartition of X and a *split system* is a collection of splits of the same set. A split $A|B$ is *proper* if both A and B are non-empty.

Definition 2.1.2. (Bryant and Dress, 2007; Spillner *et al.*, 2012) A split system $\mathcal{S}$ is *flat* if it is induced by an arrangement of pseudolines in the plane. We say that $\mathcal{S}$ is *affine* if it is induced by a collection of straight lines.

Configurations of lines, pseudolines, and points arise in a wide variety of contexts, particularly in classification (Hastie *et al.*, 2009), oriented matroids (Björner *et al.*, 1999) and statistical learning theory (Hastie *et al.*, 2009). Our original interest in these structures followed from applications in evolutionary biology, as discussed in the following chapter.

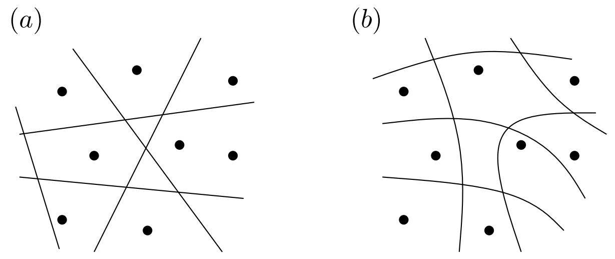


Figure 2.2: A set of points X and an arrangement of (a) lines and (b) pseudolines $\mathcal{A}$. None of the points in X appear on any of the (pseudo-)lines in $\mathcal{A}$. Each $\ell \in \mathcal{A}$ divides X into at most two parts inducing a split on X.

2.2 Network splits

Let T be phylogenetic tree, that is a labeled tree representing the evolutionary histories for a set $\mathcal{X}$ of species or individuals. Let φ be a map from taxa in $\mathcal{X}$ to vertices of the tree T, and let e be an edge of the tree. Then $T \setminus e$ consists of two connected components with corresponding vertex sets V_1 and V_2. Let $A = \varphi^{-1}(V_1)$ and $B = \varphi^{-1}(V_2)$, then we say that $A|B$ is a split that corresponds to the edge e in the tree T. The collection of splits associated with all edges from the tree T gives a split system $\mathcal{S}(T)$.

For many kinds of data, the signal in the data is more complex than can be faithfully represented by a single tree, so there has been much interest in phylogenies built using more general graphs, namely *phylogenetic networks* (Huson *et al.*, 2010). Explicit phylogenetic networks resemble trees with some additional edges indicating such evolutionary events as recombination or hybridization. Implicit networks, on the other hand are designed to show conflict in the data rather than make assumptions about the evolutionary history of the sequences in question (Huson and Bryant, 2006).

One of the more popular types of implicit phylogenetic networks are *split networks* (Huson and Bryant, 2006; Huson, 1998).

Definition 2.2.1 (Huson and Bryant, 2006)**.** Let $\mathcal{S}$ be a set of splits over a set of taxa $\mathcal{X}$. A split network $\mathcal{N}$ is a connected graph in which some of the nodes are labeled by $\mathcal{X}$ and all edges are labeled by $\mathcal{S}$, such that

(N1) Removing all edges associated with a given split $S \in \mathcal{S}$ divides $\mathcal{N}$ into two connected components, one part containing all taxa on one side of S and the other part containing all taxa on the other side.

(N2) The edges along any shortest path in $\mathcal{N}$ are all associated with different splits.

The underlying graph of a split network is a partial cube (Bryant, 2005a) as described by Wetzel (1995).

Figure 2.3a shows two trees that contain pairwise incompatible splits, i.e., $\{a, d\}|\{b, c\}$ and $\{a, b\}|\{c, d\}$, that cannot be shown on a single tree yet may be displayed by a single

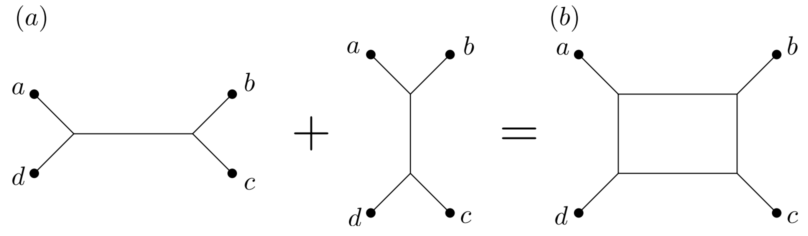


Figure 2.3: A pair of splits $\{a, d\}|\{b, c\}$ and $\{a, b\}|\{c, d\}$ that can (a) be displayed by at least two different phylogenetic trees but (b) may be contained in a single network.

split network (Figure 2.3b). For these networks, the underlying graph is a partial cube, that is, an isometric subgraph of a hypercube (Djoković, 1973), with some vertices labeled by elements in $\mathcal{X}$. Partial cubes have a rich mathematical structure. It can be shown that the edge set of a partial cube can be partitioned into classes such that (a) any shortest path contains at most one edge from each class, and (b) if a shortest path between two points contains an edge in some class then so does every path between those points. Removing all edges in a single class breaks the graph into two connected components (see Figure 2.4b). Each *edge class* $\varepsilon_i \in \Sigma$ induces a split $S_i \in \mathcal{S}$ of the label set $\mathcal{X}$. Splits arising in this way are said to be *induced splits* of the network $\mathcal{N}$.

A *drawing* of a split network is a straight line embedding of the graph into the plane so that edges in the same class are parallel and have the same length. A split network is *planar* if it has a drawing such that edges only intersect at their endpoints, each internal cell is strictly convex, and the external face contains at least one edge from each class (Figure 2.4a).

Definition 2.2.2. A collection of splits $\mathcal{S}$ has a *planar split network representation* if there exists a planar split network, partially labeled by X, which induces all the splits in $\mathcal{S}$.

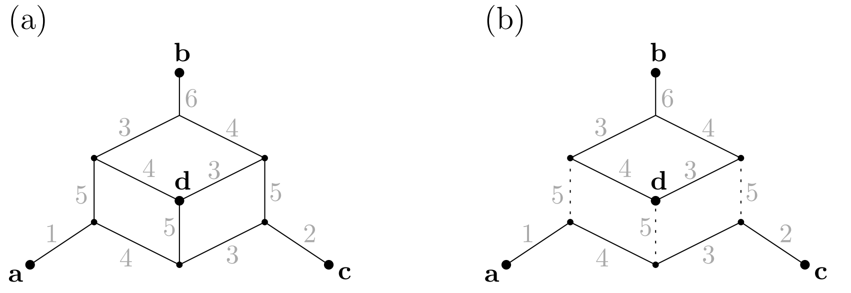


Figure 2.4: (a) An example of a planar split network. (b) If we remove edges of the split number five $ac|bd$, we get two connected components: one contains vertices labeled with $\{a, c\}$ and the other with $\{b, d\}$.

2.3 Oriented matroid splits

The third type of split collection we consider arises from oriented matroid theory. An *oriented matroid* is an abstract structure which mathematically can be used to represent point configurations over the reals, real hyperplane arrangements, convex polytopes and directed graphs (Björner *et al.*, 1999). Oriented matroids, like standard matroids, have a wide variety of different formulations and characterizations in various axiom systems. Excellent introductions to oriented matroids are found in Björner *et al.* (1999) and Richter-Gebert and Ziegler (2004).

Let E be a finite set. A *signed vector* Y is a map from E to $\{+,-,0\}$. Let Y^+, Y^0 and Y^- denote the positive, zero, and negative indices of the sign vector Y. Let Y_+ denote the vector with $Y^+ = E$ and Y_- the vector with $Y^- = E$.

For any two signed vectors V, U we define their composition $V \circ U$ by

$$V \circ U(x) = \begin{cases} V(x) & \text{if } V(x) \neq 0 \\ U(x) & \text{otherwise.} \end{cases}.$$

We say that V is a *restriction* of U if $V(x) \neq 0$ implies $V(x) = U(x)$. The *support* of U is a set $\underline{U} = \{x \in \mathcal{X} | U(x) \neq 0\}$.

Definition 2.3.1. A collection $\mathscr{T}$ of sign vectors constitute the set of topes of an oriented matroid if it satisfies the following tope axioms (Handa, 1990) :

(T0) $\mathscr{T} \neq \emptyset$

(T1) $T \in \mathscr{T} \Rightarrow -T \in \mathscr{T}$

(T2) If V is a restriction of some $T \in \mathscr{T}$ then either there is a $T' \in \mathscr{T}$ with $V \circ T' \in \mathscr{T}$ and $V \circ (-T') \notin \mathscr{T}$, or $V \circ T' \in \mathscr{T}$ for every $T' \in \mathscr{T}$.

Starting with the topes we can obtain other formulations of an oriented matroid. The maps Y such that $Y \circ T' \in \mathscr{T}$ for every $T' \in \mathscr{T}$ are called the *cocircuits* of the oriented matroid. An oriented matroid $\mathcal{M}$ can be defined by a set of elements E and a set of cocircuits that are sign vectors $\mathcal{C}^*$ satisfying cocircuit axioms given below.

Theorem 2.3.1. *(Richter-Gebert and Ziegler, 2004, Theorem 6.2.1) A collection $\mathcal{C}^*$ of sign vectors is the set of cocircuits of an oriented matroid $\mathcal{M}$ if and only if it satisfies:*

(C0) $0 \notin \mathcal{C}^*$,

(C1) $Y \in \mathcal{C}^* \Rightarrow -Y \in \mathcal{C}^*$

(C2) *For all $C, D \in \mathcal{C}^*$ we have: $\underline{C} \subseteq \underline{D} \Rightarrow C = D$ or $C = -D$*

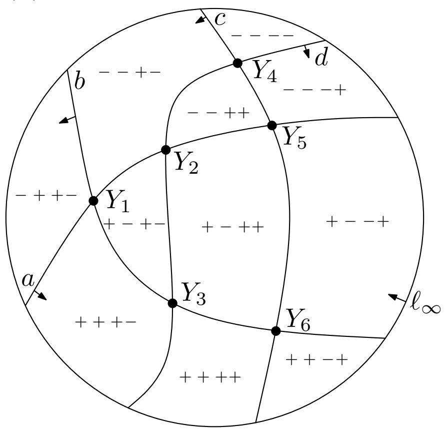
(C3) *$C, D \in \mathcal{C}^*$, $C \neq -D$, and $e \in S(C,D) \Rightarrow$ there is a $Z \in \mathcal{C}^*$ with $Z^+ \subset (C^+ \cup D^+) \setminus \{e\}$ and $Z^- \subset (C^- \cup D^-) \setminus \{e\}$.*

Here $S(C,D)$ is a separation set of the two sign vectors C and D :

$$S(C,D) = \{e \in E : C(e) = -D(e) \neq 0\}.$$

The set of cocircuits and topes is contained within a set of *covectors* of an oriented matroid. Formally, any composition of cocircuits forms a covector (Björner *et al.*, 1999, Definition 3.7.1).

(a)



⇒

(b)

Topes		Splits
$(a\,b\;c\,d)$		
$----$	⇒	~~$\vert abcd$~~
$---+$	⇒	$d\vert abc$
$--+-$	⇒	$c\vert abd$
$--++$	⇒	$cd\vert ab$
$+-+-$	⇒	$ac\vert bd$
$+-++$	⇒	$acd\vert b$
$+--+$	⇒	$ad\vert bc$

Figure 2.5: (a) Oriented matroid $\mathcal{M}$ as an arrangement of pseudo-lines in the projective plane with a bounding line at infinity; (b) A set of topes (negatives excluded) of $\mathcal{M}$ and its mapping to a split system $\mathcal{S}$. Tope '$----$' corresponds to a split that is not proper therefore $|abcd$ is not included in $\mathcal{S}$.

Theorem 2.3.2. *(Richter-Gebert and Ziegler, 2004, 6.2.1) A set $\mathcal{L}$ of sign vectors is a set of covectors of an oriented matroid if and only if it satisfies the covector axioms listed below :*

(CV0) $0 \in \mathcal{L}$,

(CV1) $C \in \mathcal{L} \Rightarrow -C \in \mathcal{L}$,

(CV2) $C, D \in \mathcal{L} \Rightarrow C \circ D \in \mathcal{L}$, *and*

(CV3) $C, D \in \mathcal{L}$, $e \in S(C, D) \Rightarrow$ *there is a* $Z \in \mathcal{L}$ *with* $Z_e = 0$ *and with* $Z_f = (C \circ D)_f$ *for* $f \in E \setminus S(C, D)$.

The *rank* r of the oriented matroid is the cardinality of the smallest subset $A \subseteq X$ which intersects the support of every cocircuit. An oriented matroid is uniform if all of its cocircuits have exactly $r - 1$ zero elements, i.e., $|Y^0| = r - 1$ for all $Y \in \mathcal{C}^*$ (Björner *et al.*, 1999).

As an example, let $\mathcal{M}$ be an oriented matroid of rank three on a set of elements $E = \{a, b, c, d\}$ with a set of cocircuits $\mathcal{C}^* = \{Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, -Y_1, -Y_2, -Y_3, -Y_4, -Y_5, -Y_6\}$ where $Y_1 = 00{+}{-}$, $Y_2 = 0{-}{+}0$, $Y_3 = {+}0{+}0$, $Y_4 = {-}{-}00$, $Y_5 = 0{-}0{+}$ and $Y_6 = {+}00{+}$. A visualization of $\mathcal{M}$ as an arrangement of pseudolines $\mathcal{A}$ in the projective plane with a bounding line at infinity is given in Figure 2.5a. Each open cell of $\mathcal{A}$ corresponds to a tope of $\mathcal{M}$. Signs in both cocircuits and topes are defined by the orientation of the elements in E.

Oriented matroids that arise from geometric situations are acyclic, that is they contain positive topes $T_+ \in \mathscr{T}$ (Björner *et al.*, 1999, Def. 3.4.7). In the following we will focus on acyclic oriented matroids that are also loop free, that is $T^0 = \emptyset$ for all $T \in \mathscr{T}$.

Definition 2.3.2. A set of splits $\mathcal{S}$ is *encoded by an oriented matroid of rank* 3 if there exists a set of topes $\mathscr{T}$ on X of a loop-free, acyclic, rank three oriented matroid such that for all $A|B \in \mathcal{S}$ there is $T \in \mathscr{T}$ such that $A = T^+$ and $B = T^-$.

Signs for the elements of each tope are assigned depending on the orientation of the corresponding lines, see Figure 2.5a. Topes (without their negatives) of the oriented matroid in Figure 2.5a are listed in Figure 2.5b.

Bryant and Dress (2007) have already briefly discussed collections of splits satisfying Definition 2.3.2, calling them *pseudo-affine*. Later Spillner *et al.* (2012) introduced the term *flat split systems* and defined them in terms of sequences of permutations.

Another structure that would be interesting to work with for exploring theory of oriented matroid splits is the Vapnik-Chervonenkis (VC) dimension which has been connected with oriented matroids by Gärtner and Welzl (1997).

2.4 Main theorem

Our main result is the equivalences summarized in Figure 2.1 on page 15.

Theorem 2.4.1. *Let $\mathcal{S}$ be a collection of splits of a finite set X. The following are equivalent:*

(i) $\mathcal{S}$ is flat, that is, $\mathcal{S}$ can be represented by an arrangement of pseudolines in the plane;

(ii) $\mathcal{S}$ has a planar split network representation;

(iii) $\mathcal{S}$ is encoded by a loop-free, acyclic oriented matroid of rank three.

The correspondence between flat split collections and collections from oriented matroids might appear to be a straight-forward application of the celebrated Topological Representation Theorem of Folkman and Lawrence (1978). However the representation we give is slightly different. Traditionally, the pseudolines in an arrangement correspond to the elements and the cells correspond to topes. The representation we describe has pseudolines corresponding to topes and points, lying in the cells, corresponding to elements.

Split networks have been studied less than oriented matroids, though methods for constructing split networks have been cited thousands of times. A connection between some classes of planar split networks and line arrangements was established by Wetzel (1995). He considered affine collections of splits where the set X of points formed the vertices of a convex polygon. These collections are called *circular*, and can be

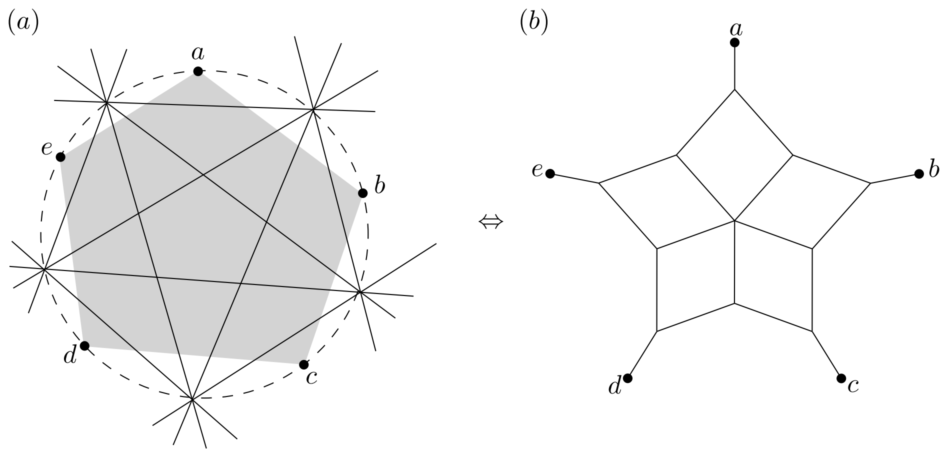


Figure 2.6: (a) An affine split system $\mathcal{S}$ on points $\{a, b, c, d, e\}$ which form a set of vertices of the convex pentagon (gray). (b) A planar outer labeled (circular) split network on $\mathcal{S}$.

characterized by the existence of an ordering $x_1, x_2, ..., x_n$ of X with the property that every split in $\mathcal{S}$ has the form

$$\{x_i, ...x_{j-1}\}|X - \{x_i, ..., x_{j-1}\}$$

for some $i < j$, see Figure 2.6a and Section 1.1.2. Later, Dress and Huson (2004) used De-Bruijn duality to prove that a collection of splits is circular if and only if it has a planar split network representation where the vertices labeled by X all lie on the external face (they are *planar outer labeled*), as shown in Figure 2.6. Neighbor-Net (Bryant and Moulton, 2004) uses this fact to produce planar split network representations of distance data.

There are many applications where it makes sense to construct split networks where some of the *internal* vertices are also allowed to be labeled. These vertices might represent ancestral species, or spatially distributed samples. Therefore it is natural to characterize which collections of splits may be represented in this way, circular collections being a special case.

Spillner *et al.* (2012) made significant progress in that direction. They started with the concept of (simple) *allowable sequences* of permutations, as introduced by Goodman and Pollack (1980, 1982), and showed that collections of splits generated from these sequences could be represented using a planar split network. The authors stated that these split collections were equivalent to those derived from pseudoline arrangements or oriented matroids, but did not provide a proof.

2.5 Proof of the main theorem

The first step is to prove the equivalence of flat split systems and split systems from oriented matroids.

Lemma 2.5.1. *Let $\mathcal{S}$ be a split system encoded by a loop-free, acyclic, rank three oriented matroid with element set $\mathcal{X}$. Then $\mathcal{S}$ is a flat split system on $\mathcal{X}$.*

Before providing a proof for Lemma 2.5.1, we introduce two different graphical representations which exist for all rank three oriented matroids (Björner *et al.*, 1999, def. 5.3.4):

Type I Arrangement of pseudolines in a projective plane.

Type II Pseudoconfiguration of points.

See Figure 2.7 for a type I and a type II representation of an oriented matroid $\mathcal{M}$. Pseudolines in the type I representation correspond to the elements in E with arrows indicating orientation of each element in $\mathcal{M}$. The sign vector of a cell in the type I representation is determined by the orientation of the pseudolines (elements). That is, if some pseudoline is oriented towards some cell, then the corresponding sign in the covector for that cell is '+' and '$-$' otherwise; in case a pseudoline passes through a point that corresponds to a covector Y, the respective sign of the covector is '0'.

Open cells of the arrangement give topes and line intersections (numbered) give cocircuits. For each tope $T \in \mathscr{T}$ at least one of T and $-T$ corresponds to a cell in the type I representation. Note that only topes that correspond to open cells bounding the line at infinity ℓ_∞ have their negatives present in the type I representation. A similar condition is valid for the cocircuits, that is, for each cocircuit $Y \in \mathcal{C}^*$ either Y or $-Y$ is present in the type I representation.

In the type II representation of $\mathcal{M}$ the role of elements and cocircuits is reversed, i.e., cocircuits form an arrangement of pseudolines and elements correspond to the labeled points of intersection. In the type II representation each of the cocircuits is assigned an orientation. As with the type I representation, for each cocircuit either Y or $-Y$ is present in the type II representation. Signs of each cocircuit Y that is present in the type II representation are then determined by the relative position of each element $e_i \in E$:

- $Y_i = +$ if the point corresponding to the element e_i is on the positive side of the pseudoline Y,
- $Y_i = -$ if the same point is on the negative side, and
- $Y_i = 0$ if the point is on the pseudoline Y.

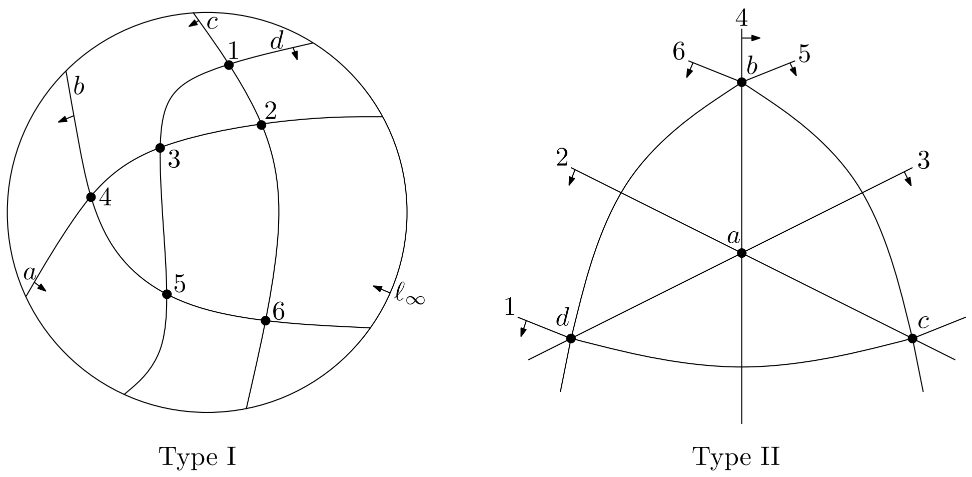


Figure 2.7: Type I and type II representations of an oriented matroid $\mathcal{M}$ with element set $E = \{a, b, c, d\}$ and cocircuits $\mathcal{C}^* = \{1, 2, 3, 4, 5, 6\}$.

The existence of a type I representation is a direct result of the Folkman and Lawrence (1978) topological representation theorem reformulated for the oriented matroids of rank three (Björner *et al.*, 1999, Thm. 6.2.3). The existence of the type II representation for the rank three follows from the existence of the oriented adjoint (Goodman, 1980; Björner *et al.*, 1999, Theorem 5.3.6). The map from the type I to the type II representation converts elements of $\mathcal{M}$ as pseudolines to elements as points and cocircuits as points to cocircuits as pseudolines. The map gives no representation for the topes of the original oriented matroid $\mathcal{M}$. Hence, to prove Lemma 2.5.1 we augment the oriented matroid $\mathcal{M}$ to a loop-free, acyclic, rank three oriented matroid $\mathcal{M}_T$ such that topes of $\mathcal{M}$ are mapped to cocircuits of $\mathcal{M}_T$.

Proof of Lemma 2.5.1. Let $\mathcal{S}$ be a split system encoded by a loop-free, acyclic, rank three oriented matroid $\mathcal{M}$ on $E = \mathcal{X}$ with tope set $\mathscr{T}$. Construct a type I representation $\mathcal{A}$ of $\mathcal{M}$ and without loss of generality assume that T_+ corresponds to a cell of $\mathcal{A}$ (Figure 2.8a). Let $\mathscr{T}' = \{T_S : S \in \mathcal{S}\}$ be a subset of topes $\mathscr{T}$ of $\mathcal{M}$ such that for each proper split $S = A|B$ there is exactly one tope $T_S \in \mathscr{T}'$ that induces S and corresponds to a cell in $\mathcal{A}$. In case there is a choice between two cells giving topes that induce the same split S, we choose one at random.

Let $P_T = \{p_S : S \in \mathcal{S}\}$ be a set of points such that $p_S \in P_T$ is any point in the interior of the cell corresponding to T_S. Let p_+ be some point in the cell associated with T_+. Add a set of oriented pseudolines $\{\ell_P\}$ to $\mathcal{A}$ so that at least two of them pass through each point $p \in P_T$ and all pseudolines are oriented towards p_+, so that the oriented matroid remains acyclic (Figure 2.8b). Let $\mathcal{M}_T$ denote the extended oriented matroid which has this type I representation. As any intersection of two or more pseudolines corresponds to a cocircuit of $\mathcal{M}_T$ we get a bijection from P_T to a subset

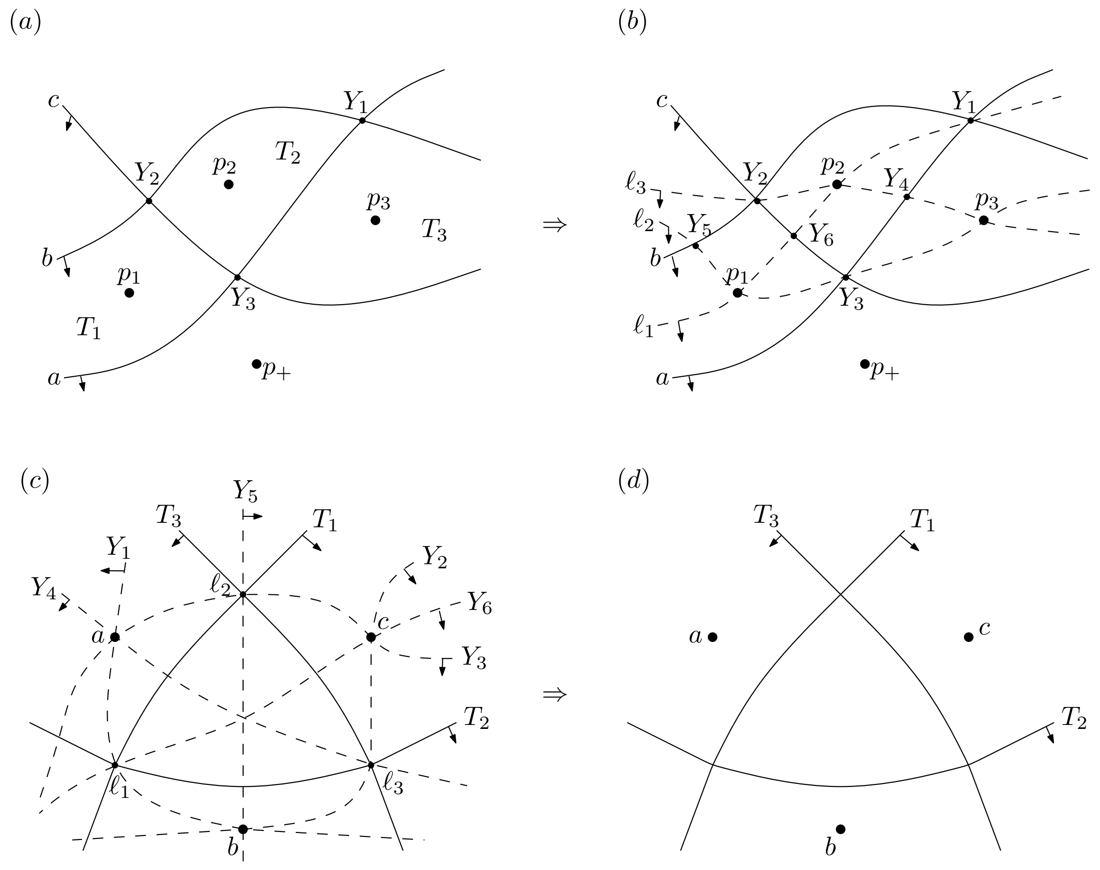


Figure 2.8: (a) An oriented matroid $\mathcal{M}$ on a set of elements $E = \{a, b, c\}$ with cocircuits $\mathcal{C}^* = \{Y_1, Y_2, Y_3\}$ inducing a split system $\mathcal{S} = \{a|bc, b|ac, c|ab\}$, then $\mathscr{T}' = \{T_1, T_2, T_3\}$. (b) An extended oriented matroid $\mathcal{M}_T$ on $E_T = E \cup \{\ell_1, \ell_2, \ell_3\}$ with $\mathcal{C}^*_T = \mathcal{C}^* \cup \{Y4, Y5\} \cup \mathcal{C}^*_P$ where $\mathscr{T}' \to \mathcal{C}^*_P$. (c) An adjoint $\mathcal{M}^{ad}_T$ of $\mathcal{M}_T$. (d) $\mathcal{M}^{ad}_T$ restricted to $\mathcal{C}^*_P$ and with points that are labeled with elements in E, i.e. a flat split system induced by the splits of the oriented matroid $\mathcal{M}$.

$\mathcal{C}^*_P \subseteq \mathcal{C}^*_T$ of cocircuits of $\mathcal{M}_T$.

Consequently we get a bijective map ϕ from topes in $\mathscr{T}'$ to cocircuits $\mathcal{C}^*_P$ and each tope $T \in \mathscr{T}'$ is the restriction of $\phi(T) \in \mathcal{C}^*_P$ to $\mathcal{X}$.

Take a type II representation of $\mathcal{M}_T$ (Figure 2.8c). Remove all pseudolines that correspond to $\mathcal{C}^*_T \backslash \mathcal{C}^*_P$. This way we eliminate all pseudolines that correspond to covectors that are not topes of $\mathcal{M}$. Points that correspond to elements in $\mathcal{X}$ lay on the intersections of pseudolines that we delete, hence to keep elements in our representation, we leave them as simple points in the plane. The remaining arrangement of pseudolines $\mathcal{C}^*_P$ (topes of $\mathcal{M}$) together with $\mathcal{X}$, then induce $\mathcal{S}$, which is therefore flat (Figure 2.8d). $\square$

Lemma 2.5.2. *Let $\mathcal{S}$ be a flat split system on $\mathcal{X}$. Then $\mathcal{S}$ is encoded by a loop-free, acyclic, rank three oriented matroid with element set $\mathcal{X}$.*

Proof. Suppose that $\mathcal{S}$ is a flat split system, induced by a set of pseudolines $\mathcal{A}$ separating points labeled by $\mathcal{X}$ in the plane. By embedding the arrangement in the projective plane we can repeatedly apply Levi's enlargement lemma (Levi, 1926; Björner *et al.*, 1999, Prop. 6.3.4) until every point labeled by $\mathcal{X}$ lies on at least two pseudolines. Orient these lines arbitrarily, making sure that all of them point towards one open cell and let $\mathcal{M}$ be the corresponding loop-free, acyclic, rank three oriented matroid. Let $\mathcal{M}^{ad}$ be an adjoint of $\mathcal{M}$.

Every $x \in \mathcal{X}$ corresponds to a cocircuit of $\mathcal{M}$ and therefore an element e_x of $\mathcal{M}^{ad}$. Furthermore, for every signed pseudoline $\ell \in \mathcal{A}$ there is a cocircuit Y_ℓ^{ad} of $\mathcal{M}^{ad}$ such that

$$Y_\ell^{ad}(e_x) = \ell(x).$$

We now restrict $\mathcal{M}^{ad}$ to elements $\{e_x : x \in \mathcal{X}\}$, i.e., we keep all elements labeled with $\mathcal{X}$ and remove others. Pseudolines in $\mathcal{A}$ that correspond to splits in $\mathcal{S}$ do not intersect any of the points labeled with $\mathcal{X}$. Thus in $\mathcal{M}^{ad}$ all $\ell \in \mathcal{A}$ become points that are not intersected by pseudolines in $\{e_x : x \in \mathcal{X}\}$, hence they correspond to points in open cells, that correspond to topes of $\mathcal{M}^{ad}$. Therefore we get that for each $\ell \in \mathcal{A}$ the cocircuit Y_ℓ^{ad} restricts to a tope T which induces the same split of $\{e_x : x \in \mathcal{X}\}$ as ℓ does of the points labeled by $\mathcal{X}$. □

We now prove the equivalence between flat splits and collections which can be represented using a planar split network. Going from flat splits to partial cubes is straight-forward: the dual of a pseudoline arrangement is a partial cube. What is more difficult is demonstrating that this graph has a straight-line embedding in the plane where edges in the same class have the same length and are parallel. For this we apply the celebrated *Bohne-Dress* theorem (Bohne, 1992), which links zonotopal tilings and oriented matroids. Our presentation draws heavily on Richter-Gebert and Ziegler (1994).

Lemma 2.5.3. *Let $\mathcal{A}$ be an arrangement of pseudolines and X a set of points in the plane. Then $\mathcal{A}$ can be extended by a line g such that all points in $\mathcal{A} \cup X$ are on the same side of g.*

Proof. Suppose that X is a set of points in the plane, let $\mathcal{A}$ be an arrangement of pseudolines and let A be a set of points induced by $\mathcal{A}$. As a direct result of the sweeping lemma (Felsner and Weil (2001), Lemma 1; Snoeyink and Hershberger (1989), Theorem 3.1) we can add a pseudoline g such that all points in A are on the one side of g, see Figure 2.9a. What we still need to show is that g can be modified so that all

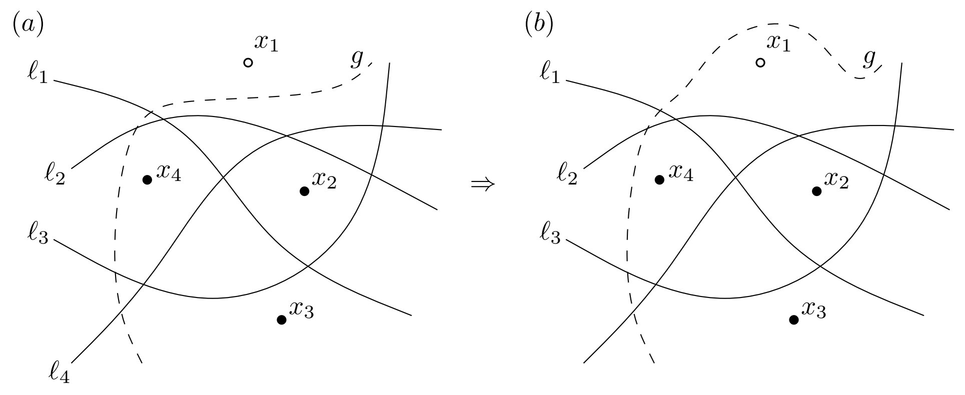


Figure 2.9: (a) Extending an arrangement of pseudolines $\mathcal{A} = \{\ell_1, \ell_2, \ell_3, \ell_4\}$ with a pseudoline g such that all points in $\mathcal{A}$ are on the one side of g. Point x_1 that does not belong to $\mathcal{A}$ is on the opposite side of g. (b) Modifying g so that all points in $X = \{x_1, x_2, x_3, x_4\}$ are on the same side of g as points in $\mathcal{A}$.

points in X are on the same side as points in A. Let x be some point in X that is on the opposite side of g and is closer to g than any other such point. As all points in A are on the same side of g, we get that g can pass through unbounded cells of $\mathcal{A}$ only. Hence x must be contained in an unbounded cell; additionally it must be one of the cells that g passes through. Then we can locally perturb g to go over the point x, see Figure 2.9b. □

Lemma 2.5.4. *Let $\mathcal{S}$ be a split system on X. If $\mathcal{S}$ is flat then $\mathcal{S}$ can be represented using a planar split network.*

Proof. Suppose that X is a set of points in the plane and let $\mathcal{A}$ be an arrangement of pseudolines which induces the collection of splits $\mathcal{S}$. Let g be an auxiliary pseudoline that we add to $\mathcal{A}$ using Lemma 2.5.3. Orient g towards the points in $\mathcal{A} \cup X$. Let p_+ be some point in the open cell of $\mathcal{A}$ where g goes to infinity and orient all pseudolines in $\mathcal{A}$ towards p_+. We obtain a loop-free, acyclic, rank three oriented matroid $\mathcal{M} = \mathcal{M}(\mathcal{A} \cup g)$ with the type I representation as described in the beginning of this chapter (see p. 23). Elements of $\mathcal{M}$ correspond to the splits in $\mathcal{S}$ with g representing an improper split. Let $\widehat{\mathcal{L}}$ be the set of covectors of the resulting oriented matroid.

Let $\ell_1, \ldots, \ell_n$ be an ordering of the lines in $\mathcal{A}$ given by their points of intersection with g, ties broken arbitrarily. Select n vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ in $\Re^2$ with increasing slopes as shown in Figure 2.10b. These constitute a realization of the contraction

$$\widehat{\mathcal{L}}/g = \Big\{ Y \in \{0, +, -\}^X : (Y, 0) \in \widehat{\mathcal{L}} \Big\}.$$

A *zonotope* is a polytope which is also a projection of a regular cube (Björner *et al.*, 1999, p. 51). The zonotope $\mathcal{Z}(\mathcal{M})$ of an oriented matroid $\mathcal{M} = (E, \mathcal{C}^*)$ is a Minkowski sum of vectors $v \in \mathbb{R}^2$ associated with elements in E (Richter-Gebert and Ziegler, 1994, def. 1.1):

$$Z(\mathcal{M}) = \sum_{i=1}^{n} [-v_i, +v_i].$$

Zonotopes are associated with sign vectors $Y \in \mathcal{L}$ by

$$Z_Y = \sum_{i \in Y^0} [-v_i, +v_i] + \sum_{i \in Y^+} v_i - \sum_{i \in Y^-} v_i,$$

see Richter-Gebert and Ziegler (1994).

Let $\mathcal{L} = \widehat{\mathcal{L}} \setminus g$ be the set of covectors after the deletion of g. The split network for $\mathcal{L}$ is now constructed as follows (see Figure 2.10c):

- The vertices correspond to topes Y of $\mathcal{L}$ such that $(Y, +) \in \widehat{\mathcal{L}}$, with coordinates

  $$Z_Y = \sum_{i \in Y^+} \mathbf{v}_i - \sum_{i \in Y^-} \mathbf{v}_i.$$

  Each point in X is mapped to the vertex corresponding to the tope of the arrangement it is contained in.
- The edges correspond to covectors Y of $\mathcal{L}$ such that $(Y, +) \in \widehat{\mathcal{L}}$ and Y^0 contains a single element. These correspond to the line segment given by the Minkowski sum

  $$Z_Y = \sum_{i \in Y^+} \mathbf{v}_i - \sum_{i \in Y^-} \mathbf{v}_i + \sum_{i \in Y^0} [-\mathbf{v}_i, \mathbf{v}_i].$$

  The zero element of these vectors Y corresponds to a pseudoline of $\mathcal{A}$, and hence a split of $\mathcal{S}$. Edges corresponding to the same split induced by the pseudoline ℓ_i have the same direction and length as they are assigned the same vector $\mathbf{v}_i$.
- The cells correspond to cocircuits Y of $\mathcal{L}$ such that $(Y, +) \in \widehat{\mathcal{L}}$ and $|Y^0| > 1$. The position of each cell is given by the same Minkowski sum

  $$Z_Y = \sum_{i \in Y^+} \mathbf{v}_i - \sum_{i \in Y^-} \mathbf{v}_i + \sum_{i \in Y^0} [-\mathbf{v}_i, \mathbf{v}_i].$$

This graph is the affine projection of a partial cube Fukuda and Handa (1993). It forms a tiling, and hence planar embedding by Theorem 4.2 of Bohne (1992) (see also Theorem 2.1 of Richter-Gebert and Ziegler (1994)). □

Eppstein (2005) provides a different presentation of similar ideas when proving that the region graph of an arrangement of pseudolines has a face-symmetric planar drawing, though key steps of the proof were omitted. A relationship between marked

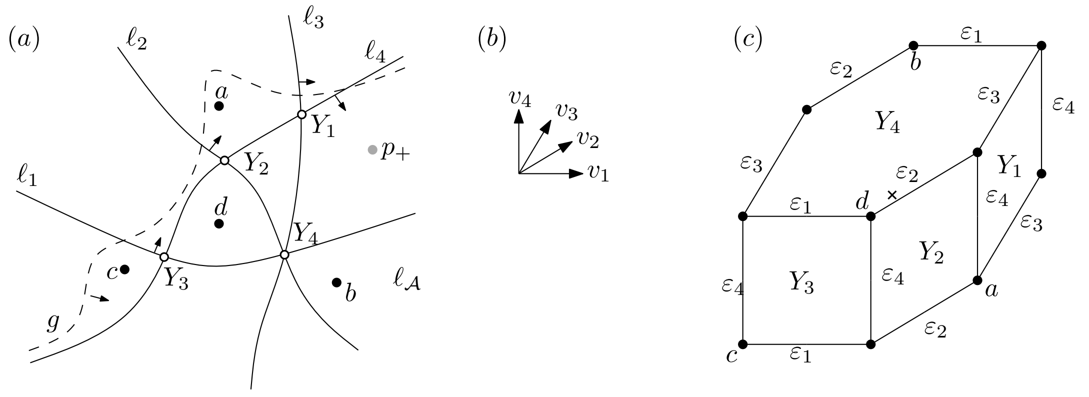


Figure 2.10: (a) An arrangement of pseudolines $\mathcal{A} = \{\ell_1, \ell_2, \ell_3, \ell_4\}$ and a set of points $\{a, b, c, d\}$. Pseudoline g (dashed) is added to the $\mathcal{A}$ using Lemma 2.5.3 and oriented so that all points in $\mathcal{A}$ and X are on the positive side of g. Pseudolines in $\mathcal{A}$ are oriented towards the dummy point p_+ and indexed in the order in which they intersect g. (b) Elements in $\mathcal{A}$ are assigned vectors $\{v_1, v_2, v_3, v_4\}$ with increasing slopes. (c) A zonotope of $\mathcal{A}$ as described above. Cross indicates the origin.

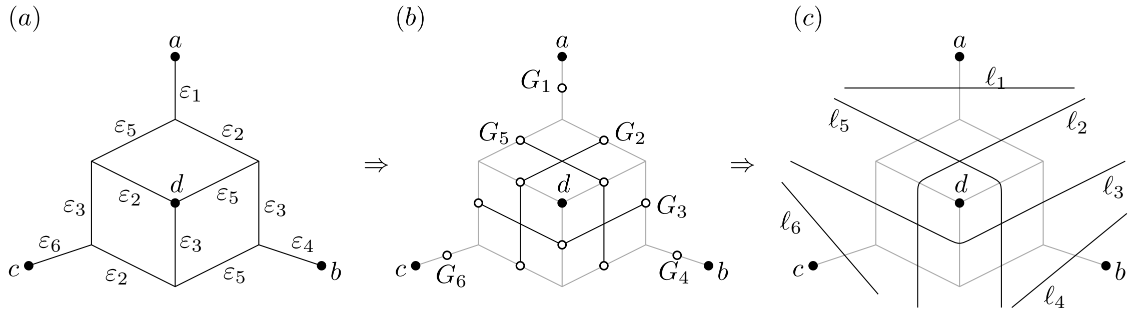


Figure 2.11: (a) A planar split network $\mathcal{N}$ on a set of taxa $\{a, b, c, d\}$ and a set of splits $\mathcal{S} = \{1, 2, 3, 4, 5, 6\}$, (b) a set of G graphs of $\mathcal{N}$, and (c) splits of $\mathcal{N}$ in the plane.

arrangements of pseudolines and marked zonotopal tilings was also proven by Felsner and Weil (2001). They proved this connection via a bijection between marked arrangements of pseudolines and allowable sequences and a bijection between allowable sequences and marked zonotopal tilings.

Lemma 2.5.5. *If $\mathcal{S}$ has a planar split network representation then $\mathcal{S}$ is flat.*

Proof. The first step of the proof is to show that if an internal face has edge e on its boundary then it has exactly one more boundary edge in the same class. Let C be the boundary of the internal face and let $e \in C$. From Lemma 2.3 of Klavzar and Mulder (2002), C contains at least one other edge in the same edge class as e, and since all the edges in this class are parallel and non-adjacent there can be at most two on the boundary of any convex region. Hence C contains zero or two edges from each edge class and any two edges from the same class will be on opposite sides of the cycle.

The second step is to use this observation to construct a collection of pseudolines from the network. For each edge class $\mathcal{E}_i$, construct a graph G_i consisting of the midpoints $\{v_e : e \in \mathcal{E}_i\}$ with edges between midpoints which lie on the same internal face (Figure 2.11b). There are at most two vertices in this graph with degree less than two; these correspond to the two edges in $\mathcal{E}_i$ lying on the external face. Furthermore G_i is connected, since otherwise removing the edges in $\mathcal{E}_i$ from the network would partition it into more than two components. It follows that G_i is a single path, terminating in midpoints on the external face of the network.

We construct a pseudoline ℓ_i by taking the path determined by G_i and extending the line to infinity in both directions in such a way that there are no new intersection points within the external face of the network. By construction, the lines that we get in this way induce exactly those splits represented by the network, see Figure 2.11c.

The third step of the proof is to show that any two pseudolines in this collection intersect at most once.

Let $\mathcal{E}_i$ and $\mathcal{E}_j$ be two edge classes associated with splits S_i and S_j respectively. Recall from Definition 2.2.1 that removing edges associated with one split divides the split network $\mathcal{N}$ into two connected components which themselves are split networks. Hence, removing edges associated with two splits, i.e., all edges $e \in \mathcal{E}_i \cup \mathcal{E}_i$ partitions $\mathcal{N}$ into at most four connected components. Respectively, pseudolines ℓ_i and ℓ_j (constructed as described above) partition the plane into at most four cells. In the following we show that this holds if and only if ℓ_i and ℓ_j intersect at no more than one point, and respectively that there may be no more than one cell with edges from both classes $\mathcal{E}_i$ and $\mathcal{E}_j$.

Let ℓ_1 and ℓ_2 be two pseudolines in the plane that intersect at k points. We show that the number of cells induced by ℓ_1 and ℓ_2 equals $k+3$. Let p_∞ be a point at infinity and let P_{int} be a set of intersection points between ℓ_1 and ℓ_2. Define a graph $H = (V, E)$ with a vertex set $V = P_{int} \cup p_\infty$ and edge set E composed of pseudoline segments as induced by $P_{int} \cup p_\infty$. Note that H constructed in this way is planar. We get that $|V| = k + 1$ and $|E| = 2|V| = 2k + 2$ (each vertex has four adjacent edges). From Euler's formula we get that the number of faces in H (including the outer face) equals $|E| - |V| + 2 = 2k + 2 - k - 1 + 2 = k + 3$. By construction, we have that each face in H corresponds to an open cell in the arrangement. Hence, number of cells induced by the arrangement of two pseudolines equals $k + 3$ where k is the number of intersection points.

Going back to the split networks, recall that by construction, ℓ_i and ℓ_j intersect only in faces that contain edges from both classes $\mathcal{E}_i$ and $\mathcal{E}_j$. Assume that there are at least two such faces. We get that ℓ_i and ℓ_j have at least two intersection points and induce at least $2 + 3 = 5$ cells. A contradiction, we previously showed that ℓ_i and ℓ_j partition the plane into at most four cells. Hence any two pseudolines intersect in at most one point.

The final step of the proof is to show that we can modify the given collection to an arrangement of pseudolines where every pair intersects *exactly* once. We claim that if there is at least one pair of pseudolines in the collection which do not intersect then we can find a non-intersecting pair which can be modified to intersect in a way which affects no other pseudolines.

Let C be a simple closed curve which contains all intersection points from the collection within its interior. Let ℓ_i and ℓ_j be two pseudolines which do not intersect, and let v_i, v_i', v_j, v_j' be points of intersection between ℓ_i, ℓ_j and C, labeled so that they appeared in the order v_i, v_j, v_j', v_i' around the curve, see Figure 2.12a.

If v_i and v_j are adjacent intersection points on the curve, then we can modify both ℓ_i and ℓ_j to add a point of intersection without affecting any other pseudolines in the collection, as in Figure 2.12b. Otherwise, there is a line ℓ_k which intersects C at some

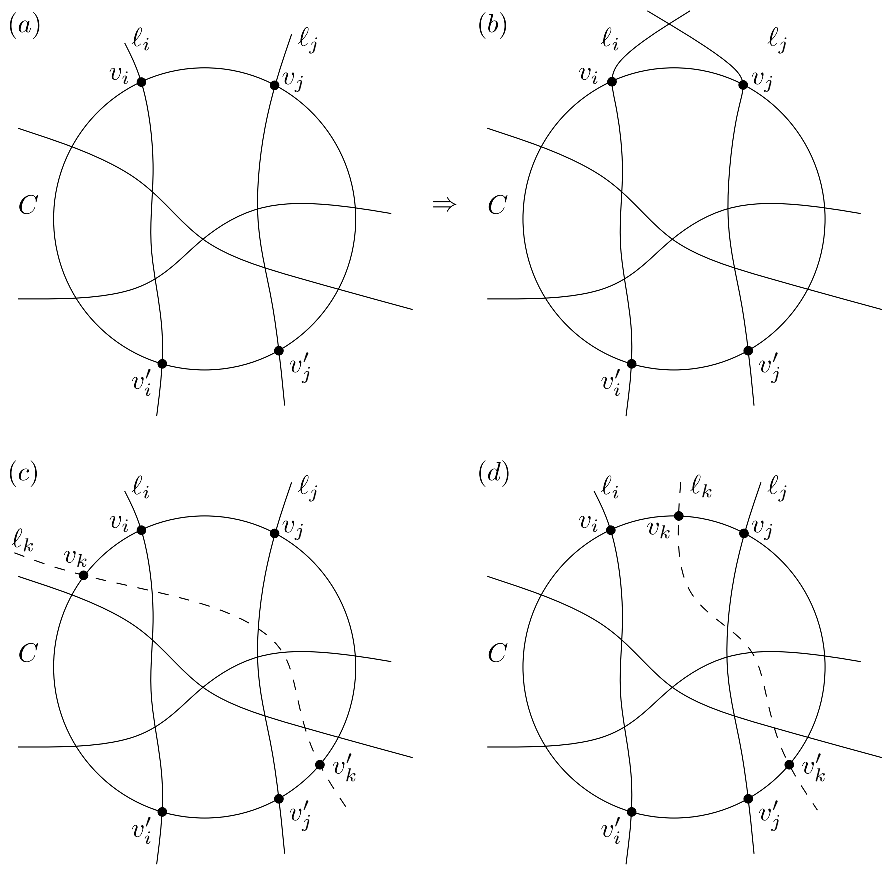


Figure 2.12: A weak arrangement of pseudolines $\mathcal{A}$ with a closed curve C bounding all intersection points. (a) Two pseudolines ℓ_i and ℓ_j do not intersect. (b) $\mathcal{A}$ can be modified so that ℓ_i and ℓ_j intersect without affecting any of the other pseudolines in $\mathcal{A} \setminus \{\ell_i, \ell_j\}$. (c) Pseudoline ℓ_k intersects both ℓ_i and ℓ_j, thus it intersects C on the different sides of ℓ_i and ℓ_j. (d) ℓ_k intersects C in between ℓ_i and ℓ_j, thus it cannot intersect both ℓ_i and ℓ_j.

point v_k between v_i and v_j. If ℓ_k intersected both ℓ_i and ℓ_j then it would intersect C between v_i and v'_i and between v_j and v'_j, see Figure 2.12c, a contradiction. Without loss of generality, suppose that ℓ_k does not intersect ℓ_i as shown in Figure 2.12d. We can then repeat the argument with ℓ_i and ℓ_k, noting that the number of intersection points on the curve between v_i and v_k is strictly less than that between v_i and v_j. In this way we eventually obtain two non-intersecting lines with adjacent intersection points on C. □

2.6 Maximal split systems

We say that a flat split system $\mathcal{S}$ is *maximal* if it is not strictly contained within any other flat split system. In this section we explore properties of maximal flat split systems. In particular we show that a flat split system is maximal exactly when it contains $\binom{n}{2}$ splits, and also that these maximal split systems have an elegant characterization as uniform, loop-free, acyclic oriented matroids of rank three

Lemma 2.6.1 (Zaslavsky, 1975; Björner *et al.*, 1999)**.** *A rank three oriented matroid on n elements has at most*

$$2\sum_{i=0}^{2}\binom{n-1}{i}$$

topes. This bound is realized exactly when the oriented matroid is uniform.

One can easily see that given bound is equal to $2+2\binom{n}{2}$.

We now prove some equivalences for the oriented matroids. These equivalences are used later on for exploring some properties of maximal flat split systems.

Lemma 2.6.2. *Let $\mathcal{M}$ be a loop-free, acyclic, rank three oriented matroid on n elements with tope set $\mathscr{T}$. The following are equivalent*

1. *$\mathcal{M}$ is uniform*
2. *$|\mathscr{T}| = 2\binom{n}{2}+2$*
3. *$\mathcal{M}$ is the only loop-free, acyclic, rank three oriented matroid with tope set which contains $\mathscr{T}$.*
4. *Every type I representation of $\mathcal{M}$ is a simple arrangement (at most two lines intersect at any one point)*

Proof. 1 $\Rightarrow$ 2. Is a direct result of Lemma 2.6.1.

2 $\Rightarrow$ 3. Let $\mathcal{M}$ be a loop-free, acyclic, rank three oriented matroid on n elements with a set of topes of size $|\mathscr{T}| = 2\binom{n}{2}+2$. Assume there is another loop-free, acyclic, rank three oriented matroid $\mathcal{M}'$ on the same set of elements with a tope set $\mathscr{T}'$ such that $\mathscr{T} \subseteq \mathscr{T}'$, from this follows that $|\mathscr{T}| \leq |\mathscr{T}'|$. From Lemma 2.6.1 we have that $|\mathscr{T}'| \leq n^2-n+2 = 2\binom{n}{2}+2$, thus $|\mathscr{T}| = |\mathscr{T}'|$, respectively $\mathscr{T} = \mathscr{T}'$ and consequently $\mathcal{M} = \mathcal{M}'$. Hence there is only one loop-free, acyclic, rank three oriented matroid with tope set which contains $\mathscr{T}$.

3 $\Rightarrow$ 4. Let $\mathcal{M}$ be the only loop-free, acyclic oriented matroid of rank three with an element set $\mathcal{X}$ and a tope set that contains $\mathscr{T}$. Suppose that there is a type I representation of $\mathcal{M}$ such that three pseudolines for elements $a, b, c \in \mathcal{X}$ all meet at a single point v. Pushing one of the pseudolines off from v creates a new bounded cell as shown in Figure 2.13 that corresponds to a new tope T'. We get a type I representation of a matroid with a tope set $\mathscr{T} \cup T'$, a contradiction.

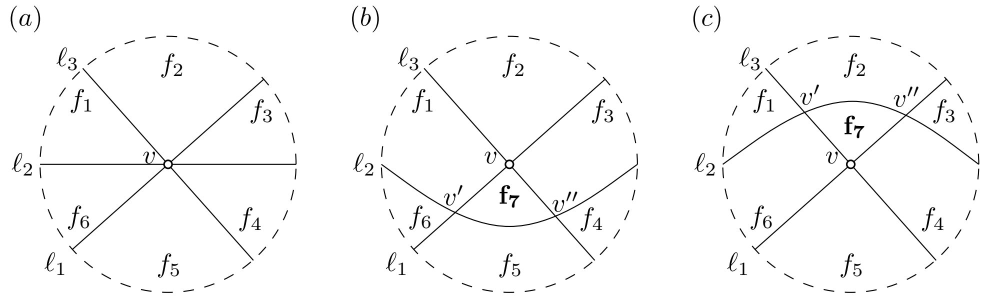


Figure 2.13: (a) A part of an arrangement of pseudolines with three pseudolines ℓ_a, ℓ_b and ℓ_c passing through a vertex v; there are six faces around v. (b),(c) Two ways resolving v into a vertex with two pseudolines passing through. Both perturbations increase the number of cells by one and do not affect the rest of the arrangement.

$4 \Rightarrow 1$. In the type I representation of $\mathcal{M}$, cocircuits correspond to points of intersection. Each such point lies on two lines, so there are only two elements in the corresponding zero set for each cocircuit. Recall that an oriented matroid is uniform if all of its cocircuits have exactly $r-1$ zero elements, where r is a rank of $\mathcal{M}$. Hence $\mathcal{M}$ is uniform. □

Next, we show that uniform, loop-free, acyclic oriented matroids of rank three correspond to flat split systems of maximal cardinality.

Theorem 2.6.3. *Let $\mathcal{S}$ be a flat split system on $\mathcal{X}$, and let $n = |\mathcal{X}|$. Then $|\mathcal{S}| \leq \binom{n}{2}$. Furthermore, the following are equivalent*

1. *$\mathcal{S}$ is the set of all splits induced by a loop-free, acyclic, uniform oriented matroid of rank three.*
2. *$|\mathcal{S}| = \binom{n}{2}$*
3. *There is no other flat split system on $\mathcal{X}$ which contains all of the splits in $\mathcal{S}$.*

In particular, all maximal flat split systems have cardinality $\binom{n}{2}$.

Proof. Let $\mathcal{S}$ be a flat split system. By Theorem 2.4.1 we have that there is an acyclic, loop-free, rank three oriented matroid with a set of topes $\mathscr{T}$ such that $\mathcal{S} = \{S(T) : T \in \mathscr{T}\}$. Hence $T_+, T_- \in \mathscr{T}$ and $|\mathscr{T}| = 2|\mathcal{S}| + 2$. Therefore we have that $|S| \leq \binom{n}{2}$. By Lemma 2.6.1 we get that maximal flat split systems are equivalent to uniform, acyclic, loop-free, rank three oriented matroids, thus, 1, 2 and 3 are equivalent by Lemma 2.6.2. □

2.7 Identifying flat split systems from partial sets of splits

We finish this chapter with an open problem: given any set of splits $\mathcal{S}$ on $\mathcal{X}$ with $|\mathcal{X}| = n$, can we tell if it is flat?

To answer this question we consider oriented matroids. One can easily translate a set of splits into a set of topes $\mathscr{T}'$. Now the question is, can we find a loop-free, acyclic oriented matroid of rank three with set of topes $\mathscr{T}$ such that $\mathscr{T}' \subset \mathscr{T}$. We can tell in polynomial time whether $\mathscr{T}' \cup \{T_+, T_-\}$ equals a set of topes of an oriented matroid by checking the tope axioms (Definition 2.3.1). However this does not provide a test for whether $\mathscr{T}'$ is *contained* in a set of topes of a rank three oriented matroid.

We do not have an answer for the general case as the problem is very likely to be NP-complete.

Conjecture 2.7.1. (Spillner *et al.* (2012)) It is a NP-complete problem to decide whether a set of splits is flat.

This assumption is based on a similar problem of extending partial chirotopes which was proved to be NP-complete (Tschirschnitz, 2001). From the set of topes $\mathscr{T}'$ we can determine the chirotope and then apply Corollary 3.6.4 of Björner *et al.* (1999) to check whether the chirotope comes from an oriented matroid of rank three.

Chapter 3

Computing flat split systems

In this chapter we present a new method, FlatNJ, for inferring flat split systems from data. FlatNJ is based on oriented matroids and employs an agglomerative approach similar to that of Neighbor-Joining (Saitou and Nei, 1987) and Neighbor-Net (Bryant and Moulton, 2004). We look at the definitions of neighbors used by the Neighbor-Joining and Neighbor-Net and use them to define neighbors in flat split systems. This method has been published in:

> Balvočiūtė, M., Spillner, A., and Moulton, V. (2014). FlatNJ: A novel network-based approach to visualize evolutionary and biogeographical relationships. *Systematic Biology*, 63 (3), 383–396.

The first part of the chapter will follow the article fairly closely.

3.1 Motivation

Split networks have been used in various applications including the evolutionary analysis of viruses (Tugume *et al.*, 2010), plants (Goremykin *et al.*, 2013), microbes (Octavia and Lan, 2006), animals (The STAR Consortium, 2008), and even languages (Dunn *et al.*, 2005). To illustrate, consider the split networks in Figure 3.1, which we generated from subcollections of a *Simian immunodeficiency virus* (SIV) data set published by Pelletier *et al.* (1995) and analyzed by Wain-Hobson *et al.* (2003) using methods for split networks available in version 3 of the SplitsTree program (Huson, 1998). Each network in this figure represents a collection of splits or bipartitions of the taxa that label the network. In particular, each split is represented by a band of parallel edges that all have the same length: the band of bold edges in network N1 represents the split that groups taxa 104 and 119 together versus the remaining taxa. This generalizes the relationship between edges of an unrooted phylogenetic tree and a splits of its leaf set.

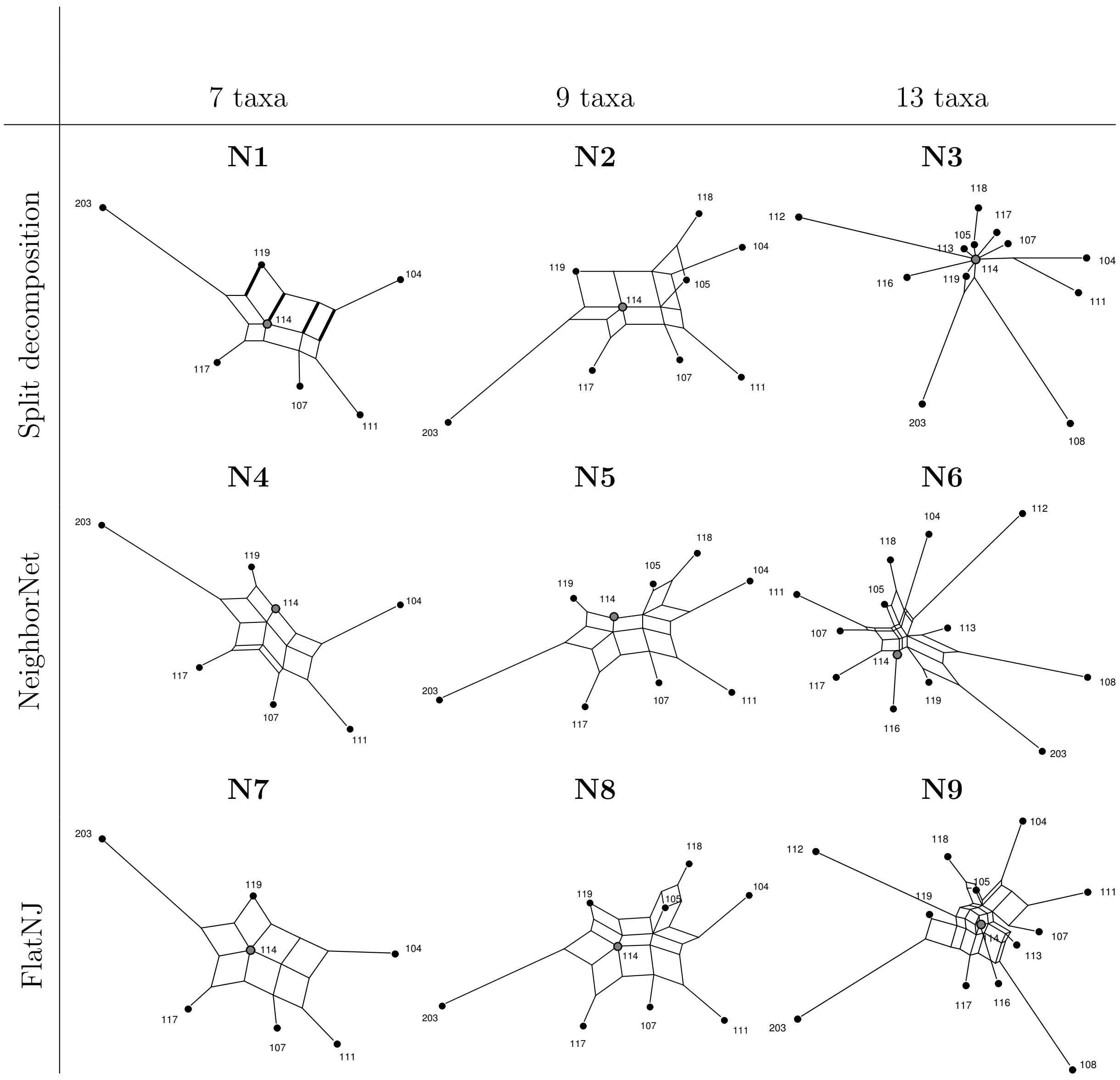


Figure 3.1: Split networks for SIV sequences (Pelletier *et al.*, 1995). Sequence labels are the same as those used in Wain-Hobson *et al.* (2003). Networks in a row are generated with the method specified in front of the row.

The boxes that appear in many of the networks in Figure 3.1 indicate pairs of splits that are *incompatible*, that is, pairs of groupings that cannot be represented simultaneously in a single phylogenetic tree. Such boxes suggest that the data are not treelike. In this particular example, some of the boxes probably result from intra-locus recombination. The box in network N1 with one vertex labeled 119 indicates that taxon 119 shares similarities with both taxon 203 and taxon 104. This suggests that taxon 119 could be a recombinant, although a more detailed analysis would have to be performed to verify this.

The two best known methods for generating split networks are split decomposition (Bandelt and Dress, 1992b) and Neighbor-Net (Bryant and Moulton, 2004). Both are implemented in the SplitsTree4 package (Huson and Bryant, 2006), and the networks generated by them for the SIV data set are depicted in the top two rows of networks in Figure 3.1. Split decomposition is useful for analysis of small data sets, but have two disadvantages in general. First, for large data sets, the resulting networks tend to be highly unresolved (network N3 in Figure 3.1, see also Winkworth *et al.* 2005). Second, split decomposition may yield networks which cannot be drawn without crossing edges (network N2 in Figure 3.1), which can make it difficult to produce a layout for these networks that can easily be interpreted. Neighbor-Net overcomes both of these issues as it can generate quite resolved networks even for much larger data sets (see, e.g., Beiko 2011), and it is guaranteed to produce a network that is *planar*, that is, that can be drawn without crossing edges. Even so, Neighbor-Net networks are constrained to be *outer-labeled*, that is, all labels must lie on the outside of the network. This may lead to situations where potentially useful information can be lost (the split represented by the bold edges in network N1 in Figure 3.1, for example, is not displayed by network N4).

In this chapter we propose a new algorithm for computing planar split networks. Our new method FlatNJ helps to rectify the difficulties with split decomposition and Neighbor-Net as it

a) does not force labels to the outside of the network (network N7 in Figure 3.1),
b) avoids crossings between edges as much as possible (network N8 in Figure 3.1) and
c) can yield informative splits even when the number of taxa increases (network N9 in Figure 3.1).

As with QNet (Grünewald *et al.*, 2007) for generating outer-labeled planar split networks, FlatNJ takes quartet-like data, namely, systems of 4-splits as an input. To construct networks FlatNJ employs an agglomerative approach similar to that used in Neighbor-Joining (Saitou and Nei, 1987) and Neighbor-Net (Bryant and Moulton, 2004). In the first stage of FlatNJ, the system of 4-splits is iteratively reduced by joining pairs of elements identified as neighbors until 4-splits for only 4 elements remain. Next step is expansion. First, we convert the remaining system of 4-splits into a flat system and then iteratively add more flat splits by reversing the agglomeration. For a high-level overview of FlatNJ see Algorithm 3.1. Once the flat split system is computed we draw it using the algorithm by Spillner *et al.* (2012).

Algorithm 3.1 FlatNJ

```
procedure FlatNJ
    if $n = 4$ then
        return InitFlatSplits
    end if
    $\{a, b\} \leftarrow$ FindNeighbors
    $c \leftarrow$ Agglomerate$(a, b)$
    FlatNJ
    $\mathcal{S} \leftarrow$ Expand$(\mathcal{S}, c, a, b)$                ▷ expand $c$ into $a$ and $b$
    return $\mathcal{S}$
end procedure
```

3.2 Systems of 4-splits

When we developed FlatNJ, we at first considered taking a matrix of pairwise distances as input, as with the Neighbor-Net method. However, this has the disadvantage that there can be more than one flat split system representing such a matrix (see, e.g., Figure 3.2 a-c). Intuitively, the problem is that pairwise distances cannot distinguish between the two fundamentally different split networks on a set of four taxa (figure 3.2b and 3.2c): Either none of the points is inside the triangle formed by the other three, or precisely one of the points is inside the triangle formed by the other three. Moreover pairwise distances do not contain much of the structural information. That is if we look at subsets of taxa of size two we can get trees composed of two leaves only, but no networks, see Figure 3.3a. If we increase this number by one and take triplets instead, as in Figure 3.3b, then we still get trees that tell nothing about a possible topology of a network. However as soon as we increase the number to four we are able to get small split networks that can have two different topologies, see Figure 3.3c. Therefore to discriminate between the two possible configurations as in Figure 3.2b-c we consider quartet-like input data like that used for the QNet method Grünewald *et al.* (2007).

We now present some definitions that are necessary for us to describe our method. For any four distinct elements a, b, c and d in $\mathcal{X}$, a 4-*split* is either of the form $\{a, b\}|\{c, d\}$ or of the form $\{a\}|\{b, c, d\}$. As with splits of the whole taxon set, $\{a, b\}|\{c, d\}$ and $\{c, d\}|\{a, b\}$ (and, similarly, $\{a\}|\{b, c, d\}$ and $\{b, c, d\}|\{a\}$) denote the same 4-split. Note

(*a*)

D	a	b	c	d
a	0	5	6	4
b	5	0	5	5
c	6	5	0	4
d	4	5	4	0

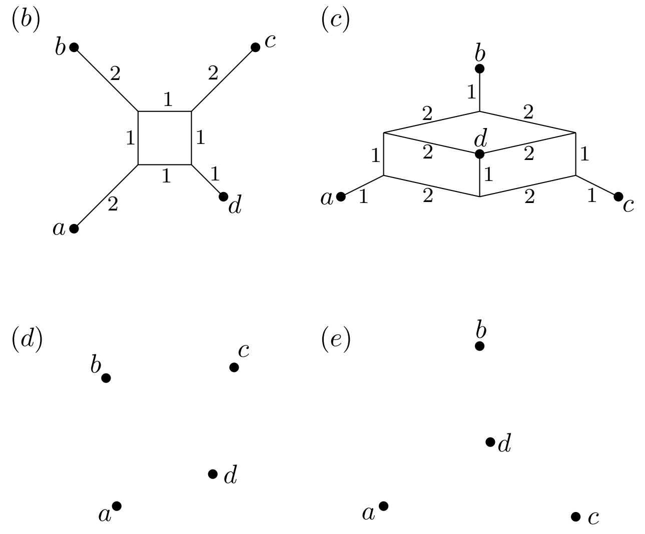


Figure 3.2: (a) A matrix of pairwise distances on the set of taxa $\mathcal{X} = \{a, b, c, d\}$. (b), (c) Two split networks representing weighted flat split systems in which the shortest path distance perfectly matches the distances given in subfigure a. For clarity, the lengths of the edges are also given as the number next to each edge. (d) A configuration of four points in the plane that corresponds to the structure of the flat split system represented in subfigure b: No taxon is inside relative to the other three. (e) A configuration of four points in the plane that corresponds to the structure of the flat split system represented in subfigure c: Taxon d is inside relative to the other three.

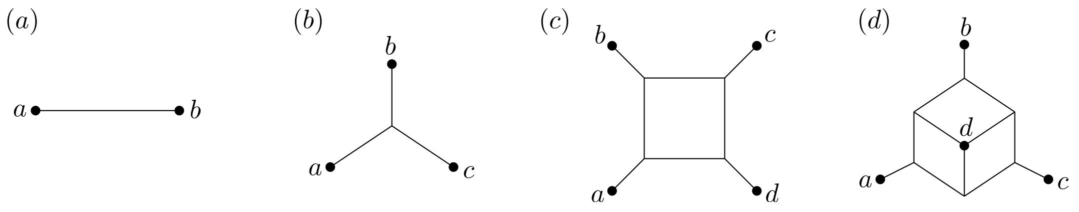


Figure 3.3: Possible split network topologies on a small number of taxa. (a) Two taxa and (b) three taxa networks are always trees whereas (c),(d) four taxa networks may admit topologies that are not tree like.

that 4-splits that group two taxa versus two other taxa are usually referred to as *quartets*. Thus, 4-splits can be viewed as a straight-forward generalization of quartets where also groupings of one taxon versus three other taxa are considered. Also note that there are precisely seven distinct 4-splits for any set of four taxa. In the following, we denote the collection of all possible 4-splits that can be formed from the taxa in $\mathcal{X}$ by $\mathcal{F} = \mathcal{F}(\mathcal{X})$ and we will usually also consider a weighting λ that assigns to every 4-split in $\mathcal{F}$ a non-negative real number. The pair $(\mathcal{F}, \lambda)$ will then be referred to as a *weighted system of* 4-splits and our method takes such a system as its input.

However, note that, unlike the QNet method, FlatNJ also assigns weights to the *trivial splits* (i.e., splits that separate one taxon from all of the rest) in the resulting flat split system. These splits correspond to "pendant" edges in the final split network. In our first experiments we found that the split systems we generated from systems of 4-splits tended to be almost circular. On investigating this phenomenon, we realized that this was probably due to the fact that any flat split system that contains all of the possible trivial splits must in fact be circular. Moreover, the presence of many 4-splits of the form $\{x\}|\{a, b, c\}$ with large weights in the input will naturally lead to flat split systems that contain the trivial split $\{x\}|X - \{x\}$, thus blocking the option of x being placed inside the resulting split network. For this reason, given a system of 4-splits $(\mathcal{F}, \lambda)$, we first compute the quantity

$$\beta(x) = \min_{\{x\}|\{a,b,c\} \in \mathcal{F}} \lambda(\{x\}|\{a, b, c\}) \tag{3.1}$$

for every $x \in \mathcal{X}$, that is, the smallest weight over all 4-splits of the form $\{x\}|\{a, b, c\}$ in $\mathcal{F}$. Then we adjust λ by subtracting $\beta(x)$ from the weight of every 4-split of this form because this amount of weight will definitely be represented in the resulting split network independently of whether x is placed inside the network or not. This is achieved in the final step of the algorithm by adding back $\beta(x)$ quantities to the weights of the respective trivial splits. A more detailed explanation of this procedure is given later when the drawing is discussed.

3.2.1 Generating systems of 4-splits

We consider two possible methods to generate systems of 4-splits: the first produces them from multiple sequence alignments using statistical geometry Eigen *et al.* (1988), and the second directly from distances between points in the plane.

For the first method, let $\mathcal{A}$ be a sequence alphabet, and let $\mathcal{D}$ denote a measure of pairwise distance between the letters in $\mathcal{A}$. Here we use $\mathcal{D}(L, L) = 0$ and $\mathcal{D}(L, L') = 1$ for any two distinct letters L and L' in $\mathcal{A}$ (see, e.g., Nieselt-Struwe and von Haeseler 2001). Then, for a multiple sequence alignment with ℓ columns $c_1, c_2, \ldots, c_\ell$, each column c_i, $1 \leq i \leq \ell$, yields a distance matrix D_i on $\mathcal{X}$ by putting $D_i(x, x') = \mathcal{D}(L, L')$, where L and L' are the letters in column c_i in the sequence corresponding to taxon x and taxon x', respectively. To obtain a weight for each 4-split in $\mathcal{F}$, we put

$$\lambda(\{a,b\}|\{c,d\}) = \frac{1}{\ell}\sum_{1\leqslant i\leqslant \ell}\frac{1}{2}\Big(\max\left\{\begin{smallmatrix} D_i(a,c)+D_i(b,d) \\ D_i(a,d)+D_i(b,c) \\ D_i(a,b)+D_i(c,d)\end{smallmatrix}\right\} - D_i(a,b) - D_i(c,d)\Big) \text{ and}$$

$$\lambda(\{a\}|\{b,c,d\}) = \frac{1}{\ell}\sum_{1\leqslant i\leqslant \ell}\frac{1}{2}\min\left\{\begin{smallmatrix} \max\{D_i(a,b)+D_i(a,c)-D_i(b,c),0\} \\ \max\{D_i(a,c)+D_i(a,d)-D_i(c,d),0\} \\ \max\{D_i(a,b)+D_i(a,d)-D_i(b,d),0\}\end{smallmatrix}\right\}$$

for any four distinct taxa a, b, c and d in $\mathcal{X}$. Note that the i-th summand in both formulae corresponds to the so-called *isolation index* (Bandelt and Dress, 1992a) of the 4-split with respect to the distance matrix D_i.

We also developed a second method for generating systems of 4-splits from distances between points in the plane (coming from, e.g., geographical coordinates for sampling locations of taxa) since we are also interested in the possibility of incorporating such information into our analyses. Recall that, for any four distinct taxa a, b, c and d, there are essentially two different ways in which the corresponding taxa locations can be arranged (Figure 3.2d and 3.2e). In each case, only six 4-splits (out of the seven possible 4-splits) are suggested by the relative position of the locations and these are exactly those 4-splits represented in the corresponding split network in Figure 3.2b and 3.2c, respectively.

To assign weights to the 4-splits, we apply the formula in Moulton and Spillner (2012, Thm. 3) to the Euclidean distances D^E between the locations. This formula will yield the unique weights for the 4-splits such that the shortest path lengths in the corresponding split network equal the given Euclidean distances. In particular, if the four taxa are arranged as in Figure 3.2d this is equivalent to weighting each 4-split by its isolation index with respect to D^E as given above (which immediately implies $\lambda(\{a,c\}|\{b,d\}) = 0$). Otherwise, if the four taxa are arranged as in Figure 3.2e, we put $\lambda(\{d\}|\{a,b,c\}) = 0$ and set

$$\begin{aligned}
\lambda(\{a\}|\{b,c,d\}) &= \frac{1}{2}(D^E(a,b) + D^E(a,c) - D^E(b,d) - D^E(c,d)),\\
\lambda(\{b\}|\{a,c,d\}) &= \frac{1}{2}(D^E(a,b) + D^E(b,c) - D^E(a,d) - D^E(c,d)),\\
\lambda(\{c\}|\{a,b,d\}) &= \frac{1}{2}(D^E(a,c) + D^E(c,b) - D^E(a,d) - D^E(b,d)),\\
\lambda(\{a,b\}|\{c,d\}) &= \frac{1}{2}(D^E(a,d) + D^E(b,d) - D^E(a,b)),\\
\lambda(\{a,c\}|\{b,d\}) &= \frac{1}{2}(D^E(a,d) + D^E(c,d) - D^E(a,c)), \text{ and}\\
\lambda(\{a,d\}|\{b,c\}) &= \frac{1}{2}(D^E(b,d) + D^E(c,d) - D^E(b,c)).
\end{aligned}$$

It is easy to verify that these weights will always be non-negative.

3.3 Neighbors in flat split systems

FlatNJ constructs a flat split system from a system of 4-splits using an agglomerative approach similar to the ones used in Neighbor-Joining (Saitou and Nei, 1987) and Neighbor-Net (Bryant and Moulton, 2004). One of the key steps in both of these previous approaches is the selection of "neighbors". The exact definition of the neighbors in a split system depends on the representation used. We give definitions in terms of planar split networks and flat split systems (i.e., point configurations).

3.3.1 Split networks

First recall how we defined neighbors in the trees and circular networks:

- In a Neighbor-Joining tree neighbors are any two taxa that form a cherry (Definition 1.1.1), see Figure 3.4a.
- In a Neighbor-Net network neighbors are two taxa that appear next to each other in a circular ordering of labeled vertices (Definitions 1.1.2) as shown in Figure 3.4b.

We generalize definitions 1.1.1 and 1.1.2 to planar split networks. Two taxa $a, b \in \mathcal{X}$ are neighbors in a planar split network $\mathcal{N}(\mathcal{X})$ if their corresponding vertices appear in the same relative position with respect to all vertices labeled with $\mathcal{X} \setminus \{a, b\}$.

Definition 3.3.1. Let $\mathcal{N}_a$ be a network obtained from $\mathcal{N}(\mathcal{X})$ by removing label a and let $\mathcal{N}_b$ be a network obtained in the same way by removing b. Let $\mathcal{N}_a^{b \to a}$ be the network $\mathcal{N}_a$ with b replaced with a. We say that a and b are neighbors in $\mathcal{N}(\mathcal{X})$ if $\mathcal{N}_a^{b \to a}$ and $\mathcal{N}_b$ induce the same split system.

For example, taxa x and y are neighbors in a planar split network shown in Figure 3.4c.

Neighbors are easier to identify in terms of 4-taxa networks. Let $Y \subseteq \mathcal{X}$ be a subset of $\mathcal{X}$ of size $|Y| = 4$ and let $\mathcal{N}$ be an planar split network on $\mathcal{X}$. Then a 4-taxa network $\mathcal{N}^4$ on Y is a network obtained from $\mathcal{N}$ by removing all labels in $\mathcal{X} \setminus Y$ and afterwards deleting edges that induce redundant splits. Let $\mathbb{N}^4$ be a set of all 4-taxa networks extracted from $\mathcal{N}$ on all possible $Y \subseteq \mathcal{X}$. Then two taxa $\{x, y\} \in \mathcal{X}$ are neighbors if and only if they are in a *neighborly position.* We say that x and y are in a neighborly position with respect to $\mathbb{N}^4$ if in all 4-taxa networks on $Y = \{a, b, c, z\}$ where $z \in \{x, y\}$ and $\{a, b, c\} \subseteq \mathcal{X} \setminus \{x, y\}$, x and y appear in the same position with respect to a, b and c and in the networks on $Y = \{a, b, x, y\}$ where $\{a, b\} \subseteq \mathcal{X} \setminus \{x, y\}$ we have that x and y are either next to each other in a circular ordering (if all four taxa are placed on the edge of the 4-taxa network), or either x is inside and y is on the outside of the network or vice versa (if one of the four taxa appears inside of the 4-taxa network).

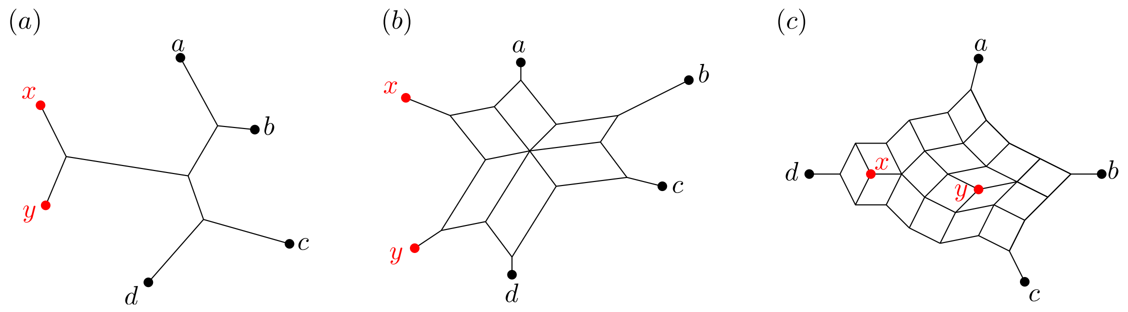


Figure 3.4: Two taxa x and y are neighbors all three planar split networks. a) A Neighbor-Joining tree where vertices labeled with x and y form cherry indicating that x and y are neighbors. b) A Neighbor-Net network with vertices x and y appearing next to each other in the circular ordering of the taxa. c) A FlatNJ network with x and y appearing in the same relative position with respect to the other labeled vertices.

For example e and f in Figure 3.5a are neighbors whereas b and f in 3.5b are not neighbors.

3.3.2 Flat splits

For flat split systems with taxa represented as a configuration of points in 2D and splits as an arrangement of pseudolines we use a slightly different definition of neighbors. As mentioned in the introduction, the splits displayed in the networks produced by Neighbor-Net can be represented by arranging points on a circle (Figure 3.6) and drawing splits as an arrangement of straight lines. Then two distinct taxa x and x' are considered to be neighbors if they correspond to consecutive points along the circle for some such ordering (e.g., b and c are neighbors in Figure 3.6a). Note, however, that this is equivalent to the following condition (shown in Figure 3.6a and Figure 3.6b):

(Nb) The straight line segment with end points x and x' does not intersect any of the straight lines through any pair of distinct elements in $X \setminus \{x, x'\}$.

The advantage of condition (Nb) is that it can readily be applied to any set of points in the plane not necessarily arranged around a circle (Figure 3.6c and Figure 3.6d). More precisely, given a flat split system $\mathcal{S}$, two taxa x and x' in $\mathcal{X}$ are *neighbors relative to* $\mathcal{S}$ if there exists some arrangement of X in the plane so that every split in $\mathcal{S}$ can be represented by a straight line and also x and x' satisfy condition (Nb). Note that there exist flat split systems for which no pair of taxa form neighbors (Figure 3.6e). Such split systems will not be generated by FlatNJ.

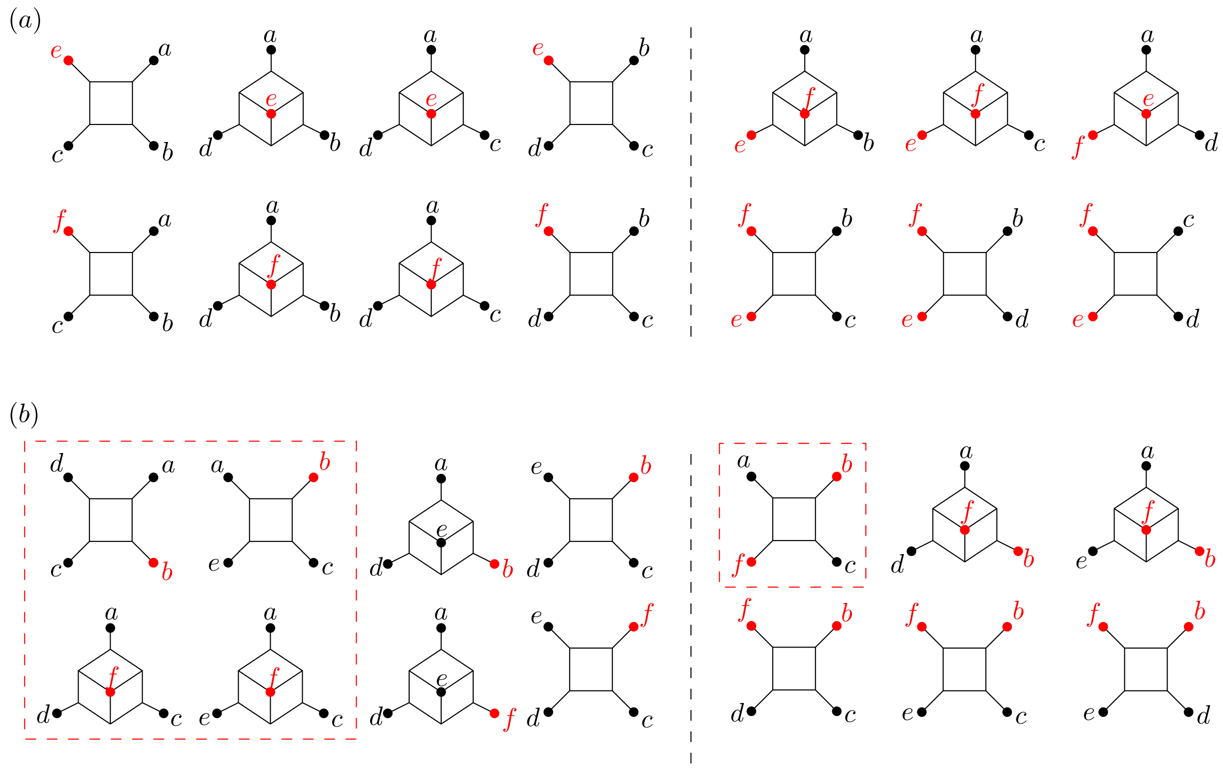


Figure 3.5: (a) Taxa e and f appear in the same position with respect to all triples of other taxa $\{a, b, c\}$, $\{a, b, d\}$, $\{a, c, d\}$ and $\{b, c, d\}$, also they are in the neighborly positions in the 4-taxa networks that contain both of them, thus e and f are identified as neighbors. (b) Taxa b and f appear in the same relative position with respect to the triplets $\{a, d, e\}$ and $\{c, d, e\}$, but are in different position with respect to $\{a, c, d\}$ and $\{a, c, e\}$, also they are not in the neighborly position in the 4-taxa network for taxa $\{a, b, c, f\}$, thus b and f are *not* neighbors.

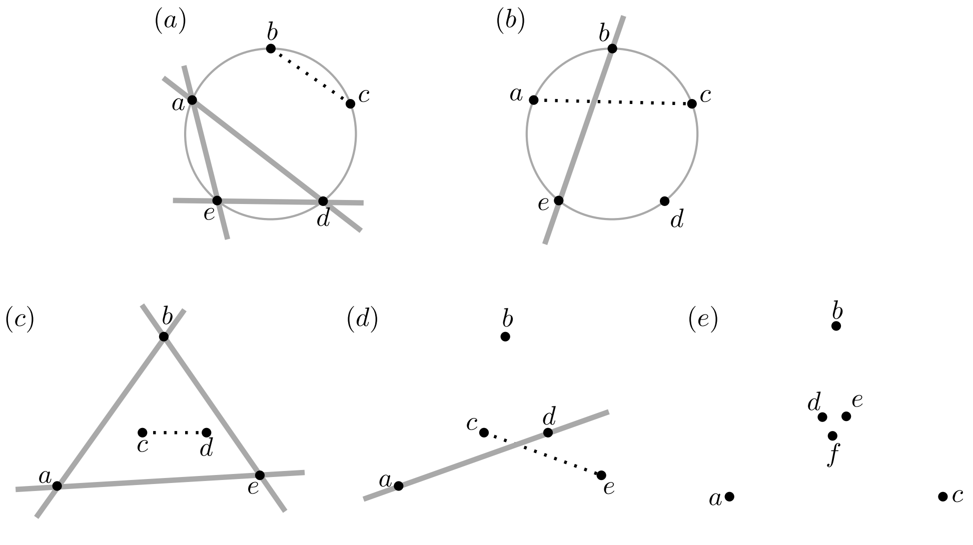


Figure 3.6: (a) Any two consecutive elements along the circle are considered to be neighbors. None of the bold gray straight lines intersects the dotted straight line segment whose end points are the neighbors b and c. (b) The bold gray straight line through b and e intersects the dotted straight line segment with end points a and c, indicating that a and c are not neighbors. (c) Taxa c and d are neighbors because none of the bold gray straight lines intersects the dotted straight line segment with end points c and d. (d) Taxa c and e are not neighbors because the bold gray straight line through a and d intersects the dotted straight line segment with end points c and e. (e) An arrangement of the taxa $X = \{a, b, \ldots, f\}$ in the plane for which there is no pair of neighbors relative to the corresponding full flat split system.

3.4 FlatNJ algorithm

Given a weighted system of 4-splits $(\mathcal{F}, \lambda)$, FlatNJ essentially works in the following stages, in a similar way to the Neighbor-Net method:

1. A pair of neighbors x' and x'' in $\mathcal{X}$ is selected using the system of 4-splits.
2. The neighbors x' and x'' are removed from $\mathcal{X}$ and replaced by a new element x representing both x' and x'' (i.e., x' and x'' are *agglomerated* into a new element x). The system of 4-splits $(\mathcal{F}, \lambda)$ is then updated to give a new system on $\mathcal{X}' = (\mathcal{X} \setminus \{x', x''\}) \cup \{x\}$. This selection and agglomeration procedure is repeated until only four elements remain.
3. The optimal flat split system is chosen for the remaining four taxa.
4. The whole agglomeration process is reversed to create a full flat split system $\mathcal{S}$.
5. The split weights are estimated for $\mathcal{S}$ relative to $(\mathcal{F}, \lambda)$, and a corresponding planar split network is then drawn.

Each of the steps is described in more detail in the following sections.

3.4.1 Choosing neighbors

As in Neighbor-Joining and Neighbor-Net algorithms, we choose neighbors by assigning scores to pairs of elements in $\mathcal{X}$. In particular, we use two scoring functions that have been chosen to ensure that the algorithm is “consistent” as explained later.

Min score

The first function is based on the following observation. Let $(\mathcal{S}, \omega)$ be a weighted flat split system and x' and x'' be two taxa that are neighbors in $\mathcal{S}$. Then, for any two distinct taxa y and y' in $\mathcal{X} \setminus \{x', x''\}$, at least one of the 4-splits:

- $\{x'\}|\{x'', y, y'\}$,
- $\{x', y\}|\{x'', y'\}$,
- $\{x', y'\}|\{x'', y\}$ and
- $\{x', y, y'\}|\{x''\}$

separating x' and x'' has the weight 0 (see Figure 3.7).

Furthermore, the sum of the weights of all the splits in $\mathcal{S}$ that extend the 4-split with weight 0 is equal to 0 (if some 4-split has weight 0, then there are no splits that would extend it). For example, for the full flat split system $\mathcal{S}$ on the set $\mathcal{X} = \{a, b, c, d, e\}$ represented by the arrangement in Figure 3.6c, the taxa c and d are neighbors and there is no split in $\mathcal{S}$ that extends the 4-split $\{c\}|\{a, b, d\}$.

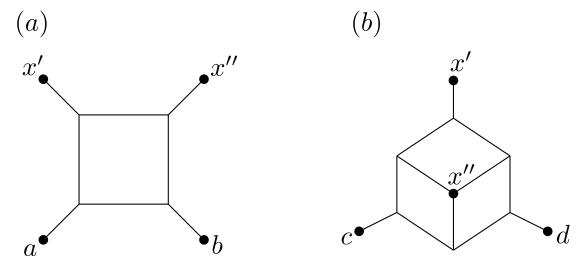


Figure 3.7: (a) Split $\{x', b\}|\{x'', a\}$ is not present in the 4-taxa network (therefore it has weight zero) indicating that x' and x'' are neighbors. Other pairs of neighbors are $\{x', a\}$, $\{a, b\}$ and $\{b, x''\}$. However $\{x', b\}$ and $\{x'', a\}$ are not neighbors because all 4-splits separating x' from b and x'' from a are present in this 4-taxa network. (b) Split $\{x''\}|\{x', c, d\}$ is not present in the 4-taxa network indicating that x'' is a neighbor with x' as well as with c and d. Because all other 4-splits are present other pairs $\{x', c\}$, $\{c, d\}$ and $\{d, x'\}$ are not neighbors.

This suggests defining the following score for any pair x' and x'' of taxa in $\mathcal{X}$:

$$\sigma_{min}(x', x'') = \sum_{\substack{y,y' \in \mathcal{X}\setminus\{x',x''\} \\ y \neq y'}} \min \left\{ \begin{matrix} \lambda(\{x'\}|\{x'',y,y'\}) \\ \lambda(\{x''\}|\{x',y,y'\}) \\ \lambda(\{x',y\}|\{x'',y'\}) \\ \lambda(\{x',y'\}|\{x'',y\}) \end{matrix} \right\}.$$

Intuitively, the score $\sigma_{min}(x', x'')$ captures the total amount of 4-split weight, over all 4-splits in $\mathcal{F}$, that will definitely *not* be represented (recovered) in the resulting flat split system if we make x' and x'' neighbors. Hence, good candidates for neighbors are taxa x' and x'' for which $\sigma_{min}(x', x'')$ is minimized.

Max score

Once we have found the pairs that minimize the score $\sigma_{min}(x', x'')$, we employ a second scoring function that aims to capture the total amount of 4-split weight, over all 4-splits in $\mathcal{F}$ of the form $\{x', x''\}|\{y, y'\}$, that *will* be represented in the resulting flat split system if we make x' and x'' neighbors. More formally:

$$\sigma_{max}(x', x'') = \sum_{\substack{y,y' \in \mathcal{X}\setminus\{x',x''\} \\ y \neq y'}} \lambda(\{x', x''\}|\{y, y'\}).$$

This function is also used in the selection of neighbors in the QNet algorithm (Grünewald *et al.*, 2007).

Hence, in summary, we choose neighbors by first computing all pairs $\{x', x''\}$ that minimize σ_{min} and then, out of these pairs, selecting a pair $\{x', x''\}$ that maximizes σ_{max}.

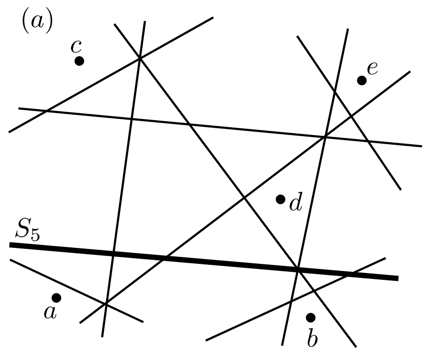
Properties of the scoring functions

We first give an example of a circular split system Σ and a pair of taxa which minimize the scoring function $\sigma_{\min}$ but are not neighbors in Σ. Let Σ be the circular split system on $\mathcal{X} = \{x_1, x_2, \dots, x_5\}$ containing the splits $\{x_i, x_{i+1}\}|\mathcal{X} \setminus \{x_i, x_{i+1}\}$, $1 \le i \le 4$, and the split $\{x_1, x_5\}|\mathcal{X} \setminus \{x_1, x_5\}$. Then the ordering θ given by x_1, x_2, x_3, x_4, x_5 of X is, up to reversal and index shifting, the unique ordering of the taxa in $\mathcal{X}$ such that, for every split $S = A|B$ in the split system, there are indices $1 \le i \le j < n$ so that either $A = \{x_i, x_{i+1}, \dots, x_j\}$ or $B = \{x_i, x_{i+1}, \dots, x_j\}$. Assigning weight $\omega(S) = 1$ to each split $S \in \Sigma$, put $\mathcal{F} = \mathcal{F}_{(\Sigma,\omega)}$. Now, using $\mathcal{F}$ as the input quadruple system, we obtain $\sigma_{\min}(x_1, x_3) = 0$. But x_1 and x_3 are not neighbors in Σ, that is, they are not consecutive in the ordering θ.

We now consider the scoring function $\sigma_{\max}$. It immediately follows by the consistency result presented in Grünewald *et al.* (2009) that, if (Σ, ω) is a weighted circular split system and $\mathcal{F} = \mathcal{F}_{(\Sigma,\omega)}$ is the associated quadruple system, then any pair of taxa that maximize $\sigma_{\max}$ are neighbors in $\mathcal{S}$. However, for more general flat split systems, $\sigma_{\max}$ does not necessarily select neighbors in this way. For example, consider the flat split system Σ depicted in Figure 3.8a. We assign weight $\omega(S) = 1$ to each split $S \in \Sigma$, except for split S_5 to which we assign weight 3. Then, using $\mathcal{F} = \mathcal{F}_{(\Sigma,\omega)}$ as the input quadruple system, $\sigma_{\max}(a, b) = 11$ is the unique maximum score over all pairs (see Figure 3.8c), but a and b are not neighbors in Σ (see Figure 3.8d). If we restrict the score $\sigma_{\max}$ to those pairs with minimum score $\sigma_{\min}$, however, we then select d and e, a pair of elements that are indeed neighbors in Σ (see Figure 3.8e).

3.4.2 Agglomeration

Once a pair of neighbors x' and x'' is selected we move to the agglomeration step. When x' and x'' are joined into x we need to update the system of 4-splits $(\mathcal{F}, \lambda)$ on $\mathcal{X}$ to form one on the set $\mathcal{X}' = (\mathcal{X} \setminus \{x', x''\}) \cup \{x\}$. First, all the 4-splits that contain neither x' nor x'' remain the same in the updated system of 4-splits on $\mathcal{X}'$. Otherwise, let a, b, c be any three distinct elements in $\mathcal{X} \setminus \{x', x''\}$ and put $Y_{x'} = \{x', a, b, c\}$ and $Y_{x''} = \{x'', a, b, c\}$. Then the 4-splits involving precisely the four taxa in $\{x, a, b, c\}$ are assigned the average of the weights of the corresponding 4-splits of $Y_{x'}$ and $Y_{x''}$, that is,



(b)

S_0	$=$	$\{a,c,d\}\|\{b,e\}$
S_1	$=$	$\{a,c\}\|\{b,d,e\}$
S_2	$=$	$\{a,b,c\}\|\{d,e\}$
S_3	$=$	$\{a,b,c,d\}\|\{e\}$
S_4	$=$	$\{a\}\|\{b,c,d,e\}$
S_5	$=$	$\{a,b\}\|\{c,d,e\}$
S_6	$=$	$\{a,b,d\}\|\{c,e\}$
S_7	$=$	$\{b,d\}\|\{a,c,e\}$
S_8	$=$	$\{b\}\|\{a,c,d,e\}$
S_9	$=$	$\{a,b,d,e\}\|\{c\}$

(c)

Pair	$\sigma_{\max}$	$\sigma_{\min}$
(a,b)	11	1
(a,c)	6	0
(a,d)	2	1
(a,e)	1	3
(b,c)	1	3
(b,d)	5	0
(b,e)	4	2
(c,d)	4	1
(c,e)	7	1
(d,e)	7	0

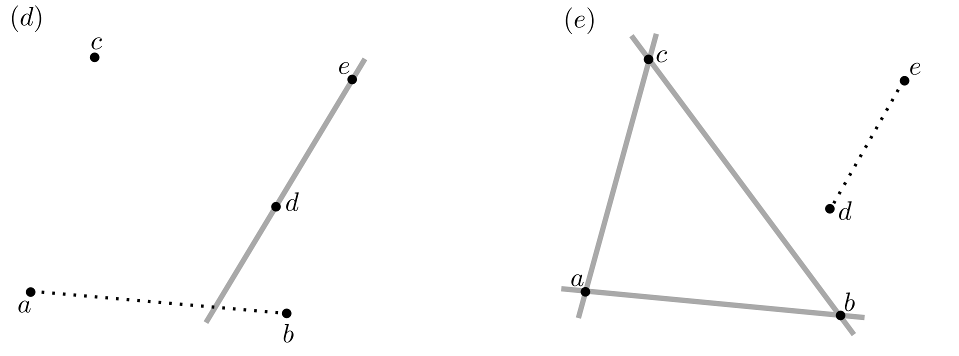


Figure 3.8: (a) A full flat split system $\mathcal{S}$ on the set $\mathcal{X} = \{a, b, c, d, e\}$. All splits are drawn as straight lines. The bold line represents the split S_5 that is assigned weight 3. All other splits are assigned weight 1. (b) The list of splits in $\mathcal{S}$. (c) The scores $\sigma_{\max}$ and $\sigma_{\min}$ for each pair of elements in X relative to the weighting of $\mathcal{S}$ given in subfigure a. (d) The elements a and b maximizing $\sigma_{\max}$ are not neighbors. (e) The neighbors d and e that maximize $\sigma_{\max}$ among all those pairs that minimizes $\sigma_{\min}$.

we put:

$$\begin{aligned}
\lambda(\{x\}|\{a,b,c\}) &= \frac{1}{2}\big(\lambda(\{x'\}|\{a,b,c\}) + \lambda(\{x''\}|\{a,b,c\})\big),\\
\lambda(\{a\}|\{x,b,c\}) &= \frac{1}{2}(\lambda(\{a\}|\{x',b,c\}) + \lambda(\{a\}|\{x'',b,c\})),\\
\lambda(\{b\}|\{x,a,c\}) &= \frac{1}{2}(\lambda(\{b\}|\{x',a,c\}) + \lambda(\{b\}|\{x'',a,c\})),\\
\lambda(\{c\}|\{x,a,b\}) &= \frac{1}{2}(\lambda(\{c\}|\{x',a,b\}) + \lambda(\{c\}|\{x'',a,b\})),\\
\lambda(\{x,a\}|\{b,c\}) &= \frac{1}{2}(\lambda(\{x',a\}|\{b,c\}) + \lambda(\{x'',a\}|\{b,c\})),\\
\lambda(\{x,b\}|\{a,c\}) &= \frac{1}{2}(\lambda(\{x',b\}|\{a,c\}) + \lambda(\{x'',b\}|\{a,c\})), \text{ and}\\
\lambda(\{x,c\}|\{a,b\}) &= \frac{1}{2}(\lambda(\{x',c\}|\{a,b\}) + \lambda(\{x'',c\}|\{a,b\})).
\end{aligned}$$

A visual example of the agglomeration procedure on 4-taxa networks is given in Figure 3.9.

3.4.3 Initializing a flat split system

Once all possible agglomerations have been performed, we are left with a set of 4 taxa $\mathcal{X}_4$ and a system of 4-splits $(\mathcal{F}_4, \lambda_4)$ on $\mathcal{X}_4$. Since $\mathcal{X}_4$ contains precisely four taxa, every split of $\mathcal{X}_4$ can be viewed as a 4-split therefore we have that

$$\mathcal{S}(\mathcal{X}_4) = \mathcal{F}_4. \tag{3.2}$$

Moreover $\mathcal{S}(\mathcal{X}_4)$ may not be flat split system since it may (and usually does) contain seven splits. Therefore, to obtain a full flat split system on $\mathcal{X}_4$ (which must contain precisely $\binom{4}{2} = 6$ splits), we need to select one split in $\mathcal{F}_4$ that will be removed. Following again the idea that we want to minimize the amount of 4-split weight that is not represented in the output, we choose a split $S_{min} \in \mathcal{F}_4$ with the minimum weight. Then we set $\mathcal{S}_4 = \mathcal{S}(\mathcal{X}_4) \setminus \{S_{min}\}$. In addition, we construct a configuration of points in $\mathcal{X}_4$ in the plane such that all the splits in $\mathcal{S}_4$ are represented by straight lines through this configuration. Figure 3.10a shows initial configuration of points for a split system on four taxa with one of the trivial splits having weight 0, Figure 3.10b shows a different initial configuration for a flat split system on $\mathcal{X}_4$ without one of the non-trivial splits, namely $\{x_1, x_3\}|\{x_2, x_4\}$.

3.4.4 Reversing the agglomeration

Next, starting with $\mathcal{X}_4$ we reverse the agglomerations one by one. For simplicity we only describe how this is done for the last reversal that replaces x in $\mathcal{X}'$ by x' and x'' to

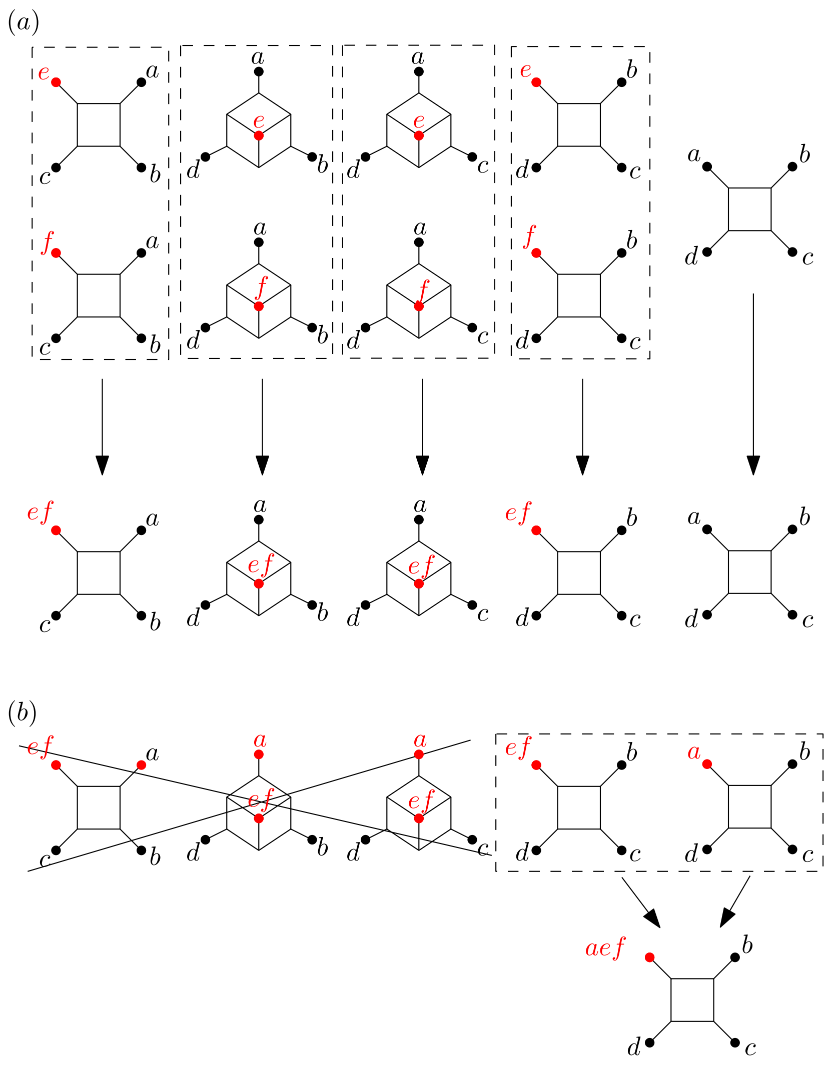


Figure 3.9: (a) Taxa e and f are identified as neighbors and joined into one. 4-taxa networks that contain either e or f are agglomerated whereas those that contain neither of the two remain untouched. 4-taxa networks and their corresponding 4-splits that contain both of the neighbors are deleted on agglomeration. (b) The last step of the agglomeration procedure. The last pair of neighbors is identified and the two 4-taxa networks that contain either of the neighbors are agglomerated into one network. The rest of the 4-taxa networks are eliminated leaving a set of taxa of size 4

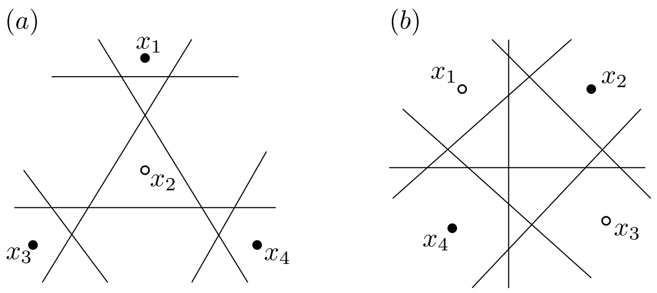


Figure 3.10: Two candidate initial flat split system on four taxa. (a) A flat split system without the split $\{x_2\}|\{x_1, x_3, x_4\}$. (b) An alternative flat split system without the split $\{x_1, x_3\}|\{x_2, x_4\}$.

obtain the set $\mathcal{X}$ since the other reversals are performed in a completely analogous way. To this end, assume that we have a full flat split system Σ' on $\mathcal{X}'$ arranged in the plane. From this we want to find a suitable arrangement of $\mathcal{X}$ in the plane that corresponds to a full flat split system $\mathcal{S}$ on $\mathcal{X}$ (see Figure 3.11a). In particular, the arrangement of $\mathcal{X}$ is obtained by replacing the point representing x in $\mathcal{X}'$ by two points representing x' and x'', respectively (cf. Figure 3.11b). These two points are placed in such a way that, for each split $S = A|B \in \mathcal{S}_4$ with $x \in A$, we have the split $(A \setminus \{x\}) \cup \{x', x''\}|B \in \mathcal{S}$. This is achieved by placing x' and x'' close enough to the original position of x. In the situation depicted in Figure 3.11b it suffices, for example, to place x' and x'' inside the shaded region.

In addition to those splits that arise from the splits in $\mathcal{S}'$, the split system $\mathcal{S}$ also contains $n - 1$ splits that separate x' and x''. The splits of this type that are contained in $\mathcal{S}$ depend on the position of x'' relative to x'. Note that there is some freedom in choosing the precise coordinates of x' and x''. Topologically, there are, however, only $2(n - 2)$ different configurations that can be described as follows. We place a suitably small disk centered at the original position of x (cf. Figure 3.11c). At the center of this disk we place x'. Then we partition the disk into $2(n - 2)$ sectors by drawing straight lines that contain x' and any of the points in $X' \setminus \{x\}$. For each of these sectors, placing x'' anywhere within that sector yields the same flat split system on $\mathcal{X}$, and placing x' in a different sector yields a different flat split system (cf. Figure 3.11d and e). Let $\mathcal{C}$ denote the resulting collection of $2(n - 2)$ different full flat split systems.

We now use the input system of 4-splits $(\mathcal{F}, \lambda)$ again to select one of the flat split systems in $\mathcal{C}$. More specifically, we select some $\mathcal{S}$ in $\mathcal{C}$ for which

$$\sum_{\substack{y,y' \in \mathcal{X} - \{x,x'\} \\ y \neq y'}} \quad \sum_{\substack{S' \text{ a 4-split of } \{x,x',y,y'\} \\ \text{and some } S \text{ in } \Sigma \text{ extends } S'}} \lambda(S'). \tag{3.3}$$

is maximum. In other words, we consider all 4-splits in $\mathcal{F}$ that involve both x' and x''

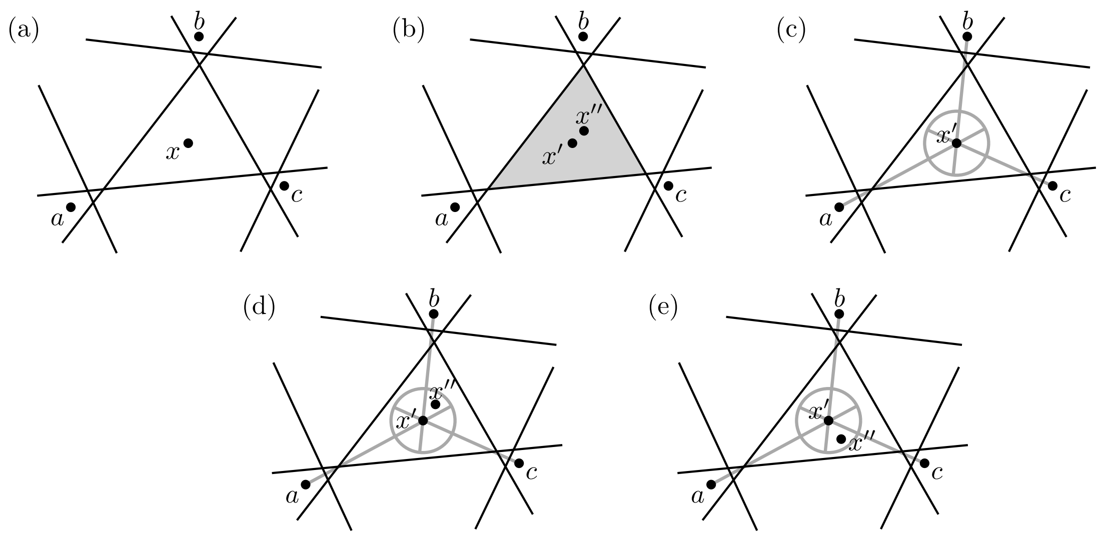


Figure 3.11: (a) A full flat split system $\mathcal{S}'$ on the set $X' = \{a, b, c, x\}$ with x representing two agglomerated elements x' and x''. The black straight lines depict the $\binom{4}{2} = 6$ splits in $\mathcal{S}'$. (b) Replacing x by two points representing x' and x''. (c) The disk sectors representing the options for placing x'' relative to x'. (d) A placement of x'' that yields the four splits $\{x', a\}|\{x'', b, c\}$, $\{x', a, b\}|\{x'', c\}$, $\{x', a, c\}|\{x'', b\}$ and $\{x', c\}|\{x'', a, b\}$ separating x' and x''. (e) An alternative placement of x'' that yields again the splits $\{x', a\}|\{x'', b, c\}$ and $\{x', a, b\}|\{x'', c\}$ but also two different splits, namely, $\{x', b\}|\{x'', a, c\}$ and $\{x', b, c\}|\{x'', a\}$.

and select a split system $\mathcal{S}$ for which the total weight of those 4-splits that are extended by some split in $\mathcal{S}$ is maximum. Note that there can be more than one $\mathcal{S}$ in $\mathcal{C}$ that maximizes (3.3). In this case we select, among those maximizing (3.3), one for which $\mathcal{S}$ contains the two trivial splits $\{x'\}|\mathcal{X} \setminus \{x'\}$ and $\{x''\}|\mathcal{X} \setminus \{x''\}$, if such a $\mathcal{S}$ exists, and an arbitrary one otherwise. This ensures that if there is a simpler way to accommodate the input data (i.e., a phylogenetic tree or a circular split system) then we choose this.

3.4.5 Weighting of a flat split system

Once we have computed a full flat split system $\mathcal{S}$ on $\mathcal{X}$ whose structure reflects that of the input system of 4-splits $(\mathcal{F}, \lambda)$, it only remains to compute non-negative weights for the splits in $\mathcal{S}$. To do this, we use an approach similar to the one used in QNet. More specifically, split weights ω are computed so that the system of 4-splits $(\mathcal{F}, \lambda_{(\Sigma,\omega)})$ on $\mathcal{X}$ (which is defined by setting, for every 4-split $S' \in \mathcal{F}$, $\lambda_{(\Sigma,\omega)}(S')$ to be the total weight of those splits S of $\mathcal{X}$ that extend S') is as close as possible to the input system of 4-splits $(\mathcal{F}, \lambda)$ in the least squares sense, that is, we minimize

$$\sum_{S' \in \mathcal{F}} \left(\lambda(S') - \lambda_{(\Sigma,\omega)}(S')\right)^2 . \tag{3.4}$$

To minimize this last expression we solve a quadratic program (see, e.g., Lawson and Hanson 1974). In the implementation of FlatNJ we use Gurobi solver (Gurobi Optimization, Inc., 2015) to find the optimal weights of splits in $\mathcal{S}$.

3.4.6 Filtering of incompatible splits

The user can then filter the resulting weighted flat split system (Σ, ω) if desired using the method described in Grünewald *et al.* (2007) to suppress splits with very low weights. In particular, the user provides a real-valued threshold t, $0 \leq t \leq 1$, which suppresses any split S in $\mathcal{S}$ for which there exists some other split S' in $\mathcal{S}$ such that S and S' are incompatible and the weight of S is less than a fraction t of the weight of S'. The splits are removed in order of increasing weight.

3.4.7 Drawing a planar split network

The resulting weighted flat split system $(\mathcal{S}, \omega)$ is represented by a planar split network $\mathcal{N}$, which is drawn using the algorithm presented in Spillner *et al.* (2012). It can then be displayed using the SplitsTree package (Huson and Bryant, 2006). Note that the filtering mentioned above can help ease interpretation of this network by reducing the number of small boxes that appear in it. At this stage, the values $\beta(x)$, defined in (3.1) for all x in $\mathcal{X}$, are also taken into account as follows. If $\mathcal{N}$ already contains a pendant edge representing the trivial split $\{x\}|\mathcal{X} - \{x\}$ then the length of this pendant edge is just increased by $\beta(x)$. Otherwise (i.e., $\mathcal{N}$ does not contain a pendant edge representing the trivial split $\{x\}|\mathcal{X} - \{x\}$ and $\beta(x) > 0$), a new pendant edge of length $\beta(x)$ is added to $\mathcal{N}$. Note that this last step can potentially produce pendant edges that must cross some other edges in the planar network $\mathcal{N}$.

3.4.8 Implementation of FlatNJ

We have implemented FlatNJ in Java. For analyzing the examples below, we ran the program on a PC with Intel i5-2300(4) CPU, with 6 GB of main memory and with the operating system Ubuntu 12.04. The run time of our implementation is superpolynomial in the worst case due to the fact that the computation of the weights for the splits involves solving a quadratic program (for this we use algorithms in the Gurobi Optimizer, version 5.0, `www.gurobi.com`), although in practice we have not found this to be a limitation for sets of up to 100 taxa. Note that the entire agglomeration process and its reversal can be done in polynomial time. More specifically, in our implementation we take $O(n^4)$ time, which is optimal since the input consists of $7 \cdot \binom{n}{4}$ 4-splits on the set $\mathcal{X}$.

3.5 Consistency of FlatNJ

An important property that any method for constructing a split network should ideally satisfy is *consistency*. This means that if the method is designed to produce a split system with a certain special property (e.g., compatible or circular), then if such a split system (or associated data) is taken as input, the same split system should result. For example, if a compatible/circular weighted split system corresponding to a phylogenetic tree/outer-labeled planar network is taken as input to Neighbor-Joining/Neighbor-Net, then it can be shown that the split system will be reproduced (Atteson, 1999; Bryant *et al.*, 2007).

By construction, FlatNJ always generates a flat split system $\mathcal{S}$ on $\mathcal{X}$ with the following special recursive property: $\mathcal{S}$ contains at least one pair of taxa that are neighbors, and if any pair of neighbors in $\mathcal{S}$ is agglomerated then a new flat split system results that has at least one pair of neighbors and that has the same property. We call such flat split systems *neighborly* (note that there are flat split systems that are not neighborly). If $(\mathcal{S}, \omega)$ is a weighted flat split system, and FlatNJ is given $(\mathcal{F}, \lambda_{(\mathcal{S},\omega)})$ as input system of 4-splits, then it can be shown that it will reproduce $(\mathcal{S}, \omega)$ if any of the following hold:

a) $\mathcal{S}$ is compatible,
b) $\mathcal{S}$ is circular, or
c) $(\mathcal{S}, \omega)$ is a neighborly, full flat split system.

Note that both of the scoring functions σ_{min} and σ_{max} are necessary to achieve consistency of FlatNJ in (a)–(c). In particular, when used on its own, the scoring function σ_{min} can fail to select neighbors even in circular split systems. Similarly, even though σ_{max} will always select neighbors in circular split systems, used alone it can fail to select neighbors in neighborly flat split systems.

In general, although we have found that there are many non-full, neighborly flat split systems for which FlatNJ is consistent, there are also such split systems that FlatNJ cannot reproduce. Ideally, we would like to give a complete and concise description of those flat split systems for which FlatNJ is consistent. However, we expect that there might not be one since such a description would probably pave the way for a polynomial time algorithm to decide whether or not an arbitrary split system is flat, a problem that we strongly suspect to be NP-complete.

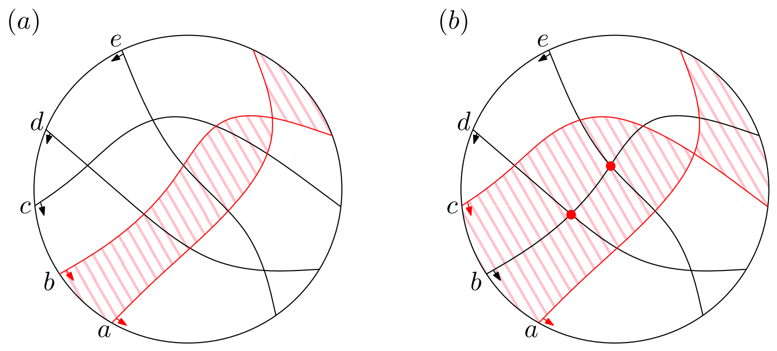


Figure 3.12: Neighbors in a loop-free, acyclic, rank three oriented matroid: (a) a and b are neighbors, (b) a and c are not neighbors because there are two crossings in between pseudolines corresponding to a and c: $b \times e$, $b \times d$.

3.6 Construction of oriented matroid splits

Equivalence between flat splits and splits of a loop-free, acyclic, rank three oriented matroid from Chapter 2 gives an interpretation of the FlatNJ algorithm from the perspective of oriented matroids. Here we define neighbors, explain initialization and expansion steps in terms of the oriented matroids, which are appealing for the implementation of FlatNJ as they are more abstract and easier to handle than pseudo-configurations of points.

3.6.1 Neighbors in oriented matroids

We designed FlatNJ algorithm to compute flat split systems as an oriented matroid of rank three, therefore we need to define neighbors in the same terms. We consider two taxa a and b to be *neighbors relative to a flat split system* $\mathcal{S}$ if no two pseudolines intersect in between pseudolines that correspond to a and b.

In terms of the wiring diagrams which are a graphical representation of the allowable sequences we define neighbors as two taxa x and x' whose respective wires have no crossings of other wires in between them. For example, a and b are neighbors in the wiring diagram depicted in Figure 3.12a whereas a and c are not neighbors in the same wiring diagram (Figure 3.12b).

3.6.2 Initialization of an oriented matroid

We initialize a flat split system by choosing one of the two loop-free, acyclic oriented matroid of rank three as shown in Figure 3.13. If initial flat split system has one of the trivial splits with weight 0, we choose an oriented matroid in Figure 3.13a and an oriented matroid in Figure 3.13b otherwise.

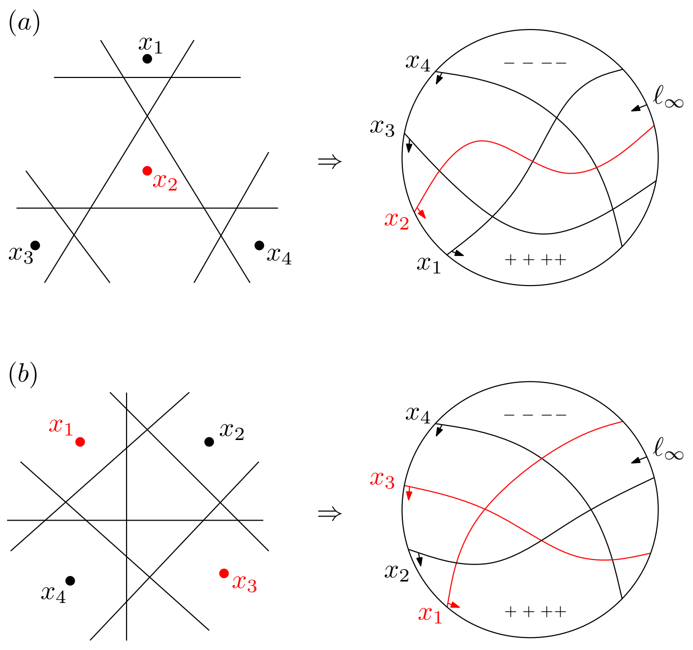


Figure 3.13: Two candidate oriented matroids for a flat split system on four taxa. (a) An arrangement $\mathcal{A}_1$ of oriented pseudolines that corresponds to a flat split system without the split $\{x_2\}|\{x_1, x_3, x_4\}$. (b) An alternative initial arrangement of pseudolines that gives a flat split system without the split $\{x_1, x_3\}|\{x_2, x_4\}$.

3.6.3 Expansion of an oriented matroid

Imagine that we are in the same situation as in Section 3.4.4. Just instead of the flat split system we have an loop-free, acyclic oriented matroid of rank three with an element x representing two neighbor elements x' and x''. We take x and split it into two taxa x' and x'' as illustrated in Figure 3.14. We double up the pseudoline x and place the copy either above or below the original pseudoline in such a way that none of the other pseudolines intersect in between the two copies of x. Then we rename x to x' and assign the other pseudoline to x'' (Figure 3.14b). Next we modify the new arrangement so that both of the new pseudolines intersect as well. We do this by crossing them in one of the cells bounded by pseudolines x' and x'' (except for the first cell as it corresponds to the same split as the last one). What oriented matroid we get depends on where we intersect x' and x'' as well as on the initial placement of x'' with respect to x'. At each expansion step we have $n - 2$ possible positions for intersecting x' and x'' where n is the number of elements in $\mathcal{X}'$ after x is expanded. All in all each expansion gives us $2(n - 2)$ possible new oriented matroids to chose from. Out of the $2(n - 2)$ potential

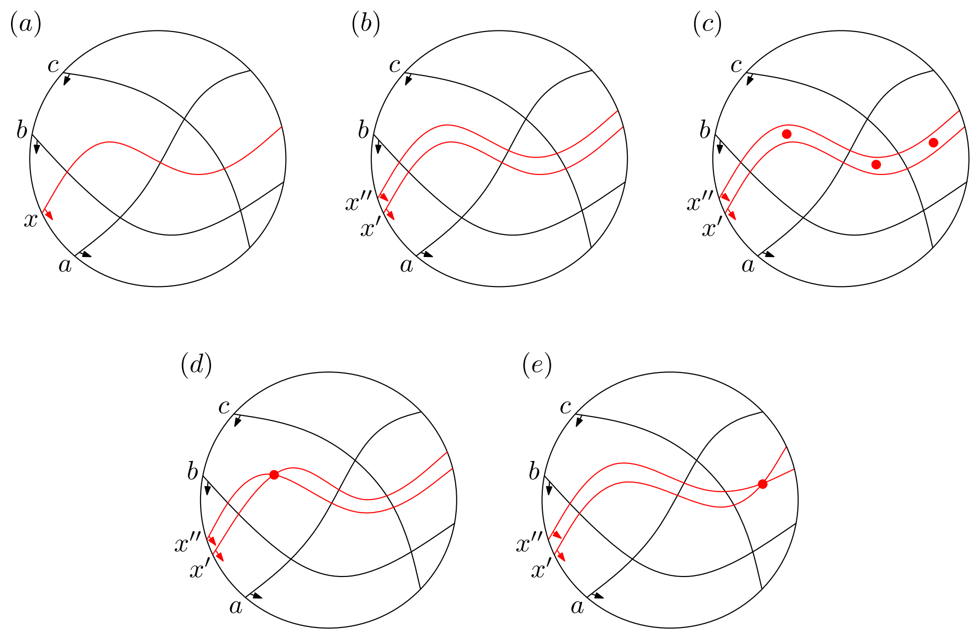


Figure 3.14: (a) An oriented matroid corresponding to a full flat split system in Figure 3.11a with x representing two agglomerated elements x' and x''. (b) Replacing pseudoline x with a pseudoline x' and adding another pseudoline x'' that is parallel to x'. Note that there are two ways to do that, i.e., x'' may be placed either above or below x'. (c) Red dots indicate cells in which x' and x'' could possibly intersect. (d) An oriented matroid that corresponds to flat split system in Figure 3.11d. (e) Choosing an alternative intersection point for x' and x'' gives a different split system as shown in Figure 3.11e.

(π, κ) we chose the one that maximizes the total recovered 4-split weight as given by Equation 3.3. In terms of the oriented matroids each step of reversing the agglomeration is actually a localization, i.e., expansion of the oriented matroid by a single element.

Chapter 4

Multidimensional scaling

4.1 Overview

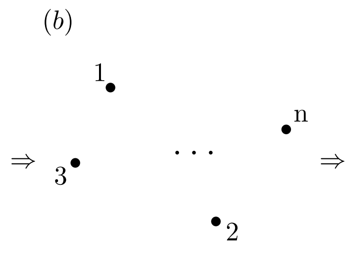
Multidimensional scaling (MDS) is a data visualization technique used for displaying high-dimensional data in a low-dimensional space. Methods for MDS are based on the pairwise proximities (similarities/dissimilarities) between objects in a high-dimensional space. In MDS solutions, objects are mapped to points in the low-dimensional space and proximities are represented as distances between the points (France and Carroll, 2011), see Figure 4.1. Multidimensional scaling was first introduced by Young and Householder (1938). A couple of decades later Torgerson (1952) proposed the algorithm for MDS which is now known as classical scaling.

Algorithms for multidimensional scaling work on proximities δ_{ij} which are derived from the object $\times$ stimuli (dimension) data using some suitable distance metric (France and Carroll, 2011). The metric may be distances between points in a high-dimensional space, dissimilarities between biologic sequences or any other source of distance data. Distances $D = \{d_{ij}\}$ in the MDS solution, on the other hand, are (usually) defined

(a)

	1	2	3	...	n
1	0	δ_{12}	δ_{13}	...	δ_{1n}
2	δ_{12}	0	δ_{23}	...	δ_{2n}
3	δ_{13}	δ_{23}	0	...	δ_{3n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	δ_{1n}	δ_{2n}	δ_{3n}	...	0

$\Rightarrow$



$\Rightarrow$

(c)

	1	2	3	...	n
1	0	d_{12}	d_{13}	...	d_{1n}
2	d_{12}	0	d_{23}	...	d_{2n}
3	d_{13}	d_{23}	0	...	d_{3n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	d_{1n}	d_{2n}	d_{3n}	...	0

Figure 4.1: A general multidimensional scaling scheme. (a) An $n \times n$ input proximity matrix Q which is used to compute (b) an MDS representation of n points. (c) An $n \times n$ distance matrix D that contains the (Euclidean) distances between points in (b).

as Euclidean distances between points. Differences between proximities and their corresponding distances give us a *representational error*:

$$e_{ij} = \delta_{ij} - d_{ij},$$

defined for each pair of objects i and j.

4.1.1 Stress

Metric multidimensional scaling methods aim at finding a solution such that distances d_{ij} are as similar to the input proximities δ_{ij} as possible. This similarity is measured using a function called stress which may differ depending on the method used. Consequently most of the multidimensional scaling algorithms compute an MDS embedding by iteratively minimizing the value of a chosen stress function. Here we discuss a few commonly used stress functions and give a graphical overview in Figure 4.2

The simplest stress function is *raw stress* (Kruskal, 1964, p. 8) which is equal to a sum of squared differences between input proximities δ_{ij} and output distances d_{ij}:

$$\sigma_r^2 = \sum_{i,j} [\delta_{ij} - d_{ij}]^2$$

Another type of stress function, which is optimized by the ALSCAL algorithm (Takane *et al.*, 1977), is known as *SStress*:

$$\sigma_s^2 = \sum_{i,j} \left[\delta_{ij}^2 - d_{ij}^2\right]^2 . \tag{4.1}$$

SStress is often preferred due to its connection to principal coordinate analysis (Mardia, 1978, p. 1234). However SStress emphasizes large dissimilarities and overlooks small (Borg and Groenen, 2005, p. 252).

Values of both σ_r^2 and σ_s^2 are themselves not very informative as they depend on the scale of the proximities. To overcome this dependence σ_r^2 is normalized by the sum of squared output distances, giving a *normalized stress*:

$$\sigma_1^2 = \frac{\sigma_r^2}{\sum d_{ij}^2} .$$

In practice σ_1^2 values tend to be very small and, as shown in Figure 4.2, not very sensitive for small differences between input proximities and output distances. Therefore Kruskal (1964) suggests using the square root of σ_1^2 which in the literature is referred to as *Stress-1* or just *Stress*, see Figure 4.2:

$$\sigma_1 = \sqrt{\sigma_1^2} \tag{4.2}$$

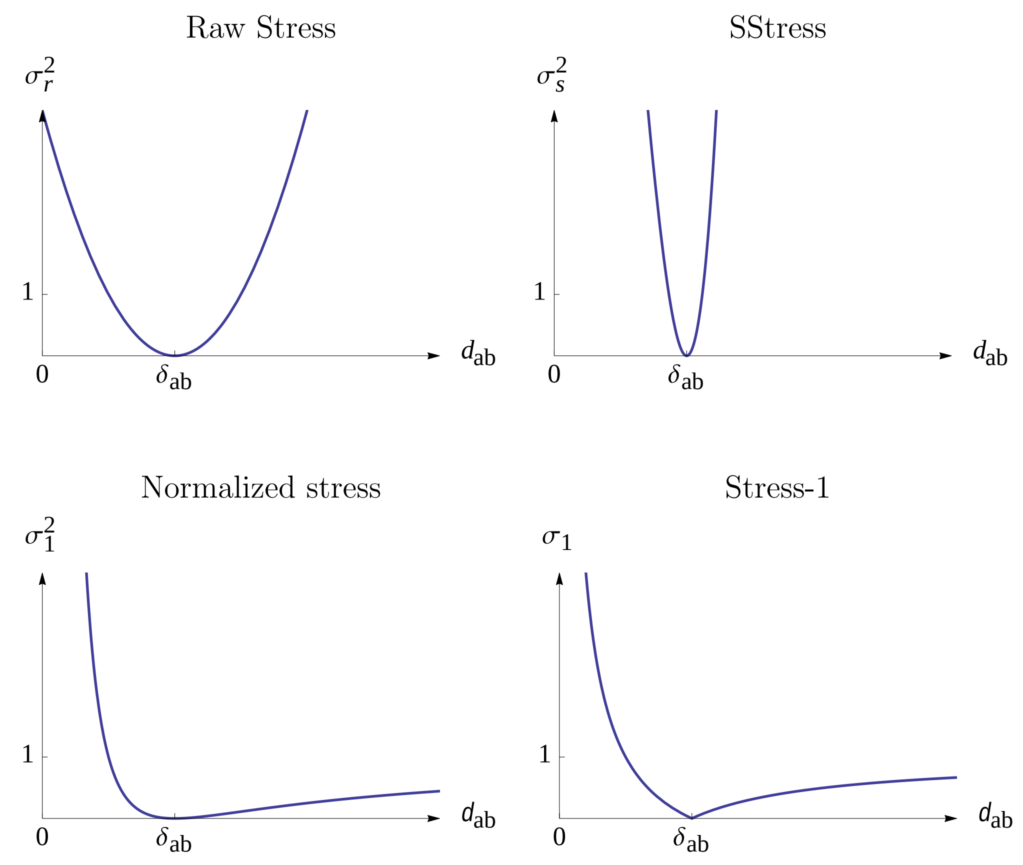


Figure 4.2: Graphs of four different stress functions (Raw stress, SStress, Normalized stress and Stress-1) for a (single) distance between objects a and b as a function of d_{ab}. All graphs are drawn with the same scale and a fixed value for δ_{ab}, they show how stress changes with respect to the low-dimensional distance d_{ab}. All of the stress functions equal zero when $d_{ab} = \delta_{ab}$. Raw stress and SStress are both dependent on the scale of measurement and equally penalize deviations to both sides of the high-dimensional distance δ_{ab}. Normalized stress and Stress-1 do not depend on the scale. Both approach one as d_{ab} grows bigger. Also very small values d_{ab} are heavily penalized, preventing solutions from collapsing into a single point. Normalized stress is less sensitive to small differences between δ_{ab} and d_{ab}. Stress-1 (σ_1) is preferred as its values are easier to discriminate.

In some cases, for example when dealing with missing or inaccurate data, another variation of the Stress-1 known as the *weighted stress* is used (France and Carroll, 2011):

$$\sigma_w = \sqrt{\frac{\sum_i \sum_{j \neq i} w_{ij}(\delta_{ij} - d_{ij})^2}{\sum_i \sum_{j \neq i} w_{ij} d_{ij}^2}}$$

To estimate the fit of individual points a *stress per point* is computed. Given the formula for stress and a point i, we extract only those terms of the summation involving i. For example raw stress for the point i equals:

$$\sigma_r^2(i) = \sum_{j \neq i} \left[\delta_{ij} - d_{ij}\right]^2 .$$

We mainly use stress σ_1 for the evaluation and a *weighted* variation of the raw stress for computations:

$$\sigma_{rw}^2 = \sum_{ij} w_i w_j \left(\delta_{ij} - d_{ij}\right)^2 . \tag{4.3}$$

4.2 Existing methods for multidimensional scaling

In this section we give a detailed review of the two most widely used multidimensional scaling techniques, classical scaling and iterative methods, and provide a brief overview of some of the other approaches.

4.2.1 Classical multidimensional scaling

Classical multidimensional scaling (cMDS) was the first practical method for MDS. It was introduced by Torgerson (1952, 1958) and Gower (1966) and is also known as Torgerson scaling or Torgerson-Gower scaling. Classical scaling is attractive since it can be solved analytically, requiring no iterations. Also, its solutions are nested, that is the first two dimensions of a 3D solution are the same as the two dimensions of a 2D solution (Borg and Groenen, 2005).

Classical scaling starts from the matrix of squared proximities $Q^{(2)} = \{\delta_{ij}^2\}$ and uses it to compute a B matrix:

$$B = -\frac{1}{2} J Q^{(2)} J,$$

where J is a centering matrix:

$$J = I - n^{-1}\mathbf{1}\mathbf{1}^T.$$

Entries of the matrix B can also be computed directly using the following formula:

$$b_{ij} = -\frac{1}{2}\left(\delta_{ij}^2 - \frac{1}{n}\sum_k \delta_{ik}^2 - \frac{1}{n}\sum_k \delta_{jk}^2 + \frac{1}{n^2}\sum_l \sum_k \delta_{lk}^2\right) \tag{4.4}$$

Let $B = V\Lambda V^T$ be an eigendecomposition of B, let Λ_+ be a matrix composed of first d positive eigenvalues and $V+$ corresponding columns of V. Then let X be $V_+\Lambda_+^{1/2}$. The rows of X define coordinates of the embedded objects in the d-dimensional space.

Mardia (1978) showed that classical scaling solutions are optimal with respect to the matrix B as they minimize

$$\sum_i \sum_j (b_{ij} - \hat{b}_{ij})^2 = tr(B - \hat{B})^2$$

where $\hat{B}$ is a B matrix for the output distances. If the $\{\delta\}$ matrix is Euclidean then an even stronger optimality condition holds: Classical scaling solutions minimize the SStress σ_s (Formula 4.1) for the Euclidean proximity matrices (Mardia, 1978).

Classical scaling solutions are sometimes used as a starting configuration for iterative MDS algorithms minimizing different stress functions (Malone *et al.*, 2002). For example, PROXCAL (Busing *et al.*, 1997) lets user choose classical (Torgerson) scaling to generate the initial configuration. However, the computational complexity of the classical scaling is dominated by the $O(n^3)$ complexity of eigendecomposition calculation (Yang *et al.*, 2006).

4.2.2 Spring system algorithms

Another class of multidimensional scaling methods treat the embedding problem by converting it into an equivalent spring system and iteratively minimizing the chosen stress function. Pairs of points affect each other through the connecting springs as shown in Figure 4.3. If $\delta_{ij} < d_{ij}$ then the corresponding spring is stretched and a force pulls one point towards the other whereas in case $\delta_{ij} > d_{ij}$ the spring is compressed and therefore pushes points away from each other. In other words the force with which some point p_i affects another point p_j is proportional or equal to the difference between δ_{ij} and d_{ij} (Chalmers, 1996; France and Carroll, 2011). In case both input (δ) and output (d) distances are equal, the spring is at its rest, i.e., optimal length.

In the original version of the algorithm used for graph drawing by Eades (1984), all vertices (points) are initially placed at random positions and then the spring system iteratively moves them to the equilibrium (locally optimal) positions. At each iteration, all points are considered and the force for each point is computed as a function of forces coming from all other points. Therefore each iteration has a complexity of $O(n^2)$. The number of iterations used often depends on the size of the data, making the overall complexity $O(n^3)$ (Morrison *et al.*, 2003). To make this approach applicable to large data sets Chalmers (1996) and Morrison *et al.* (2003) came up with modifications that significantly reduce the computational complexity of the original algorithm.

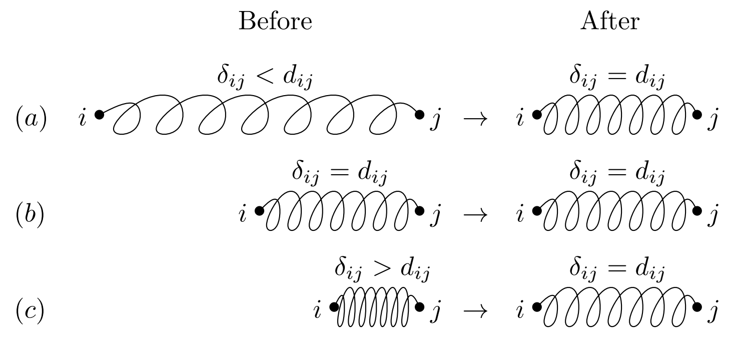


Figure 4.3: Embeddings of two points i and j as spring systems. (a) i and j are placed too far from each other. Original proximity δ_{ij} is lower than low-dimensional distance thus the connecting spring is stretched. After the system is let go, the spring contracts making d_{ij} equal to δ_{ij}. (b) Points are placed in such a way that $d_{ij} = \delta_{ij}$, therefore the spring is at rest and does not move; (c) i and j are too close to each other and the spring is compressed. After letting go, it expands as points repulse each other via the spring.

The first modification, suggested by Chalmers (1996), reduces the overall complexity to $O(n^2)$. In this variation of the algorithm, each point i is connected by springs to a certain subset $S_i^* \subset S$ of points rather than all of them. The subset S_i^* is different for each point and consists of its k nearest neighbors and r random points. Nearest neighbors are found in the input data and do not change over the iterations, whereas random points are resampled each time. The complexity of each iteration depends on the sizes of the sets S_i^*. When $|S_i^*|$ is bounded by a constant for all i, each iteration runs in linear time. Chalmers (1996) experimented with different numbers of neighbors and random points and found that relatively small sizes such as 5 neighbors and 10 random points work just as well as, for example, 20 and 30. However they only experimented with a relatively small data set of 831 objects, hence small neighborhood and random samples might actually depend on the size of the whole data set as, for example, $\sqrt[4]{831} \approx 5.4$.

Morrison *et al.* (2003) introduced a hybrid method that uses spring systems as well as an interpolation technique. It starts by sampling a subset S of $\sqrt{n}$ objects and running Chalmers (1996) algorithm on S as explained above. This gives $O(\sqrt{n}^2) = O(n)$ complexity of the initial step. The remaining $n - \sqrt{n}$ points are then placed using an *interpolation* technique. For each object in $i \notin S$, we find an object $x \in S$ such that δ_{ix} is minimum and draw a circle around x with radius r proportional to δ_{ix}, then determine which quadrant is most likely to be best for positioning i and perform a binary search on that quadrant to find the most suitable initial position i_s for i. A

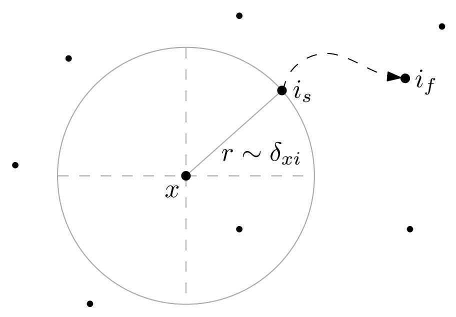


Figure 4.4: A point i is initially placed at the position i_s on the circle around another object x. Then forces coming from other points (not only x) iteratively move it from i_s through some intermediate positions to its final position i_f (Morrison *et al.*, 2003, Figure 2).

random sample $S' \subseteq S$ of points is selected and $i \cup S'$ is embedded into a spring system which moves i to its best position with respect to S', see Figure 4.4. Each object that does not belong to the initial random sample S is compared to all $\sqrt{n}$ objects in S, thus the complexity of the algorithm is $O(n\sqrt{n})$ as other operations are performed in constant time. Later it was reduced to $O(n\sqrt[4]{n})$ with the use of pivots for the nearest neighbor search (Morrison and Chalmers, 2003). Additionally, to refine the final solution a constant number iterations of the Chalmers (1996) algorithm are run. This does not influence the overall complexity as each iteration of Chalmers (1996) algorithm has linear runtime complexity.

4.2.3 Other methods for MDS

The two methods that we have just described are just a few out of many approaches to the multidimensional scaling problem. There are a lot of algorithms targeting different aspects of the MDS. A good review can be found in France and Carroll (2011). Here we briefly review a few commonly used approaches.

Iterative stress minimization

Sammon (1969) developed a very successful iterative method for non-linear dimensionality reduction. It starts with a random initial point configuration and iteratively moves points using steepest descent optimization.

Another class of somewhat similar methods for iterative stress minimization are *majorization algorithms*. Stress functions are quite difficult to minimize due to their complicated derivatives. Majorization approaches do not work with actual stress functions, instead they minimize a simpler auxiliary functions (Borg and Groenen, 2005, Chapter 8). Let $f(x)$ be a stress function and let $g(x, z)$ be a function such that

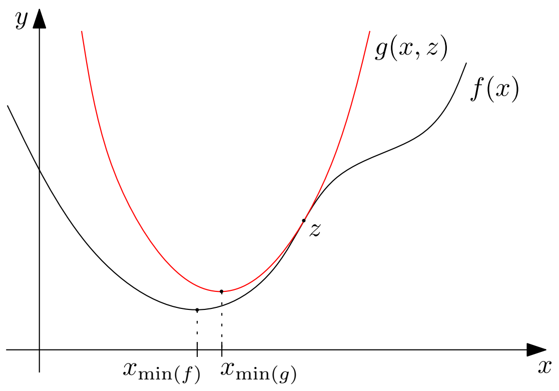


Figure 4.5: Function $g(x,z)$ is a majorizing function of $f(x)$ that touches $f(x)$ at the point z and for all $x : g(x,z) \geq f(x)$.

$g(x,z) \geq f(x)$ and $f(z) = g(z,z)$ as shown in Figure 4.5. Then majorization works as described in Algorithm 4.1.

Algorithm 4.1 Majorization

```
1: $z \leftarrow z_0$
2: $x' \leftarrow \text{argmin}_x(g(x,z))$
3: if $f(z) - f(x') < \varepsilon$ then
4:     return $x'$
5: else
6:     $z \leftarrow x'$
7:     goto 2
8: end if
```

Algorithms that are based on the majorization technique are SMACOF (De Leeuw, 1977; De Leeuw and J., 1977) and PROXSCAL (Busing *et al.*, 1997).

Local scaling

Local scaling methods focus on the good embeddings of the neighborhoods by emphasizing small distances over large. Locally linear embedding (LLE) by Roweis and Saul (2000) computes low-dimensional embeddings by preserving neighborhoods. The global structure of the data is determined from the local symmetries. Laplacian Eigenmaps is another local scaling approach proposed by Belkin and Niyogi (2003). It finds embeddings which optimally preserve local neighborhoods by using the Laplacian of the neighborhood graph.

Very high-dimensional data

Very high-dimensional data suffers from a phenomena known as the *curse of dimensionality.* As the dimensionality grows, relative differences between lowest and highest proximities become smaller due to the concentration of measure phenomenon (Donoho, 2000), see Figure 4.6. Some methods for MDS are designed to deal with this issue specifically. For example, DD-HDS (Lespinats *et al.*, 2007) uses a specific weighting scheme that emphasizes small proximities even when the difference between small and large proximities is relatively minor. They use a function based on the Gaussian probability density function f with mean μ and standard deviation σ (Lespinats *et al.*, 2007, Formula 6). The weight for proximity between i and j is given

$$w_{ij} = k(\min(\delta_{ij}, d_{ij})) = 1 - \int_{-\infty}^{\min(\delta_{ij},d_{ij})} f(u, \mu, \sigma)du.$$

The $\min(\delta_{ij}, d_{ij})$ ensures that weighting is symmetric and emphasizes small proximities as well as small low-dimensional distances. The stress function is (Lespinats *et al.*, 2007, Formula 7):

$$\sigma_{DD-HDS} = \sum_{i<j} |d_{ij} - \delta_{ij}| \left(1 - \int_{-\infty}^{\min(\delta_{ij},d_{ij})} f(u, \mu, \sigma)du\right).$$

Absolute difference between proximities and low-dimensional distance is used to avoid over representing large proximities which usually result in large differences. (Lespinats *et al.*, 2007).

Nonmetric methods

Unlike approaches discussed above that rely on the proximities themselves, non-metric methods focus on the order of the proximities instead. For example if we have three objects a, b and c with pairwise proximities such that $\delta_{ab} < \delta_{ac} < \delta_{bc}$, then nonmetric methods will try to preserve this order and find embeddings for which $d_{ab} < d_{ac} < d_{bc}$ holds. One of the nonmetric scaling methods is RankVisu by Lespinats *et al.* (2009). It ranks neighborhoods of the points and aims at preserving this ranking in a low-dimensional space as much as possible.

4.3 A new agglomerative approach

In general the task of multidimensional scaling is very similar to that of distance-based phylogenetics. The objective of MDS is to find a configuration of points that best fits given proximity matrix Q. Meanwhile in distance-based phylogenetics one wants

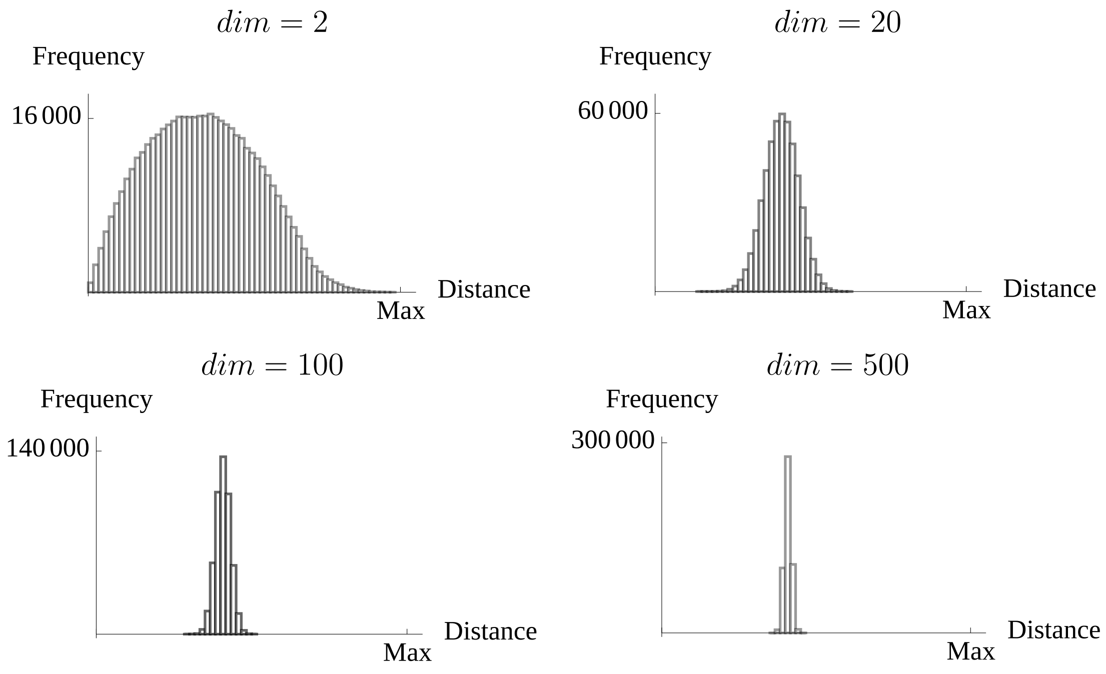


Figure 4.6: Concentration of measure phenomenon. We generated random data sets of 1000 uniformly distributed points in 2, 20, 100 and 500 dimensions and computed pairwise distances between all pairs of points. Histograms show the distribution of distances for each dimension. Maximum distance is the square root of the dimension.

to estimate a network or a tree structure that best resembles dissimilarities between biological sequences. In both cases we want to fit some sort of "distances" between objects on some geometric/topological structure. Inspired by this similarity we apply the agglomerative approach (used in e.g. NeighbourJoining (Saitou and Nei, 1987), NeighbourNet (Bryant and Moulton, 2004), FlatNJ (Chapter 3, Balvočiūtė *et al.*, 2014) and hierarchical clustering) to multidimensional scaling. Agglomerative methods for trees are flexible, produce high quality trees and are computationally efficient even for large data sets (Walter *et al.*, 2008). At each agglomeration two objects are joined into one, thus the variance is reduced and accuracy is increasing at every step.

Our algorithm operates either on a proximity matrix $\{\delta\}$ or directly on objects in a high-dimensional space. Low-dimensional embeddings are then computed by repeating the following steps:

1. Neighbor identification and agglomeration,
2. Initial embedding, and
3. Expansion or reversal.

See Algorithm 4.2 for more details. The actual implementation of each step depends on the chosen low-dimensional space and type of the input data. In the following we explain our algorithm for embedding points in one and then in two dimensions.

Algorithm 4.2 Agglomerative MDS

```
1: procedure Embedd
2:     if $n = d + 1$ then
3:         return InitEmbedding        ▷ initialize embedding in a $d$-dimensional space
4:     end if
5:     $\{a, b\} \leftarrow$ FindNeighbors
6:     $c \leftarrow$ Agglomerate($a, b$)
7:     Embedd
8:     $\mathcal{P} \leftarrow$ Expand($\mathcal{P}, c, a, b$)        ▷ expand $c$ into $a$ and $b$
9:     return $\mathcal{P}$
10: end procedure
```

4.4 Embeddings on a real line

Scaling in one dimension is usually referred to as *unidimensional scaling* or *seriation* and it is a specific case of the multidimensional scaling where output also gives an order of objects. Thus it can be viewed more like a problem of ordering rather than embedding. Unidimensional scaling originates from archeology (Petrie, 1899; Robinson, 1951) where chronological order is of interest. Since then it has been related to the consecutive ones problem in binary matrices (Kendall, 1969) and used for the representation of graphs on the real line (Lekkeikerker and Boland, 1962). The assembly of DNA sequences is another interesting application of unidimensional scaling (Garriga *et al.*, 2011) as genes admit a linear (or circular) ordering in the chromosomes.

Essentially unidimensional scaling is a combinatorial problem of finding the optimal ordering. Using exhaustive search, exact solutions can only be found for very small data sets as the number of possible orderings grows exponentially with respect to the size of the data set (Buchta *et al.*, 2008).

4.4.1 Neighbor identification

First we introduce a definition of neighbors on a real line and then formulate criterion that could be used for the neighbor identification.

Let ℓ be a real line, and let $P \subseteq \ell$ be a set of points on ℓ with x_i the coordinate of the point i on ℓ. We define neighbors as follows.

Definition 4.4.1. Two points p' and p'' are *neighbors* in P if for all $p_i \in P \setminus \{p', p''\}$ either $x_i \leq x_{p'}$ and $x_i \leq x_{p''}$ or $x_i \geq x_{p'}$ and $x_i \geq x_{p''}$.

In other words two points p' and p'' are neighbors if none of the other points in P lie between them on ℓ; see Figure 4.8a.

We formulate our criterion for selecting neighbors based on the observation that a pair of closest points on the line are always neighbors by Definition 4.4.1. Thus a pair p' and p'' for which

$$\delta_{p'p''} = \min_{x,y \in P} \delta_{xy} \tag{4.5}$$

holds is selected as neighbors. We experimented with some other formulations of the criterion which were computationally more demanding and proved to have the same accuracy with respect to the final result, for more details see Appendix B.

Solving 4.5 exactly takes $O(n^2)$ operations in each iteration making overall complexity $O(n^3)$. To reduce the complexity we relax the criterion and look for the reciprocal nearest neighbors instead. Benzécri (1982) suggested a single cluster algorithm for hierarchic clustering based on the nearest neighbor chains (NN-chain) which are defined as follows:

$$i,\ NN(i) = j,\ NN(j) = k,\ \ldots,\ NN(p) = q,\ NN(q) = p. \tag{4.6}$$

NN-chains are characterized by three propositions (Murtagh, 1984):

1. Inter-object dissimilarities are monotonically decreasing along the NN-chain (see Figure 4.7).

2. The final link always connects a pair of reciprocal nearest neighbors (RNNs), i.e., the final link connects two objects that are each other's nearest neighbors.
3. The NN-chain does not contain a circuit of more than two nodes.

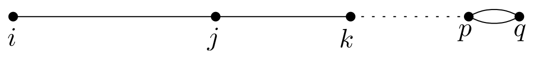


Figure 4.7: Nearest neighbor chain (NN-chain). See Equation 4.6.

The single cluster algorithm starts growing a NN-chain from some random point, once a pair of reciprocal nearest neighbors is found, it is agglomerated into one point. The algorithm continues from the point that preceded the pair on the NN-chain or from a random point if the chain is empty. See Algorithm 4.3 for more details. Implementation of the agglomeration procedure (line 7) depends on the type of the input data and we explain it in the following section.

Algorithm 4.3 Single cluster algorithm (Murtagh, 1984, Algorithm B)

```
 1: $NNchain \leftarrow \emptyset$
 2: while $\mathcal{X} \neq \emptyset$ do
 3:     $r \leftarrow$ RANDOM($\mathcal{X}$)
 4:     PUSH($NNchain, r$)
 5:     $p \leftarrow$ NEARESTNEIGHBOR($\mathcal{X}, r$)        ▷ finds nearest neighbor for $r$ in $\mathcal{X}$
 6:     if CONTAINS($NNchain, p$) then
 7:         $c \leftarrow$ AGGLOMERATE($r, p$)
 8:         $\mathcal{X} \leftarrow \mathcal{X} \cup c \setminus \{r, p\}$
 9:         POP($NNchain$)                        ▷ removes $r$ from the $NNchain$
10:         POP($NNchain$)                        ▷ removes $p$ from the $NNchain$
11:         if $NNchain = \emptyset$ then
12:             goto 3
13:         else
14:             $r \leftarrow$ PEEK($NNchain$)
15:             goto 5                        ▷ returns last element without removal
16:         end if
17:     else
18:         $r \leftarrow p$
19:         goto 4
20:     end if
21: end while
```

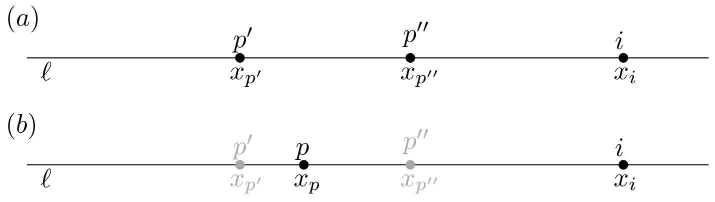


Figure 4.8: Agglomeration of points on the line. (a) Two points p' and p'' are identified as neighbors and (b) they are joined into p.

4.4.2 Agglomeration

Once a pair of neighbors p' and p'' are selected they are merged together into a new object p as if they were lying on some real line ℓ as shown in Figure 4.8. Let $\{w_i\}$ be a set of weights and initially set $w_i = 1$ for all points $i \in P$. When the input data is a proximity matrix Q, we update it by adding a row and a column for the new proximities between all objects $i \in Q \setminus \{p', p''\}$ and p which are computed as follows

$$\delta_{ip} = \frac{w_{p'}\delta_{ip'} + w_{p''}\delta_{ip''}}{w_{p'} + w_{p''}}. \tag{4.7}$$

The proximity matrix Q is then reduced by removing all rows and columns that correspond to p' and p''.

If the input is not a matrix of proximities, but instead vectors in high-dimensional space, we remove p' and p'' from the set and assign p a weighted average of the coordinates of p' and p'':

$$x_p(i) = \frac{w_{p'}x_{p'}(i) + w_{p''}x_{p''}(i)}{w_{p'} + w_{p''}}. \tag{4.8}$$

Here $x^p(i)$ is the coordinate of p in the i-th dimension. The weight w_p of the new point p is the sum of the weights of p' and p'':

$$w_p = w_{p'} + w_{p''}. \tag{4.9}$$

That is, weight of some point i is equal to the sum of weights of all points that were agglomerated into i. When we start with the unit weights, then w_i equals the number of points that are contained in i.

Neighbor identification and agglomeration is repeated until only two objects remain. These are placed on a real line ℓ and the whole agglomeration process is reversed.

4.4.3 Initial positioning

Let a and b be the remaining two objects in Q. Map a on ℓ with $x_a = 0$ and b with $x_b = \delta_{ab}$ as shown in Figure 4.9. This gives a perfect fit for two points because distance $d_{ab} = |x_a - x_b| = \delta_{ab}$, thus stress is equal to zero independent of the stress formula used.

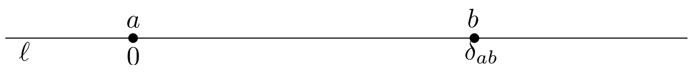


Figure 4.9: Initial positioning of two points a and b on a real line ℓ.

4.4.4 Expansion

Expansion works by separating points in the reverse order that they were agglomerated. At each expansion step we select the last agglomerated point p and translate all points on ℓ so that p appears at the origin:

$$x_i = x_i - x_p.$$

The point p is expanded into two points p' and p'' with coordinates $x_{p'}$ and $x_{p''}$ such that

$$x_{p'} \frac{w_{p'}}{w_{p'} + w_{p''}} = -x_{p''} \frac{w_{p''}}{w_{p'} + w_{p''}},$$

hence

$$x_{p''} = -x_{p'} \frac{w_{p'}}{w_{p''}}.$$

We compute coordinates $x_{p'}$ and $x_{p''}$ by minimizing the weighted raw stress σ_{rw}. Let $x_{p'} = x$ and $x_{p''} = -x \frac{w_{p'}}{w_{p''}}$, then weighted raw stress equals:

$$\begin{aligned} \sigma_{rw} = &\sum_i \left[w_i w_{p'} \left(\delta_{ip'} - |x_i - x|\right)^2 + w_i w_{p''} \left(\delta_{ip''} - |x_i + x\frac{w_{p'}}{w_{p''}}|\right)^2 \right] \\ &+ w_{p'} w_{p''} \left(\delta_{p'p''} - \left(\frac{w_{p'}}{w_{p''}} + 1\right) |x|\right)^2 \end{aligned}$$

We only consider distances to points p' and p'' as other pairwise distances are not affected by the expansion of p. The stress is minimum when

$$\frac{\delta \sigma_{rw}}{\delta x} = 0.$$

To reduce the number of norms in the derivative, we take advantage of the fact that we know the position of each point in $P \setminus \{p', p''\}$ with respect to p. That is, for each point i we know whether $x_i < x_p$, $x_i = x_p$ or $x_i > x_p$. Thus, we divide $P \setminus \{p', p''\}$ into two subsets $L = \{l \in P : x_l \leq x_p\}$ and $M = \{m \in P : x_m > x_p\}$ and express stress σ_{rw} as:

$$\begin{aligned}\sigma_{rw} =& \sum_{l\in L}\left[w_l w_{p'}\left(\delta_{lp'}-(x-x_l)\right)^2 + w_l w_{p''}\left(\delta_{lp''}-\left(-x\frac{w_{p'}}{w_{p''}}-x_l\right)\right)^2\right]\\ &+\sum_{m\in M}\left[w_m w_{p'}\left(\delta_{mp'}-(x_m-x)\right)^2 + w_m w_{p''}\left(\delta_{mp''}-\left(x_m+x\frac{w_{p'}}{w_{p''}}\right)\right)^2\right]\\ &+w_{p'}w_{p''}\left(\delta_{p'p''}-\left(\frac{w_{p'}+w_{p''}}{w_{p''}}\right)|x|\right)^2 \end{aligned} \tag{4.10}$$

Derivative of the stress (Equation 4.10) with respect to the coordinate x is

$$\begin{aligned}\frac{\delta\sigma_{rw}}{\delta x} =& x\underbrace{\left(\frac{w_{p'}}{w_{p''}}w_p(w_p+1)\sum_{i\in L\cup M} w_i\right)}_{\text{a}}+\\ &\underbrace{+\,w_{p'}\left(\sum_{l\in L} w_l\left(\delta_{lp''}-\delta_{lp'}\right)-\sum_{m\in M} w_m\left(\delta_{mp''}-\delta_{mp'}\right)\right)}_{\text{b}}+\\ &-sign(x)\underbrace{\left(w_{p'}w_p\delta_{p'p''}\right)}_{\text{c}}.\end{aligned}$$

Assuming that $x \neq 0$ we get that $\frac{\delta\sigma_{rw}}{\delta x}=0$ has two solutions:

$$\begin{aligned}x_1 =& \frac{-b-c}{a},\\ x_2 =& \frac{-b+c}{a},\end{aligned}$$

Out of x_1 and x_2 we choose the one that minimizes the weighted raw stress.

4.4.5 Optimal weighting and local optimality condition

Once a point p is expanded into p' and p'' and their respective coordinates are chosen such that stress over p' and p'' is minimal, the overall stress for all other points is no longer guaranteed to be minimum. To account for that we explore local optimality criterion and its application for weighted points.

Pliner (1984) has shown that the local minimum condition for some ordering of points on a line is reached if and only if

$$x_i = \frac{1}{n}\sum_{j=1}^{n}\delta_{ij}sign(x_i-x_j) \tag{4.11}$$

for all $i = 1, \ldots, n$ and $x_i \neq x_j$ for all $i \neq j$.

We modify this condition for points with weights. To obtain a weighted version of the equation 4.11 we take the weighted raw stress

$$\sigma_{rw} = \sum_{i<j} w_i w_j (\delta_{ij} - |x_i - x_j|)^2$$

and modify it to include a sign function instead of a norm:

$$\sigma_{rw} = \sum_{i<j} w_i w_j (\delta_{ij} - (x_i - x_j) sign(x_i - x_j))^2 \tag{4.12}$$

To find a condition for a local minimum we compute the derivative of 4.12 with respect to x_i:

$$\frac{\delta \sigma_{rw}}{\delta x_i} = -2w_i \sum_j sign(x_i - x_j) w_j (\delta_{ij} - (x_i - x_j) sign(x_i - x_j))$$

And find when $\frac{\delta \sigma_{rw}}{\delta x_i} = 0$:

$$\begin{aligned} -2w_i \sum_j sign(x_i - x_j) w_j (\delta_{ij} - (x_i - x_j) sign(x_i - x_j)) &= 0 \\ \sum_j [sign(x_i - x_j) w_j \delta_{ij} - w_j (x_i - x_j)] &= 0 \\ \sum_j w_j \delta_{ij} sign(x_i - x_j) - x_i \sum_j w_j + \sum_j w_j x_j &= 0 \end{aligned}$$

From here we can obtain the value for x_i:

$$\begin{aligned} x_i \sum_j w_j &= \sum_j w_j \delta_{ij} sign(x_i - x_j) + \sum_j w_j x_j \\ x_i &= \frac{1}{\sum_j w_j} \sum_j w_j \delta_{ij} sign(x_i - x_j) + \frac{1}{\sum_j w_j} \sum_j w_j x_j \end{aligned}$$

The second term $\frac{1}{\sum_j w_j} \sum_j w_j x_j$ does not depend on x_i and can be ignored, hence a local minimum condition for the weighted case is:

$$x_i = \frac{1}{\sum_j w_j} \sum_j w_j \delta_{ij} sign(x_i - x_j) \tag{4.13}$$

We adjust coordinates after each expansion to make sure that local optimality condition is satisfied at every stage.

Preserving local optimality on expansion

Ideally, we would like to perform an expansion in such a way that optimality is not violated after each step. To see whether and in what cases it is possible to achieve this stability we look into the criterion for local optimality (4.13) and explore when it does not change for all points $i \in P \setminus \{k\}$ after p is expanded into p' and p''. Because the weight of each point equals the number of objects contained in it we have that :

$$\begin{aligned} w_p &= w_{p'} + w_{p''} \\ \delta_{ip} &= \frac{w_{p'}}{w_p}\delta_{ip'} + \frac{w_{p''}}{w_p}\delta_{ip''}. \end{aligned} \tag{4.14}$$

When the expansion is performed in such a way that does not change the order of the points we also have that:

$$sign(x_i - x_p) = sign(x_i - x_{p'}) = sign(x_i - x_{p''}) \tag{4.15}$$

If the optimality criterion is satisfied after the expansion then the term including point p in the equation 4.13 must be equal to the sum of terms including p' and p'':

$$w_p \delta_{ip} sign(x_i - x_p) = w_{p'}\delta_{ip'} sign(x_i - x_{p'}) + w_{p''}\delta_{ip''} sign(x_i - x_{p''}) \tag{4.16}$$

When we incorporate equations 4.14 and 4.15 into 4.16, we get:

$$\begin{aligned} w_p(\frac{w_{p'}}{w_p}\delta_{ip'} + \frac{w_{p''}}{w_p}\delta_{ip''}) &= w_{p'}\delta_{ip'} + w_{p''}\delta_{ip''} \\ w_{p'}\delta_{ip'} + w_{p''}\delta_{ip''} &= w_{p'}\delta_{ip'} + w_{p''}\delta_{ip''} \end{aligned} \tag{4.17}$$

From 4.17 we see that if 4.16 holds, the local optimality condition is preserved after each expansion. Consider another case when we allow points to change order. That is, if after expansion we get that for some point i

$$sign(x_i - x_p) = -sign(x_i - x_{p'}). \tag{4.18}$$

Then local optimality is preserved if:

$$\begin{aligned} w_p \delta_{ip} sign(x_i - x_p) &= w_{p'}\delta_{ip'} sign(x_i - x_{p'}) + w_{p''}\delta_{ip''} sign(x_i - x_{p''}) \\ w_{p'}\delta_{ip'} + w_{p''}\delta_{ip''} &= -w_{p'}\delta_{ip'} + w_{p''}\delta_{ip''} \\ 2w_{p'}\delta_{ip'} &= 0 \end{aligned} \tag{4.19}$$

From 4.19 we get that 4.18 holds and the optimality criterion is preserved is when:

$$w_{p'} = 0 \text{ or } \delta_{ip'} = 0.$$

4.4.6 Implementation and computational complexity

The computational complexity of agglomeration depends on the clustering algorithm and input update. The reduction formula (4.7) satisfies the reducibility condition of Murtagh (1984), so neighbor finding is based on the single cluster algorithm with complexity $O(n^2)$ for proximity matrices and $O(n^2k)$ for objects in k-dimensional space. Complexity of the input update also depends on the type of the data we are using. It is $O(n)$ per iteration for a proximity matrix and $O(k)$ for k-dimensional objects, overall complexities are $O(n^2)$ and $O(nk)$ respectively. To sum up, runtime complexity of the agglomeration is $O(n^2)$ for proximity matrix and $O(n^2k)$ for k-dimensional objects.

Initial embedding is straightforward and requires a constant number of operations for proximity matrices and $O(k)$ operations for k-dimensional objects.

The most computationally expensive step is the expansion. For the proximity matrix it requires $O(n)$ operations to compute coordinates for the expanded points and another $O(n)$ to evaluate both solutions. Applying local optimality criterion requires $O(n)$ operations per point, hence $O(n^2)$ per iteration and $O(n^3)$ overall. Therefore runtime complexity of the expansion is $O(n^3)$ for proximity matrix and $O(n^3k)$ for k-dimensional objects. By tracking the change of locally optimal coordinates we could reuse the optimal coordinates and adjust them with respect to the expanded points only. This would reduce the complexity to $O(n^2)$ and $O(n^2k)$.

4.5 Embedding points in the two-dimensional space

After exploring how agglomeration works for a relatively simple unidimensional case we consider a more complicated problem of scaling into two-dimensional spaces. Here we focus on the computational complexity as well as accuracy. To keep runtime as low as possible different neighbor finding strategies are used for the proximity matrices and for high-dimensional objects.

4.5.1 Neighbor identification and agglomeration

We agglomerate points in two dimensions in the same manner as we did for unidimensional case, i.e., we take two points p' and p'' that we choose as neighbors and replace them with a new point p that is placed between p' and p'' as shown in Figure 4.10. The actual position of the p depends on the weights of $w_{p'}$ and $w_{p''}$.

We use the same formulas for updating weights and coordinates as we did in the unidimensional case (Formula 4.9 and Formula 4.8 accordingly). However, for the distance update we can no longer take the weighted average. Instead we take a weighted mean (Gower, 1967a) as it is done in the weighted mean-pair method by Sokal and

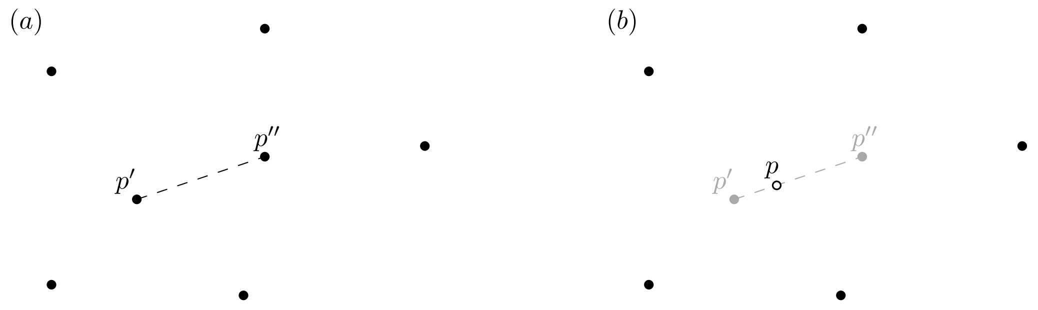


Figure 4.10: Agglomeration in two dimensions. (a) A set of points before the agglomeration of p' and p''. (b) Points p' and p'' are agglomerated into p that is placed in between p' and p''.

Michener (1958). This approach is often referred to as *median linkage* clustering by Gower (1967a).

We compute new distances as follows. Let $\mathcal{P}$ be the set of points and let $D = \{d\}$ be the pairwise distance matrix on $\mathcal{P}$. Let $p', p'' \in \mathcal{P}$ be neighbors and let $i \in \mathcal{P} \setminus \{p', p''\}$ be any other point in $\mathcal{P}$, then distance d_{ip} equals the length of the *weighted median* of the triangle formed by p', p'' and i:

$$d_{ip} = \sqrt{\frac{w_{p'}}{w_p} d^2_{ip'} + \frac{w_{p''}}{w_p} d^2_{ip''} - \frac{w_{p'} w_{p''}}{w^2_p} d^2_{p'p''}},$$

see Figure 4.11. We substitute distances d to proximities δ and use the following formula to update proximity matrix on agglomeration:

$$\delta_{ip} = \sqrt{\frac{w_{p'}}{w_p} \delta^2_{ip'} + \frac{w_{p''}}{w_p} \delta^2_{ip''} - \frac{w_{p'} w_{p''}}{w^2_p} \delta^2_{p'p''}}, \tag{4.20}$$

with $w_p = w_{p'} + w_{p''}$ as in the Formula 4.9.

The single cluster algorithm that we used for the unidimensional scaling requires the agglomeration formula to satisfy the *reducibility* property which requires that

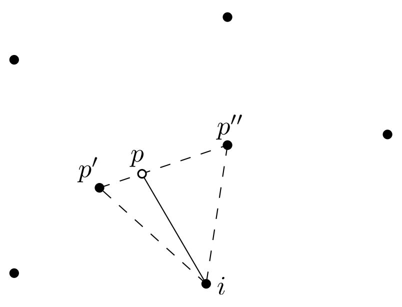


Figure 4.11: Agglomeration of p' and p'' into p. Distance between i and p is equal to the length of the weighted median ip of the triangle $p'p''i$.

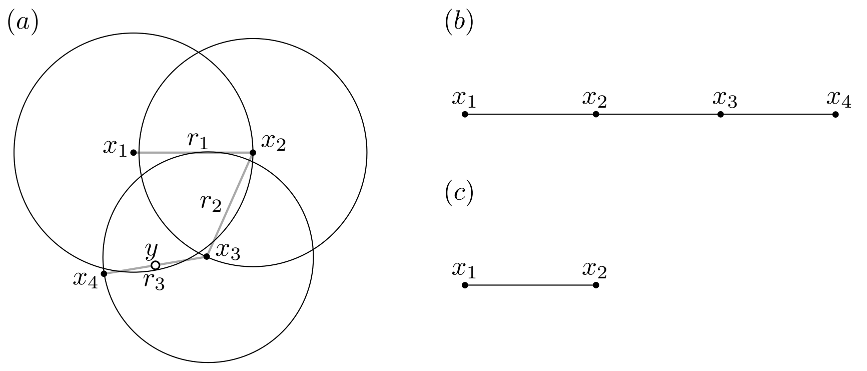


Figure 4.12: Single cluster algorithm applied to points in 2D space. (a) A set of four points $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ in a two-dimensional space. Radii of circles around the points are such that $r_1 > r_2 > r_3$. (b) A nearest neighbor chain with x_1 as a starting point. Points x_3 and x_4 are the reciprocal nearest neighbors, that are joined into y as shown in (a). (c) NN-chain after the agglomeration. If we continue from x_2 we get that x_1 and x_2 are the pair of the reciprocal nearest neighbors, however point y is closer to x_1 than is x_2.

agglomeration of two objects $p', p'' \in \mathcal{X}$ cannot produce a new object p that would be closer to any of the other objects in $\mathcal{X}$ than was p' or p'' (Murtagh, 1984, Section 3); see Figure 4.12 for an example where this property fails when applying the single cluster algorithm for points in two-dimensional space.

Müllner (2011) suggested a generic clustering algorithm that supports any proximity update formulae and does not require for the reducibility property to hold. This algorithm has $O(n^3)$ worst case and $O(n^2)$ best case runtime complexity (Müllner, 2011). We can break down the complexity into two major components: the number of iterations and the complexity of each iteration. There is not much we can do about the number of iterations, but there are tricks for finding nearest neighbors that can reduce complexity of each iteration. While exact nearest neighbor search requires $O(n)$ operations for each point, we can easily reduce it if we relax neighbor criterion and look for approximately nearest neighbors instead.

4.5.2 Nearest neighbor search strategies

Let $\mathcal{X}$ be a set of objects and let $a \in \mathcal{X}$. Then $b \in \mathcal{X}$ is said to be the nearest neighbor of a if $|a - b| \leq |a - i|$ for all $i \in \mathcal{X} \setminus \{a, b\}$. Finding such an object b for a is a task of $O(nd)$ complexity ($n = |\mathcal{X}|$ and d is the dimension of objects in $\mathcal{X}$). Thus it takes quite a lot of time for large data sets of high dimension. However, in some cases an approximately nearest neighbor is just as good and accuracy can be exchanged for speed. Most of the algorithms for finding approximately nearest neighbors comprise of two steps (Buaba *et al.*, 2014), namely (1) preprocessing of the input data into some simpler structure, and (2) localized search in this structure.

A common method for approximate neighbor finding is locality sensitive hashing by Indyk and Motwani (1998) or LSH for short. This algorithm hashes objects into buckets and assumes that close (similar) objects will appear in the same bucket. It has a query complexity of $O(dn^q)$ where $q \in (0, 1)$. A similar strategy is used in random mapping onto one dimension by Kaski (1998). Let $\{r_i\}$ be a set of random numbers with $|\{r_i\}| = d$, then random projection of some object x with coordinates $(x_1, \dots, x_d)$ in d-dimensional space is (Kaski, 1998, Formula 2):

$$p(x) = \sum_{i}^{d} r_i x_i$$

See Figure 4.13 for an example how random embedding works for a set of points in two-dimensional space.

Random mapping projects objects onto a continuous line giving us a linear search space. Let $k \leq n$ be the size of search space and let $x \in \mathcal{X}$ be an object with a random projection $p(x)$ on the line ℓ_r. We find an approximately nearest neighbor of x by considering points whose projections are in the set of $p(x)$-nearest neighbors on ℓ_r. The rationale behind this approach is the assumption that points that are close in the high-dimensional space will also be near each other on the projective line (Kaski, 1998). For example, take the set of points in Figure 4.13 choose some point x and compare its k nearest neighbors with k nearest neighbors in the projection as shown in Figure 4.14. We experimented with different values of k and found that $k \approx \sqrt{n}$ performs better than small constant numbers and is still significantly smaller than n.

Agglomeration

Algorithm 4.4 is the modification of the generic agglomerative clustering algorithm by Müllner (2011). We use the random projection when working with high-dimensional objects and no projection for the distances. The procedure for finding (approximately) nearest neighbors is given in Algorithm 4.5. Procedures that differ for the proximity data and high-dimensional objects are described separately in Algorithm 4.6 and Algorithm 4.7 accordingly. Random projection allows us to limit the complexity of the nearest neighbor search and loops in the lines 16 and 22 of Algorithm 4.4 to $O(k)$ thus reducing the overall complexity to approximately $O(nk)$ best and $O(nk^2)$ in the worst case. See Section 4.5.6 for more details on the runtime complexity.

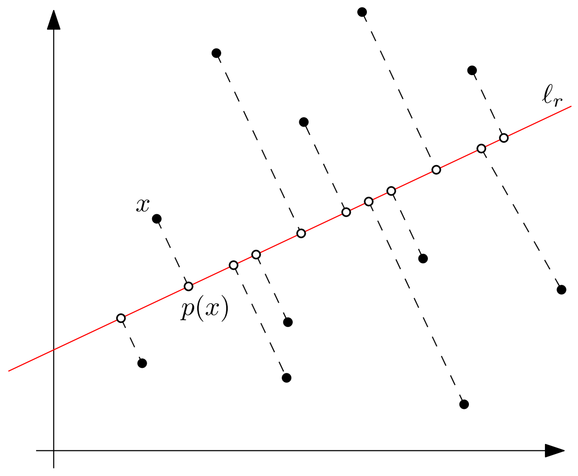


Figure 4.13: A set of points (black) and their projections (white) onto a random line ℓ_r. For example $p(x)$ is the projection of x on ℓ_r.

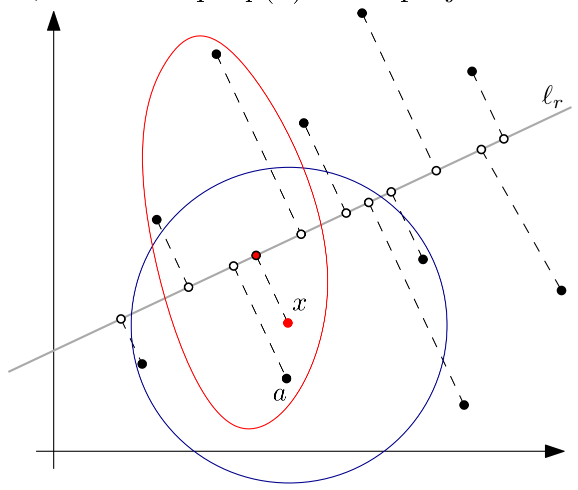


Figure 4.14: Three nearest neighbors of x. Blue circle encloses true nearest neighbors whereas red curve bounds its nearest neighbors on the projective line ℓ_r.

Algorithm 4.4 Approximate clustering.

```
procedure ApproximateClustering($\mathcal{X}$)
    $L \leftarrow \emptyset$
    $nnghbr \leftarrow []$
    $mindist \leftarrow []$
    for $x \in \mathcal{X}$ do                                    ▷ makes a list of nearest neighbors
        UpdateNearestNeighbor($x, nnghbr, mindist$)
    end for
    $Q \leftarrow$ MakePriorotyQueue($\mathcal{X}, mindist$)       ▷ sorts $\mathcal{X}$ by values in $mindist$
    while $|\mathcal{X}| > 1$ do
        $a \leftarrow$ Pop($Q$)
        $b \leftarrow nnghbr[a]$
        Remove($Q, b$)                                            ▷ removes $b$ from $Q$
        $c \leftarrow$ Agglomerate($a, b$)
        $\mathcal{X} \leftarrow \mathcal{X} \cup c \setminus \{a, b\}$          ▷ updates $\mathcal{X}$
        Push($L, a, b$)                                          ▷ saves $a$ and $b$ as neighbors
        for $x \in$ GetNeighborhood($a$) $\cup$ GetNeighborhood($b$) do
            if $nnghbr[x] = a$ or $nnghbr[x] = b$ then
                $nnghbr[x]$ = FindNearestNeighbor($x$)
                UpdateQ($Q, x, mindist[x]$)
            end if
        end for
        for $x \in$ GetNeighborhood($c$) do
            if GetDistance($x, c$) $< mindist[x]$ then
                $nnghbr[x] = c$
                $mindist[x]$ = GetDistance($x, c$)
                UpdateQ($Q, x, mindist[x]$)
            end if
        end for
        UpdateNearestNeighbor($c, nnghbr, mindist$)
        UpdateQ($Q, c, mindist[c]$)
    end while
    return $L$
end procedure

procedure UpdateNearestNeighbor(x,nnghbr,mindist)
    $nnghbr[x] \leftarrow$ FindNearestNeighbor($x$)
    $mindist[x] \leftarrow$ GetDistance($x, nnghbr[x]$)
end procedure
```

Algorithm 4.5 Find nearest neighbor.

```
procedure FindNearestNeighbor(y)
    $neighbor \leftarrow 0$
    $min \leftarrow \infty$
    for $x \in$ GetNeighborhood($y$) do
        if GetDistance($x, y$) $< min$ then
            $neighbor \leftarrow x$
            $min \leftarrow$ GetDistance($x, y$)
        end if
    end for
    return $neighbor$
end procedure
```

Algorithm 4.6 Procedures for the proximity matrix.

```
procedure Agglomerate(a,b)
    for $x \in \mathcal{X} \setminus c$ do
        $\delta_{xc} \leftarrow \sqrt{\frac{2\delta_{xa}^2 + 2\delta_{xb}^2 - \delta_{ab}^2}{4}}$
    end for
    $\delta_{cc} = 0$
    return $c$
end procedure

procedure GetNeighborhood(x)
    return $\mathcal{X} \setminus x$
end procedure

procedure GetDistance($x, y$)
    return $\delta_{xy}$
end procedure
```

Algorithm 4.7 Procedures for high-dimensional objects.

```
procedure AGGLOMERATE(a,b)
    $c \leftarrow \frac{1}{2}(a+b)$        ▷ creates $c$ with coordinates that are average of $a$ and $b$
    UPDATEPROJECTION($a.b.c$)
    return $c$
end procedure

procedure GETNEIGHBORHOOD(x)
    return NEIGHBORSINPROJECTION($x, k$)        ▷ returns $k$ nearest neighbors for $x$
end procedure

procedure GETDISTANCE($x, y$)
    return COMPUTEDISTANCE($x, y$)
end procedure
```

4.5.3 Initial positioning

Every Euclidean metric on n points can be embedded into $n-1$ dimensions (Borg and Groenen, 2005, p. 419). Therefore any three points, assuming that they satisfy triangle inequality, can be placed in a two-dimensional space with stress equal to zero. We embed three points p_1, p_2 and p_3 by assigning them the following coordinates as shown in Figure 4.15a. Point p_1 is placed at the origin, p_2 is placed in the horizontal axis to the right of p_1 with the coordinates $x_1(p_2) = \delta_{12}$ and $x_2(p_2) = 0$. Coordinates for the last point p_3 are then computed using the Pythagorean theorem:

$$(x_1(p_1) - x_1(p_3))^2 + (x_2(p_1) - x_2(p_3))^2 = \delta_{13}^2$$
$$(x_1(p_2) - x_1(p_3))^2 + (x_2(p_2) - x_2(p_3))^2 = \delta_{23}^2$$

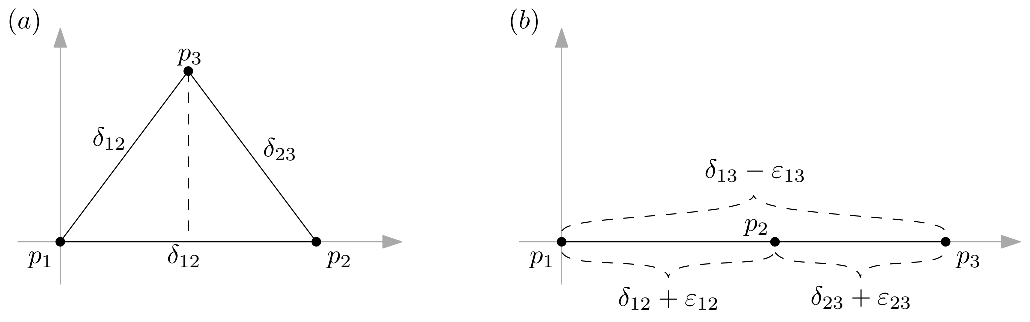
By inserting the coordinates of p_1 and p_2 we have:

$$x_1(p_3)^2 + x_2(p_3)^2 = \delta_{13}^2$$
$$(\delta_{12} - x_1(p_3))^2 + x_2(p_3)^2 = \delta_{23}^2$$

From here we get:

$$x_1(p_3) = \frac{\delta_{12}^2 + \delta_{13}^2 - \delta_{23}^2}{2\delta_{12}}$$
$$x_2(p_3) = \sqrt{\delta_{13}^2 - x_1(p_3)^2}$$

The second coordinate $x_2(p_3)$ can be either positive or negative. Without loss of generality we always place p_3 above the first (horizontal) axis, i.e., choose the positive coordinate.



(a)
p_3
δ_{12}
δ_{23}
p_1
δ_{12}
p_2
(b)
$\delta_{13}-\varepsilon_{13}$
p_2
p_1
p_3
$\delta_{12}+\varepsilon_{12}$
$\delta_{23}+\varepsilon_{23}$

Figure 4.15: Initial placement of three points p_1, p_2 and p_3 in the two-dimensional space. (a) Points form a triangle when proximities satisfy the triangle inequality, and (b) points are placed on the line when the triangle inequality is not satisfied.

If proximities between the three remaining objects do not satisfy the triangle inequality, we cannot find an exact embedding. Let δ_{13} be the largest proximity between the three remaining objects and let

$$\varepsilon = \delta_{13} - (\delta_{12} + \delta_{23}) \tag{4.21}$$

be the residual. If we add ε to $\delta_{12} + \varepsilon_{23}$ then the *triangle equality* is satisfied. Split the ε into three components:

$$\varepsilon_{12} + \varepsilon_{23} + \varepsilon_{13} = \varepsilon$$

and rewrite Equation 4.21 as follows

$$(\delta_{12} + \varepsilon_{12}) + (\delta_{23} + \varepsilon_{23}) = (\delta_{13} - \varepsilon_{13}).$$

Let the distances in the embedding be

$$\begin{aligned} d_{12} =& \delta_{12} + \varepsilon_{12}, \\ d_{23} =& \delta_{23} + \varepsilon_{23}, \text{ and} \\ d_{13} =& \delta_{13} - \varepsilon_{13} = \delta_{13} - \varepsilon + \varepsilon_{12} + \varepsilon_{23}. \end{aligned}$$

The weighted raw stress is minimized when

$$w_1 w_2 \varepsilon_{12}^2 + w_2 w_3 \varepsilon_{23}^2 + w_1 w_3 (\varepsilon - \varepsilon_{12} - \varepsilon_{23})^2 \tag{4.22}$$

is minimized. By minimizing Equation 4.22 we get:

$$\begin{aligned} \varepsilon_{12} =& \frac{w_3}{w_1 + w_2 + w_3} \varepsilon, \\ \varepsilon_{23} =& \frac{w_1}{w_1 + w_2 + w_3} \varepsilon, \text{ and} \\ \varepsilon_{13} =& \frac{w_2}{w_1 + w_2 + w_3} \varepsilon. \end{aligned}$$

Hence we place p_1 at the origin, p_2 at $(0, \delta_{12} + \varepsilon_1 2)$ and p_3 at $(0, \delta_{13} - \varepsilon_{13})$ as shown in Figure 4.15b.

4.5.4 Expansion

The expansion step consists of two parts. First we take the pair that was agglomerated last and separate it into two points which are placed on the opposite sides of the agglomerated point. Next we adjust the two expanded points and the neighborhood of the agglomerated point in the same way as described by Morrison *et al.* (2003).

Placing the expanded points

To expand some point c which is an agglomerate of a and b we will refer to some other two points $p_1 \neq p_2$. Let (x_1, y_1) and (x_2, y_2) be the coordinates of p_1 and p_2 respectively. When performing the agglomeration to get the coordinates of c we took the weighted average of the high-dimensional coordinates of a and b. We assume that the same also holds in the two-dimensional space, i.e.,

$$\begin{aligned} \frac{w_a}{w_c}x_a + \frac{w_b}{w_c}x_b &= x_c \\ \frac{w_a}{w_c}y_a + \frac{w_b}{w_c}y_b &= y_c \end{aligned} \tag{4.23}$$

For the sake of simplicity assume that at each expansion step we translate all points so that c is placed at the origin. Then from the equations in 4.23 we have that:

$$\begin{aligned} x_a &= -\frac{w_b}{w_a}x_b \\ y_a &= -\frac{w_b}{w_a}y_b \end{aligned} \tag{4.24}$$

Also, as c is between a and b we get that $d_{ac} = \frac{w_a}{w_c}d_{ab}$ and $d_{bc} = \frac{w_b}{w_c}d_{ab}$. Thus:

$$\begin{aligned} x_a^2 + y_a^2 &= \frac{w_a^2}{w_c^2}d_{ab}^2 \\ x_b^2 + y_b^2 &= \frac{w_b^2}{w_c^2}d_{ab}^2 \end{aligned} \tag{4.25}$$

To start with, the only coordinates that we know are those of p_1, p_2 and c. After the expansion of c, $\overline{p_1c}$ and $\overline{p_2c}$ become weighted medians of the triangles $\triangle ap_1b$ and $\triangle ap_2b$ respectively, see Figure 4.16. Hence we will refer to these segments as weighted medians and denote $m_1 = d_{p_1c}$ and $m_2 = d_{p_2c}$. Since c was placed at the origin we have that:

$$\begin{aligned} x_1^2 + y_1^2 &= m_1^2 \\ x_2^2 + y_2^2 &= m_2^2. \end{aligned} \tag{4.26}$$

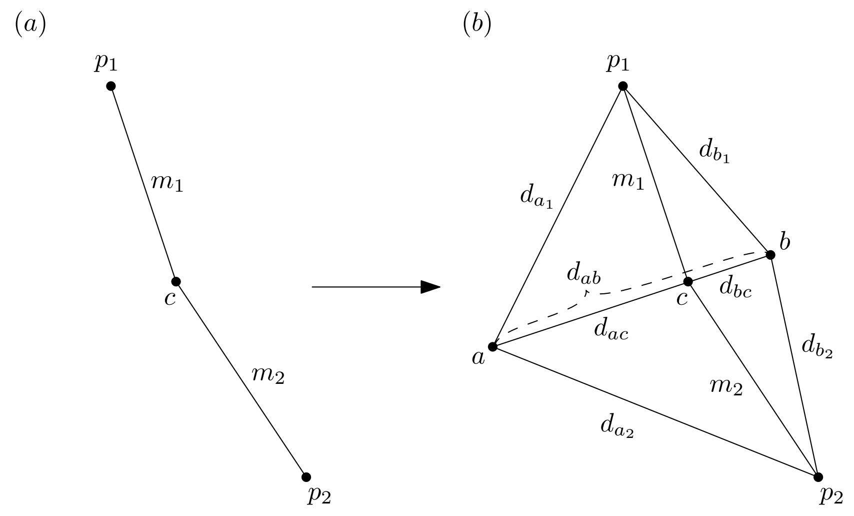


Figure 4.16: Initial expansion of the two points a and b with respect to some other two points $p_1 \neq p_2$.

Corresponding medians in high-dimensional space we denote as μ_1 and μ_2:

$$\begin{aligned}\mu_1 &= \delta_{p_1 c}\\ \mu_2 &= \delta_{p_2 c}.\end{aligned}$$

Ideally we would have $m_1 = \mu_1$ and $m_2 = \mu_2$. If this is the case, then we can easily find coordinates of a and b from Q distances. However, this quite likely might not be the case, thus we need to use scaling. Let $s_1 = \frac{m_1}{\mu_1}$ and $s_2 = \frac{m_2}{\mu_2}$ be the scaling factors for proximities in triangles $\triangle a p_1 b$ and $\triangle a p_2 b$ respectively.

From the Pythagorean theorem we have that:

$$\begin{aligned}d_{a_1}^2 &= (x_a - x_1)^2 + (y_a - y_1)^2\\ d_{a_2}^2 &= (x_a - x_2)^2 + (y_a - y_2)^2\\ d_{b_1}^2 &= (x_b - x_1)^2 + (y_b - y_1)^2\\ d_{b_2}^2 &= (x_b - x_2)^2 + (y_b - y_2)^2. \end{aligned} \tag{4.27}$$

With the use of equations 4.24, 4.25 and 4.26 we can rewrite 4.27 to

$$\begin{aligned}d_{a_1}^2 &= \frac{w_a^2}{w_c^2} d_{ab}^2 - 2x_a x_1 - 2y_a y_1 + m_1^2\\ d_{a_2}^2 &= \frac{w_a^2}{w_c^2} d_{ab}^2 - 2x_a x_2 - 2y_a y_2 + m_2^2\\ d_{b_1}^2 &= \frac{w_b^2}{w_c^2} d_{ab}^2 - 2x_b x_1 - 2y_b y_1 + m_1^2\\ d_{b_2}^2 &= \frac{w_b^2}{w_c^2} d_{ab}^2 - 2x_b x_2 - 2y_b y_2 + m_2^2.\end{aligned}$$

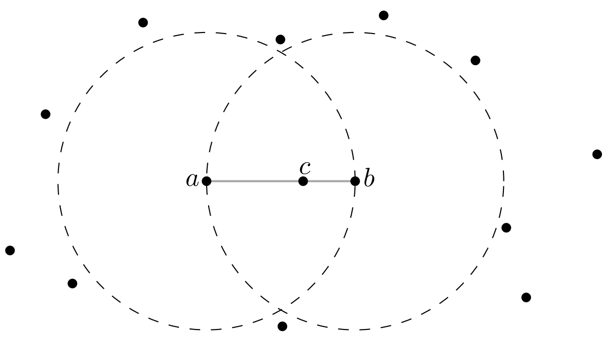


Figure 4.17: Neighborhood of the expanded points.

We replace low-dimensional distances (d) with scaled proximities (δ):

$$\begin{aligned}
s_1^2\delta_{a_1}^2 &= \frac{w_a^2}{w_c^2}s_1^2\delta_{ab}^2 - 2x_a x_1 - 2y_a y_1 + s_1^2\mu_1^2 \\
s_2^2\delta_{a_2}^2 &= \frac{w_a^2}{w_c^2}s_2^2\delta_{ab}^2 - 2x_a x_2 - 2y_a y_2 + s_2^2\mu_2^2 \\
s_1^2\delta_{b_1}^2 &= \frac{w_b^2}{w_c^2}s_1^2\delta_{ab}^2 - 2x_b x_1 - 2y_b y_1 + s_1^2\mu_1^2 \\
s_2^2\delta_{b_2}^2 &= \frac{w_b^2}{w_c^2}s_2^2\delta_{ab}^2 - 2x_b x_2 - 2y_b y_2 + s_2^2\mu_2^2.
\end{aligned} \tag{4.28}$$

From equations 4.28 we can express the coordinates of a and b which we do not provide here due to their complexity. We only need to compute coordinates for either a or b as from Equation 4.24 b can be expressed in terms of a and vice versa.

Local adjustment of the expanded points

After initial expansion we adjust coordinates of points using a heuristic approach based on the spring systems as described by Morrison *et al.* (2003) (see Section 4.2.2). Since each point is adjusted with respect to its nearest neighbors as well as a set of random points we need to keep neighborhoods of points up to date. We expect each expansion to have a local effect only. This is because in the agglomeration we assume two points a and b to be each other's (approximately) nearest neighbors; this means that there should be no other points in the radius δ_{ab} from either a or b as shown in Figure 4.17. Therefore when updating neighbors we look at the points that are nearest neighbors of c and also at the neighbors of the neighbors of c. We check whether a and/or b might be closer to some point in the set than any of the current nearest neighbors. If this is the case, the nearest neighbor list of that point is updated. We also look at the neighborhoods of a and b. Initially we assign both of them the same neighborhoods as c had and then check whether any of the points in the set appear closer. The greedy way would be to go through all of the points that are present in the current configuration, but because we assume a and b to be an approximately closest pair of points, we also assume that neighborhoods of both of them should not be very different from the neighborhood of the point c. This allows us to reduce the search space significantly, especially in case of large data sets. Each point in the neighborhood as well as a and b is assigned a set of random points to correct for the local forces. Then we use a spring system model to move each point to a new (likely more optimal) position.

Each coordinate of the point p is translated by a force vector which is an average of all forces coming from neighboring and random points. Each interaction between point p and some other point i is assigned a weight w_{ip}:

$$w_{ip} = w_i w_p \max\left(\frac{\delta_{ip}}{d_{ip}}, \frac{d_{ip}}{\delta_{ip}}\right),$$

here w_j is the weight of the point j which is equal to the number of objects that j contains at the current stage, i.e. the number of objects that were agglomerated into j. Interaction between some two points is actually an interaction between pairs of all objects in both points, therefore we want to emphasize distances between "heavy" points more than between "light" ones. The other component $\max\left(\frac{\delta_{ip}}{d_{ip}}, \frac{d_{ip}}{\delta_{ip}}\right)$ makes sure that we give more weight to the currently worse fits. Now considering weights and distances in low and high-dimensional spaces we compose the force vector for all coordinates p_j of the point p in the low-dimensional space:

$$f(p_j) = \frac{\sum_i w_{ip}(i_j - p_j)\frac{d_{ip}-\delta_{ip}}{d_{ip}}}{\sum_p w_{ip}}$$

4.5.5 Global adjustment

At each step we perform local adjustments, however expansion of some two points might also have an impact on some points further away as well. To compensate for that a few times during the expansion and also at the end we perform a global adjustment procedure. It adjusts all of the points in the current configuration in the same way as local adjustment, only we do not need to update any of the neighborhoods as all of them are already up to date. However, random point sets still need to be updated and preferably before every adjustment as this enhances performance of the algorithm and helps to avoid getting stuck in a local minima.

4.5.6 Computational complexity

We have already discussed computational complexity of the agglomeration procedure which is $O(n^2)$ best and $O(n^3)$ worst case for the proximity data and $O(n\sqrt{(n)}d)$ best and $O(n^2 d)$ worst case for objects in d-dimensional space.

Initial embedding has the same complexity as for unidimensional scaling, that is, we have $O(c)$ for proximity and $O(d)$ for vector data.

Expansion consists of the initial expansion which runs in constant time and refinement which depends on the size of neighborhoods k and number of random points r that we use to adjust each expansion. We choose $k = \sqrt[4]{n}$ and $r = ck$. Then each adjustment requires $(O(\sqrt[4]{n}))$ operations. During the expansion we also perform some global adjustments each of which requires $O(n\sqrt[4]{n})$ computations. Hence the overall expansion complexity is $O(n\sqrt[4]{n})$ for proximities and $O(n\sqrt[4]{n}d)$ for the vectors.

Summing up, we get that the most computationally costly step for the two-dimensional embedding is the agglomeration. It determines the overall complexity making it $O(n^2)$ best and $O(n^3)$ worst case for the proximity data and $O(n\sqrt{(n)}d)$ best and $O(n^2d)$ worst case for vector data. Random embedding of d-dimensional objects requires $O(nd)$ operations and thus does not affect the overall complexity of the algorithm.

4.5.7 Performance

We implemented the algorithm for agglomerative multidimensional scaling as described in this section in java programing language. To test its performance we generated synthetic data sets each containing:

- n_c clusters,
- m points per cluster
- in d dimensions.

Boundaries of each cluster were defined as cubes in d dimensional space with edges of length l.

By running multiple tests we got very promising results concerning both accuracy and speed out method. For example a good quality embedding ($\sigma_1 = 0.166$) for a set of 100,000 points ($n_c = 10$, $m = 10,000$, $l = 0.2$) in five dimensions was computed in only two minutes on a laptop with an *i5* processor.

To compare our algorithm to other available methods we used smaller data sets due to the high computational complexities of some of the other methods. Figure 4.18 shows multidimensional scaling plots of four different methods for a set points with parameters: $n_c = 10$, $m = 200$, $d = 5$ and $l = 0.2$. Sammon mapping produced a plot with lowest stress and very easily identifiable clusters, but was relatively slow as it took a few minutes compute a plot for 2,000 points. Agglomerative clustering performed almost as good as Sammon mapping on this data set and took only a few seconds to compute the plot. Classical scaling and PROXSCAL produced plots with relatively high stress values and fewer clusters than expected.

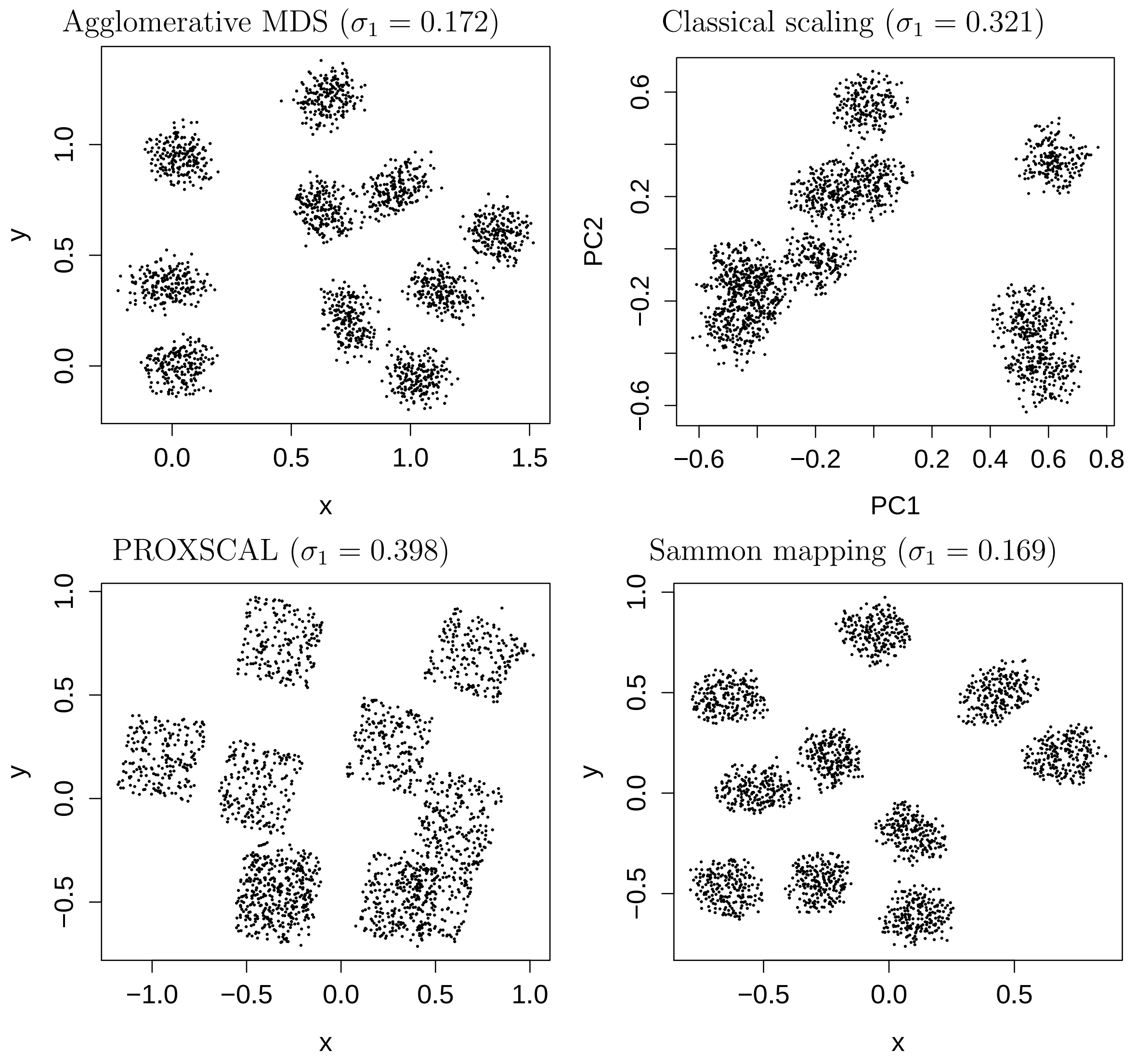


Figure 4.18: MDS plots for a set of 2,000 random points in five dimensions with 10 clusters computed with different methods.

4.6 Alternative strategies

4.6.1 Expansion using numerical optimization

Initially we did not adjust points and instead chose to work on finding the globally optimal initial placement. To place a and b in such a way that Equation 4.24 holds and the stress for a and b is minimized we need to find the points where the stress gradient vanishes. For that we look at the weighted raw stress:

$$\sigma_{rw} = \sum_i \left[w_a w_i(\delta_{ai} - d_{ai})^2 + w_b w_i(\delta_{bi} - d_{bi})^2\right] \\ + w_a w_b(\delta_{ab} - d_{ab})^2$$

High-dimensional distances d are already known as are coordinates of all points $i \notin \{a, b\}$. Therefore we can write low-dimensional distances as a function of a and/or b and by substituting b with $-\frac{w_b}{w_a}a$ (Equation 4.24) as a function of a only. Thus σ_{rw} becomes:

$$\sigma_{rw} = \sum_i \left[w_a w_i(\delta_{ai} - |i-a|)^2 + w_b w_i(\delta_{bi} - |i + \frac{w_b}{w_a}a|)^2\right] \tag{4.29}$$

$$+ w_a w_b(\delta_{ab} - (1 + \frac{w_b}{w_a})|a|)^2 \tag{4.30}$$

And its derivative with respect to a is:

$$\frac{\delta\sigma_{rw}}{\delta a} = \sum_i \left[w_a w_i\left(\frac{\delta_{ai}}{|i-a|} - 1\right)(i-a) - w_b w_i\left(\frac{\delta_{bi}}{|i + \frac{w_b}{w_a}a|} - 1\right)\left(i + \frac{w_b}{w_a}a\right)\right] \\ - (w_a w_b + w_b^2)\left(\frac{\delta_{ab}}{|a|} - (1 + \frac{w_b}{w_a})\right)a \tag{4.31}$$

Finding a solution for $\frac{\delta\sigma_{rw}}{\delta a} = 0$ for more than one point i is computationally too difficult, but can be achieved using numerical methods. We employed Newton's method for multiple dimensions as $a = \{x, y\}$:

$$a_{n+1} = a_n - \gamma\left[H\sigma_{rw}(a_n)\right]^{-1}\nabla\sigma_{rw}(a_n)$$

Here $H\sigma_{rw}(a_n)$ is a Hessian matrix:

$$Hf(a_n) = \begin{bmatrix} \frac{\delta^2\sigma_{rw}}{\delta x_n^2} & \frac{\delta^2\sigma_{rw}}{\delta x_n \delta y_n} \\ \frac{\delta^2\sigma_{rw}}{\delta y_n \delta x_n} & \frac{\delta^2\sigma_{rw}}{\delta y_n^2} \end{bmatrix}$$

We start computations from four different starting points a_0: $\{\frac{\delta_{ab}}{2}, 0\}$, $\{0, \frac{\delta_{ab}}{2}\}$, $\{-\frac{\delta_{ab}}{2}, 0\}$ and $\{0, -\frac{\delta_{ab}}{2}\}$. At the end, if the computations converge to different points we choose the one for which 4.29 is minimal as the solution.

Applying numerics for expansion is computationally inefficient as finding a global optimal position requires $O(n)$ computations at each iteration. However it might be useful for the initial expansions with respect to the points in the neighborhood. To test for that we compare numerics to the methods of random pair and average coordinates. As shown in Figure 4.19 applying numerics helps to find more optimal local positions, however this does not guarantee better overall stress as in many cases the suboptimal local placements that we get using other two approaches resulted in better global stress values.

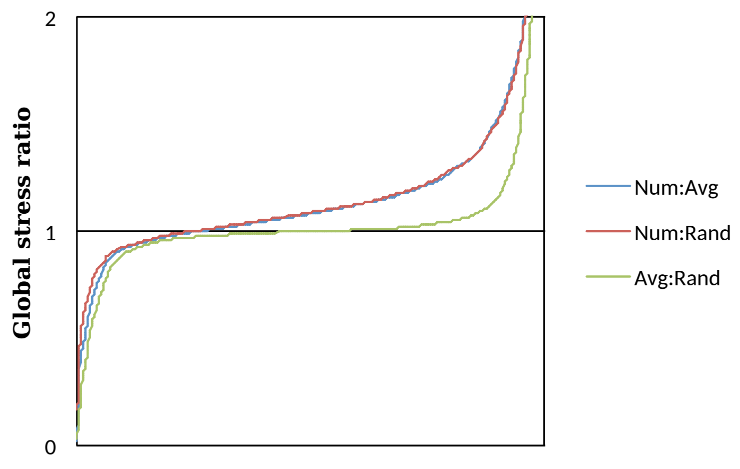
4.6.2 Alternative initial expansion strategies

One should also consider that unless $d = 2$ the coordinates that we get depend on the choice of p_1 and p_2. To check whether it would be better to consider all pairs of p_1 and p_2 from the neighborhood of c we ran a simulation. We generated a random set of approximately 3000 points in a 100-dimensional space and ran our algorithm. In each expansion step, we tried both strategies as well as the numerical optimization and evaluated global and local stress values for the expanded points a and b. Local stress was estimated by considering points in the neighborhood of c, whereas global stress was computed by taking all points into account. Results of the simulation are shown in the graphs in Figure 4.19. Stress values do not favor any of the two strategies as ratios between stress for taking the average over all pairs and selecting a random pair are concentrated around 1. We persisted to expand points with respect to a set of random points as this approach is computationally faster.

4.6.3 Embeddings in higher dimensions

Algorithm for computing embeddings in the plane can be easily extended to more than two dimensions. Let q be the dimensionality of the embedding. The agglomeration procedure would not need to change as update formulas for two dimensions fit higher dimensions just as well. The initial embedding could be performed with, for example classical scaling onto the q dimensions, followed by the expansion in the q-dimensional space. Formulas for the initial expansion can be easily extended to higher dimensions by simply taking q points each time and finding initial embeddings with respect to them. Refinement, just like agglomeration, would not need any modifications.

(*a*)



(*b*)

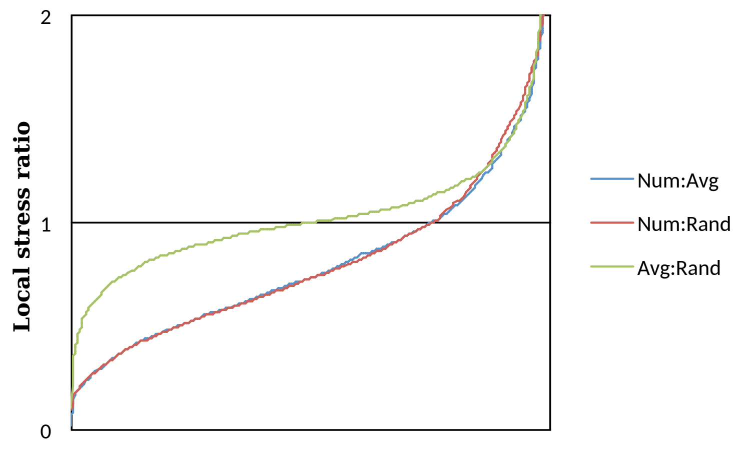


Figure 4.19: Comparison of the three initial expansion strategies (random pair, average of all pair, numeric with respect to all pairs): (a) with respect to all pairs of points in the neighborhood, and (b) one random pair from the neighborhood. At each iteration all approaches were tested and the resulting stress values compared. Graphs show distribution of ratios of the stress values that we get with different approaches at each iteration. Only stress scores of the expanded points are considered. Simulation was run on a random data set of 3000 objects in a 100-dimensional space. Global stress evaluates the overall placement of the expanded points whereas local stress shows how well they fit in the neighborhood. Ratios lower than one indicate that the first approach produce lower stress whereas higher than one indicate better performance of the second approach in each of the comparisons.

Chapter 5

Applications

In this chapter we demonstrate how planar split networks and multidimensional scaling plots can be used to visualize and analyze data. We do this with four data sets that highlight different benefits of the planar split networks and MDS plots versus trees and outer-labeled networks. The first example that we use consists of whole genome HIV sequences and illustrates a good application of planar split networks for the study of *recombination*. Next, we consider a data set that contains *ancestral sequences* of certain fluorescent proteins as well as sequences from currently living organisms. The third example, a collection of mitochondrial sequences from gall wasps, illustrates FlatNJ's applicability for *biogeographical* studies. Finally, we construct a data set from coordinates of some of the European capitals and show how different methods behave with data that is almost completely planar in nature. For all these data sets we compare FlatNJ networks to Neighbor-Joining trees, Neighbor-Net networks and multidimensional scaling plots. We build our discussion on the results published in (Balvočiūtė *et al.*, 2014) and complement it with some observations concerning multidimensional scaling.

All trees and outer-planar networks were computed with Neighbor-Joining and Neighbor-Net algorithms respectively as implemented in the program SplitsTree version 4.13.1 (Huson and Bryant, 2006). Neighbor-Net and FlatNJ networks were filtered as described in Section 3.4.6 with threshold 0.15. Multidimensional scaling plots were computed with the algorithm presented in the previous chapter with additional iterations of greedy refinement. All networks (including trees) and MSD plots were evaluated by estimating the Stress-1 σ_1 (Formula 4.27).

5.1 A circulating recombinant form of HIV

The first example involves the study of recombination in viruses, for which split networks have been commonly used. In this example, we applied FlatNJ to analyze the circulating recombinant form *CRF49* of HIV reported in de Silva *et al.* (2010).

We aligned the three whole genome sequences representing *CRF49* (accession numbers HQ385477, HQ385478 and HQ385479) published in de Silva *et al.* (2010) together with reference sequences for the collection

$$Sub = \{A, B, C, D, F, G, H, J, K\}$$

of known subtypes of HIV (see supplementary material for details). We used the reference alignment (version 2010) from the HIV databases at the Los Alamos National Laboratory (`www.hiv.lanl.gov`) to get the representative sequences of known subtypes. A multiple sequence alignment of all sequences was generated with *mafft* (Katoh *et al.*, 2002), v6.864b, maxiterate – 1000, localpair. The alignment was edited with *GBlocks* (Castresana, 2000), v.0.91b, minimum length of a block – 2, allowed gap positions – half.

In Figure 5.1 we present the networks produced by FlatNJ and Neighbor-Net, Neighbor-Joining and a MDS plot for this data set.

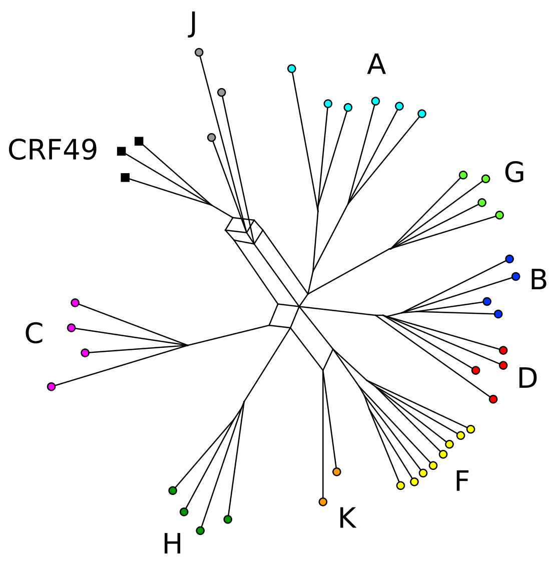
It was found in de Silva *et al.* (2010) that *CRF49* is composed of the known subtypes:

- J (48%),
- A (23% of total sequence length),
- C (18%),
- K (5%) and,
- unknown (6%).

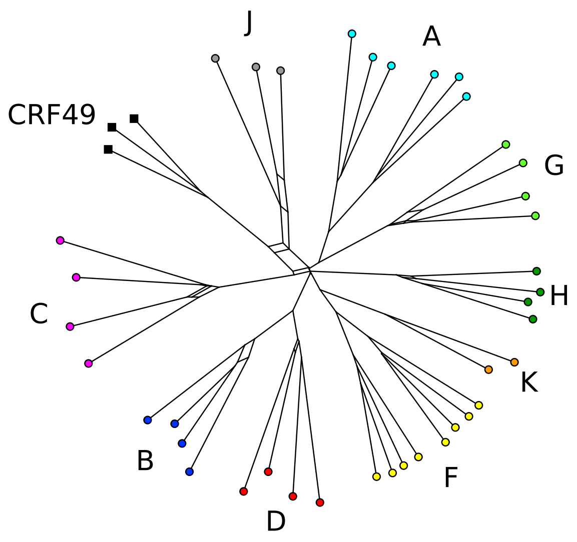
This composition is reflected by the fact that both the Neighbor-Net and FlatNJ networks contain the splits $S_J = \{CRF49, J\}|Sub - \{J\}$ and $S_C = \{CRF49, C\}|Sub - \{C\}$. The Neighbor-Joining tree only contains the split S_J. Moreover, the weight assigned to these splits in both networks is quite similar to the relative contribution of subtypes J and C to *CRF49*.

Note that the FlatNJ network also contains the split $S_{A,G} = \{CRF49, A, G\}|Sub - \{A, G\}$, which indicates that subtypes A and/or G could have contributed to *CRF49*. According to de Silva *et al.* (2010), subtype A contributed to *CRF49*, but this cannot be easily deduced from the NeighborNet network. In fact, it is impossible to display the three splits S_J, S_C and $S_{A,G}$ together in *any* outer-labeled split network. Hence, the FlatNJ network provides a more complete visualization of the composition of *CRF49* inferred in de Silva *et al.* (2010).

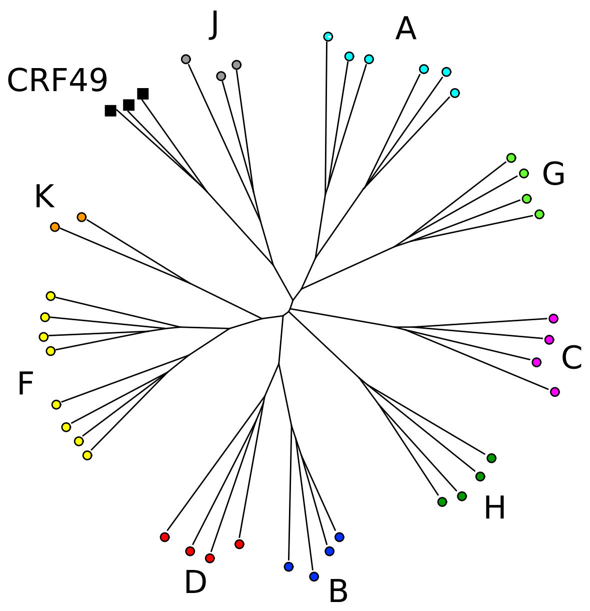
In the multidimensional scaling plot all of the contributing subtypes are placed closer to *CRF49* than any other subtype. However, note a somewhat concentric placement



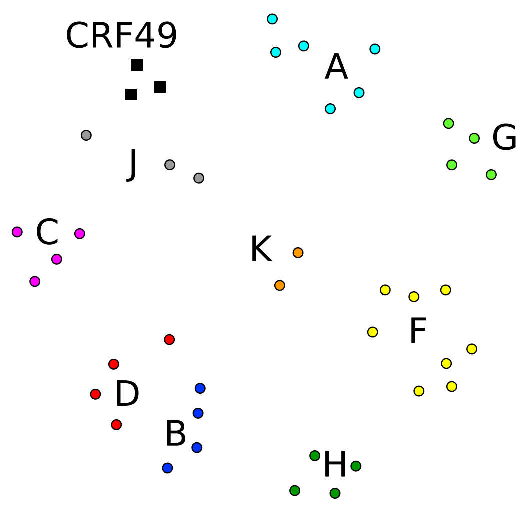
FlatNJ ($\sigma_1 = 0.280$)



Neighbor-Net ($\sigma_1 = 0.115$)



Neighbor-Joining ($\sigma_1 = 0.132$)



Multidimensional scaling ($\sigma_1 = 0.354$)

Figure 5.1: Split networks and a multidimensional scaling plot for the HIV data set.

of the subtypes in the plot that is typical for the very high-dimensional data. Indeed, as we took whole genome sequences, the alignment is approximately 8,450 bases long. All pairwise distances for this data set are shown in Table A.3 in Appendix A.3. Most of the distances between different subtypes are between 0.13 and 0.15; this lack of variation is quite typical for high dimensional data. Pairwise distances within subtypes are smaller and mostly vary between 0.06 and 0.08, thus sequences that belong to the same subtype cluster together in the MDS plot.

Significantly lower Stress-1 values for the networks than for the MDS plot suggest that networks may suit this high dimensional data better than two-dimensional plots.

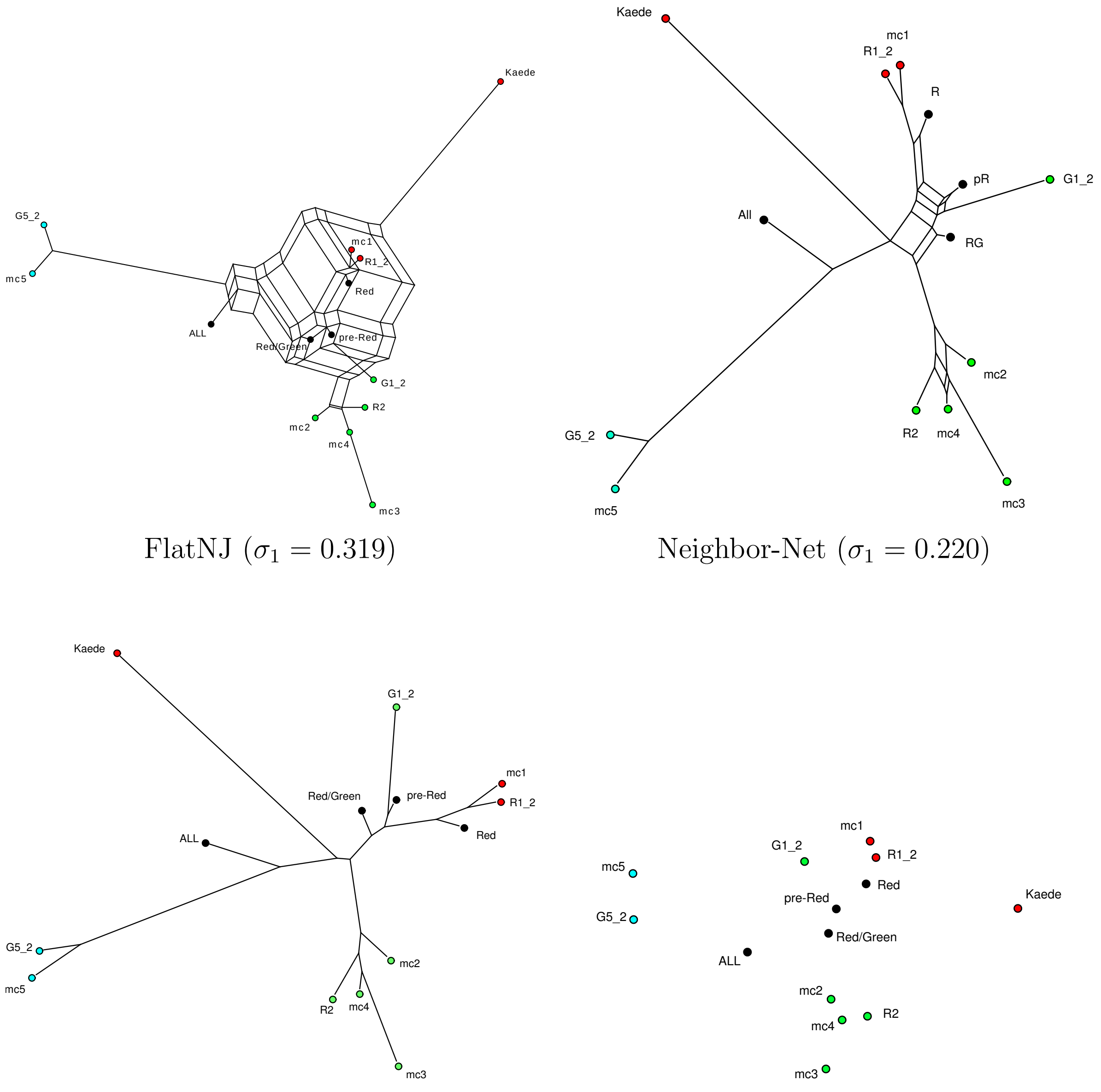
5.2 Ancestral forms of fluorescent proteins

We now consider a data set presented in Ugalde *et al.* (2004) to investigate the evolution of fluorescent proteins in corals. This data set consists of previously published proteins and reconstructed ancestral sequences presented in Ugalde *et al.* (2004).

Here we focus on those groups of proteins for which an ancestral sequence was presented in Ugalde *et al.* (2004): $Red = \{Kaede, mc1, R1\_2\}$, $pre\text{-}Red = \{G1\_2\} \cup Red$, $Red/Green = \{R2, mc2, mc3, mc4\} \cup pre\text{-}Red$ and $ALL = \{G5\_2, mc5\} \cup Red/Green$. The sequences included in this data set are listed in Table A.1 (Appendix A.1). A multiple sequence alignment of all sequences was generated with *mafft* (Katoh *et al.*, 2002), v6.864b, maxiterate – 1000, localpair. The alignment was edited with *GBlocks* (Castresana, 2000), v.0.91b, minimum length of a block – 2, allowed gap positions – none.

Networks and a MDS plot are depicted in Figure 5.2. We use the same labels as in Ugalde *et al.* (2004); names of the groups above are used to indicate the corresponding ancestral sequences.

Both Neighbor-Net and FlatNJ networks group sequences emitting the same color (red, green or cyan) together. However, the networks also contain many pairs of incompatible splits suggesting a complex, non-treelike evolution of fluorescent proteins in corals. This is in agreement with the findings in Kelmanson and Matz (2003), suggesting that intra-locus recombination could be one of the mechanisms that produced the sequence diversity we see today. This data set illustrates that it could be useful to allow internal labels when ancestral sequences are present. Indeed, in contrast to the Neighbor-Net, FlatNJ places all four ancestral sequences inside the network. Moreover, their placement relative to one another also better reflects the groups of proteins given above.



FlatNJ ($\sigma_1 = 0.319$)

Neighbor-Net ($\sigma_1 = 0.220$)

Neighbor-Joining ($\sigma_1 = 0.260$)

Multidimensional scaling ($\sigma_1 = 0.142$)

Figure 5.2: Split networks and a multidimensional scaling plot for the fluorescent protein data set.

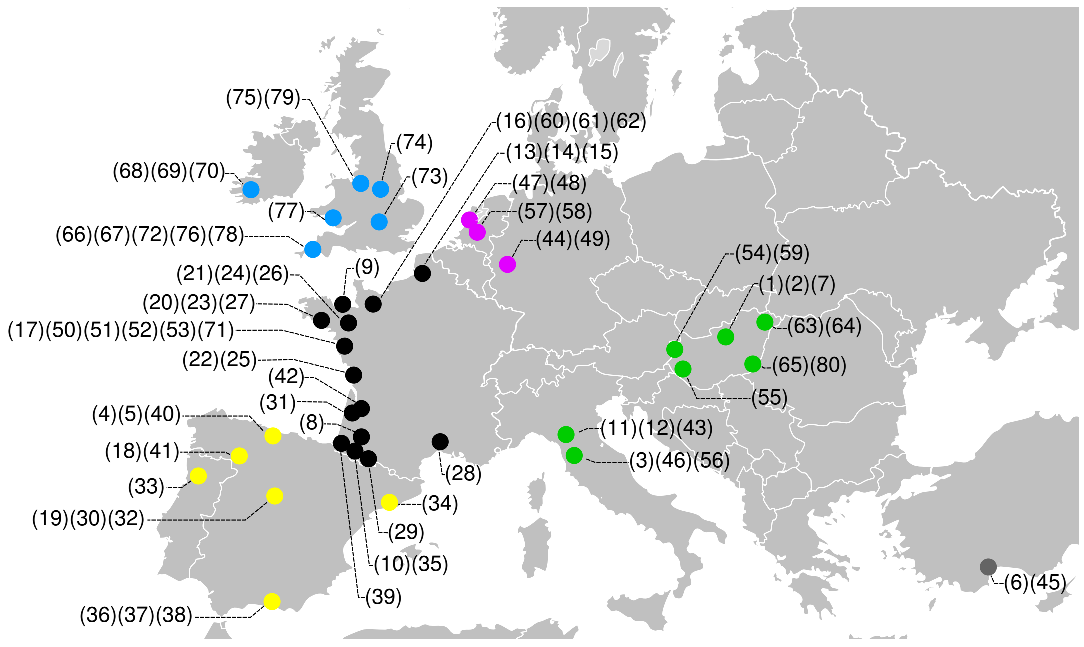


Figure 5.3: A map with the sampling locations of the sequences in the gall wasp data set. The accession numbers corresponding to the labels used in the map and the networks can be found in Table A.2.

A good fit for the MDS plot agrees with the suggested non-tree like evolution. Note that we can use MDS plot to generate a flat split system by adding pseudolines to it. The most straightforward way to do it would be to cut the plane with straight lines separating points in the plot. The fact that FlatNJ network has relatively high stress could mean that neighborliness condition might be too strong for this data set. The affine split network generated by cutting MDS plot with straight lines, on the other hand, does not have to be neighborly.

5.3 Biogeography of gall wasps

In our next example we consider a data set of 80 mitochondrial DNA sequences sampled from individuals of the species *A. kollari* (oak gall wasp) for which geographic coordinates for the sampling locations corresponding to each sequence are known (see Figure 5.3). This data set was also used in Spillner *et al.* (2012) to illustrate how, in a somewhat ad-hoc fashion, flat split systems can also be generated using multidimensional scaling as implemented in R (Spillner *et al.*, 2012).

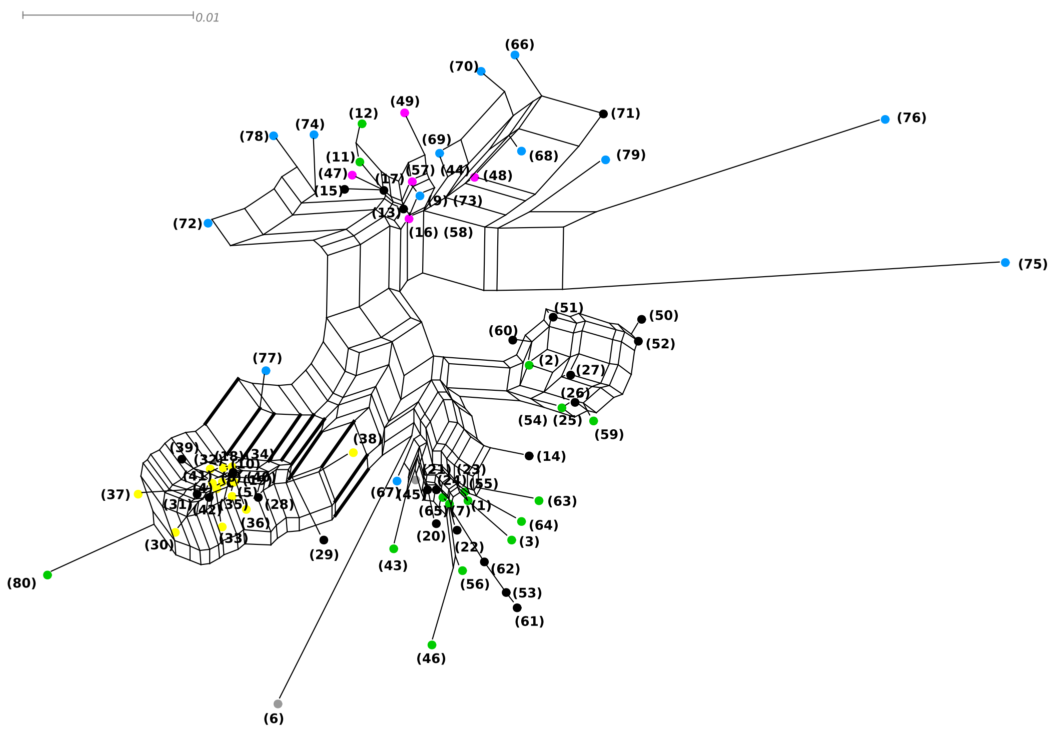
Accession numbers for the sequences included in the data set are listed in Appendix A.2 (Table A.2) along with the country in which the sequence was sampled. The sequences all have the same length and the alignment contains no gaps.

A. kollari is native to regions at the latitude of the Mediterranean from Portugal to Iran. In Stone *et al.* (2001, 2007) a study of the colonization of Northern Europe, in particular the British Isles, by this species is presented concluding that the data suggests that a large number of individuals of *A. kollari* that came originally from the Eastern Mediterranean were introduced to Britain by human trade. One step taken to reach this conclusion was the generation of a NeighborNet network for the sequences, which suggested that a tree-based analysis was not sufficient to fully assess the data. A Neighbor-Joining tree shown in Figure 5.5 has a significantly higher Stress-1 value than networks in Figure 5.4 or MDS plot.

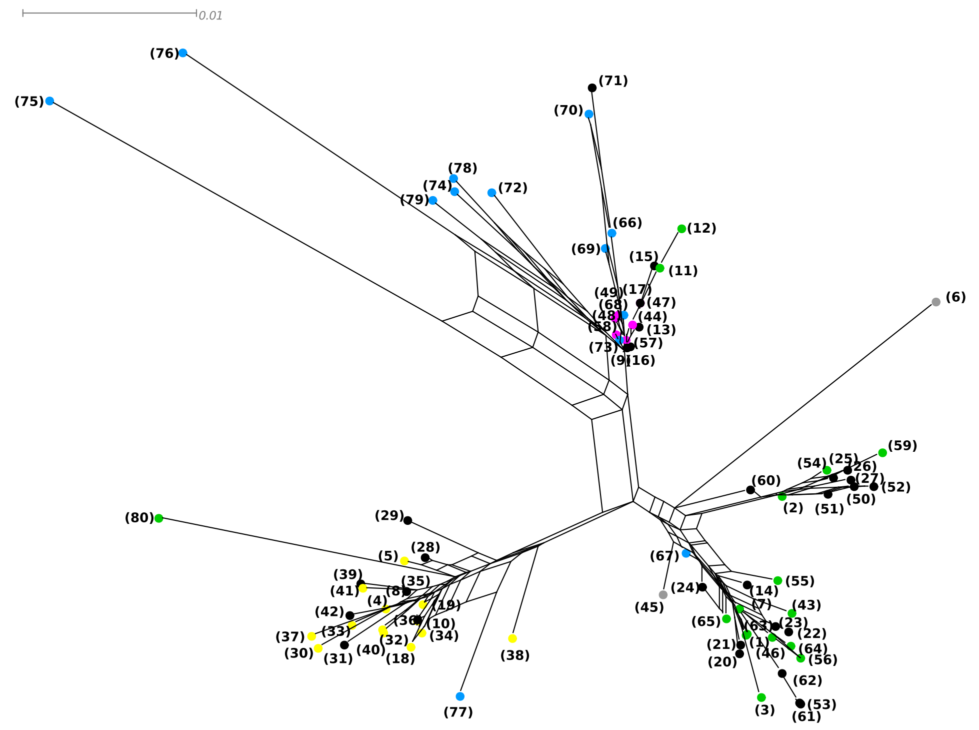
The network estimated with FlatNJ is quite similar to the network produced by Neighbor-Net, with 45% of the total weight of all splits in the FlatNJ network corresponding to splits that are also represented in the Neighbor-Net. This is somewhat reassuring as we feel that it is desirable for FlatNJ not to behave too differently from the well-established Neighbor-Net method, at least for the data that is planar in nature.

The multidimensional scaling plot for this data set has even better stress value. This once again confirms that for data that is somewhat planar in nature; two-dimensional embeddings may be more accurate than networks.

We next explored a way to visualize the relationship between the geographic and genetic data using split networks. More specifically, we generated the flat split system Σ_{geo} from Euclidean distances between the sampling locations and, to investigate which of the splits in Σ_{geo} are supported by the genetic distances, we reassigned weights to the splits in Σ_{geo} by minimizing the objective function (3.4) for the 4-split weights obtained from the sequence alignment. The split network representing the resulting weighted flat split system is depicted in Figure 5.6. The network displays a clear-cut geographic structure, although it is quite different from the FlatNJ network in Figure 5.4. Even so, the split highlighted in bold is present (up to sequence 80) in both networks, which might represent a signal for a geographical divide between the sequences from Iberia and South-Western France and the other sequences. Note that such a major divide has also been observed for other species from the genus *Andricus* (Stone *et al.*, 2007).

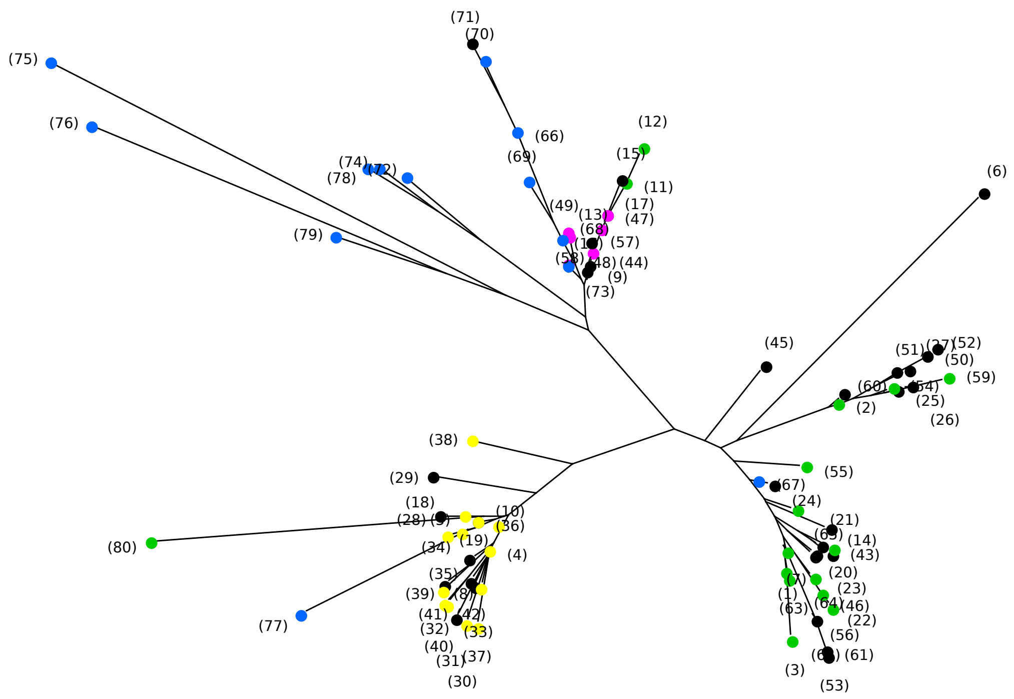


FlatNJ ($\sigma_1 = 0.174$)

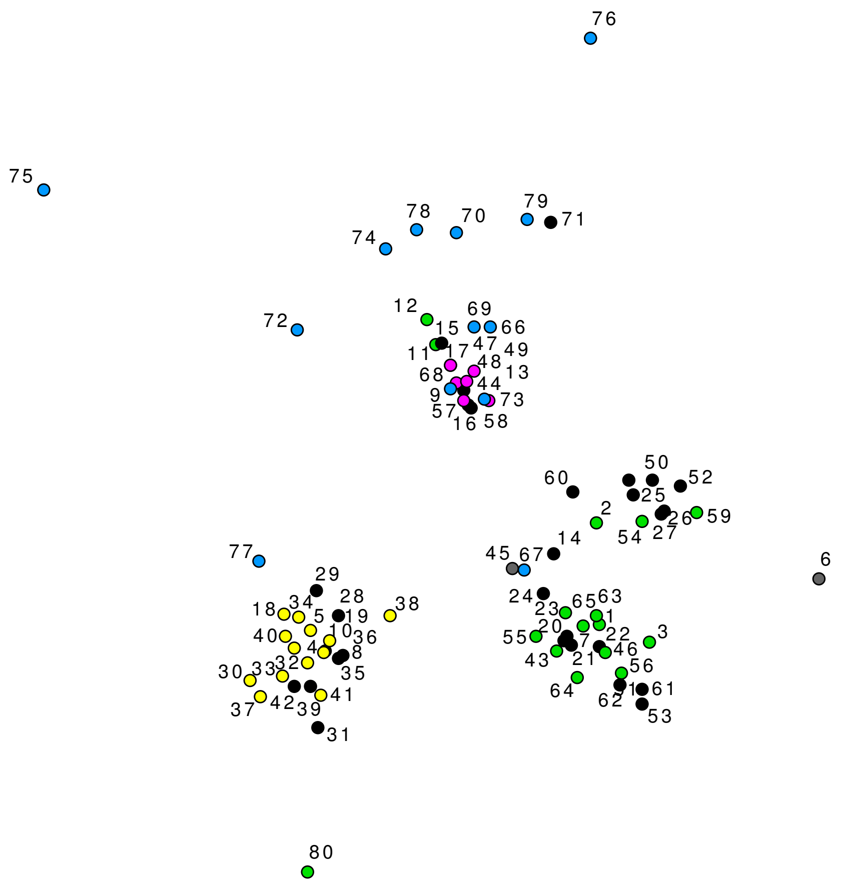


Neighbor-Net ($\sigma_1 = 0.231$)

Figure 5.4: Split networks produced by Neighbor-Net and FlatNJ from the sequence alignment for the gall wasp data set. The coloring scheme is the same as in Figure 5.3.



Neighbor-Joining ($\sigma_1 = 0.336$)



Multidimensional scaling ($\sigma_1 = 0.153$)

Figure 5.5: A Neighbor-Joining tree and a multidimensional scaling plot for the gall wasp data set from sequence data.

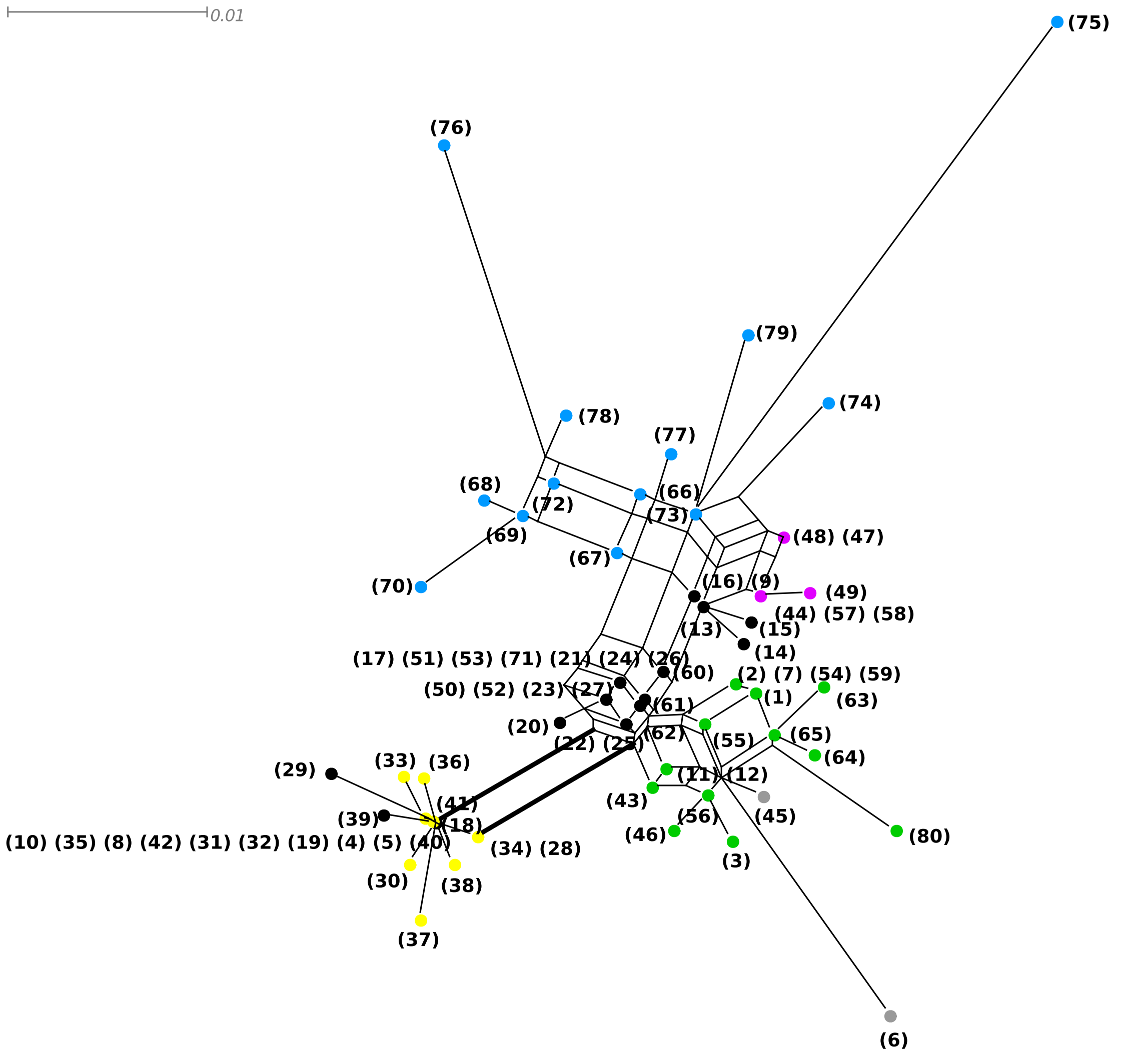


Figure 5.6: Split network produced by FlatNJ for the gall wasp data set from geographic coordinates with split weights computed using the 4-splits generated from the sequence alignment as described in the text. The split highlighted in bold separates the sequences from Iberia and Southern France from the other sequences. The coloring scheme is the same as in Figure 5.3.

5.4 European capitals data set

In our last example we look at data that is almost completely Euclidean. To demonstrate FlatNJ's potential in revealing spatial information, we applied FlatNJ to the system of 4-splits generated from geographic coordinates of some of the European capitals as explained in Section 3.2.1.

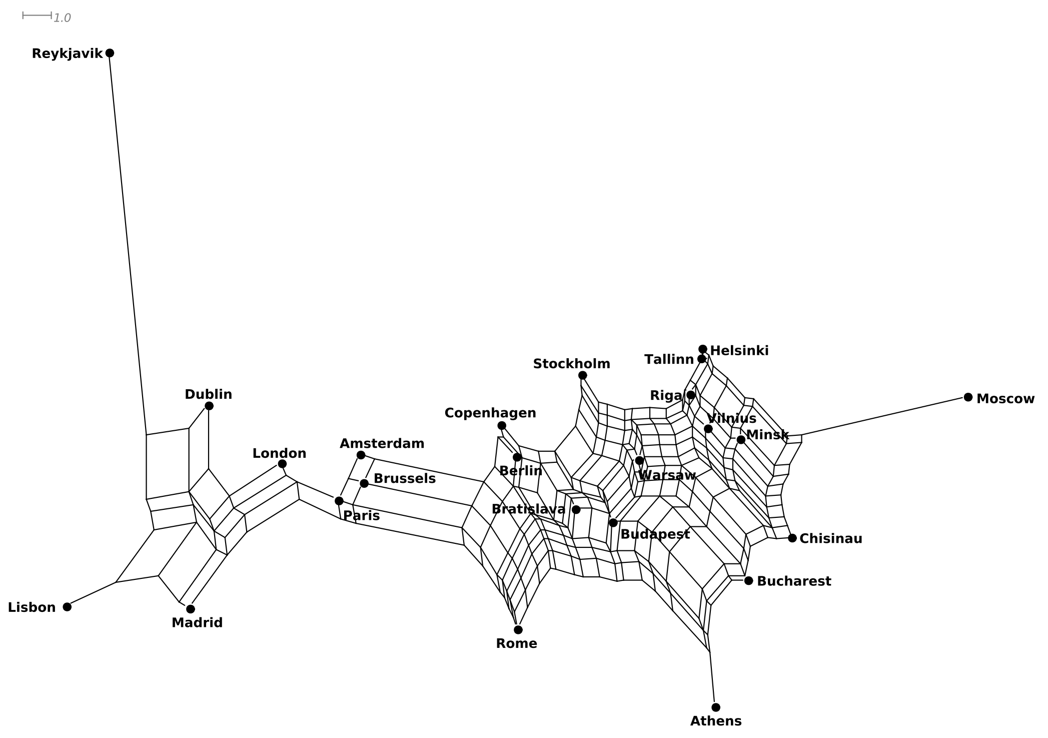
The resulting network together with a Neighbor-Net network, both of which have been manually adjusted for layout purposes, are depicted in Figure 5.7. The FlatNJ network has captured much of the spatial distribution of the capitals and as expected placed some of them in the inside of the network. Neighbor-Net also does quite well at capturing the relative positions of the capitals with some distortion in the part of the network that represents Eastern Europe what is due to the outer-planar nature of the Neighbor-Net networks. This might also explain why the network produced with FlatNJ appear more "boxy" than that computed with Neighbor-Net, FlatNJ is using more splits to avoid distorting geography, but at the price of having to introduce more incompatibility.

Recall from Section 3.2.1 that FlatNJ uses Euclidean norm to estimate distances. For consistency, we compute input distances for MDS as Euclidean norms, thus ignoring the curvature of the globe. Hence we get that multidimensional scaling for European capitals dataset behaves exactly as expected and generates a plot with Stress-1 equal to zero, see Figure 5.9.) Neighbor-Joining estimated a tree with very high Stress-1 value $\sigma_1 = 0.397$ (see Figure 5.8), proving its unsuitability for the planar data.

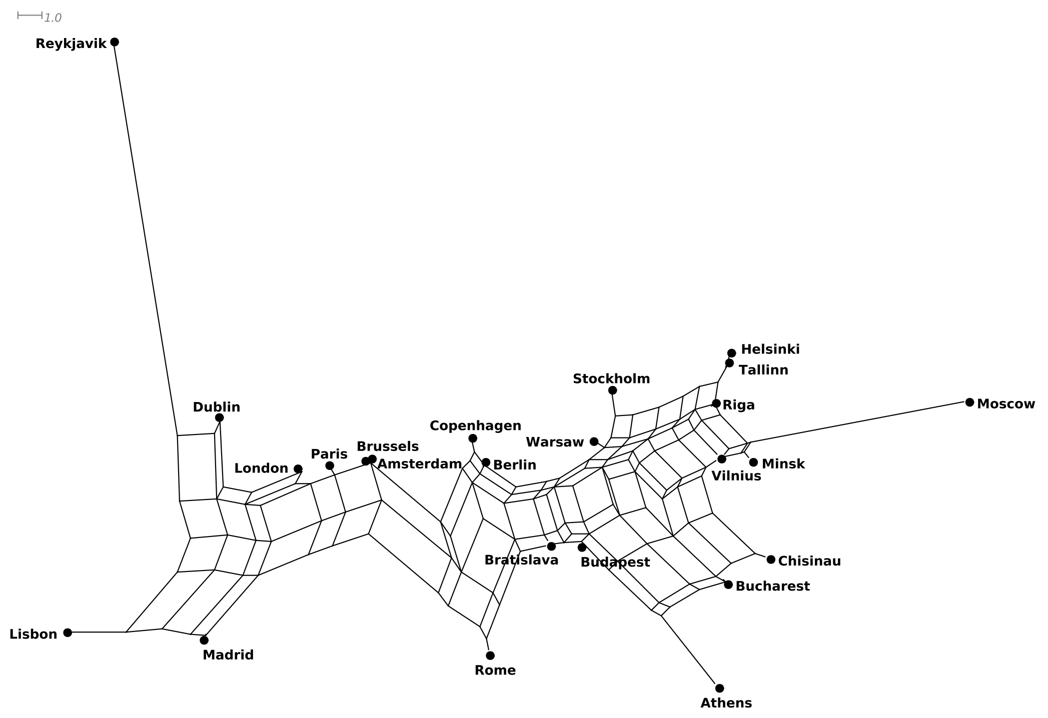
5.5 Discussion

In the analysis of recombination the added value of having labels inside the network is mainly the flexibility gained by representing collections of splits that cannot be displayed with outer-labeled networks. We also saw that ancestral sequences can be naturally placed by FlatNJ in the interior of the network, which not only helps avoid unnecessary distortion in the representation, but might potentially help to identify candidate ancestral sequences in situations where these are not known. In the third data set geographic considerations were of interest and we demonstrated that FlatNJ could also be useful for analyzing and visualizing such data. The final example illustrates FlatNJ's suitability for the data that is planar. However, one should be careful when applying any of the methods discussed and ideally try at least a few of them to find the best fit.

Even though we have found that FlatNJ is able to visualize more information than Neighbor-Net this can come at a price: To avoid distortion FlatNJ sometimes uses

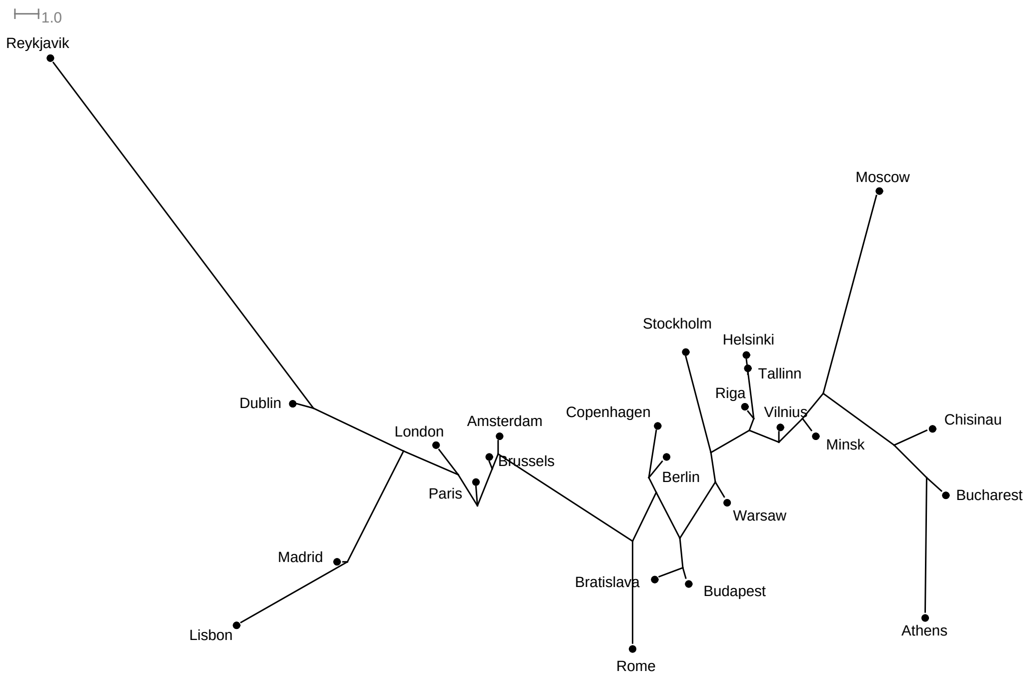


FlatNJ ($\sigma_1 = 0.191$)



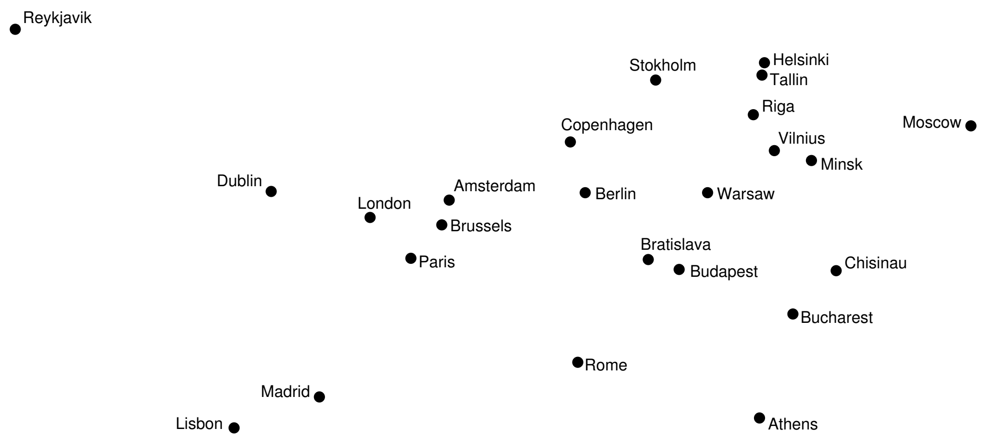
Neighbor-Net ($\sigma_1 = 0.231$)

Figure 5.7: FlatNJ and Neighbor-Net networks of the European capitals.



Neighbor-Joining ($\sigma_1 = 0.397$)

Figure 5.8: A Neighbor-Joining tree of the European capitals.



Multidimensional scaling ($\sigma_1 = 0.000$)

Figure 5.9: A multidimensional scaling plot of the European capitals.

more pairs of incompatible splits to represent the data than NeighborNet (see, e.g., the networks N5 and N8 in Figure 3.1). Moreover, we have found that producing a suitable layout of the labels of interior vertices can be quite challenging, especially for data where large groups of taxa label the inside of the network as in Figure 5.4. Developing alternative ways to draw the network that address this would be desirable. More generally, although having a planar network can be useful for interpreting data, as noted in Huson and Bryant (2006), some data sets are intrinsically better represented by high-dimensional, non-planar networks such as the ones that can be generated using split decomposition. It is therefore still an interesting challenge to develop methods to help effectively construct and visualize such networks.

Appendices

Appendix A

Data sets

A.1 Fluorescent protein data set

Id	Accession number	Color	Organism	Source
mc1	AY181552	red	*Montastraea cavernosa*	Kelmanson and Matz (2003)
mc2	AY181553	green	*Montastraea cavernosa*	Kelmanson and Matz (2003)
mc3	AY181554	green	*Montastraea cavernosa*	Kelmanson and Matz (2003)
mc4	AY181555	green	*Montastraea cavernosa*	Kelmanson and Matz (2003)
mc5	AY181556	cyan	*Montastraea cavernosa*	Kelmanson and Matz (2003)
G1_2	AY182020	green	synthetic	Kelmanson and Matz (2003)
G5_2	AY182023	cyan	synthetic	Kelmanson and Matz (2003)
R1_2	AY182013	red	synthetic	Kelmanson and Matz (2003)
R2	AY182014	green	synthetic	Kelmanson and Matz (2003)
Kaede	AB085641	red	*Trachyphyllia geoffroyi*	Ando *et al.* (2002)
ALL	AY648253	—	synthetic	Ugalde *et al.* (2004)
Red/Green	AY648241	—	synthetic	Ugalde *et al.* (2004)
pre-Red	AY648264	—	synthetic	Ugalde *et al.* (2004)
Red	AY648275	—	synthetic	Ugalde *et al.* (2004)

Table A.1: Sequences identifiers, accession numbers and additional information for the amino acid sequences in the fluorescent protein data set.

A.2 Gall wasp data set

Id	Accession number	Country	Id	Accession number	Country	Id	Accession number	Country
(1)	AF242739	Hungary	(28)	EF031373	France	(55)	EF031411	Hungary
(2)	AF242740	Hungary	(29)	EF031374	France	(56)	EF031412	Italy
(3)	AF242741	Italy	(30)	EF031375	Spain	(57)	EF031414	Netherlands
(4)	AF242742	Spain	(31)	EF031376	France	(58)	EF031416	Netherlands
(5)	AF242743	Spain	(32)	EF031378	Spain	(59)	EF031418	Hungary
(6)	AF242744	Turkey	(33)	EF031379	Portugal	(60)	EF031419	France
(7)	AF242746	Hungary	(34)	EF031380	Spain	(61)	EF031420	France
(8)	AF242747	France	(35)	EF031381	France	(62)	EF031421	France
(9)	AF242749	France	(36)	EF031385	Spain	(63)	EF031422	Hungary
(10)	AF242752	France	(37)	EF031386	Spain	(64)	EF031423	Hungary
(11)	AF242753	Italy	(38)	EF031387	Spain	(65)	EF031424	Hungary
(12)	AF242754	Italy	(39)	EF031388	France	(66)	EF031431	UK
(13)	AF242757	France	(40)	EF031390	Spain	(67)	EF031432	UK
(14)	AF242758	France	(41)	EF031391	Spain	(68)	EF031433	Ireland
(15)	AF242759	France	(42)	EF031392	France	(69)	EF031434	Ireland
(16)	AF242761	France	(43)	EF031393	Italy	(70)	EF031435	Ireland
(17)	AF242764	France	(44)	EF031394	Germany	(71)	EF031438	France
(18)	AF242765	Spain	(45)	EF031397	Turkey	(72)	EF031439	UK
(19)	AF242766	Spain	(46)	EF031398	Italy	(73)	EF031440	UK
(20)	EF031335	France	(47)	EF031399	Netherlands	(74)	EF031442	UK
(21)	EF031337	France	(48)	EF031400	Netherlands	(75)	EF031443	UK
(22)	EF031338	France	(49)	EF031401	Germany	(76)	EF031444	UK
(23)	EF031339	France	(50)	EF031402	France	(77)	EF031445	UK
(24)	EF031350	France	(51)	EF031403	France	(78)	EF031446	UK
(25)	EF031351	France	(52)	EF031404	France	(79)	EF031447	UK
(26)	EF031352	France	(53)	EF031406	France	(80)	EF031437	Hungary
(27)	EF031353	France	(54)	EF031408	Hungary			

Table A.2: Sequence identifiers with corresponding accession numbers and countries of origin for the gall wasp data set. UK stands for United Kingdom.

A.3 HIV pairwise dissimilarity data

	CRF49			A						B				C				D				F								G				H				J			K	
CRF49	0.00	0.06	0.07	0.14	0.14	0.14	0.15	0.15	0.15	0.14	0.15	0.15	0.15	0.14	0.14	0.13	0.14	0.14	0.15	0.14	0.15	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.14	0.14	0.12	0.12	0.11	0.15	0.14
	0.06	0.00	0.06	0.14	0.13	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.15	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.12	0.11	0.11	0.14	0.14
	0.07	0.06	0.00	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.15	0.14	0.14	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.12	0.11	0.11	0.14	0.14
A	0.14	0.14	0.14	0.00	0.08	0.08	0.12	0.12	0.12	0.14	0.15	0.14	0.15	0.15	0.15	0.15	0.15	0.14	0.15	0.15	0.15	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.13	0.14	0.14	0.14	0.15	0.14	0.15	0.15	0.14	0.14	0.14	0.14
	0.14	0.13	0.14	0.08	0.00	0.07	0.11	0.12	0.11	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14
	0.14	0.14	0.14	0.08	0.07	0.00	0.11	0.12	0.11	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.15	0.13	0.14	0.14	0.14
	0.15	0.14	0.14	0.12	0.11	0.11	0.00	0.07	0.07	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.15	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.15	0.14
	0.15	0.15	0.15	0.12	0.12	0.12	0.07	0.00	0.07	0.14	0.14	0.14	0.15	0.15	0.14	0.14	0.15	0.14	0.14	0.15	0.15	0.15	0.15	0.15	0.15	0.14	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.15	0.14	0.15	0.15	0.14
	0.15	0.14	0.14	0.12	0.11	0.11	0.07	0.07	0.00	0.14	0.14	0.14	0.15	0.15	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.15	0.14
B	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.00	0.07	0.05	0.07	0.13	0.14	0.14	0.14	0.09	0.11	0.10	0.10	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.13	0.13	0.13	0.13
	0.15	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.07	0.00	0.07	0.08	0.14	0.14	0.14	0.15	0.10	0.11	0.11	0.11	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13
	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.05	0.07	0.00	0.07	0.14	0.14	0.14	0.14	0.10	0.11	0.11	0.11	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.13
	0.15	0.14	0.14	0.15	0.14	0.14	0.14	0.15	0.15	0.07	0.08	0.07	0.00	0.14	0.14	0.14	0.15	0.11	0.12	0.11	0.11	0.13	0.13	0.13	0.13	0.14	0.13	0.14	0.14	0.15	0.15	0.15	0.15	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.13
C	0.14	0.14	0.14	0.15	0.14	0.14	0.15	0.15	0.15	0.13	0.14	0.14	0.14	0.00	0.07	0.08	0.09	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14
	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.07	0.00	0.08	0.09	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.15	0.15	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14
	0.13	0.13	0.13	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.08	0.08	0.00	0.08	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.14	0.15	0.14	0.14	0.14	0.13	0.13	0.13	0.14	0.14	0.13	0.14	0.14	0.13
	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.15	0.15	0.14	0.15	0.14	0.15	0.09	0.09	0.08	0.00	0.14	0.15	0.15	0.15	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.14	0.14	0.14	0.14	0.15	0.14	0.15	0.15	0.14
D	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.09	0.10	0.10	0.11	0.14	0.14	0.13	0.14	0.00	0.08	0.08	0.08	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.13	0.14	0.13	0.13
	0.15	0.15	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.11	0.11	0.11	0.12	0.14	0.14	0.14	0.15	0.08	0.00	0.10	0.10	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.13	0.14	0.13	0.13
	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.15	0.14	0.10	0.11	0.11	0.11	0.14	0.14	0.14	0.15	0.08	0.10	0.00	0.08	0.14	0.13	0.14	0.14	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.13
	0.15	0.15	0.14	0.15	0.14	0.14	0.15	0.15	0.14	0.10	0.11	0.11	0.11	0.14	0.14	0.14	0.15	0.08	0.10	0.08	0.00	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.13
F	0.14	0.14	0.15	0.15	0.14	0.14	0.14	0.15	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.13	0.00	0.06	0.06	0.08	0.10	0.10	0.10	0.10	0.14	0.14	0.15	0.14	0.14	0.14	0.13	0.14	0.15	0.14	0.14	0.12	0.12
	0.15	0.14	0.15	0.15	0.14	0.14	0.14	0.15	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.13	0.14	0.13	0.13	0.06	0.00	0.06	0.08	0.10	0.10	0.10	0.10	0.14	0.14	0.14	0.14	0.13	0.14	0.13	0.14	0.14	0.14	0.14	0.12	0.11
	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.13	0.06	0.06	0.00	0.07	0.10	0.10	0.10	0.10	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.14	0.14	0.12	0.11
	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.15	0.13	0.13	0.14	0.13	0.08	0.08	0.07	0.00	0.10	0.10	0.10	0.10	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.14	0.12	0.12
	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.14	0.14	0.14	0.13	0.14	0.13	0.13	0.13	0.13	0.10	0.10	0.10	0.10	0.00	0.07	0.07	0.06	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.13	0.13	0.12	0.11
	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.13	0.14	0.13	0.14	0.13	0.13	0.10	0.10	0.10	0.10	0.07	0.00	0.07	0.07	0.14	0.14	0.14	0.14	0.13	0.14	0.13	0.13	0.14	0.14	0.14	0.12	0.12
	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.14	0.13	0.13	0.13	0.14	0.14	0.14	0.13	0.14	0.13	0.14	0.13	0.13	0.10	0.10	0.10	0.10	0.07	0.07	0.00	0.06	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.12	0.12
	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.10	0.10	0.10	0.10	0.06	0.07	0.06	0.00	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.13	0.12	0.12
G	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.00	0.08	0.07	0.09	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.15	0.14
	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.08	0.00	0.07	0.07	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14
	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.14	0.15	0.14	0.14	0.14	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.07	0.07	0.00	0.08	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14
	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.14	0.15	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.09	0.07	0.08	0.00	0.14	0.15	0.14	0.14	0.14	0.13	0.13	0.14	0.14
H	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.00	0.09	0.09	0.08	0.14	0.13	0.13	0.14	0.13
	0.15	0.14	0.14	0.15	0.14	0.14	0.14	0.15	0.14	0.13	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.13	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.14	0.15	0.09	0.00	0.06	0.08	0.14	0.14	0.14	0.14	0.13
	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.13	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.09	0.06	0.00	0.08	0.14	0.14	0.14	0.13	0.13
	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.08	0.08	0.08	0.00	0.14	0.13	0.13	0.14	0.13
J	0.12	0.12	0.12	0.15	0.14	0.15	0.15	0.15	0.15	0.14	0.14	0.14	0.15	0.15	0.15	0.14	0.15	0.14	0.15	0.14	0.15	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.00	0.09	0.09	0.14	0.13
	0.12	0.11	0.11	0.14	0.14	0.13	0.14	0.14	0.14	0.13	0.14	0.13	0.14	0.14	0.14	0.13	0.14	0.13	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.13	0.14	0.14	0.13	0.09	0.00	0.07	0.13	0.13
	0.11	0.11	0.11	0.14	0.13	0.14	0.14	0.15	0.14	0.13	0.14	0.13	0.14	0.14	0.14	0.14	0.15	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.13	0.13	0.13	0.14	0.13	0.13	0.14	0.14	0.13	0.09	0.07	0.00	0.13	0.13
K	0.15	0.14	0.14	0.14	0.14	0.14	0.15	0.15	0.15	0.13	0.13	0.14	0.14	0.14	0.14	0.14	0.15	0.13	0.13	0.14	0.14	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.15	0.14	0.14	0.14	0.14	0.14	0.13	0.14	0.14	0.13	0.13	0.00	0.07
	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.14	0.14	0.13	0.14	0.13	0.13	0.13	0.13	0.12	0.11	0.11	0.12	0.11	0.12	0.12	0.12	0.14	0.14	0.14	0.14	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.07	0.00

Table A.3: Pairwise dissimilarities between sequences in the HIV data set. Cells are colored by the dissimilarity measures, the redder the cell, the more similar are the corresponding sequences.

Appendix B

Neighbor finding on a real line

It is easy to see that for any three points p_1, p_2, p_3 on ℓ such that $x_1 \leq x_2 \leq x_3$ the following holds:

$$d_{12} + d_{23} = d_{13} \tag{B.1}$$

Here $d_{ij} = |x_j - x_i|$ is a distance between two points i and j.

From equation B.1 we derive our first condition that could be used for the identification of neighbors:

$$\sum_{i \in P \setminus \{p', p''\}} (|\delta_{ip'} - \delta_{ip''}| - \delta_{p'p''})^2 = 0 \tag{B.2}$$

If any of the points in $P \setminus \{p', p''\}$ lay in between p' and p'' then equation B.2 is not satisfied. However, if proximities come from data that has dimension d_h higher than one, then eq. B.2 would not hold either. Therefore instead of looking for a pair of objects that satisfy B.2, we look for a pair for which this sum is minimal:

$$\sum_{i \in P \setminus \{p', p''\}} (|\delta_{ip'} - \delta_{ip''}| - \delta_{p'p''})^2 \to \min\,. \tag{C1}$$

Another, much simpler criterion comes from the same definition 4.4.1. It picks as neighbors two objects p' and p'' with the lowest pairwise proximity:

$$\delta_{p'p''} \to \min\,. \tag{C2}$$

As the order in which we agglomerate has an effect on the ordering that we get on expansion, we decide to add one more criterion which has better defined order of agglomeration. Let point i be the point that has smallest average proximity to all other points, then we take i and its nearest neighbor as a pair of neighbors:

$$\delta_{ij} \to \min : \ \delta_{ik} \to \min \frac{1}{n} \sum_{j}^{n} \tag{C3}$$

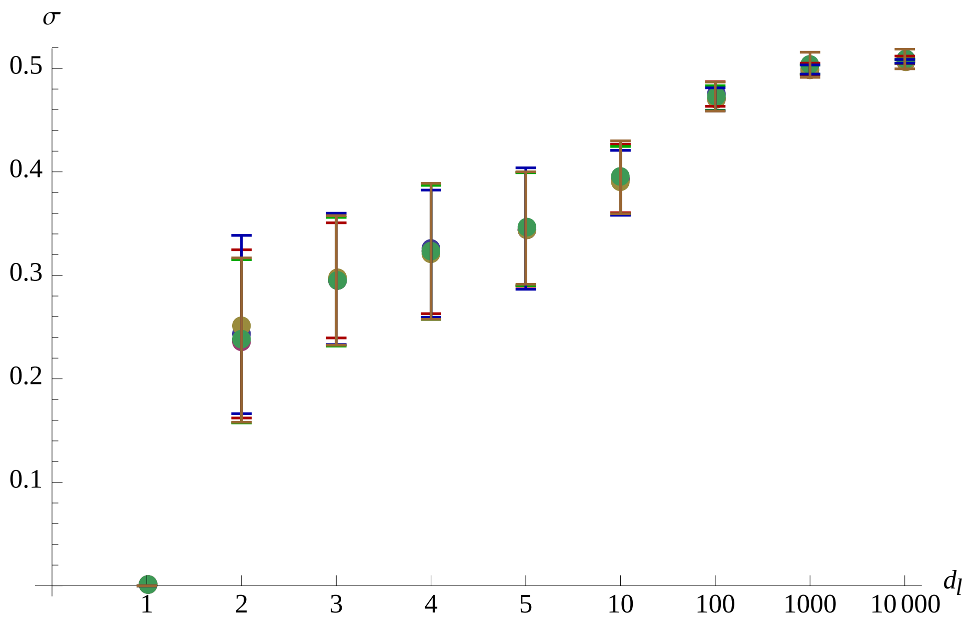


Figure B.1: Error bars for all four neighbor selection criteria discussed in the text.

Complexity of the C3 is the same as C2 as it takes $O(n^2)$ computations to identify the first neighbor and then $O(n)$ to find its nearest neighbor.

All three criteria are effective as they identify true neighbors when applied to the one dimensional data. However, their computational complexities are far from ideal. In case of the criterion C1 computing score for each pair requires $O(n)$ operations. Multiplying it by the number of pairs which is quadratic and the linear time of iterations we find that overall complexity is $O(n^4)$. Second criterion is somewhat lighter as it takes only one operation to compute score for each pair. Consequently overall complexity for the criterion C2 is $O(n^3)$. It is still too large for large datasets. Therefore we add one more criteria which is a generalization of C2. We say that two objects p' and p'' are neighbors if they are each others reciprocal nearest neighbors:

$$\begin{aligned} NN(p') &= p'' \text{ and} \\ NN(p'') &= p' \end{aligned} \tag{C4}$$

Graph in Figure B.1 shows error bars that we get when using each of the criteria C1, C2, C3 and C4 for different dimensions. The resulting stress values do not differ among criteria much, therefore we choose to use the one with lowest computational complexity, that is C4.

References

Ando, R., Hama, H., Yamamoto-Hino, M., Mizuno, H., and Miyawaki, A. (2002). An optical marker based on the UV-induced green-to-red photoconversion of a fluorescent protein. *Proceedings of the National Academy of Sciences of the United States of America*, *99*(20), 12651–12656.

Atteson, K. (1999). The performance of the neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, *25*, 251–278.

Balaram, P. and Kaas, J. (2014). Current research on the organization and function of the visual system in primates. *Eye and Brain*, *6*(Thematic series: Organization and function of the visual system in primates), 1–4.

Balvočiūtė, M., Spillner, A., and Moulton, V. (2014). FlatNJ: A novel network-based approach to visualize evolutionary and biogeographical relationships. *Systematic Biology*, *63*(3), 383–396.

Bandelt, H.-J. and Dress, A. (1993). A relational approach to split decomposition. In O. Opitz, B. Lausen, and R. Klar (Eds.), *Information and Classification*, 123–131. Springer, Berlin Heidelberg.

Bandelt, H.-J. and Dress, A. W. (1992a). A canonical decomposition theory for metrics on a finite set. *Advances in Mathematics*, *92*(1), 47–105.

Bandelt, H.-J. and Dress, A. W. (1992b). Split decomposition: A new and useful approach to phylogenetic analysis of distance data. *Molecular Phylogenetics and Evolution*, *1*(3), 242–252.

Barthelemy, J. (1989). From copair hypergraphs to median graphs with latent vertices. *Discrete Mathematics*, *76*(1), 9–28.

Beiko, R. (2011). Telling the whole story in a 10,000-genome world. *Biology Direct*, *6*(1), 34.

Belkin, M. and Niyogi, P. (2003). Laplacian Eigenmaps for dimensionality reduction and data representation. *Neural Computation*, *15*(6), 1373–1396.

Benzécri, J.-P. (1982). Construction d'une classification ascendante hiérarchique par la recherche en chaîne des voisins réciproques. *Cahiers de l'analyse des données*, *7*(2), 209–218.

Björner, A., Vergnas, M. L., Sturmfels, B., White, N., and Ziegler, G. M. (1999). *Oriented Matroids.* Cambridge University Press, New York.

Bohne, J. (1992). *Eine kombinatorische Analyse zonotopaler Raumaufteilungen.* Ph. D. thesis, Bielefeld University.

Borg, I. and Groenen, P. J. F. (2005). Classical scaling. In *Modern Multidimensional Scaling*, Springer Series in Statistics, 261–267. Springer-Verlag, New York.

Bryant, D. (2005a). Extending tree models to split networks. In L. Pachter and B. Sturmfels (Eds.), *Algebraic Statistics for Computational Biology*, 322–334. Cambridge University Press.

Bryant, D. (2005b). On the uniqueness of the selection criterion in neighbor-joining. *Journal of Classification*, *22*(1), 3–15.

Bryant, D. and Dress, A. (2007). Linearly independent split systems. *European Journal of Combinatorics*, *28*(6), 1814–1831.

Bryant, D. and Moulton, V. (2004). Neighbor-Net: An agglomerative method for the construction of phylogenetic networks. *Molecular Biology and Evolution*, *21*(2), 255–265.

Bryant, D., Moulton, V., and Spillner, A. (2007). Consistency of the neighbor-net algorithm. *Algorithms for Molecular Biology*, *2*(1), 8.

Buaba, R., Homaifar, A., Hendrix, W., Son, S. W., Liao, W.-k., and Choudhary, A. (2014). Randomized algorithm for approximate nearest neighbor search in high dimensions. *Journal of Pattern Recognition Research*, *1*, 111–122.

Buchta, C., Hornik, K., and Hahsler, M. (2008). Getting things in order: an introduction to the R package seriation. *Journal of Statistical Software*, *25*(3), 1–34.

Buneman, P. (1971). The recovery of trees from measures of dissimilarity. In F. H. et al. (Ed.), *Mathematics in the Archaeological and Historical Sciences*, 387–395. Edinburgh University Press, Edinburgh.

Busing, F., Commandeur, J. J., Heiser, W. J., Bandilla, W., and Faulbaum, F. (1997). PROXSCAL: A multidimensional scaling program for individual differences scaling with constraints. *Softstat*, *97*, 67–74.

Castresana, J. (2000). Selection of conserved blocks from multiple alignments for their use in phylogenetic analysis. *Molecular Biology and Evolution*, *17*(4), 540–552.

Chalmers, M. (1996). A linear iteration time layout algorithm for visualising high-dimensional data. In *Visualization '96. Proceedings*, 127–131.

Charleston, M. A., Hendy, M. D., and Penny, D. (1993). Neighbor-joining uses the optimal weight for net divergence. *Molecular phylogenetics and evolution*, *2*(1), 6–12.

Charleston, M. A., Hendy, M. D., and Penny, D. (1994). The effects of sequence length, tree topology, and number of taxa on the performance of phylogenetic methods. *Journal of Computational Biology*, *1*(2), 133–151.

Clark, D. P. and Pazdernik, N. J. (2013). *Molecular Biology* (2nd ed.). Academic Cell, Waltham.

Cuadros, A., Paulovich, F., Minghim, R., and Telles, G. (2007). Point placement by phylogenetic trees and its application to visual analysis of document collections. In *Visual Analytics Science and Technology, 2007. VAST 2007. IEEE Symposium on*, 99–106.

De Leeuw, J. (1977). Applications of convex analysis to multidimensional scaling. In *Recent Developments in Statistics*, 133–145. North-Holland Publishing Company, Amsterdam.

De Leeuw, J. and J., H. W. (1977). Convergence of correction-matrix algorithms for multidimensional scaling. In *Geometric Representations of Relational Data*, 735–751. Mathesis Press, Michigan.

de Silva, T., Turner, R., Hué, S., Trikha, R., van Tienen, C., Onyango, C., Jaye, A., Foley, B., Whittle, H., Rowland-Jones, S., and Cotten, M. (2010). HIV-1 subtype distribution in the Gambia and the significant presence of CRF49_cpx, a novel circulating recombinant form. *Retrovirology*, *7*(82), 1–14.

DeBry, R. W. (1992). The consistency of several phylogeny-inference methods under varying evolutionary rates. *Molecular Biology and Evolution*, *9*(3), 537–551.

Djoković, D. (1973). Distance-preserving subgraphs of hypercubes. *Journal of Combinatorial Theory, Series B*, *14*(3), 263–267.

Donoho, D. L. (2000). High-dimensional data analysis: The curses and blessings of dimensionality. Aide-Memoire of a Lecture at. In *AMS Conference on Math Challenges of the 21st Century*.

Dress, A. W. and Huson, D. H. (2004). Constructing splits graphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *1*(3), 109–115.

Dunn, M., Terrill, A., Reesink, G., Foley, R., and Levinson, S. (2005). Structural phylogenetics and the reconstruction of ancient language history. *Science*, *309*, 2072–2075.

Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, *42*, 146–160.

Eickmeyer, K., Huggins, P., Pachter, L., and Yoshida, R. (2008). On the optimality of the neighbor-joining algorithm. *Algorithms for Molecular Biology*, *3*, 1 – 11.

Eigen, M., Winkler-Oswatitsch, R., and Dress, A. (1988). Statistical geometry in sequence space: a method of quantitative comparative sequence analysis. *Proceedings of the National Academy of Sciences of the United States of America*, *85*, 5913–5917.

Engel, D., Rosenbaum, R., Hamann, B., and Hagen, H. (2011). Structural decomposition trees. *Computer Graphics Forum*, *30*(3), 921–930.

Eppstein, D. (2005). Algorithms for drawing media. In J. Pach (Ed.), *Graph Drawing*, Volume 3383 of *Lecture Notes in Computer Science*, 173–183. Springer Berlin Heidelberg.

Eppstein, D., Falmagne, J.-C., and Ovchinnikov, S. (2008). *Media Theory*. Springer-Verlag, Berlin Heidelberg.

Felsner, S. and Weil, H. (2001). Sweeps, arrangements and signotopes. *Discrete Applied Mathematics*, *109*(1–2), 67–94. 14th European Workshop on Computational Geometry.

Florek, K., Łukaszewicz, J., Perkal, J., Steinhaus, H., and Zubrzycki, S. (1951). Sur la liaison et la division des points d'un ensemble fini. *Colloquium Mathematicae*, *2*(3–4), 282–285.

Folkman, J. and Lawrence, J. (1978). Oriented matroids. *Journal of Combinatorial Theory, Series B*, *25*(2), 199–236.

France, S. and Carroll, J. (2011). Two-way multidimensional scaling: A review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *41*(5), 644–661.

Fukuda, K. and Handa, K. (1993). Antipodal graphs and oriented matroids. *Discrete Mathematics*, *111*(1–3), 245–256.

Garriga, G. C., Junttila, E., and Mannila, H. (2011). Banded structure in binary matrices. *Knowledge and Information Systems*, *28*(1), 197–226.

Gärtner, B. and Welzl, E. (1997). Vapnik-Chervonenkis dimension and (pseudo-)hyperplane arrangements.

Gascuel, O. (1997). BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, *14*(7), 685–695.

Goodman, J. E. (1980). Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discrete Mathematics*, *32*(1), 27–35.

Goodman, J. E. and Pollack, R. (1980). On the combinatorial classification of nondegenerate configurations in the plane. *Journal of Combinatorial Theory, Series A*, *29*(2), 220–235.

Goodman, J. E. and Pollack, R. (1982). A theorem of ordered duality. *Geometriae Dedicata*, *12*(1), 63–74.

Goremykin, V. V., Nikiforova, S. V., Biggs, P. J., Zhong, B., Delange, P., Martin, W., Woetzel, S., Atherton, R. A., Mclenachan, P. A., and Lockhart, P. J. (2013). The evolutionary root of flowering plants. *Systematic Biology*, *62*(1), 50–61.

Gower, J. C. (1966). Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, *53*(3–4), 325–338.

Gower, J. C. (1967a). A comparison of some methods of cluster analysis. *Biometrics*, *23*(4), 623–637.

Gower, J. C. (1967b). Multivariate analysis and multidimensional geometry. *Journal of the Royal Statistical Society. Series D (The Statistician)*, *17*(1), 13–28.

Grünewald, S., Forslund, K., Dress, A., and Moulton, V. (2007). QNet: an agglomerative method for the construction of phylogenetic networks from weighted quartets. *Molecular Biology and Evolution*, *24*, 532–538.

Grünewald, S., Moulton, V., and Spillner, A. (2009). Consistency of the QNet algorithm for generating planar split graphs from weighted quartets. *Discrete Applied Mathematics*, *157*, 2325–2334.

Gupta, A. (2000). Embedding tree metrics into low-dimensional Euclidean spaces. *Discrete & Computational Geometry*, *24*(1), 105–116.

Gurobi Optimization, Inc. (2015). Gurobi Optimizer Reference Manual,. http://www.gurobi.com.

Handa, K. (1990). A characterization of oriented matroids in terms of topes. *European Journal of Combinatorics*, *11*(1), 41–45.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning* (2nd ed.). Springer-Verlag, New York.

Haws, D., Hodge, T., and Yoshida, R. (2011). Optimality of the neighbor joining algorithm and faces of the balanced minimum evolution polytope. *Bulletin of Mathematical Biology*, *73*(11), 2627–2648.

Huson, D., Rupp, R., and Scornavacca, C. (2010). *Phylogenetic networks.* Cambridge University Press, New York.

Huson, D. H. (1998). SplitsTree: analyzing and visualizing evolutionary data. *Bioinformatics*, *14*(1), 68–73.

Huson, D. H. and Bryant, D. (2006). Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, *23*(2), 254–267.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 604–613. ACM.

Kaski, S. (1998). Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, Volume 1, 413–418. IEEE.

Katoh, K., Misawa, K., Kuma, K., and Miyata, T. (2002). MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, *30*(14), 3059–3066.

Kelmanson, I. V. and Matz, M. V. (2003). Molecular basis and evolutionary origins of color diversity in great star coral Montastraea cavernosa (Scleractinia: Faviida). *Molecular Biology and Evolution*, *20*(7), 1125–1133.

Kendall, D. (1969). Incidence matrices, interval graphs and seriation in archeology. *Pacific Journal of Mathematics*, *28*(3), 565–570.

Klavzar, S. and Mulder, H. M. (2002). Partial cubes and crossing graphs. *SIAM Journal on Discrete Mathematics*, *15*(2), 235–251.

Kruskal, J. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, *29*(1), 1–27.

Kuhner, M. K. and Felsenstein, J. (1994). A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, *11*(3), 459–468.

Kumar, S. and Gadagkar, S. R. (2000). Efficiency of the neighbor-joining method in reconstructing deep and shallow evolutionary relationships in large phylogenies. *Journal of Molecular Evolution*, *51*(6), 544–553.

Lawson, C. and Hanson, R. (1974). *Solving Least Squares Problems.* Prentice Hall Inc., Englewood Cliff, New Jersey.

Lekkeikerker, C. and Boland, J. (1962). Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, *1*(51), 45–64.

Lespinats, S., Fertil, B., Villemain, P., and Hérault, J. (2009). RankVisu: Mapping from the neighborhood network. *Neurocomputing*, *72*(13), 2964–2978.

Lespinats, S., Verleysen, M., Giron, A., and Fertil, B. (2007). DD-HDS: A method for visualization and exploration of high-dimensional data. *IEEE Transactions on Neural Networks*, *18*(5), 1265–1279.

Levi, F. (1926). Die Teilung der projektiven Ebene durch Gerade oder Pseudogerade. *Ber. Math.-Phys. Kl. Sächs. Akad. Wiss*, *78*, 256–267.

Malone, S. W., Tarazaga, P., and Trosset, M. W. (2002). Better initial configurations for metric multidimensional scaling. *Computational Statistics & Data Analysis*, *41*(1), 143–156.

Mardia, K. V. (1978). Some properties of classical multi-dimesional scaling. *Communications in Statistics-Theory and Methods*, *7*(13), 1233–1241.

McQuitty, L. L. (1957). Elementary linkage analysis for isolating orthogonal and oblique types and typal relevancies. *Educational and Psychological Measurement*, *17*(2), 207–229.

Morrison, A. and Chalmers, M. (2003). Improving hybrid MDS with pivot-based searching. In *Proceedings of the Ninth Annual IEEE Conference on Information Visualization*, INFOVIS'03, Washington, DC, USA, 85–90. IEEE Computer Society.

Morrison, A., Ross, G., and Chalmers, M. (2003). Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization*, *2*(1), 68–77.

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

Müllner, D. (2011). Modern hierarchical, agglomerative clustering algorithms. arXiv preprint arXiv:1109.2378.

Murtagh, F. (1984). Complexities of hierarchic clustering algorithms: State of the art. *Computational Statistics Quarterly*, *1*(2), 101–113.

Nei, M. (1991). Relative efficiencies of different tree-making methods for molecular data. In *Phylogenetic Analysis of DNA Sequences*, 90–128. Oxford University Press, New York.

Nieselt-Struwe, K. and von Haeseler, A. (2001). Quartet-mapping, a generalization of the likelihood-mapping procedure. *Molecular Biology and Evolution*, *18*, 1204–1219.

Octavia, S. and Lan, R. (2006). Frequent recombination and low level of clonality within *Salmonella enterica* subspecies I. *Microbiology*, *152*, 1099–1108.

Ottino, J. M. (2003). Is a picture worth 1,000 words? *Nature*, *421*(6922), 474–476.

Paiva, J., Florian, L., Pedrini, H., Telles, G., and Minghim, R. (2011). Improved similarity trees and their application to visual data classification. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2459–2468.

Pelletier, E., Saurin, W., Cheynier, R., Letvin, N. L., and Wain-Hobson, S. (1995). The tempo and mode of SIV quasispecies development *in vivo* calls for massive viral replication and clearance. *Virology*, *208*, 644–652.

Petrie, W. F. (1899). Sequences in prehistoric remains. *Journal of the Anthropological Institute of Great Britain and Ireland*, *29*(3/4), 295–301.

Pliner, V. M. (1984). A class of metric scaling models. *Automation and Remote Control*, *45*(6), 789–794.

Richter-Gebert, J. and Ziegler, G. M. (1994). Zonotopal tilings and the Bohne-Dress theorem. In *Jerusalem Combinatorics*, 211–232.

Richter-Gebert, J. and Ziegler, G. M. (2004). Oriented matroids. In *Handbook of Discrete and Computational Geometry*, 129–152. Chapman and Hall/CRC, Boca Raton.

Robinson, W. S. (1951). A method for chronologically ordering archaeological deposits. *American Antiquity*, *16*(4), 293–301.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*(5500), 2323–2326.

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

Sammon, J. W. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, *18*(5), 401–409.

Semple, C. and Steel, M. (2003). *Phylogenetics.* Oxford University Press, New York.

Shor, P. (1991). Stretchability of pseudoline arrangements is NP-hard,. In *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, 531–554. American Mathematical Society, Providence, RI.

Sneath, P. H. (1957). The application of computers to taxonomy. *Journal of General Microbiology*, *17*(1), 201–226.

Snoeyink, J. and Hershberger, J. (1989). Sweeping arrangements of curves. In *Proceedings of the fifth annual symposium on Computational geometry*, 354–363. ACM.

Sokal, R. R. and Michener, C. D. (1958). A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, *38*, 1409–1438.

Sørensen, T. (1948). A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biologiske Skrifter*, *5*, 1–34.

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

Stone, G., Atkinson, R., Rokas, A., Csóka, G., and Nieves-Aldrey, J. (2001). Differential success in northwards range expansion between ecotypes of the marble gallwasp *Andricus kollari*: a tale of two lifecycles. *Molecular Ecology*, *10*, 761–778.

Stone, G., Challis, R., Atkinson, R., Csóka, G., Hayward, A., Melika, G., Mutun, S., Preuss, S., Rokas, A., Sadeghi, E., and Schönrogge, K. (2007). The phylogeographical clade trade: tracing the impact of human-mediated dispersal on the colonization of northern Europe by the oak gallwasp *Andricus kollari. Molecular Ecology*, *16*, 2768–2781.

Studier, J. and Kepler, K. (1988). A note on the neighbour-joining method of Saitou and Nei. *Molecular Biology and Evolution*, *5*, 729–731.

Takane, Y., Young, F., and de Leeuw, J. (1977). Nonmetric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features. *Psychometrika*, *42*(1), 7–67.

The STAR Consortium (2008). SNP and haplotype mapping for genetic analysis in the rat. *Nature Genetics*, *40*, 560–566.

Torgerson, W. (1952). Multidimensional scaling: I. Theory and method. *Psychometrika*, *17*(4), 401–419.

Torgerson, W. S. (1958). *Theory and methods of scaling.* John Wiley & Sons, New York.

Tschirschnitz, F. (2001). Testing extendability for partial chirotopes is NP-complete. In *Proceedings of the 13th Canadian Conference on Computational Geometry, U. of*, 165–168.

Tugume, A., Mukasa, S., Kalkkinen, N., and Valkonen, J. (2010). Recombination and selection pressure in the ipomovirus sweet potato mild mottle virus (*Potyviridae*) in wild species and cultivated sweetpotato in the centre of evolution in East Africa. *Journal of General Virology*, *91*, 1092–1108.

Ugalde, J. A., Chang, B. S. W., and Matz, M. V. (2004). Evolution of coral pigments recreated. *Science*, *305*(5689), 1433.

Vach, W. and Degens, P. (1991). Least-squares approximation of additive trees to dissimilarities-characterizations and algorithms. *Computational Statistics Quarterly*, *3*, 203–218.

Wain-Hobson, S., Renoux-Elbé, C., Vartanian, J.-P., and Meyerhans, A. (2003). Network analysis of human and simian immunodeficiency virus sequence sets reveals massive recombination resulting in shorter pathways. *Journal of General Virology*, *84*(4), 885–895.

Walter, B., Bala, K., Kulkarni, M., and Pingali, K. (2008). Fast agglomerative clustering for rendering. In *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, 81–86. IEEE.

Wetzel, R. (1995). *Zur Visualisierung abstrakter Ähnlichkeitsbeziehungen.* Ph. D. thesis, Bielefeld University.

Winkworth, R., Bryant, D., Lockhart, P., Havell, D., and Moulton, V. (2005). Biogeographic interpretation of split graphs: least squares optimization of edge lengths. *Systematic Biology*, *54*, 56–65.

Yang, T., Liu, J., McMillan, L., and Wang, W. (2006). A fast approximation to multidimensional scaling. In *IEEE workshop on Computation Intensive Methods for Computer Vision*.

Young, G. and Householder, A. S. (1938). Discussion of a set of points in terms of their mutual distances. *Psychometrika*, *3*(1), 19–22.

Zaslavsky, T. (1975). Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Memoirs of the American Mathematical Society*, *1*(154), 1–102.