Divide-and-Conquer Neighbor-Joining Algorithm: $\mathcal{O}(N^3)$ Neighbor-Joining on Additive Distance Matrices

CHAN, Ho Fai

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Mathematics

Thesis/Assessment Committee

Prof. Thomas K.K. AU (Chairperson)

Prof. Raymond H.F. CHAN (Thesis supervisor)

Prof. Jun ZOU (Committee member)

Prof. Michael K.P. NG (External Examiner)

Abstract

An evolutionary history of a set of species can be visualized by phylogenetic trees. After giving pairwise distances between the species in the form of a distance matrix, Neighbor-Joining (NJ) is a well-known greedy algorithm that constructs such a tree with branch lengths. Owing to its speed and accuracy, NJ has been widely used by the phylogeny community.

In the thesis, we explore the properties of binary trees, where a binary tree is the mathematical model of a phylogenetic tree. We also prove a necessary and sufficient condition for neighbor pairs on a binary tree given the distance matrix corresponding to this tree. Based on these, we propose an algorithm of NJ on additive matrices using the idea of divide-and-conquer. Our experiments on the additive matrices outperforms the current algorithms of NJ implemented in MEGA and PHYLIP.

摘要

生物的演化歷史可用親緣樹(phylogenetic tree)來圖象化。鄰近連接法(Neighbor-Joining, NJ)是一個著名的貪婪算法(greedy algorithm)。它可找出一親緣樹來表示一些生物間之互相距離，這些距離通常會用距離矩陣(distance matrix)的形式來儲存。由於NJ的速度及準確度，它已被親緣分析者廣泛使用。

本文會研究親緣樹之數學模型——二分樹(Binary Tree)。我們也會證出一個充要條件去用對應於一棵二分樹之距離矩陣去找出它所有的鄰對(neighbor pairs) 。基於這些結果，我們會對NJ提出一個有分治理念(divide-and-conquer)的新算法并將其應用在相離矩陣(additive matrix)。在對於相離矩陣之實驗中，這個新算法的速度比知名軟件MEGA、PHYLIP中的NJ算法更快。

ACKNOWLEDGMENTS

The completion of a thesis is never easy. I would like to express my thanks to Prof. Raymond H.F. Chan, his patience, always–listening, professorship and professionship has great influence in my personal philosophy and my ways of thinking.

I am very whole–heartedly grateful to my parents, whom have been unreservingly supportive and caring. I also wish to give my thanks to Ms. Kitty H.K. Chan, Ms. Kaman K.M. Chu for their sweetness, and also other family members for their supports. A few of my very best friends are also worth to mention as they have been very important to my growth: Mr. Wai-yip Leung, Mr. Lawrence K.T. Leung, Mr. Chris T.K. Wong. Last but not least, I have to thank my colleagues under the same big umbrella: Mr. Jian-feng Cai, Mr. Daniel K.T. Cheng, Ms. Yi-qiu Dong, Mr. Eric S.H. Lau, Ms. Hai-xia Liang, Mr. Wei Wang, Mr. Jesse Z.X. Wang.

Contents

Chapter 1

Introduction

The phylogenetic reconstruction problem is to determine evolutionary relationships from biological data. Usually, relationships between different organisms can be found by using external characteristics of a species, or, by the DNA sequences of a species. The overall relationship can be visualized using a tree, where the tips of the tree represent the current species and the internal nodes represent common ancestors of those descendants. The distance between any two species in the tree represents their level of dissimilarity. Phylogenetics itself is an important branch in biology, and also it has important applications such as classifying unknown species, for example, in the quick design of influenza vaccines when there is an outbreak.

Phylogenetics treats each species as a group of lineage-connected objects in time, where different species are linked by the process of evolution. Evolution is regarded as a diversifying process, whereby existing species populations are altered over time: differentiate into separate branches, hybridize together, or terminate by extinction. Therefore a tree model is very suitable in representing this biological structure. The 'Tree of Life' model (Figure 1.1) is believed within the biologist community and it follows the underlying principle of the 'Evolutionary Theory' proposed by Darwin. Both of them postulates that all different species existed on Earth have a common ancestor. Figure 1.2 is an example of a phy-

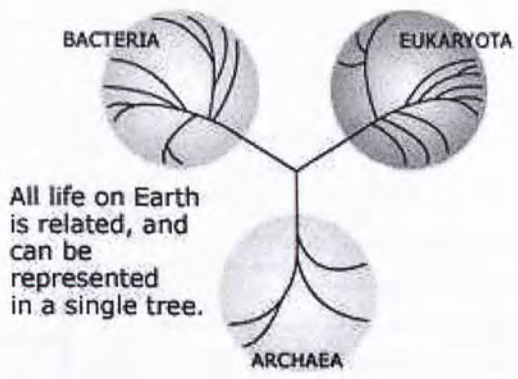
BACTERIA
EUKARYOTA
All life on Earth is related, and can be represented in a single tree.
ARCHAEA

Figure 1.1: The Tree of Life model.

logenetic tree appeared in [2]. For a detailed introduction on phylogeny, readers may refer to [10].

Over the past several decades, many methods in constructing phylogenetic trees have been proposed: Maximum Parsimony (Minimum Evolution) [7], Maximum Likelihood [5], Neighbor-Joining (NJ) [11], Unweighted Pair-Group Method using Arithmetic mean (UPGMA) [13]. Besides representing biological distances using trees, several other methods used other structures such as networks, for example, Splitstree [9], Quartnet [8]. Several computer programmes are available to draw the phylogenetic trees using NJ or other methods, a few to mention are MEGA [16], PHYLIP [6], PAUP [15], Splitstree [9].

The famous Neighbor-Joining algorithm was proposed by Saitou and Nei [11], and was improved to $\mathcal{O}(N^3)$ by Studier and Keppler [14], where N is the number of taxa. After given the pairwise biological distances between the interested species, the algorithm outputs a tree with branch lengths representing the dissimilarity between connected nodes. Besides NJ, there are other neighbor-joining

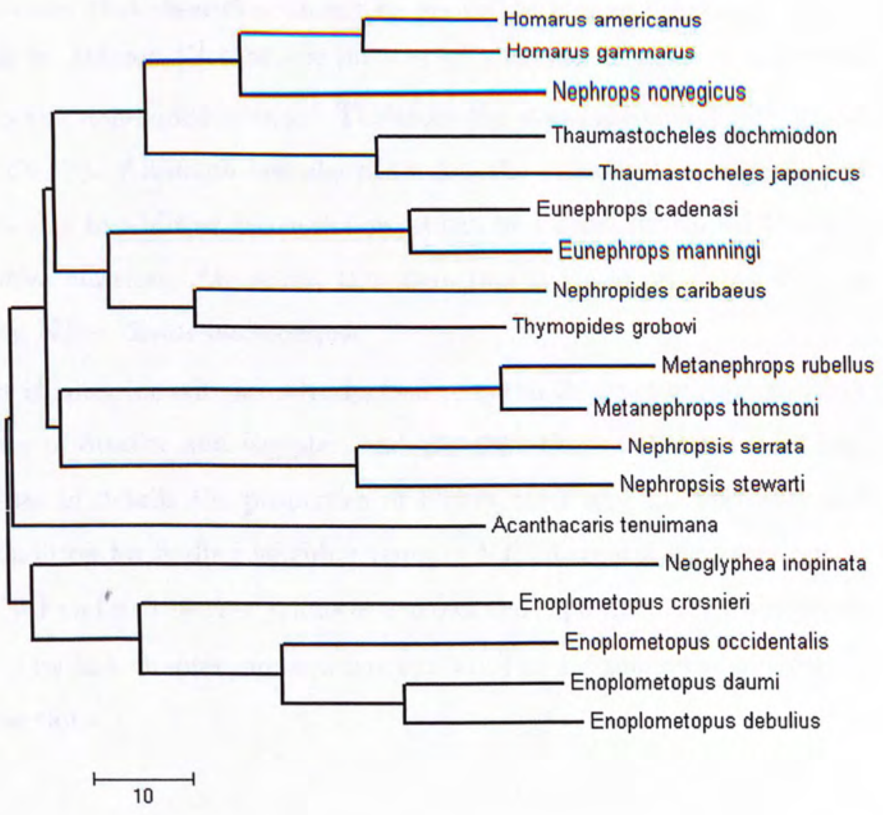

Figure 1.2: A phylogenetic tree of Nephropidae appeared in [2].

methods, such as, ADDTREE [12], Fast Neighbor-Joining (FNJ) [3], Relaxed Neighbor-Joining (RNJ) [4].

In the thesis, we develop a new algorithm named Divide-and-Conquer Neighbor-Joining (DCNJ) for additive matrices on implementing NJ. Although Waterman et al. [17] has already proposed an $\mathcal{O}(N^2)$ algorithm on implementing NJ on additive matrices, that algorithm cannot be applied to non-additive matrices. It is mentioned in Atteson [1] that the number of additive matrices is too small compared to the non-additive ones. Therefore the standard complexity in NJ now is still $\mathcal{O}(N^3)$. Although our algorithm has the complexity of $\mathcal{O}(N^3)$ and is only applicable to additive matrices now, it can be further developed to apply to non-additive matrices. Moreover, this algorithm is based on a new idea on implementing NJ — divide-and-conquer.

The next chapter includes an introduction to graph theory, the NJ algorithm, the algorithm of Studier and Keppler, and the algorithm of Waterman. Chapter 3 discusses in details the properties of binary trees and the necessary and sufficient condition for finding neighbor pairs in NJ. Chapter 4 describes our algorithm DCNJ and its theories. Chapter 5 shows the experimental results on the algorithm. The last chapter summarizes our work and discusses about further research directions.

Chapter 2

Current methods on Neighbor-Joining

In this chapter, an introduction to graph theory will be given first, followed by a general discussion on visualizing distance matrices. Then we introduce the original Neighbor-Joining (NJ) algorithm and its speedup versions proposed by Studier and Keppler [14], and Waterman et al. [17].

2.1 Introduction to graph theory

To construct a phylogenetic tree visualizing a distance matrix, it is better to understand the properties of trees first. Hence we briefly recall several results in graph theory related to trees. For further references, see [18].

Definition 2.1 (graph, vertex, edge). *A graph G is a collection of vertices (nodes) and a collection of edges (lines) that connect pairs of vertices. Usually, we use $v \in G$ to say that v is a vertex in G.*

Definition 2.2 (subgraph). *A graph S is a subgraph of G if all the vertices and edges in S appears in G.*

Definition 2.3 (adjacent vertex, vertex degree). *Two vertices are adjacent if there exists an edge between them. The degree of a vertex v, $\deg(v)$, is defined as the number of vertices adjacent to it.*

Definition 2.4 (end-vertex, internal vertex). *In a tree T, a vertex with degree 1 is called an end-vertex. We call a vertex on T an internal vertex if this vertex is not an end-vertex.*

Definition 2.5 (path). *A path is a finite collection of vertices and edges alternatively: $\{v_0, e_1, v_1, e_2, \cdots, e_K, v_K\}$, where e_i connects exactly the vertices v_{i-1} and v_i. A path for vertices i, j, denoted by $P_{i,j}$, is a path such that $v_0 = i$, $v_K = j$.*

Definition 2.6 (connected, disconnected). *A graph is connected if there exists at least one path between any two vertices. A graph is disconnected if it is not connected.*

Definition 2.7 (weighted graph). *A graph G is a weighted graph if there is a positive number associated with each edge. The weight (or length) of the edge is denoted by $w(e)$ (or $l(e)$).*

Figure 2.1 is an example of a weighted graph.

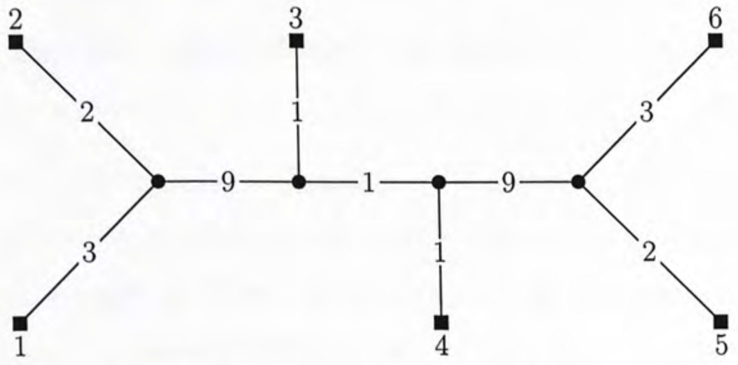

Figure 2.1: A graph with branch lengths.

Definition 2.8 (path length). *Let $P := \{v_0, e_1, v_1, e_2, \cdots, e_K, v_K\}$ be a path in a weighted graph G. The length of the path P, denoted by $l(P)$, is $\sum_{i=1}^{K} l(e_i)$.*

Proposition 2.1 (Handshaking Lemma). *In a graph G, the total sum of vertex degrees is equal to two times the number of edges, i.e.*

$$\sum_{v \in G} \deg(v) = 2 \times |E|, \tag{2.1}$$

where $|E|$ is the number of edges in G.

Proof. Since for each edge joining a pair of vertices, the total sum of vertex degree will be increased by two. Hence, $\sum_{v \in G} \deg(v) = 2 \times |E|$. $\square$

Definition 2.9 (tree). *A tree is a connected graph such that any pair of vertices are connected by a unique path.*

Definition 2.10 (distance between vertices). *If u, v are vertices in a connected tree T, then the distance between u, v in T is defined as:*

$$dist_T(u, v) := l(P_{u,v}).$$

Proposition 2.2 ([18] Theorem 9.1). *A tree T with M vertices has $M - 1$ edges.*

Definition 2.11 (binary tree). *A tree T is called a binary tree if all vertex degrees are either one or three.*

Figure 2.2 is an example of (unweighted) binary tree, with 13 edges and 14 vertices: 8 end-vertices and 6 internal vertices. In the thesis, a binary tree will always mean a weighted binary tree (see Figure 2.1) with end-vertices labeled $1, \cdots, N$, with $N \geq 4$ unless otherwise stated.

Definition 2.12 (neighbor pair). *In a binary tree T, two distinct end-vertices i, j form a neighbor pair, denoted by $i \sim j$, if the path $P_{i,j}$ contains only two edges.*

For example in Figure 2.2, $1 \sim 2, 5 \sim 6$ and $7 \sim 8$.

Usually, we view a binary tree as a composition of branch lengths and topology.

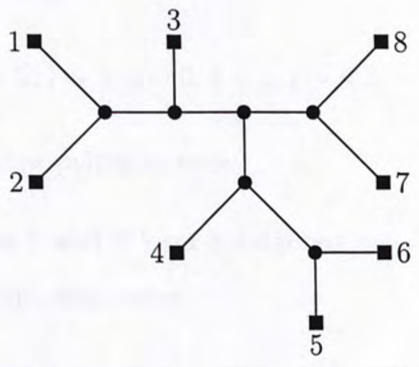

Figure 2.2: A binary tree with 13 edges (line) and 14 vertices: 8 end-vertices (square nodes) and 6 internal vertices (circle nodes).

Definition 2.13 (topology of binary tree). *Given a binary tree T, we can compute a topology of T, denoted by $\mathcal{T}$, by joining neighbor pairs step-by-step. Precisely, in each step, if we have $a \sim b$, then a, b are removed from T. Now the vertex adjacent to both a and b becomes a new end-vertex and is named $a \sim b$ or simply a.*

For example, given the binary tree in Figure 2.2, we can find a topology of the tree by joining neighbor pairs step-by-step:

1. $1 \sim 2$, and renamed as 1, see Figure 2.3,
2. $(1 \sim 2) \sim 3$ or simply $1 \sim 3$, and then renamed as 1,
3. $5 \sim 6$, and renamed as 5,
4. $4 \sim (5 \sim 6)$ or simply $4 \sim 5$, and then renamed as 4
5. $((1 \sim 2) \sim 3) \sim (4 \sim (5 \sim 6))$ or simply $1 \sim 4$, and then renamed as 1,
6. $((1 \sim 2) \sim 3) \sim (4 \sim (5 \sim 6)) \sim 7$ or simply $1 \sim 7$, and then renamed as 1, see Figure 2.5,
7. $(((1 \sim 2) \sim 3) \sim (4 \sim (5 \sim 6)) \sim 7) \sim 8$ or simply $1 \sim 8$.

Therefore a topology of T is:

$$\mathcal{T}_1 = [1 \sim 2, 1 \sim 3, 5 \sim 6, 4 \sim 5, 1 \sim 4, 1 \sim 7, 1 \sim 8]\,. \tag{2.2}$$

We have several important points to note:

1. If the end-vertices 1 and 2 form a neighbor pair, we always write $1 \sim 2$ instead of $2 \sim 1$ for consistency.

2. The bracket (.) is necessary, e.g. $(((1 \sim 2) \sim 3) \sim 4) \sim 5 \neq (1 \sim 2) \sim ((3 \sim 4) \sim 5)$.

3. Although 7 and 8 form a neighbor pair in T, $7 \sim 8$ never appeared in $\mathcal{T}_1$. Actually it is hidden among $1 \sim 7$, $1 \sim 8$.

4. As we can have more than one neighbor pairs to choose in some steps, we can construct another topology of T, e.g.

$$\mathcal{T}_2 = [5 \sim 6, 1 \sim 2, 7 \sim 8, 1 \sim 3, 1 \sim 7, 1 \sim 4, 1 \sim 5]\,. \tag{2.3}$$

5. In the second last step (Figure 2.5), we are left with three end-vertices and any two of them can form a neighbor pair.

Given a topology of a binary tree, we can reconstruct the shape of the binary tree (i.e. the binary tree with branch lengths missing). For example, given the topology $\mathcal{T}_1$ in (2.2) of a binary T, we can construct the shape of T by joining the end-vertices according to the order of the neighbor-pairs on $\mathcal{T}$. First, we join 2 to 1 in Figure 2.7 to form Figure 2.8. Second, we join 3 to $1(= 1 \sim 2)$ forming Figure 2.9. Third, we join 6 to 5 forming Figure 2.10, and etc. At last, we join 8 to 14 in Figure 2.11 to recover the shape of the binary tree T, which has already been shown in Figure 2.2.

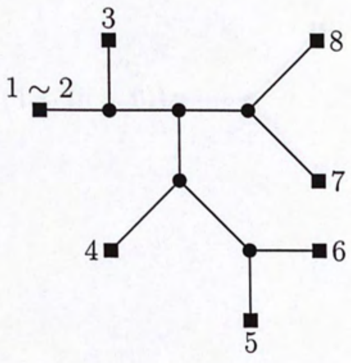

Figure 2.3: The end-vertices $1, 2$ are joined together to form $1 \sim 2$.

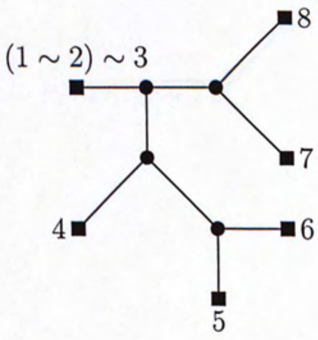

Figure 2.4: After joining $1, 2$ to form $1 \sim 2$, it continues to form pair with 3.

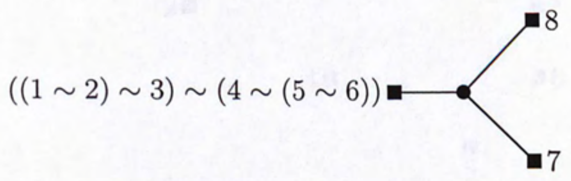

Figure 2.5: Only three end-vertices are left, any two end-vertices form a neighbor pair.

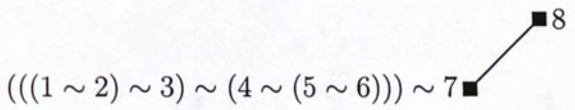

Figure 2.6: Finally, we join the last pair.

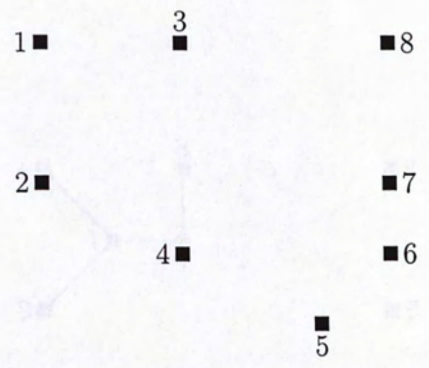

Figure 2.7: We begin with a graph with no edges.

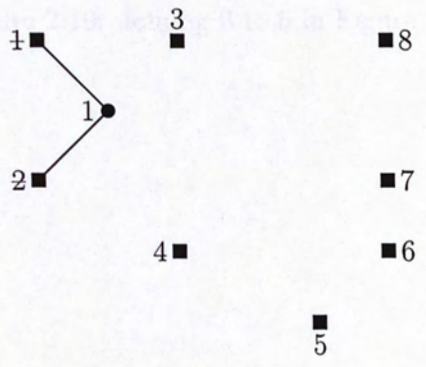

Figure 2.8: Joining 2 to 1 in Figure 2.7.

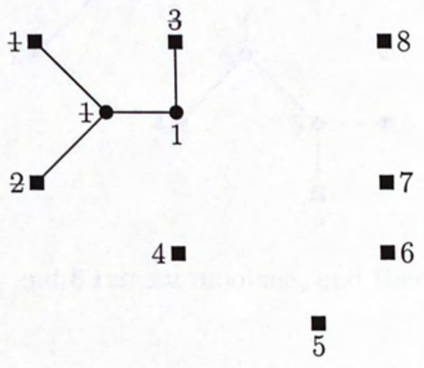

Figure 2.9: Joining 3 to 1 in Figure 2.8.

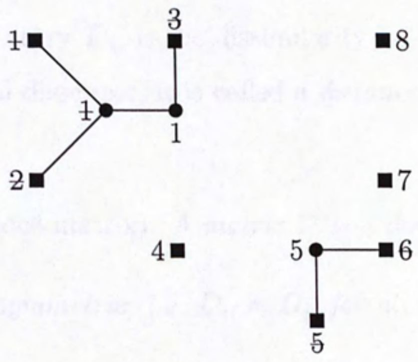

Figure 2.10: Joining 6 to 5 in Figure 2.9.

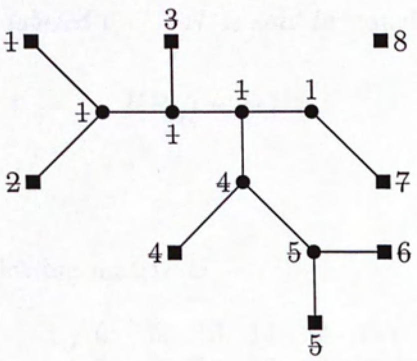

Figure 2.11: Only 1 and 8 remain unjoined, and therefore we join 8 to 1.

2.2 General discussion on visualizing distance matrices by binary trees

Pairwise biological distance data are given in the form of a matrix, usually abbreviated by D, where the entry D_{ij} is the dissimilarity between taxa i, j. Since D is representing biological distances, it is called a *distance matrix* and is assumed certain properties:

Definition 2.14 (distance matrix). *A matrix D is a distance matrix if:*

1. *The matrix D is symmetric, i.e. $D_{ij} = D_{ji}$ for all i, j.*
2. *The diagonal entries of D are zero, i.e. $D_{ii} = 0$ for all i.*
3. *The non-diagonal entries of D are positive, i.e. $D_{ij} > 0$ for all $i \neq j$.*

The neighbor-joining algorithms or other representations are trying to 'visualize' the distance matrix, for example using a binary tree. Hence, scientists can make judgements or get insights by viewing a graph instead of a matrix.

Definition 2.15 (visualize). *Let D be a distance matrix of size $N \times N$. A binary tree T with end-vertices labeled $1, \cdots, N$ is said to visualize D if*

$$l(P_{i,j}) = D_{ij},$$

for all $i, j = 1, \cdots N$.

For example, the following matrix D,

$$D = \begin{pmatrix} 0 & 5 & 13 & 14 & 24 & 25 \\ 5 & 0 & 12 & 13 & 23 & 24 \\ 13 & 12 & 0 & 3 & 13 & 14 \\ 14 & 13 & 3 & 0 & 12 & 13 \\ 24 & 23 & 13 & 12 & 0 & 5 \\ 25 & 24 & 14 & 13 & 5 & 0 \end{pmatrix},$$

is visualized by the binary tree in Figure 2.1. We can easily check that D is a distance matrix.

Given a tree we can always construct a distance matrix D corresponding to T, i.e. for all $i, j = 1, \cdots, N$,

$$D_{ij} := l(P_{i,j}). \tag{2.4}$$

The fundamental questions about visualizing distance matrices will be — correctness and feasibility (i.e. do-able):

1. Can we know if a binary tree T is correctly visualizing a distance matrix?
2. Can all the distance matrices be visualized correctly using binary trees?

To answer the first question, we compute the distance matrix $\tilde{D}$ from the binary tree T using (2.4), and then compare $\tilde{D}$ with the given distance matrix D.

The answer to the second question is negative, as we can show that the following distance matrix cannot be visualized by any binary trees:

$$D = \begin{pmatrix} 0 & 7 & 12 & 9 \\ 7 & 0 & 9 & 14 \\ 12 & 9 & 0 & 7 \\ 9 & 14 & 7 & 0 \end{pmatrix}.$$

To see that, we note that since there are only four end-vertices, the binary tree has a unique topology (Figure 2.12). For instance if we set $[i, j, k, l] = [1, 2, 3, 4]$, then we have to solve the following for a, b, c, d, e:

$$\begin{cases} a+b &= D_{12}, \\ a+e+c &= D_{13}, \\ a+e+d &= D_{14}, \\ b+e+c &= D_{23}, \\ b+e+d &= D_{24}, \\ c+d &= D_{34}. \end{cases}$$

Here we have $(a+e+c)+(b+e+d) = D_{13} + D_{24} = 26 \neq 18 = D_{14} + D_{23} = (a+e+d)+(b+e+c)$, but the leftmost term equals the rightmost term! Therefore the system of equations has no solution. That means no binary tree with these end-vertex labels can visualize D. Similarly we find that the remaining cases $[1,3,2,4]$ and $[1,4,2,3]$ have no solution.

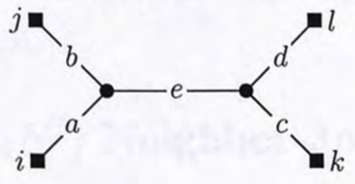

Figure 2.12: The only topology for binary trees having four end-vertices.

Though the answer to the second question is negative, we have a condition to check if a matrix is visualizable — 'Additive matrix'.

Definition 2.16 (additive matrix). *A distance matrix D is additive or is called an additive matrix, if for all distinct i, j, k, l, we have*

$$D_{ij} + D_{kl} < \max\{D_{il} + D_{jk}, D_{ik} + D_{jl}\}. \tag{2.5}$$

The following theorem relates additive matrices and visualization.

Theorem 2.3 (Visualization of additive matrices, [17] Theorem 1). *Given an additive matrix D, there exists a unique binary tree T that visualizes D.*

Proposition 2.4. *If T is a binary tree, and D is the distance matrix computed from T, then D is additive.*

Proof. Since every four points of T must have the topology like Figure 2.12, the additive condition must be satisfied for any 4-tuples from D. □

Proposition 2.5. *A square submatrix of an additive matrix D is additive.*

Proof. It is automatic by definition, as the indices in the submatrix is a subset of D. □

As now we know only some distance matrices D can be visualized, how can we represent the non-additive ones visually? Naturally, there are two ways. The first is to search for a binary tree T, such that the corresponding distance matrix is 'near' to D. Another is to use structures other than binary trees to visualize D, for example, networks [8], [9].

2.3 Original $\mathcal{O}(N^5)$ Neighbor-Joining algorithm

Neighbor joining (NJ) [11] is an algorithm that computes the topology and branch lengths of a binary tree T after given a distance matrix D. In NJ, the topology of the tree T, denoted by $\mathcal{T}$, is found by:

1. Given a distance matrix D, define $\mathcal{T} = \emptyset$, N :=size of D, **indx**$=[1, 2, \cdots, N]$.

2. For each pair of taxa i, j, compute the (i, j)-th entry of the matrix W_{NJ} from D:

$$W_{NJ}(i,j) := \frac{1}{2(N-2)}\sum_{k\neq i,j}(D_{ik}+D_{jk})+\frac{1}{2}D_{ij}+\frac{1}{(N-2)}\sum_{k<l,k,l\neq i,j}D_{kl}. \quad (2.6)$$

3. For all i, $W_{NJ}(i,i) := \infty$.

4. Find α, β such that $W_{NJ}(\alpha, \beta)$ is a minimum entry of W_{NJ}.

5. For all k, set $D_{\alpha k} := (D_{\alpha k} + D_{\beta k})/2$ and $D_{k\alpha} := D_{\alpha k}$.

6. Remove the β-th row and column from D, remove the β-th entry from **indx**.

7. $N := N - 1$.

8. $\mathcal{T} := [\mathcal{T}, indx(\alpha) \sim indx(\beta)]$.

9. If $N > 3$, repeat from Step 2.

10. Else $\mathcal{T} := [\mathcal{T}, indx(1) \sim indx(2), indx(1) \sim indx(3)]$.

After finding the topology of T, the branch lengths of T can be estimated by the formula mentioned in [7]. If the end-vertices α, β are joined in the topology $\mathcal{T}$, then $l(e_\alpha), l(e_\beta)$ can be estimated by:

$$\begin{aligned} l(e_\alpha) &= \frac{1}{2}(D_{\alpha\beta} + D_{\alpha Z} - D_{\beta Z}), && (2.7) \\ l(e_\beta) &= \frac{1}{2}(D_{\alpha\beta} + D_{\beta Z} - D_{\alpha Z}), && (2.8) \end{aligned}$$

where $D_{\alpha Z} = \sum_{k \neq \alpha,\beta} D_{\alpha k}/(N-2)$.

Theorem 2.6 (Correctness of NJ on additive matrices [11] P.411-412). *Let T be a binary tree with N end-vertices, and D be the additive matrix corresponding to T. Then the binary tree computed from D by NJ equals T.*

We now give the complexity of NJ. The computation of each $W_{NJ}(i,j)$ in (2.6) needs $\mathcal{O}(N^2)$ operations. Since there are $\mathcal{O}(N^2)$ entries in W_{NJ}, we need $\mathcal{O}(N^4)$ operations to compute W_{NJ}. After that $\mathcal{O}(N^2)$ is needed to find the argmin of W_{NJ}. Finally we have to repeat the above steps $\mathcal{O}(N)$ times to join the N end-vertices. Thus the NJ algorithm has an overall complexity of $\mathcal{O}(N^5)$. Hence, it is quite slow.

2.4 Speedup of NJ

There are algorithms to implement NJ faster: $\mathcal{O}(N^3)$ for arbitrary distance matrices and $\mathcal{O}(N^2)$ for additive matrices.

2.4.1 $\mathcal{O}(N^3)$ NJ for arbitrary distance matrices

Studier and Keppler [14] spotted that the complexity of NJ can be significantly reduced to $\mathcal{O}(N^3)$ from $\mathcal{O}(N^5)$. To see that, they rewrite (2.6):

$$\begin{aligned} W_{NJ}(i,j) &= \frac{1}{2(N-2)}\sum_{k\neq i,j}(D_{ik}+D_{jk}) + \frac{1}{2}D_{ij} + \frac{1}{(N-2)}\sum_{k<l,k,l\neq i,j} D_{kl} \\ &= \frac{1}{2(N-2)}(S_i - D_{ij} + S_j - D_{ij}) + \frac{1}{2}D_{ij} \\ &\quad + \frac{1}{N-2}(S - (S_i + S_j - D_{ij})) \\ &= \frac{2S - S_i - S_j}{2(N-2)} + \frac{1}{2}D_{ij}, \end{aligned} \tag{2.9}$$

where $S_i := \sum_{k=1}^{N} D_{ik}$ and $S := \sum_{1\leq i<j\leq N} D_{ij}$. As only the argmin of W_{NJ} is concerned in Step 4 of the NJ algorithm, the computation of S in (2.9) becomes unnecessary. Therefore the formula of W_{NJ} in (2.6) can be replaced by W_{SK} in [14]:

$$W_{SK}(i,j) := (N-2)D_{ij} - S_i - S_j. \tag{2.10}$$

As $W_{SK} = 2(N-2)W_{NJ} - 2S$, we can see W_{NJ} and W_{SK} have identical argmin. Thus the replacement of W_{NJ} by W_{SK} will not affect the outcome of NJ.

We now analyze the complexity of this approach. The computation of all S_i, where $i = 1, \cdots, N$ needs $\mathcal{O}(N^2)$. Therefore the computation of W_{SK} takes $\mathcal{O}(N^2)$, and finding the argmin needs $\mathcal{O}(N^2)$. Since there are $\mathcal{O}(N)$ steps to do, the overall complexity becomes $\mathcal{O}(N^3)$, which is a huge speedup.

2.4.2 $\mathcal{O}(N^2)$ NJ on additive matrices

On additive matrices, the complexity of NJ is reduced to $\mathcal{O}(N^2)$ by the algorithm of Waterman et al. [17]:

1. Pick any two end-vertices and construct the unique binary tree T (actually a line).

2. While there are end-vertices not added to T:
    (a) Pick a remaining end-vertex $k \notin T$ and two end-vertices $i, j \in T$.
    (b) Form the unique binary tree T' visualizing the distances among i, j, k.
    (c) If the internal vertex in T' does not coincide with any internal vertices lying in the path $P_{i,j}$ in T, k has been properly added to T.
    (d) Else select another end-vertex in T not tested before to replace j and repeat from Step 2b.

Theorem 2.7 (Correctness of Waterman's algorithm on additive matrices [17] Theorem 2). *Let T be a binary tree with N end-vertices, and D be the additive matrix corresponding to T. Then the binary tree computed from D by Waterman's algorithm equals T.*

Since at each step, we have at most N end-vertices to test, the complexity of the algorithm is in total $\mathcal{O}(N^2)$.

Although this algorithm is an $\mathcal{O}(N^2)$ algorithm on implementing NJ on additive matrices, it cannot be applied to non-additive matrices. It is mentioned in Atteson [1] that the number of additive matrices is too small compared to the non-additive ones. Therefore the standard complexity in NJ now is still $\mathcal{O}(N^3)$. Although our algorithm, the DCNJ algorithm, has the complexity of $\mathcal{O}(N^3)$ and only applicable to additive matrices now, it can be further developed to apply to non-additive matrices. Moreover, this algorithm is based on a new philosophy on implementing NJ — decoupling the computation of W_{SK} (2.10). The further application of the philosophy may speed up NJ to $\mathcal{O}(N^2 \log N)$.

Chapter 3

Finding neighbor pairs

In this chapter, we investigate the properties of binary trees. Moreover, we propose and prove the equivalent condition on the branch length matrix W_{SK} for finding neighbor pairs, where W_{SK} is defined by (2.10). For simplicity, we use W to denote W_{SK} onwards.

3.1 Properties of Binary trees

In Neighbor-Joining (NJ) or other neighbor-joining methods, phylogenetic information in a distance matrix is represented by a binary tree. Therefore, we first understand the properties of binary trees, especially their internal vertices. Recall that all internal vertices in a binary tree have exactly degree three by definition.

Proposition 3.1. *Given a binary tree T with $N \geq 4$ end-vertices, we have*

1. *T has $N-2$ internal vertices;*

2. *T has $2N-2$ vertices;*

3. *T has $2N-3$ edges.*

Proof. Suppose there are I internal vertices in T. By Proposition 2.2, there are

$N + I - 1$ edges. Using the handshaking lemma, the sum of vertex degrees is:

$$\sum_{v \in T} \deg(v) = 2(N + I - 1). \tag{3.1}$$

On the other hand, we have N vertices of degree 1 and I vertices of degree 3. Therefore the sum of vertex degrees is

$$\sum_{v \in T} \deg(v) = N + 3I. \tag{3.2}$$

Since (3.1) equals (3.2), we have $I = N - 2$. Thus there are $2N - 2$ $(= N + N - 2)$ vertices. Hence by Proposition 2.2, we have $2N - 3$ $(= N + I - 1)$ edges in T. □

Definition 3.1 (parent/link/bridge vertex). *In a binary tree T, an internal vertex is a parent/link/bridge vertex, if it is adjacent to 2/1/0 end-vertices (i.e. 1/2/3 internal vertices) respectively. The number of respective vertices are denoted by $P/L/B$.*

Note that $P + L + B$ is the number of internal vertices in T if T has at least four end-vertices, and also two end-vertices form a neighbor pair if and only if they are adjacent to the same parent vertex. Figure 3.1 gives an example of parent, link and bridge vertices, where $P = 3, L = 2, B = 1$.

Proposition 3.2. *In a binary tree T with $N \geq 4$ end-vertices, the number of parent vertices is equal to the number of bridge vertices plus two, i.e.*

$$P = B + 2. \tag{3.3}$$

Proof. Let T be a binary tree with $N \geq 4$ end-vertices. By Proposition 3.1, we have

$$P + L + B = N - 2. \tag{3.4}$$

Let T_I be the connected subtree of T containing only the internal vertices and the edges connecting them. Therefore, T_I has $N - 2$ vertices. By Proposition 2.2,

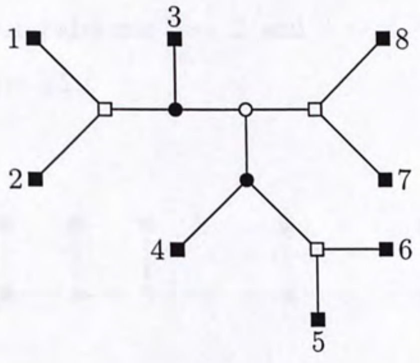

Figure 3.1: The vertices in a binary tree are categorized into 4 types: 1) black square — end-vertex, with degree one; 2) white square — parent vertex, adjacent to one internal vertex and two taxa; 3) black circle — link vertex, adjacent to two internal vertices and one taxon; 4) white circle — bridge vertex, adjacent to internal vertices only.

T_I has $N-3$ edges. By the handshaking lemma, sum of vertex degrees in T_I is $2N-6$. On the other hand, as T_I contains internal vertices only, the sum of vertex degrees in T_I is $P+2L+3B$. Hence

$$P+2L+3B = 2N-6. \tag{3.5}$$

Considering $2\times(3.4)-(3.5)$, we have $P-B=2$, i.e. $P=B+2$. □

Proposition 3.3. *In a binary tree T with $N \geq 4$ end-vertices, the number of parent vertices is greater than or equal to two, i.e.*

$$P \geq 2. \tag{3.6}$$

Proof. Since the number of bridge vertices in a binary tree is always non-negative, by Proposition 3.2, the number of parent vertex is greater than or equal to two. □

Proposition 3.4. *If a binary tree T has only two neighbor pairs say $1\sim 2$ and $3\sim 4$, then its topology is exactly as shown in Figure 3.2.*

Proof. We know T has only two parent vertices, hence by Proposition 3.2, we have no bridge vertices. Therefore the remaining internal vertices must be link vertices. If the neighbor pairs are $1 \sim 2$ and $3 \sim 4$, the tree must have the topology shown in Figure 3.2. □

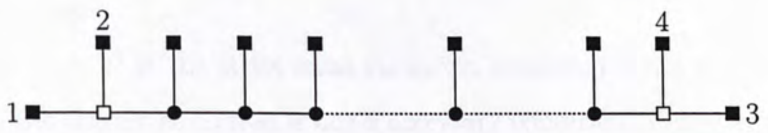

Figure 3.2: A binary tree with two parent vertices must have no bridge vertices.

3.2 Similar rows: finding all neighbor pairs in additive matrices

In each step of NJ, the matrix W is computed from D using (2.10). Then the global minimum on W is found and a neighbor pair is joined. By Proposition 3.3, every binary tree has at least two neighbor pairs. Therefore the number of steps in NJ can at least be halved by joining two neighbor pairs in every step. In this section, we describe an equivalent condition, called similar rows, for finding neighbor pairs using W. In this section, we assume all W are computed from a given additive matrix D using (2.10), where D corresponds to a binary tree T.

We first give a definition which will be useful later.

Definition 3.2 (strict local minimum). *An entry of a matrix M, denoted by M_{ij}, is said to be a strict local minimum of M if M_{ij} is simultaneously the row and the column minimum of M, i.e.*

$$M_{ij} < M_{ik} \text{ and } M_{ij} < M_{kj}, \qquad \forall k \neq i, j. \tag{3.7}$$

It was shown in Saitou and Nei [11] that if i, j form a neighbor pair in T, then W_{ij} is a strict local minimum of the matrix W. Therefore, the strict local minimum condition is a necessary condition for neighbor pairs. On the other hand, Studier and Keppler [14] proved that if (i, j) is an argmin of W, then i and j must be a neighbor pair in T. Therefore, the argmin is a sufficient condition for neighbor pairs.

We now show that the strict local minimum condition is not a sufficient condition and the argmin condition is not a necessary condition. Consider the binary tree T_1 shown in Figure 3.3. We can compute D_1 corresponding to T_1:

$$D_1 = \begin{pmatrix} 0 & 2 & 22 & 22 & 11 & 12 \\ 2 & 0 & 22 & 22 & 11 & 12 \\ 22 & 22 & 0 & 2 & 13 & 12 \\ 22 & 22 & 2 & 0 & 13 & 12 \\ 11 & 11 & 13 & 13 & 0 & 3 \\ 12 & 12 & 12 & 12 & 3 & 0 \end{pmatrix}.$$

Hence W_1 computed from D_1 using (2.10) is:

$$W_1 = \begin{pmatrix} 0 & -130 & -52 & -52 & -76 & -72 \\ -130 & 0 & -52 & -52 & -76 & -72 \\ -52 & -52 & 0 & -134 & -70 & -74 \\ -52 & -52 & -134 & 0 & -70 & -74 \\ -76 & -76 & -70 & -70 & 0 & -90 \\ -72 & -72 & -74 & -74 & -90 & 0 \end{pmatrix}. \tag{3.8}$$

Since $W_1(5, 6)$ is a strict local minimum on W_1 and $5, 6$ do not form a neighbor pair in T_1, a strict local minimum is not a sufficient condition for neighbor pairs. Since $W_1(1, 2)$ is not an minimum entry in W and $1 \sim 2$ in T_1, being an argmin is not a necessary condition for neighbor pairs.

We now propose the necessary and sufficient condition for two end-vertices to form a neighbor pair in T.

Definition 3.3 (similar rows). *Suppose $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ are two distinct rows of a matrix M. They are called similar rows in M, denoted by $\boldsymbol{m}_i \sim \boldsymbol{m}_j$, if*

$$M_{ik} = M_{jk}, \qquad \forall k \neq i, j. \tag{3.9}$$

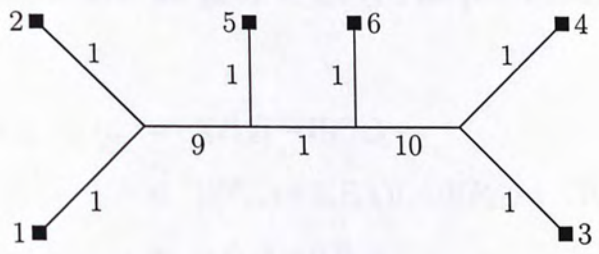

Figure 3.3: The binary tree T_1 with $1 \sim 2$ and $3 \sim 4$ but $W_{34} < W_{12}$.

For example of the matrix W_1 in (3.8), we have $\mathbf{w}_1 \sim \mathbf{w}_2$ and $\mathbf{w}_3 \sim \mathbf{w}_4$.

Theorem 3.5 (Necessary and sufficient condition for neighbor pairs). *Let T be a weighted binary tree and D be the additive matrix corresponding to T. Let W be computed from D using (2.10) and $\boldsymbol{w}_i$, $\boldsymbol{w}_j$ be two rows of W. We have*

$$i \sim j \text{ in } T \iff \boldsymbol{w}_i \sim \boldsymbol{w}_j \text{ in } W. \tag{3.10}$$

Proof. WLOG, assume T has N end-vertices. First we have for all $k \neq i, j$,

$$\begin{aligned} W_{ik} = W_{jk} &\iff (N-2)D_{ik} - S_i - S_k = (N-2)D_{jk} - S_j - S_k \\ &\iff (N-2)(D_{ik} - D_{jk}) = S_i - S_j \\ &\iff D_{ik} - D_{jk} = \frac{S_i - S_j}{N-2}. \end{aligned} \tag{3.11}$$

($\Longleftarrow$) If $\mathbf{w}_i \sim \mathbf{w}_j$, then we have $D_{ik} - D_{jk} = \dfrac{S_i - S_j}{N-2}$ for all $k \neq i, j$ from (3.11). Therefore

$$(D_{im} - D_{jm}) - (D_{in} - D_{jn}) = 0, \qquad \forall m, n \neq i, j. \tag{3.12}$$

If i, j are not neighbor pairs, then path $P_{i,j}$ contains at least three edges (Figure 3.4). Therefore the path $\tilde{P} := (P_{i,j} \setminus \{e_i, e_j\}) \neq \emptyset$ and there must be two other taxa a and b such that $\tilde{P} \subset P_{a,b}$. Hence, $(D_{ia} - D_{ja}) - (D_{ib} - D_{jb}) = 2 \times l(\tilde{P}) \neq 0$, a contradiction to (3.12).

($\Longrightarrow$) If $i \sim j$, then they have a common parent vertex p (Figure 3.5). Since paths in a tree are unique, the paths $P_{i,k}$, $P_{j,k}$ must pass through p. Hence for all $k \neq i, j$,

$$\begin{aligned} D_{ik} - D_{jk} &= l(P_{i,k}) - l(P_{j,k}) \\ &= [l(P_{i,p}) + l(P_{p,k})] - [l(P_{j,p}) + l(P_{p,k})] \\ &= l(P_{i,p}) - l(P_{j,p}). \end{aligned}$$

Therefore

$$\begin{aligned} \frac{S_i - S_j}{N-2} &= [\sum_{k=1}^{N}(D_{ik} - D_{jk})]/(N-2) \\ &= [\sum_{k \neq i,j}(D_{ik} - D_{jk})]/(N-2) \\ &= (N-2)[l(P_{i,p}) - l(P_{j,p})]/(N-2) \\ &= l(P_{i,p}) - l(P_{j,p}) \\ &= D_{ik} - D_{jk}, \end{aligned}$$

for all $k \neq i, j$. From (3.11), we have $\mathbf{w}_i \sim \mathbf{w}_j$. □

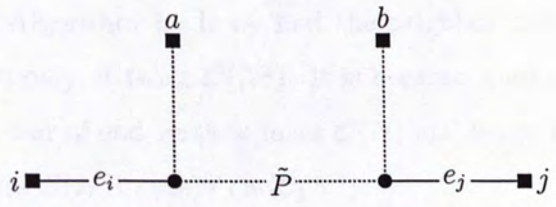

Figure 3.4: i, j do not form neighbor pairs.

We know that from Theorem 2.3 that a binary tree is exactly corresponding to one and only one additive matrix. Also, by Theorem 3.5 we can find the neighbor pairs in the binary tree T after given the additive matrix D corresponding to T only. Therefore we can say 'the neighbor pair $i \sim j$ in D' to mean '$i \sim j$ is a neighbor pair in the binary tree corresponding to D'. We can also say 'the

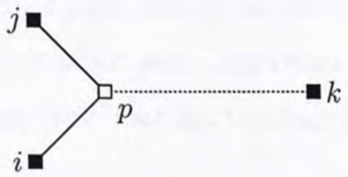

Figure 3.5: i, j form neighbor pairs.

number of neighbor pairs in an additive matrix D is Q' to mean 'the number of neighbor pairs in the binary tree corresponding to D is Q'.

We use Theorem 3.5 in Algorithm 1 to find all neighbor pairs in an additive matrix D.

```
Algorithm 1: Finding Neighbor Pairs.
  Input: Additive distance matrix $D$
  Output: Neighbor pairs: $i_1 \sim j_1, i_2 \sim j_2, \cdots$, where $i_1 < j_1, i_2 < j_2, \cdots$
1 Compute $W$ from $D$ using (2.10)
2 Find all strict local minima (3.7) on the upper triangular part of $W$.
3 For each strict local minima, if similar row condition (3.9) is satisfied, store
  $i \sim j$ as a neighbor pair
```

Note that we make use of the strict local minimum condition (3.7) to reduce the complexity of Algorithm 1. If we find the neighbor pairs in D by similar row condition (3.9) only, it takes $\mathcal{O}(N^3)$. It is because checking the similar row condition for each pair of end-vertices takes $\mathcal{O}(N)$ and hence to check every pair of end-vertices takes $\mathcal{O}(N^2) \times \mathcal{O}(N) = \mathcal{O}(N^3)$.

We now give the complexity of Algorithm 1. First, we compute $W(:= W_{SK})$ (2.10) in $\mathcal{O}(N^2)$. Then, we find all the strict local minima in the upper triangular part of W in $\mathcal{O}(N^2)$. Note that there are at most N strict local minima in the upper triangular part of W. After that, for each strict local minimum, we test the similar row condition on it, which takes $\mathcal{O}(N)$. Therefore, testing the similar row condition for all strict local minima takes at most $\mathcal{O}(N^2)$. Hence, the complexity of Algorithm 1 is $\mathcal{O}(N^2)$.

After knowing the neighbor pairs, how do we make use of it? For example if a binary tree T has only two neighbor pairs, Algorithm 2 can visualize this special T. It is possible because we know that the topology is unique from Proposition 3.3.

```
Algorithm 2: Visualize $D$ having only 2 neighbor pairs.
  Input: Additive distance matrix $D$ with 2 neighbor pairs
  Output: Topology $\mathcal{T}$ of the binary tree visualizing $D$
1 Find neighbor pairs using Algorithm 1, and assume they are
  $1 \sim 2, (N-1) \sim N$
2 for $k = 3, \cdots, N-2$ do
3     Compute Skel($k$) $= (D_{1k} + D_{1N} - D_{Nk})/2$ (see Figure 3.6)
4 end
5 Define Skel(1)=Skel(2)=Skel($N-1$)=Skel($N$)=$\infty$
6 SortSkel:=Sort Skel in ascending order, and assume
  Skel($u_1$) $\leq$Skel($u_2$) $\leq \cdots \leq$Skel($u_{N-4}$)
7 return $\mathcal{T} := [1 \sim 2, 1 \sim u_1, \cdots, 1 \sim u_{N-4}, 1 \sim (N-1), 1 \sim N]$
```

The graphical meaning of Skel(k) is shown in Figure 3.6.

Proposition 3.6. *Algorithm 2 is correct for any additive matrices with only two neighbor pairs.*

Proof. The neighbor pairs found by Algorithm 1 must be correct, as guaranteed by Theorem 3.5. WLOG, assume the neighbor pairs found are $1 \sim 2$ and $(N-1) \sim N$. Therefore the true binary tree must have the topology shown in Figure 3.6. Hence the order of the numbers Skel(k) $:= (D_{1k} + D_{1N} - D_{Nk})/2$ must represent the correct order of the end-vertices k in T, where $k = 3, \cdots, N-2$. Therefore Algorithm 2 is correct. □

We give the complexity of Algorithm 2. First, we use Algorithm 1 to find the neighbor pairs in $\mathcal{O}(N^2)$. Then, we sort N numbers and find their respective indices using $\mathcal{O}(N \log N)$. Therefore the overall complexity of Algorithm 2 is $\mathcal{O}(N^2)$.

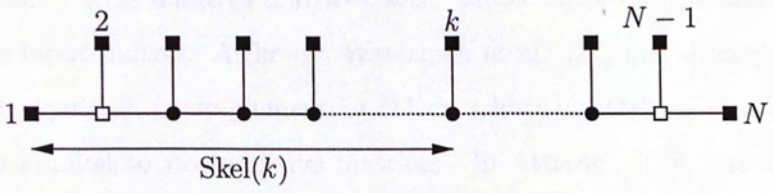

Figure 3.6: Graphical meaning of Skel(k).

Chapter 4

Divide-and-Conquer Neighbor-Joining

We propose a new algorithm — Divide-and-Conquer Neighbor-Joining (DCNJ) to implement NJ on additive matrices with complexity $\mathcal{O}(N^3)$, where N is the size of the input matrix. Although Waterman et al. [17] has already proposed an $\mathcal{O}(N^2)$ algorithm on implementing NJ on additive matrices, that algorithm cannot be applied to non-additive matrices. In Atteson [1], it was mentioned that the number of additive matrices is too small compared to the non-additive ones. Therefore the standard complexity in NJ now is still $\mathcal{O}(N^3)$. Although our algorithm, the DCNJ algorithm, has the complexity of $\mathcal{O}(N^3)$ and is only applicable to additive matrices now, it can be further developed to apply to non-additive matrices. Moreover, this algorithm is based on a new philosophy on implementing NJ — decoupling the computation of W. The further application of the philosophy may speed up NJ to $\mathcal{O}(N^2 \log N)$. Therefore, this algorithm serves as a stepping stone to a more general algorithm.

In Section 4.1, we give an example to illustrate the idea of DCNJ and the algorithm. In Section 4.2, we explain the theories of DCNJ: its correctness and complexity.

4.1 DCNJ Algorithm

Given the binary tree T shown in Figure 4.1, let D be the additive matrix corresponding to it. We give an example on applying DCNJ to find the topology of T given D. In the DCNJ algorithm, there are two phases. In phase one, we first find the number of pairs in D by applying Algorithm 1 to D. If the number of neighbor pairs found $:= Q \geq 4$, then we add all the neighbor pairs found to the topology $\mathcal{T}_P$ and update D. For example, if $i \sim j$ is one of the neighbor pairs found by Algorithm 1, we will add $i \sim j$ to $\mathcal{T}_P$ and then remove the j-th row and column from D. In fact, this step is exactly doing the same as NJ, but we just join more than one neighbor pairs. After that, we repeat this phase with the updated D. We will end this phase and go to phase two when we find that the number of pairs Q remaining in D is less than four.

In phase two, we apply different methods to D according to the number of pairs Q remaining in D. If $Q = 3$, we start our divide-and-conquer procedure to find the topology of the remaining end-vertices in D. Or if $Q = 2$, we can just apply Algorithm 2 to D to find the topology of the remaining end-vertices in D. In either case, the topology found will be added to the topology $\mathcal{T}_P$ to form the final topology $\mathcal{T}$ for output.

To begin with the example, we start with the empty topology $\mathcal{T}_P := \emptyset$ and we apply Algorithm 1 to D and find four neighbor pairs: $1 \sim 2, 6 \sim 7, 8 \sim 9$ and $12 \sim 13$. As now the number of pairs $Q = 4$, we add the neighbor pairs to $\mathcal{T}_P$ consecutively to form

$$\mathcal{T}_P = [1 \sim 2, 6 \sim 7, 8 \sim 9, 12 \sim 13]. \tag{4.1}$$

After that, we remove the rows and columns corresponding to the end-vertices $2, 7, 9$ and 13 from D to form D'. In terms of the trees, we remove the end-vertices $2, 7, 9$ and 13 from T to form T' (Figure 4.2).

We now repeat phase one, which means again we look for neighbor pairs in D'. Therefore we apply Algorithm 1 to D' and find the neighbor pairs $5 \sim 6, 8 \sim 10$ and $12 \sim 14$ in T'. This time the number of pairs $Q = 3$. Therefore we go to phase two and begin the divide-and-conquer procedure.

In this procedure, we first form one cluster for each neighbor pair, and then we put the remaining end-vertices to either one of these clusters. In our case, we form the cluster of $5 \sim 6$, $8 \sim 10$ and $12 \sim 14$. Then we put the remaining end-vertices $1, 3, 4, 11$ and 15 into either one of these clusters. To determine which cluster should the end-vertex 1 be put into, we consider $T^{(1)}$, which is the binary tree containing only the end-vertices $1, 5, 8$ and 12 (Figure 4.3). Visually we see that $1 \sim 12$ in $T^{(1)}$, and therefore we put end-vertex 1 into the cluster of 12. Numerically for end-vertex 1, we compute $D^{(1)}$ which is the submatrix of D' containing the pairwise distances between $1, 5, 8$ and 12. We apply Algorithm 1 to $D^{(1)}$ and find $1 \sim 12$. Therefore we put 1 into the cluster of 12. The case is similar for end-vertices $3, 4$ and 15.

For end-vertex 11, we compute $D^{(11)}$ which is the submatrix of D' containing the pairwise distances between $11, 5, 8$ and 12. This matrix is represented visually by the binary tree $T^{(11)}$ in Figure 4.4. We can find $8 \sim 11$ by applying Algorithm 1 to $D^{(11)}$ and therefore we put 11 in the cluster of 8.

All the remaining end-vertices have been clustered. We see that the cluster of 5, Y_5, does not receive any extra end-vertices. Summing up, we have:

1. The cluster of neighbor pair $5 \sim 6 := Y_5 = [5, 6]$,

2. The cluster of neighbor pair $8 \sim 10 := Y_8 = [8, 10, 11]$,

3. The cluster of neighbor pair $12 \sim 14 := Y_{12} = [12, 14, 1, 3, 4, 15]$.

We now illustrate the graphical meaning of these clusters. If we consider the binary tree T', there is only one bridge vertex B in it. After removing B and its adjacent edges from T', there are three separated trees remaining. Each cluster is exactly containing every end-vertices in one such tree.

Now we find the topology of each cluster. Since Y_5 has only two end-vertices, obviously, the topology of Y_5 is $[5 \sim 6]$. Here the end-vertex 6 will be joined with 5 and therefore the end-vertex 5 is the unjoined vertex in Y_5.

To find the topology of the cluster Y_8, we cannot follow the same approach as Y_5. It is because Y_8 has more than two end-vertices and therefore there are more than one choices on the topology, for instance $[8 \sim 10, 8 \sim 11]$ and $[10 \sim 11, 8 \sim 10]$. We can see from T' that the first one is correct while the second one is wrong. To obtain the correct topology of Y_8, we need to use some tricks. First, we append an end-vertex outside Y_8, say 12, to Y_8 to form $\tilde{Y}_8 = [8, 10, 11, 12]$. Let $\tilde{T}_8$ be the binary tree containing the end-vertices in $\tilde{Y}_8$ (Figure 4.5) and we see the topology of $\tilde{T}_8 = [8 \sim 10, 8 \sim 11, 8 \sim 12]$. As we know the end-vertex 12 is external to $\tilde{Y}_8$, we remove the neighbor pair containing it, i.e. $8 \sim 12$, to obtain the topology of $Y_8 = [8 \sim 10, 8 \sim 11]$.

Numerically, we first let $\tilde{D}_8$ be the submatrix of D' containing the pairwise distances between the end-vertices in $\tilde{Y}_8$. Visually, $\tilde{D}_8$ is corresponding to the binary tree $\tilde{T}_8$. Therefore there are two neighbor pairs in $\tilde{D}_8$ and hence we can apply Algorithm 2 to them. In fact, we will prove in Proposition 4.3 that every such appended cluster contains exactly two neighbor pairs. In our case, we apply Algorithm 2 to $\tilde{D}_8$ and get the topology $[8 \sim 10, 8 \sim 11, 8 \sim 12]$. As we know that the end-vertex 12 is outside Y_8, we remove the last neighbor pair $8 \sim 12$ from this topology to get $[8 \sim 10, 8 \sim 11]$, which is the correct topology of Y_8. Since the end-vertices 10 and 11 will be joined with 8 successively in this topology, the unjoined vertex in Y_8 is the end-vertex 8.

Since the cluster Y_{12} has more than two end-vertices, we use the same approach as Y_8 to find its topology. We first append the end-vertex 5 to Y_{12} to form $\tilde{Y}_{12} = [12, 14, 1, 3, 4, 15, 5]$. Let $\tilde{D}_{12}$ be the submatrix of D' containing the pairwise distances between the end-vertices in $\tilde{Y}_{12}$. Visually, $\tilde{D}_{12}$ is corresponding to the binary tree $\tilde{T}_{12}$ shown in Figure 4.6. Again we see that there are only two neighbor pairs in the appended cluster $\tilde{T}_{12}$, as guaranteed by Proposition 4.3, and therefore it is valid to apply Algorithm 2 to $\tilde{D}_{12}$ to find the topology of $\tilde{Y}_{12}$. The topology found is $[12 \sim 14, 12 \sim 15, 1 \sim 12, 1 \sim 3, 1 \sim 4, 1 \sim 5]$, which is exactly the topology of $\tilde{T}_{12}$. As we know the end-vertex 5 is not in Y_{12}, we remove the last neighbor pair $1 \sim 5$ from this topology to get $[12 \sim 14, 12 \sim 15, 1 \sim 12, 1 \sim 3, 1 \sim 4]$. The unjoined end-vertex in Y_{12} is the end-vertex 1 instead of 12 because we have to keep the index notation consistent, so we write $1 \sim 12$ instead of $12 \sim 1$. Summing up, we have

1. The topology of $Y_5 := \mathcal{T}_5 = [5 \sim 6]$,
2. The topology of $Y_8 := \mathcal{T}_8 = [8 \sim 10, 8 \sim 11]$,
3. The topology of $Y_{12} := \mathcal{T}_{12} = [12 \sim 14, 12 \sim 15, 1 \sim 12, 1 \sim 3, 1 \sim 4]$.

After computing the topologies for each cluster, there are only three unjoined end-vertices 5, 8 and 1 from the clusters Y_5, Y_8 and Y_{12} respectively. As mentioned in the notes after the definition of topology (Definition 2.13), we know that any two end-vertices form a neighbor pair if there are only three end-vertices left. Therefore, we form the topology of the three remaining end-vertices

$$\mathcal{T}_U = [1 \sim 5, 1 \sim 8]. \tag{4.2}$$

At last, we recombine the topologies: $\mathcal{T}_P$ in (4.1), $\mathcal{T}_5, \mathcal{T}_8, \mathcal{T}_{12}$ and $\mathcal{T}_U$ in (4.2) to get the topology of the binary tree that visualizes D:

$$\begin{aligned} \mathcal{T} \;=\; & [1 \sim 2, 6 \sim 7, 8 \sim 9, 12 \sim 13, 5 \sim 6, 8 \sim 10, 8 \sim 11, \\ & \quad 12 \sim 14, 12 \sim 15, 1 \sim 12, 1 \sim 3, 1 \sim 4, 1 \sim 5, 1 \sim 8], \end{aligned}$$

which is exactly the topology of T in Figure 4.1. To see that, we can join neighbor pairs in $\mathcal{T}$ step-by-step, i.e. first join the end-vertices 1 and 2 in T to 1, and then join the end-vertices 6 and 7 in T to 6, and etc. To find the branch lengths of T, we follow the same approach as NJ, which has been described in Section 2.3.

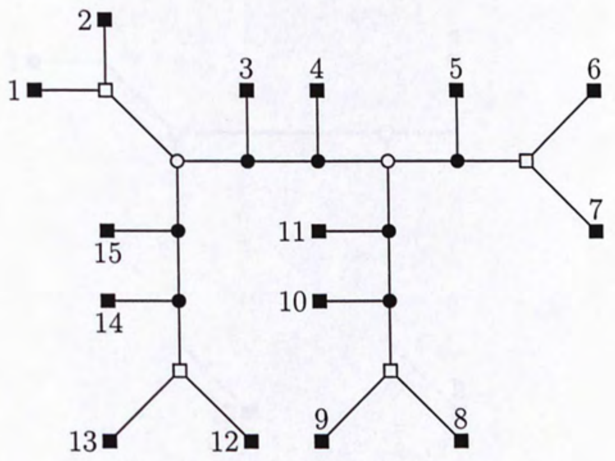

Figure 4.1: A binary tree T with fifteen end-vertices.

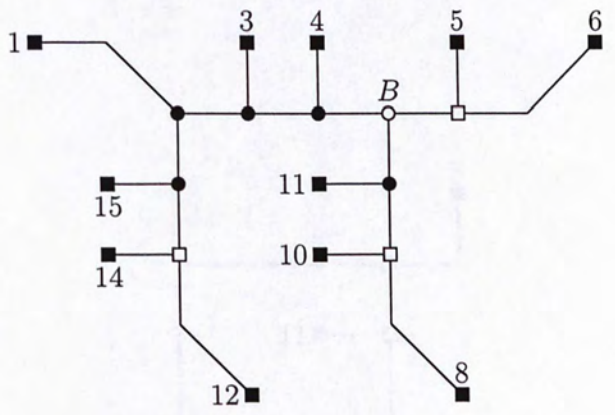

Figure 4.2: A binary tree T' with eleven end-vertices.

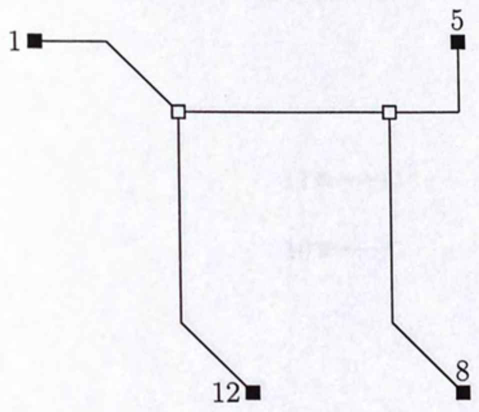

Figure 4.3: The binary tree $T^{(1)}$ with four end-vertices corresponding to $D^{(1)}$. We see that $1 \sim 12$.

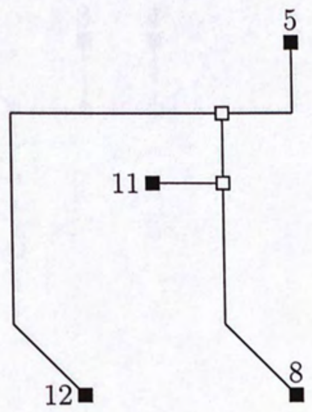

Figure 4.4: The binary tree $T^{(11)}$ with four end-vertices corresponding to $D^{(11)}$. We see that $8 \sim 11$.

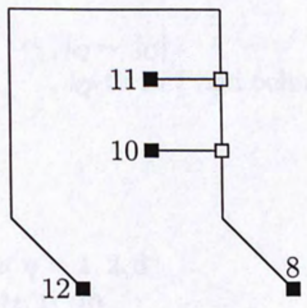

Figure 4.5: The binary tree $T^{(8)}$ containing only the end-vertices of the cluster $\tilde{Y}_8 = [8, 10, 11, 12]$.

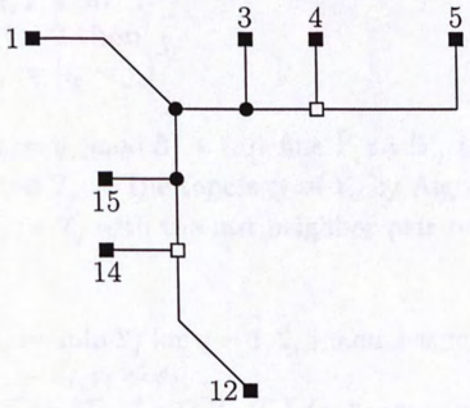

Figure 4.6: The binary tree $T^{(12)}$ containing only the end-vertices of the cluster $\tilde{Y}_{12} = [12, 14, 1, 3, 4, 15, 5]$.

Algorithm 3: Divide-and-Conquer Neighbor-Joining Algorithm (DCNJ)

```
Input: Additive distance matrix $D$
Output: Topology $\mathcal{T}$ of a binary tree visualizing $D$
while size of $D \geq 4$ do
    Find all neighbor pairs in $D$: $i_1 \sim j_1, \cdots, i_Q \sim j_Q$ by Algorithm 1
    if $Q \geq 4$ then
        $\mathcal{T}_P := [\mathcal{T}_P, i_1 \sim j_1, \cdots, i_Q \sim j_Q]$
        Remove the $j_1, j_2, \cdots, j_Q$-th row and column from $D$
    else break
    end
end
if $Q = 3$ then
    Define $Y_q := [i_q, j_q]$ for $q = 1, 2, 3$
    forall $v \neq i_1, j_1, \cdots, i_3, j_3$ do
        Define $\mathbf{x} := [v, i_1, i_2, i_3]$
        for $a, b = 1, \cdots, 4$ do
            $D^{(v)}(a, b) := D(x(a), x(b))$
        end
        Find all the neighbor pairs in $D^{(v)}$ by Algorithm 1
        if $v \sim i_q$ in $D^{(v)}$ then
            $Y_q := [Y_q, v]$
        end
    end
    for $q = 1, 2, 3$ do
        if $|Y_q| = 2$ then
            $\mathcal{T}_q := [i_q \sim j_q]$
        else
            $m := q \pmod 3 + 1$, define $\tilde{Y}_q := [Y_q, i_m]$
            Find $\tilde{\mathcal{T}}_q :=$ the topology of $\tilde{Y}_q$ by Algorithm 2
            $\mathcal{T}_q := \tilde{\mathcal{T}}_q$ with the last neighbor pair removed
        end
    end
    Define $s_q := \min Y_q$ for $q = 1, 2, 3$, and assume $s_1 < s_2 < s_3$
    $\mathcal{T}_U := [s_1 \sim s_2, s_1 \sim s_3]$
    return $\mathcal{T} := [\mathcal{T}_P, \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_U]$ /* Recombine $\mathcal{T}_q$'s */
else
    Apply Algorithm 2 to $D$ to get $\mathcal{T}'$
    return $\mathcal{T} := [\mathcal{T}_P, \mathcal{T}']$
end
```

4.2 Theories of DCNJ on additive matrices: Correctness and Complexity

In this section, we provide theories of DCNJ. We now give several propositions useful in proving the correctness of DCNJ. Let us recall from Proposition 2.5, that any square submatrices of an additive matrix are additive, and therefore it is valid to talk about neighbor pairs in submatrices. Another point to remind is from P. 31 that the sentence '$i \sim j$ in D' is to mean '$i \sim j$ in the unique binary tree that corresponds to D'.

Proposition 4.1. *If $i \sim j$ is a neighbor pair in an additive matrix D of size N, then $i \sim j$ is a neighbor pair in any square submatrices of D containing the end-vertices i, j.*

Proof. WLOG, we assume $i = 1$ and $j = 2$. Let D' be the $(N-1) \times (N-1)$ square submatrix of D, where the N-th row and column of D are removed. By Theorem 3.5, we have for $k = 3, 4, \cdots, N$,

$$W_D(1,k) = W_D(2,k), \tag{4.3}$$

where W_D is computed from D using (2.10).

Let $W_{D'}$ be computed from D' using (2.10). Let $S_D(k) := \sum_{m=1}^{N} D(k,m)$ for $k = 1, \cdots, N$ and $S_{D'}(l) := \sum_{m=1}^{N-1} D'(l,m)$ for $l = 1, \cdots, N-1$. Then for

$n = 3, \cdots, N-1$, we have

$$\begin{aligned}
W_{D'}(1,n) - W_{D'}(2,n) &= (N-3)[D'(1,n) - D'(2,n)] - S_{D'}(1) + S_{D'}(2) \\
&= (N-3)[D(1,n) - D(2,n)] - S_{D'}(1) + S_{D'}(2) \\
&= (N-2)[D(1,n) - D(2,n)] - [D(1,n) - D(2,n)] \\
&\qquad -S_D(1) + D(1,N) + S_D(2) - D(2,N) \\
&= (N-2)[D(1,n) - D(2,n)] - S_D(1) + S_D(2) \\
&\qquad -[D(1,n) - D(2,n)] + D(1,N) - D(2,N) \\
&= W_D(1,n) - W_D(2,n) - [D(1,n) - D(2,n)] \\
&\qquad +D(1,N) - D(2,N) \\
&= -[D(1,n) - D(2,n)] + D(1,N) - D(2,N), \qquad (4.4)
\end{aligned}$$

where the last inequality follows from (4.3). As $1 \sim 2$ in D, we have $D(1,N) + D(2,n) - D(1,n) - D(2,N) = 0$. Therefore from (4.4), we have for $n = 3, \cdots, N-1$,

$$W_{D'}(1,n) - W_{D'}(2,n) = 0,$$

which means $1 \sim 2$ in D' by Theorem 3.5.

Any square submatrices of D can be formed by removing a row and the corresponding column from D consecutively. Therefore the above argument can be repeated each time and thus we have $i \sim j$ in any square submatrices of D containing the end-vertices i, j. □

Proposition 4.2. *Let D be an additive distance matrix of size $N \geq 4$, then the number of neighbor pairs in any square submatrices of D is less than or equal to D.*

Proof. Let T be the binary tree corresponding to D. Let D' be the square submatrix of D, where the N-th row and column of D is removed. Recall that the indices in the matrix D are corresponding to only end-vertices in T. Let N be an end-vertex in T, and b be the internal vertex adjacent to N in T. Let c, d be the other two vertices adjacent to b in T (see Figure 4.7).

Let T' be the binary tree constructed by the following steps: first, we remove the end-vertex N, the internal vertex b and also the edges e_c, e_d, e_N from T; then we add an edge e_{cd} to connect c, d with length

$$l(e_{cd}) := l(e_c) + l(e_d). \tag{4.5}$$

Now T' looks exactly like Figure 4.8. We can see that T' is a binary tree because the vertex degrees of c, d are unchanged.

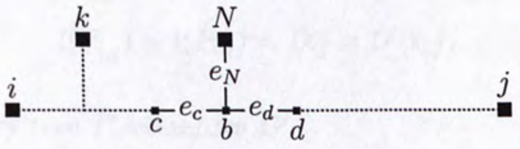

Figure 4.7: The binary tree T in Proposition 4.2.

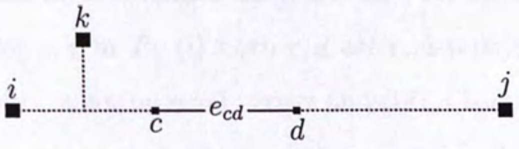

Figure 4.8: The binary tree T' in Proposition 4.2

We want to show that the binary tree T' constructed this way is visualizing D'. It means we have to show $l(P'_{i,j}) = D'(i,j)$ for all $i, j = 1, \cdots, N-1$, where $P'_{i,j}$ is the path of i, j in T'.

First, we want to show $l(P'_{i,j}) = l(P_{i,j})$, where $P_{i,j}$ is the path of i, j in T. To see that, if the path $P'_{i,k}$ in T' does not contain the edge e_{cd}, then it is exactly the path $P_{i,k}$ in T. Therefore $l(P'_{i,k}) = l(P_{i,k})$. If the path $P_{i,j}$ in T' contains the edge e_{cd}, then the path $P_{i,j}$ in T contains the edges e_c, e_d. Therefore we have

$$\begin{aligned}
l(P'_{i,j}) &= l(P'_{i,c}) + l(P'_{c,d}) + l(P'_{d,j}) \\
&= l(P'_{i,c}) + l(P'_{c,d}) + l(P'_{d,j}) \\
&= l(P_{i,c}) + l(e_{cd}) + l(P_{d,j}) \\
&= l(P_{i,c}) + l(e_c) + l(e_d) + l(P_{d,j}) \\
&= l(P_{i,c}) + l(P_{c,a}) + l(P_{a,d}) + l(P_{d,j}) \\
&= l(P_{i,j}).
\end{aligned}$$

Therefore we have for all $i, j = 1, \cdots, N-1$

$$l(P'_{i,j}) = l(P_{i,j}) = D_{ij} = D'(i,j),$$

and hence the binary tree T' visualizes D'.

We now prove that the operations on T to form T' will not increase the number of parent vertices in T'. Since only the vertices b, c, d, N are modified from T to T', we investigate the cases on them only. As b, N are removed in T', we only consider the cases for c, d in T: (i) both c, d are end-vertices, (ii) c, d are both internal vertices, only c is an internal vertex and (iii) d is an end-vertex.

Case(i): We now prove that the case (i) is not possible. Recall that any internal vertices in a binary tree are of degree three, and any end-vertices in it are of degree one. Now the internal vertex b is connecting exactly the three end-vertices c, d, N in T, and therefore b, c, d, N cannot connect to other vertices in T. Hence if there are another vertex in T, then T will be disconnected. Therefore T has only three end-vertices, i.e. $N = 3$, which contradicts to our assumption that $N \geq 4$ stated in the beginning of the proposition.

Case(ii): If both c, d are internal vertices in T, then c, d remain as internal vertices in T', e.g. if c is a link vertex in T', then c will remain a link vertex in T'.

Case(iii): If c is an internal vertex and d is an end-vertex in T, then b is a parent vertex in T as it is adjacent to both d and N in T. No matter what c will change to, the number of neighbor pairs in T' will not be increased as a parent vertex b in T has already been removed in T'.

Summing up the three cases, we have that the number of parent vertices in T' is less than or equal to T. As each parent vertex corresponds to a neighbor pair, the number of neighbor pairs in T' is less than or equal to T, and therefore the number of neighbor pairs in D' is less than or equal to D.

Any square submatrices of D can be formed by removing a row and the corresponding column from D consecutively. Therefore the above argument can be repeated each time and thus we have the number of neighbor pairs in any square submatrices of D is less than or equal to D. □

Proposition 4.3. *Let T be a binary tree with N end-vertices and let $a \sim b$, $c \sim d$ and $e \sim f$ be the only neighbor pairs in T. Let the set Y be defined as:*

$$Y := \{a, b, c, v_1, \cdots, v_k\}, \tag{4.6}$$

where $v_i \in Y$ if and only if $v_i \sim a$ in the binary tree containing only v_i, a, c and e. Then the binary tree T' containing the end-vertices in Y has only two neighbor pairs.

Proof. By Proposition 3.3, we know T' has at least two neighbor pairs. Therefore we want to show that if T' has more than two neighbor pairs, we will have a contradiction. We separate the proof into three cases: $|Y| < 6$, $|Y| = 6$ and $|Y| \geq 6$.

If $|Y| < 6$, then T' must have two neighbor pairs because we need two end-vertices to form a neighbor pair. If $|Y| = 6$, then T' will look like either Figure 4.9 or Figure 4.10. Note that the existence of $a \sim b$ in T' is guaranteed by Proposition 4.1.

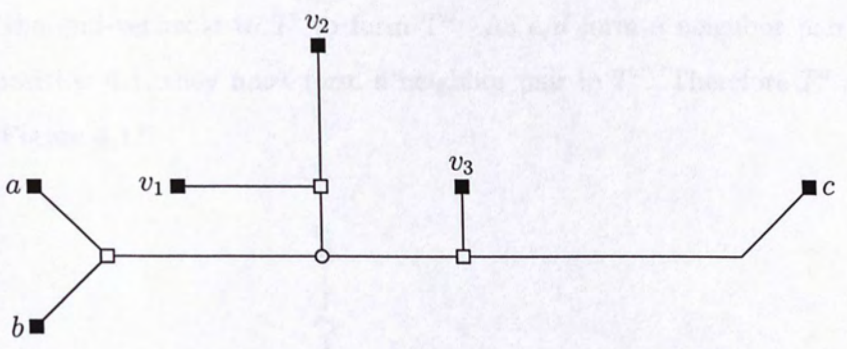

Figure 4.9: A binary tree T'.

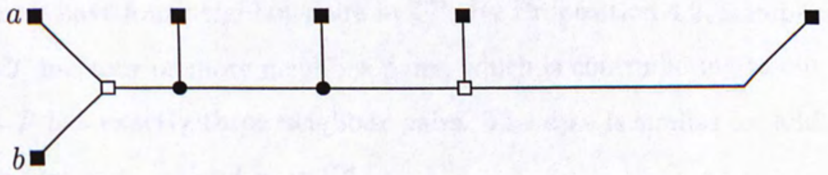

Figure 4.10: A binary tree having two neighbor pairs.

We will show that the first case is not possible and therefore T' must look like Figure 4.10. Suppose T' is the binary tree as shown in Figure 4.9. We add the end-vertex d to T' to form T''. As c, d form a neighbor pair in T, by Proposition 4.1, they must form a neighbor pair in T''. Therefore T'' must look like Figure 4.11.

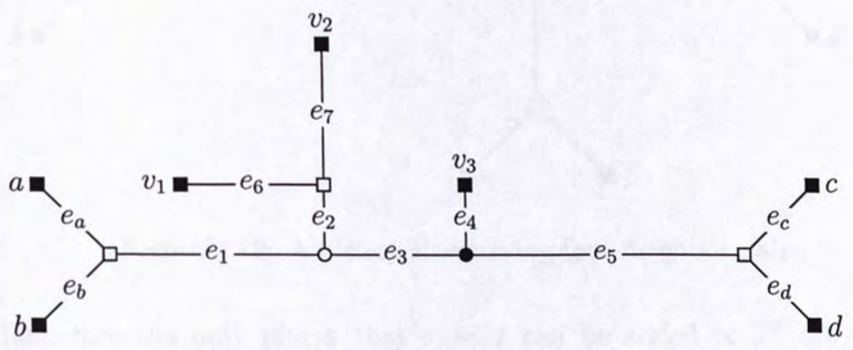

Figure 4.11: The binary tree T''.

Now we add e, f to T'' to form T'''. By Proposition 4.1, $e \sim f$ must be a neighbor pair in T'''. If $e \sim f$ are added at the edge e_5 in T'' to form Figure 4.12, then we have four neighbor pairs in T'''. By Proposition 4.2, it implies the binary tree T has four or more neighbor pairs, which is contradicting to our assumption that T has exactly three neighbor pairs. The case is similar for adding $e \sim f$ at the edges e_1, e_2, e_3 and e_4 in T''.

If $e \sim f$ were added in between the neighbor pair $a \sim b$ or $c \sim d$ in T'', i.e. e_a, e_b, e_c or e_d in Figure 4.11, then $a \sim b$ or $c \sim d$ would not be a neighbor pair in T'''. Since $a \sim b$ and $c \sim d$ are neighbor pairs in T, by Proposition 4.1, they must also be neighbor pairs in T''', hence we have a contradiction and therefore $e \sim f$ cannot be added in between the neighbor pair $a \sim b$ or $c \sim d$.

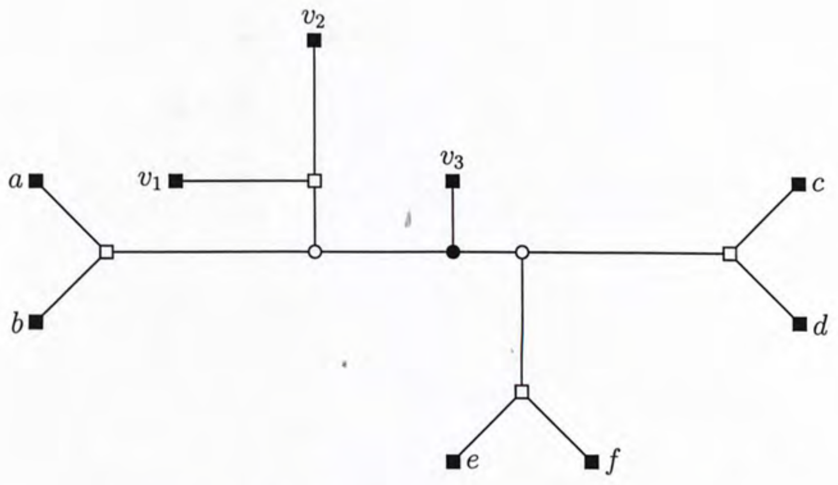

Figure 4.12: A binary tree having four neighbor pairs.

Therefore the only places that $e \sim f$ can be added to T'' are the edges between the neighbor pair $v_1 \sim v_2$, i.e. e_6, e_7. WLOG, we assume $e \sim f$ is added in between e_6 to T'' and forming T''' as shown in Figure 4.13. However, if we consider only the binary tree containing the end-vertices v_1, a, c and e, we see that $v_1 \sim e$ which implies $v_1 \notin Y$, a contradiction to the definition of the set Y in (4.6).

We have shown that existence of T''' lead to contradictions, and therefore the binary tree T' in Figure 4.9 is wrong. Therefore T' must look like Figure 4.10 and hence it has only two neighbor pairs.

For $|Y| > 6$, and if T' has three or more neighbor pairs, then there exists v_i, v_j that form a neighbor pair in T'. Using a similar approach in the above, we can add d and $e \sim f$ to T' to obtain a contradiction. Therefore the binary tree T' containing the end-vertices in Y has only two neighbor pairs. □

Proposition 4.4. *The divide-and-conquer procedure (line 10 to 32) in Algorithm 3 is correct, i.e. given a binary tree T with exactly three neighbor pairs, the topology obtained from the procedure given the input D is the topology of T,*

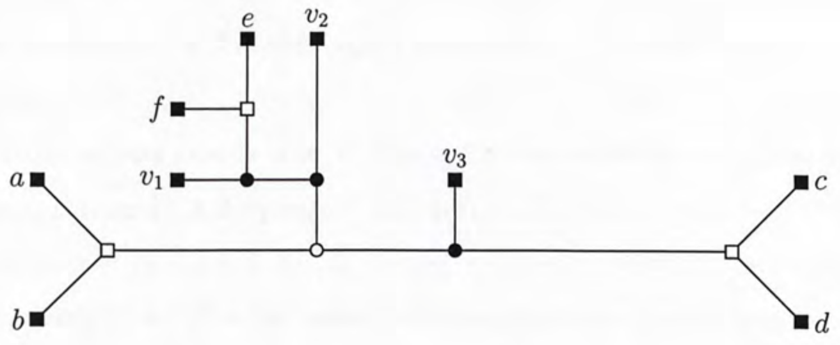

Figure 4.13: The binary T'''.

where D is the additive matrix corresponding to T.

Proof. WLOG assume the neighbor pairs in T are $1 \sim 2$, $3 \sim 4$ and $5 \sim 6$. Since T has three neighbor pairs, by Proposition 3.2, T has only one bridge vertex. Therefore we can assume T looks like Figure 4.14 and the topology of T is

$$\begin{aligned}\mathcal{T} &= [1 \sim 2, 1 \sim v_1, \cdots, 1 \sim v_n, 3 \sim 4, 3 \sim u_1, 3 \sim u_2, \cdots, 3 \sim u_m, \\ &\qquad 5 \sim 6, 5 \sim w_1, \cdots, 5 \sim w_r, 1 \sim 3, 1 \sim 5].\end{aligned}$$

We now show that $\mathcal{T}'$, the topology found by the divide-and-conquer procedure given D, is exactly $\mathcal{T}$. To prove this, we will show both cases

$$\mathcal{T}' = [\cdots, 1 \sim u_1, \cdots], \tag{4.7}$$

i.e. u_1 is in the cluster of the neighbor pair $1 \sim 2$,

$$\mathcal{T}' = [\cdots, 1 \sim v_2, 1 \sim v_1 \cdots], \tag{4.8}$$

i.e. the ordering of v_1, v_2 from the topology of cluster 1 is wrong, are not possible.

For the first case, if $1 \sim u_1$ is found in $\mathcal{T}'$, it means that u_1 will form a neighbor with the end-vertex 1 in the binary tree containing $u_1, 1, 3$ and 5 only.

However, as we can see from T, it is not possible as u_1 will form a neighbor with the end-vertex 3 in this binary tree instead of 1. Hence the first case is not possible.

If the second case is true, it means that the topology computed on the appended cluster $\{1, 2, 3, v_1, v_2, \cdots, v_n\}$ is $[1 \sim 2, 1 \sim v_2, 1 \sim v_1, \cdots, 1 \sim 3]$ instead of the correct one $[1 \sim 2, 1 \sim v_1, 1 \sim v_2, \cdots, 1 \sim 3]$. However, as we have proved in Proposition 4.3 that the appended cluster has exactly two neighbor pairs, we can apply Algorithm 2 to the appended cluster to find the topology. By Proposition 3.6, Algorithm 2 will return the correct topology of the appended cluster, and hence the second case is not possible.

We have proved both (4.7) and (4.8) are not possible. The end-vertices u_1, v_1 and v_2 in the above argument can be generalized to any u_i, v_j, v_k. Therefore the only topology which the divide-and-conquer can find is exactly the same as $\mathcal{T}$. □

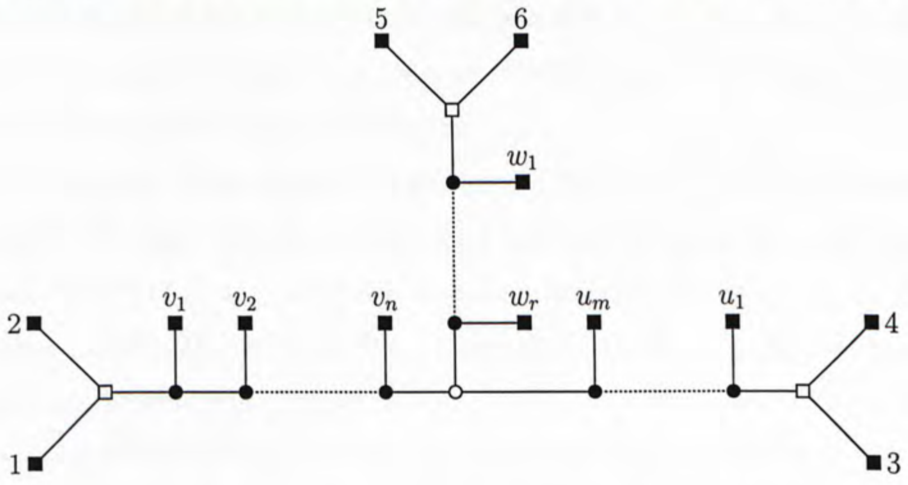

Figure 4.14: Binary tree T corresponding to Proposition 4.4.

Theorem 4.5 (Correctness of DCNJ on additive matrices). *Let T be a binary tree with N end-vertices and D be the additive matrix corresponding to T. Then the binary tree computed from D by DCNJ equals T.*

Proof. The DCNJ algorithm has two phases (see beginning of Section 4.1). The phase one is combining several neighbor pairs found by Algorithm 1. Since the end-vertices joined must be neighbor pairs, as guaranteed by Theorem 3.5, phase one is correct. In phase two, we will either apply the divide-and conquer procedure or Algorithm 2. Their correctness have been proved in Proposition 4.4 and Proposition 3.6 respectively. Hence DCNJ is correct. □

Proposition 4.6 (Complexity of DCNJ). *DCNJ is an $\mathcal{O}(N^3)$ algorithm, where N is the size of the input matrix D.*

Proof. The DCNJ algorithm has two phases. In phase one, the process of finding all neighbor pairs in D by Algorithm 1 needs $\mathcal{O}(N^2)$. If the number of pairs $Q \geq 4$, then we add all the neighbor pairs to the topology $\mathcal{T}$ in $\mathcal{O}(N)$. After that we remove Q rows and columns from D, and therefore it needs at most $\mathcal{O}(N^2)$. Hence the complexity for each run of this phase is $\mathcal{O}(N^2)$. We need to repeat phase one at most $\mathcal{O}(N)$ times, as there are N end-vertices in D. Hence, we need at most $\mathcal{O}(N^3)$ before going to phase two.

In phase two, if the number of pairs $Q = 3$, we start our divide-and-conquer approach. For each end-vertex v that does not form neighbor pairs, we find the submatrix $D^{(v)}$ of D, where the size of this matrix is 4. Therefore computing the neighbor pairs in $D^{(v)}$ by Algorithm 1 needs $\mathcal{O}(4^2) = \mathcal{O}(1)$. Hence, the process of dividing the remaining end-vertices into different clusters takes $\mathcal{O}(N)$ as there are at most $\mathcal{O}(N)$ end-vertices that do not form a neighbor pair. We know that the size of each cluster is at most $\mathcal{O}(N)$, and therefore applying Algorithm 2 to compute the topology on each cluster takes $\mathcal{O}(N^2)$. There are three clusters in total and therefore the complexity to compute the topology for each of them is $3 \times \mathcal{O}(N^2) = \mathcal{O}(N^2)$. Hence, the complexity of divide-and-conquer procedure is $\mathcal{O}(N^2)$. In phase two, if the number of pairs $Q = 2$, we can apply Algorithm 2 to get the topology using $\mathcal{O}(N^2)$.

We need $\mathcal{O}(N^3)$ in phase one, $\mathcal{O}(N^2)$ in phase two and $\mathcal{O}(N)$ to combine the topologies found in both phases. Therefore the total complexity for DCNJ is $\mathcal{O}(N^3)$. □

Although DCNJ has the same complexity compared to NJ, our experiments in Chapter 5 show that DCNJ outperforms NJ.

Chapter 5

Experimental Results

Although Waterman et al. [19] has already proposed an $\mathcal{O}(N^2)$ algorithm on implementing NJ on additive matrices, this algorithm cannot be applied to non-additive matrices. It is mentioned in Atteson [1] the number of additive matrices is too small compared to the non-additive ones, and therefore the standard complexity in NJ now is still $\mathcal{O}(N^3)$. Although our algorithm has the complexity of $\mathcal{O}(N^3)$ and only applicable to additive matrices now, it can be further developed to apply to non-additive matrices. Therefore, this algorithm serves as a stepping stone to a more general algorithm. Hence, we only compare the speed of our algorithm to the algorithm of Studier and Keppler (Section 2.4) implemented in two famous software packages containing NJ: PHYLIP [6] and MEGA [17]. Both of them are written in C++ and we download the executables directly. Our algorithm is written in MATLAB.

All experiments were done in a desktop computer having 3.2G CPU and 1G Ram on a Windows XP environment. We first generate a binary tree with arbitrary topology and then assign a random positive number to each branch as branch length. Next, we compute the distance matrix corresponding to the binary tree. By Proposition 2.4, we know that the distance matrix must be additive. After that, we input the matrix into PHYLIP [6], MEGA [17] and our algorithm DCNJ separately. Finally, we take the average time of ten independent runs. On additive distance matrices, as we have proved in Chapter 4, all of our results are identical to NJ. We can see from Table 5.1 that the speed of DCNJ outperforms the other two algorithms, despite they all have the same complexity $\mathcal{O}(N^3)$. The size of the matrices in the real applications ranges from tens to several hundreds (for example in [9] and [18]).

Matrix Size, N	NJ_P [6]	NJ_M [17]	DCNJ
100	< 1	< 1	0.1
200	< 1	< 1	0.3
400	1.6	1.9	0.4
800	9.8	6.4	3.3
1600	75.3	40	5.2
2000	150.6	81.3	11.2

Table 5.1: Comparison of NJ, DCNJ on additive matrices with NJ implemented in PHYLIP (NJ_P) and MEGA (NJ_M). Time is measured in seconds.

Chapter 6

Conclusions

In the thesis, we showed various properties of binary trees. Also, we proposed and proved the equivalent condition for neighbor pairs on W. Using this condition, we gave a new algorithm named DCNJ on implementing Neighbor-Joining in $\mathcal{O}(N^3)$ operations on additive matrices. Besides, we proved the correctness of this algorithm. Our experiments show that this algorithm is much faster than the NJ method implemented in MEGA and PHYLIP, despite they are of the same complexity. Although this algorithm is not as good as Waterman's algorithm on additive matrices in terms of complexity, from our experiments we believe DCNJ can be modified to achieve the complexity of $\mathcal{O}(N^2 \log N)$.

On the other hand, we are trying to relax the similar row condition in finding neighbor pairs in DCNJ such that it can be applied to non-additive matrices. In DCNJ, the similar condition $\mathbf{w}_i \sim \mathbf{w}_j$ can be rewritten as

$$\sum_{k \neq i,j} |W_{ik} - W_{jk}| = 0.$$

In non-additive matrices, the above equation has to be relaxed to, for instance,

$$\sum_{k \neq i,j} |W_{ik} - W_{jk}| < \epsilon,$$

where ϵ is a parameter to be determined after giving the distance matrix. Our target will be to estimate ϵ such that DCNJ can compute the binary tree exactly as in NJ.

Bibliography

[1] K. Atteson. The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25(2-3):251–278, 1999.

[2] K. H. Chu, C. P. Li, and J. Qi. Ribosomal RNA as molecular barcodes: a simple correlation analysis without sequence alignment. *Bioinformatics*, 22(14):1690–1701, 2006.

[3] I. Elias and J. Lagergren. Fast neighbor-joining. In *Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1263–1274. Springer-Verlag, 2005.

[4] J. Evans, L. Sheneman, and J. Foster. Relaxed neighbor-joining: A fast distance-based phylogenetic tree construction method. *Journal of Molecular Evolution*, 62(6):785–792, 2006.

[5] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, 17:368–376, 1981.

[6] J. Felsenstein. *PHYLIP (Phylogeny Inference Package) version 3.5c*. Distributed by the author. Department of Genetics, University of Washington, Seattle., 1993.

[7] W.M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.

[8] S. Grenewald, K. Forslund, A. W. M. Dress, and V. Moulton. Qnet: An agglomerative method for the construction of phylogenetic networks from weighted quartets. *Mol. Biol. Evol.*, 24(2):532–538, 2007.

[9] Daniel H. Huson. Splitstree: analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998.

[10] Regents of the University of California: History of life through time (2008). *http://www.ucmp.berkeley.edu/exhibits/historyoflife.php.*

[11] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4(4):406–425, 1987.

[12] S. Sattath and A. Tversky. Additive similarity trees. *Psychometrika*, 42(3):319–345, 1977.

[13] P.H.A. Sheath and R. R. Sokal. *Numerical Taxonomy.* Freeman, San Francisco, California, 1973.

[14] J.A. Studier and K.J. Keppler. A note on the neighbor-joining method of Saitou and Nei. *Mol. Biol. Evol.*, 5:729–731, 1981.

[15] D. L. Swofford. *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods) Version 4.* Sinauer Associates, Sunderland, Massachusetts, 2003.

[16] K. Tamura, J. Dudley, M. Nei, and S. Kumar. MEGA4: Molecular Evolutionary Genetics Analysis (MEGA) Software Version 4.0. *Mol. Biol. Evol.*, 24(8):1596–1599, 2007.

[17] M. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *J. Theo. Bio.*, 64:199–213, 1977.

[18] R. J. Wilson. *Introduction to Graph Theory (4th ed.).* John Wiley & Sons, Inc., New York, NY, USA, 1996.