Evaluating Phylogenetic Trees by Matrix Decomposition

S. Qiao

Department of Computer Science and Systems
McMaster University
Hamilton, Ontario L8S 4K1
Canada

W. S-Y. Wang

Department of Electronic Engineering
City University of Hong Kong
Kowloon, Hong Kong

(Also at the University of California at Berkeley, CA, USA)

Suggested running head: Evaluating Phylogenetic Trees by Marix

Mailing address:

Prof. W. S-Y. Wang

Department of Electronic Engineering

City University of Hong Kong

Kowloon, Hong Kong

Phone: (852)2788-7196

Fax: (852)2788-7791

1. Introduction

Suppose we are given N objects, and the distances between all pairs of these objects. We will call the matrix of these distances the *Input Matrix.* Our aim is to construct an unrooted binary tree from the Input Matrix with the tips of the tree representing the N objects. In an unrooted binary tree, each node connects exactly three branches, and the branches are not directional. Note that tips are not nodes. The distance between any pair of tips is the sum of the lengths of the branches which connect them. From these distances we may construct an *Output Matrix.* A *Difference Matrix* can then be obtained by subtracting the Output Matrix from the Input Matrix. The tree should have the property that the sum of the squares of the entries in the Difference Matrix should be minimal [8].

An application of these procedures is from linguistics, where the objects are languages. The distance between any pair of languages may be measured in terms of some similarity index, based, for example, on the amount of shared basic vocabulary. For example, each entry of the following matrix

$$
S = \begin{array}{l} \\ \text{English} \\ \text{German} \\ \text{Swedish} \\ \text{French} \\ \text{Spanish} \\ \text{Italian} \\ \text{Russian} \end{array}
\begin{array}{c} \begin{array}{ccccccc} \text{EN} & \text{GE} & \text{SW} & \text{FR} & \text{SP} & \text{IT} & \text{RU} \end{array} \\
\begin{pmatrix}
100 & 58 & 59 & 24 & 24 & 25 & 24 \\
58 & 100 & 70 & 24 & 25 & 27 & 25 \\
59 & 70 & 100 & 24 & 25 & 26 & 25 \\
24 & 24 & 24 & 100 & 73 & 80 & 22 \\
24 & 25 & 25 & 73 & 100 & 79 & 23 \\
25 & 27 & 26 & 80 & 79 & 100 & 24 \\
24 & 25 & 25 & 22 & 23 & 24 & 100
\end{pmatrix} \end{array}
$$

shows the similarity between a pair of European languages [4]. The corresponding Input Matrix may be obtained by

$$
D = -\log(S/100) = \begin{pmatrix}
0 & 0.5447 & 0.5276 & 1.4271 & 1.4271 & 1.3863 & 1.4271 \\
0.5447 & 0 & 0.3567 & 1.4271 & 1.3863 & 1.3093 & 1.3863 \\
0.5276 & 0.3567 & 0 & 1.4271 & 1.3863 & 1.3471 & 1.3863 \\
1.4271 & 1.4271 & 1.4271 & 0 & 0.3147 & 0.2231 & 1.5141 \\
1.4271 & 1.3863 & 1.3863 & 0.3147 & 0 & 0.2357 & 1.4697 \\
1.3863 & 1.3093 & 1.3471 & 0.2231 & 0.2357 & 0 & 1.4271 \\
1.4271 & 1.3863 & 1.3863 & 1.5141 & 1.4697 & 1.4271 & 0
\end{pmatrix}.
$$

To solve this problem, we need two procedures. First, we need to generate trees of N tips, which can be achieved by adding branches to a smaller tree, starting with the trivial tree of three tips. Second, after a tree of N tips is constructed, we need to calculate the distances of all pairs of tips. A proper data representation of trees can lead to efficient algorithms. Trees can be represented by, for example, linked lists. In this paper, we propose a matrix representation of trees and show how to add branches and calculate the distances between tips using this matrix representation. Matrices can be manipulated easily and efficiently. Moreover, in Section 4.5, we show an efficient method for computing the distances between tips using this matrix representation.

The tree which results from the procedures discussed in the present paper would allow us to go beyond pairwise distances, and show how the languages are related to each other as a group. This relationship may then be studied further with respect to other linguistic implications, such as vertical and horizontal transmissions.

2. Neighbor-joining Method

The neighbor-joining method proposed by Saitou and Nei [10] constructs the tree by repeatedly joining a pair of tips or nodes. We illustrate their method using the seven language example in Section 1. We denote the number of objects by N. In our example, N equals 7. We also number the languages in the order shown in the matrix S in Section 1. That is EN (English) is numbered by 1, GE (German) by 2, and so forth. Suppose we choose to join tips 1(EN) and 2(GE) using a new node n. As shown in Figure 1, we denote the third neighbor of n by m and the three branches connected to n by b_1, b_2, and b_3.

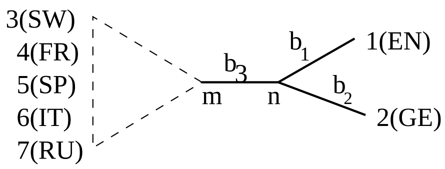


Figure 1. Join tips 1(EN) and 2(GE) by node n.

Let B_m be the sum of the lengths of the branches connecting node m and tips 3 to 7, then the sum of the distances between tip 1(EN) and those five tips is $(N-2)(b_1+b_3)+B_m$. Since the distance between tips 1(EN) and 2(GE) is b_1+b_2, the sum of the distances between tip 1(EN) and the other six $(N-1)$ tips is

$$S_1 = b_1 + b_2 + (N-2)(b_1+b_3) + B_m. \tag{1}$$

Similarly, the sum of the distances between tip 2(GE) and the other six $(N-1)$ tips is

$$S_2 = b_1 + b_2 + (N-2)(b_2+b_3) + B_m. \tag{2}$$

The key measurement in the neighbor-joining method is the sum of the lengthes of the branches connecting the new node and all the tips. Denoting this sum as S_n, similar to (1) and (2), we have

$$S_n = b_1 + b_2 + (N-2)b_3 + B_m. \tag{3}$$

The above expression for S_n contains unknown branch lengths and B_m. However, comparing (3) with the sum of (1) and (2), we obtain

$$S_n = \frac{1}{2}(S_1 + S_2 - (N-2)(b_1+b_2)), \tag{4}$$

which can be computed from the Input Matrix D as follows. Since S_1 is the sum of the distances between tip 1(EN) and other six tips, it is simply the sum of the entries in the first row of D. In our example $S_1 = 6.7399$. Similarly, $S_2 = 6.4104$ is the sum of the entries in the second row of D. The term b_1+b_2 in (4) is the distance between tips 1(EN) and 2(GE), i.e., the (1,2)-entry, denoted by D_{12}, of the Input Matrix D. The neighbor-joining method searches for a pair of tips to minimize

$$\tilde{S} = S - S_n$$

where S is the total sum of the distances of all possible pairs of tips. In terms of the Input Matrix D, S is the sum of the entries above (or below) the zero diagonal of D. Table 1 lists the values of $\tilde{S}$ and S_n for all possible pairs.

pairs	(1,2)	(1,3)	(2,3)	(1,4)	(2,4)	(3,4)	(1,5)	(2,5)	(3,5)	(4,5)
$\tilde{S}$	5.213	3.614	3.562	4.074	4.107	4.105	4.085	4.098	4.096	3.569
S_n	3.625	5.267	5.529	2.969	2.804	2.814	2.912	2.849	2.860	5.490

pairs	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)
$\tilde{S}$	4.094	4.088	4.105	3.553	3.570	3.846	3.858	3.856	3.930	3.919	3.927
S_n	2.869	2.896	2.812	5.573	5.485	4.108	4.045	4.055	3.687	3.741	3.702

Table 1. The values of $\tilde{S}$ and S_n for all pairs.

Since the pair $(4, 6)$ gives the smallest $\tilde{S}$, we join tips 4(FR) and 6(IT) using node n as shown in Figure 2. Next, we compute the lengths of the branches b_4 and b_6 and reduce the problem size (the number of tips) by merging tips 4(FR) and 6(IT) into the new node n.

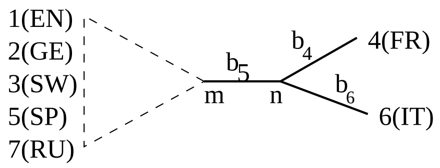


Figure 2. Join tips 4(FR) and 6(IT) by node n.

The branch lengths of b_4 and b_6 can be computed as follows. Analogous to (1) and (2), the sum of the distances between tip 4(FR) and other six tips is

$$S_4 = b_4 + b_6 + (N-2)(b_4 + b_5) + B_m,$$

and the sum of the distances between tip 6(IT) and other six tips is

$$S_6 = b_4 + b_6 + (N-2)(b_5 + b_6) + B_m.$$

It follows that

$$(S_4 - S_6)/(N-2) = b_4 - b_6$$

where S_4 and S_6 are given by the sums of the entries in the row 4 and row 6 respectively. Consequently, the length of b_4 can be calculated by

$$(S_4 - S_6)/(N-2) + (b_4 + b_6) = 2b_4$$

where $b_4 + b_6$, the distance between tips 4(FR) and 6(IT), is the (4,6)-entry denoted by D_{46} of D. The length of b_6 can be easily obtained by $b_6 = D_{46} - b_4$. For this example, we get

$$b_4 = 0.152 \quad \text{and} \quad b_6 = 0.0711.$$

Having selected the pair (4,6) and computed the branch lengths for b_4 and b_6, we merge tips 4(FR) and 6(IT) into node n and thus reduce the problem size by one. However, in order to continue the procedure, it is necessary to determine the distances between the new node n and the other five tips and update the distance matrix D. The neighbor-joining method defines the distance between the new node n and one of the five tips, say 1(EN), as the average of the distances between tips 4(FR) and 1(EN) and between tips 6(IT) and 1(EN). Following the notations used above, $D_{1n} = (D_{14} + D_{16})/2$. In terms of the Input Matrix D, we delete its row 6 and column 6 and replace row 4 and column 4 by the average of rows 4 and 6 or the average of columns 4 and 6. The following $\tilde{D}$ is the updated matrix after tips 4(FR) and 6(IT) have been merged into n.

$$\tilde{D} = \begin{array}{c} \\ \text{EN} \\ \text{GE} \\ \text{SW} \\ n \\ \text{SP} \\ \text{RU} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{EN} & \text{GE} & \text{SW} & n & \text{SP} & \text{RU} \end{array} \\ \begin{pmatrix} 0 & 0.5447 & 0.5276 & 1.4067 & 1.4271 & 1.4271 \\ 0.5447 & 0 & 0.3567 & 1.3682 & 1.3863 & 1.3863 \\ 0.5276 & 0.3567 & 0 & 1.3871 & 1.3863 & 1.3863 \\ 1.4067 & 1.3682 & 1.3871 & 0 & 0.2752 & 1.4706 \\ 1.4271 & 1.3863 & 1.3863 & 0.2752 & 0 & 1.4697 \\ 1.4271 & 1.3863 & 1.3863 & 1.4706 & 1.4697 & 0 \end{pmatrix} \end{array}.$$

This procedure is repeated $N-3$ times and terminates at three tips or nodes. It should be pointed out that the distance D_{1n} includes the average distance from n to tips 4(FR) and 6(IT) in addition to the distance from n to tip 1(EN). Consequently, in the computation of the branches in the subsequent steps, it is necessary to make adjustments. We refer details to Saitou and Nei [10].

To summarize, we present the neighbor-joining algorithm using the notations introduced above.

Algorithm NJ

Input Matrix D;

Repeat $N-3$ times,

Compute S, the total sum of distances, from D;

For each pair (i,j),

$S_n = (S_i + S_j - (N-2)D_{ij})/2$;

$\tilde{S} = (S - S_n)/(N-2)$;

Save data for the smallest $\tilde{S}$;

Compute branches b_i and b_j (make adjustments);

Compute $D_{kn} = (D_{ik} + D_{jk})/2$ for all k $(k \neq i,j)$;

Replace i with n and delete j;

Update the Input Matrix;

Find the branches of the last three nodes.

To conclude this section, we propose the following two modifications. First, we propose to compute S_n instead of $\tilde{S}$. Since S, the total sum of distances, is a constant for all pairs and $\tilde{S} = (S - S_n)/(N-2)$, minimizing $\tilde{S}$ is equivalent to maximizing S_n. This requires less computation and simplifies the computation of the branches. In particular, (1) and (3) show that

$$b_1 = (S_1 - S_n)/(N-2).$$

Since S_n is the sum of the distances from node n to other tips, maximizing S_n can be interpreted as finding a pair which is as remote to other tips as possible. Second, we suggest to compute the distance

$$D_{kn} = (D_{ik} + D_{jk} - D_{ij})/2,$$

Since this D_{nk} excludes the average distance from node n to its descendants i and j, no adjustments are necessary for computing the branches in the subsequent steps. Consequently, it saves computations and gives more accurate branch lengths.

We note that an early decision on pairing objects does not guarantee the final optimal solution.

3. Fitch and Margoliash's Method

We describe the Fitch and Margoliash's method based on Felsenstein's implementation [5]. We start with a tree of three tips. There is only one tree of three. Its three branch lengths are calculated using the distances of all pairs of the three objects given in the Input Matrix D. Then we construct trees of four by adding a new branch to any of the three branches of the tree of three. Thus there are three trees of four. For each tree, we find the optimal branch lengths in the sense that

$$\sqrt{\frac{\sum_i \sum_j (D_{ij} - \hat{D}_{ij})^2 / D_{ij}^2}{N(N-1)}} \times 100 \tag{5}$$

is minimized, where D_{ij} is the distance between objects i and j given by the (i,j)-entry of the Input Matrix D, $\hat{D}_{ij}$ is the distance induced by adding branch lengths between tips i and j in the tree, and N, currently 4, is the number of objects. The quantity (5) is called percent standard deviation (p.s.d.) by Fitch and Margoliash [6]. Thus it is a weighted linear least squares problem. Felsenstein [5] proposes an iterative method called alternating least squares approach to this problem. Then we select the tree with the smallest p.s.d., add another new branch to any of its branches to generate trees of five. The whole procedure is repeated until we reach a tree of N tips. We note that an early decision on selecting the tree with smallest p.s.d. may not lead to the final optimal tree. Felsenstein [5] suggests a rearrangement technique to heuristically overcome this problem. Also in [5], Felsenstein proposes a technique which disallows negative branch lengths in the solution.

To summarize, we present the following Fitch-Margoliash algorithm.

Algorithm FM.

Input Matrix D;

Construct a tree of three tips;

Repeat $N-3$ times,

  For each branch of the tree,

    Add a new branch to construct a new tree;

    Find branch lengths minimizing p.s.d.;

  Save the tree with minimal p.s.d.;

Example 3.1. Using the Input Matrix D of seven European languages in Section 1, we start with constructing a tree of three tips, namely 1(EN), 2(GE), and 3(SW), as shown in Figure 3. Then we add a new branch with tip 4(FR) to any of its three branches and find the branch lengths based on the Fitch and Margoliash's criterion. Figure 4 shows the three resulting trees. Their p.s.d. values are listed in Table 2. Thus tree (a) in Figure 4, which has the smallest p.s.d., survives, trees (b) and (c) are discarded, and the whole procedure is repeated.

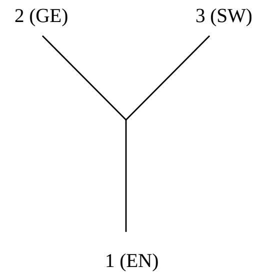


Figure 3. Initial tree of three tips.

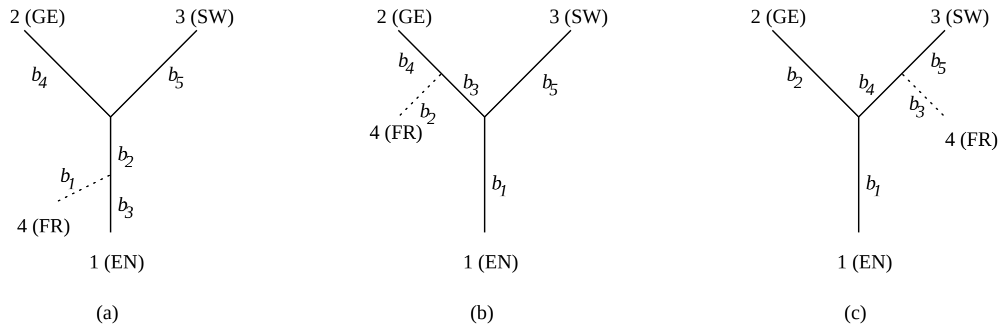


Figure 4. Three trees of four tips.

Trees	(a)	(b)	(c)
p.s.d.	0.32	29.7	29.7

Table 2. Percent standard deviations of the three trees in Figure 4.

4. Matrix Decomposition Method

In this section, we first establish a relation between trees and matrices. Then we show how to update the matrices when a new branch is added to the tree. Thus we can generate trees in the form of matrices. The matrix representing the optimal tree, in the least squares sense, can be found using standard matrix techniques for solving the linear least square problems. Then the matrix can be recomposed into tree form.

4.1. Representing a Tree by Matrices

We call a matrix binary, if its entries are binary. Given a tree of N tips, we first appoint one of its nodes as the center. Since a node has exactly three branches connected to it, there are three binary subtrees connected to the center as shown in Figure 5 where we have ordered the subtrees clockwise.

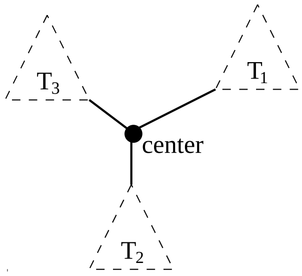


Figure 5. A tree and its three binary subtrees connected to the center.

We then order the branches by traversing the three subtrees clockwise. For each tree, we traverse the branches in the following order. That is we traverse recursively the left subtree, the root of the subtree, and then the right subtree. For example, Figure 6 shows an unrooted tree of $N = 6$ tips. Given the center indicated in the figure, the subtree T_1 contains tips 1, 4, and 6, T_2 contains 2 and 5, and T_3 has only tip 3. The branches in T_1 are traversed first followed by those in T_2 and then T_3. In each subtree, we traversed the branches in-order as illustrated in Figure 6.

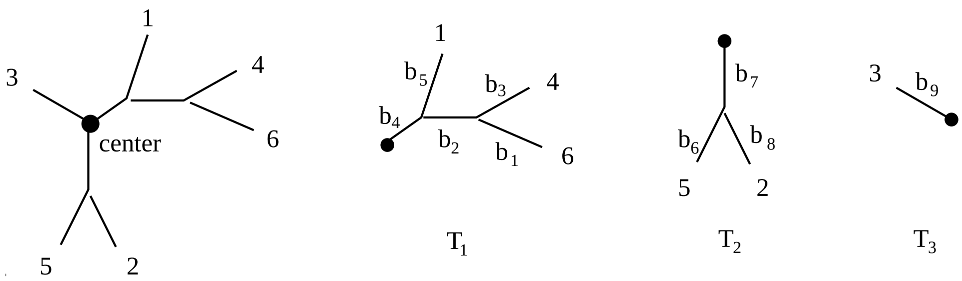


Figure 6. A tree of six tips and its three subtrees with their branches labeled.

Having labeled the branches, we are ready to represent the tree by an N-by-$(2N-3)$ binary matrix P. Note that $2N-3$ is the number of branches in a tree of N tips. The (i,j)-entry of P is 1 if and only if the branch b_j is on the path from the center to tip i. For example, the matrix representing the tree in Figure 6 is

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array},$$

where we bordered the matrix with the tips and branches. We make the following remarks on the association between a tree and its matrix P.

- The row i of P is associated with the path from the center to tip i in the tree. For example, in the tree in Figure 6, since the branches b_4 and b_5 are on the path from the center to tip 1, both the $(1,4)$-entry P_{14} and the $(1,5)$-entry P_{15} in the first row elqual 1 and the rest of the entries in the first row are zero.
- The column j of P is associated with the branch b_j of the tree. For example, in the tree in Figure 6, the branch b_2 is on the paths from the center to tips 4 and 6, so both entries P_{42} and P_{62} in the second column equal 1 and the rest of the entries in the second column are zero.
- If column j of P has exactly one non-zero element, then the branch b_j is a tip branch connecting a tip. For example, column 3 of P contains only one nonzero element in the fourth row, thus b_3 is a tip branch with tip 4.

As pointed out in Introduction, in this problem, we need to generate trees of N tips by adding branches to existing trees starting with the trivial tree. In order to implement this procedure using the matrix representation, we extract three submatrices from P to represent the three subtrees. Then in Section 4.3 we describe the procedure of adding branches using the matrix representation of the subtrees. Consider the subtree T_1 in Figure 6, it has tips 1, 4, and 6. We extract the corresponding rows 1, 4, and 6 from P. On the other hand, since the five branches in T_1 are labeled first, we extract the first five columns from P. Thus the submatrix

$$P_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} b_1 & b_2 & b_3 & b_4 & b_5 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

consisting of the entries in rows 1, 4, and 6 and in the first five columns of P is sufficient for representing the subtree T_1. Note that we have bordered P_1 with the tips and branches in the subtree T_1. Similarly, the submatrices

$$P_2 = \begin{array}{c} \\ 2 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} b_6 & b_7 & b_8 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \quad \text{and} \quad P_3 = \begin{array}{c} \\ 3 \end{array} \begin{array}{c} b_9 \\ \begin{pmatrix} 1 \end{pmatrix} \end{array}$$

represent the other two subtrees T_2 and T_3 respectively. In general, if the subtree T_1 has k tips $n_1, n_2, ..., n_k$, then it has $2k-1$ branches $b_1, b_2, ..., b_{2k-1}$, because it is a binary tree. Since T_1 is traversed first, its branches are the first $2k-1$ branches of T, namely $b_1, b_2, ..., b_{2k-1}$. Thus T_1 can be represented by a k-by-$(2k-1)$ binary matrix P_1 bordered by the tips $n_1, n_2, ..., n_k$ and the branches $b_1, b_2, ..., b_{2k-1}$. Similarly, we can determine P_2 and P_3. Obviously, the total storage required by P_1, P_2, and P_3 is less than that required by P. Thus, this scheme also saves storage in the intermediate steps when adding branches. The final matrix P representing the tree of N tips can be easily reconstructed from its submatrices P_1, P_2, and P_3 using the tips (as row indices) and branches (as column indices) on their borders.

4.2. Constructing the Tree From a Matrix

In the previous subsection, we showed how to represent a subtree by a binary matrix. In this subsection, we describe the construction of the subtree given its corresponding submatrix.

Suppose the submatrix representing the subtree T_1 is given by

$$P_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

We find that column 6 has no zeros. This means that the branch b_6 is on the paths from the center to all tips in this subtree. We call this branch the root branch of the subtree. Obviously, one end of the root branch is connected to the center. If the subtree has only one tip, then the other end of the root branch is connected to the tip. If the subtree has more than one tip, then the other end of the root branch is connected to other two branches. To find these two branches, we first search for a column which has more 1s than any other column except column 6. That is column 2 in P_1, which has three 1s. Then we know that b_2 is one of these two branches connected to the root branch b_6. For the another branch connected to b_6, we search for a column which is the binary complement of column 2, i.e., it equals column 2 XORed by column 6 (a column of 1s). For convenience, we provide the definition of the XOR (exclusive-OR) operation:

a	0	0	1	1
b	0	1	0	1
a XOR b	0	1	1	0

We find that column 8 is the binary complement of column 2. So, the branches b_2 and b_8 are connected to the root branch b_6 as shown in Figure 7.

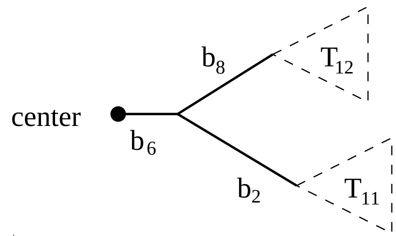


Figure 7. Two branches b_2 and b_8 are connected to the root branch b_6.

Column 8 is the binary complement of column 2, because when we traverse from the root branch b_6 to a tip in the subtree, we must pass either b_2 or b_8. To construct the subtree T_{11} in Figure 7, we extract all rows which contain a 1 in the second column and form a matrix. They are the rows indexed 4, 5, and 7 in P_1 and

$$P_{11} = \begin{array}{c} \\ 4 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

is the matrix associated with the subtree T_{11} in Figure 7. Applying the above procedure to P_{11} except that we ignore column 6 which is already done, we first find that column 2 of P_{11} has no zeros, i.e., b_2 is the root branch of the subtree T_{11}. Then we find that column 4 has more 1s than any other column in P_{11} except columns 2 and 6. We also find that column 1 is the binary complement of column 4. Thus, b_4 and b_1 are the two branches connected to b_2. Repeating this procedure until we exhaust all columns and using the tip numbers on the borders of P_{11}, we get the subtree T_{11} shown in Figure 8.

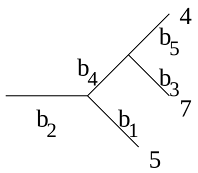


Figure 8. The subtree T_{11} in Figure 7.

Similarly, we can find the subtree T_{12} in Figure 7. Figure 9 shows the subtree T_1 constructed from the matrix P_1.

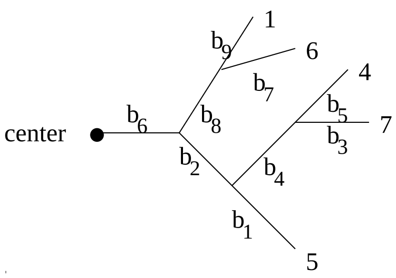


Figure 9. The subtree T_1 constructed from the submatrix P_1.

4.3. Updating the Matrix

In this subsection, we show how to add a new tip branch to a subtree using the matrix representation. We assume that the new branch is always added to the left of an existing branch. We discuss this updating problem in two cases.

4.3.1. Adding a new branch to a tip branch

We first consider the case when a branch with a new tip is added to a tip branch of a subtree. The figure on the left of Figure 10 shows that a branch (dashed line) with a tip n is added to a tip branch with tip i. The figure on the right of Figure 10 shows the labels of the three new branches, recalling that we traverse the left branch followed by the root branch and then the right branch. Then we can see that after a new branch is added, branch b_k becomes b_{k+2} and two branches b_k and b_{k+1} are inserted.

Figure 10. A new branch is added to a tip branch b_k.

For example, in Figure 11, a branch with a new tip $n = 8$ is added to the tip branch b_3 of a subtree T_1 on the left. The new subtree $\hat{T}_1$, after relabeling the branches, is shown on the right of Figure 11.

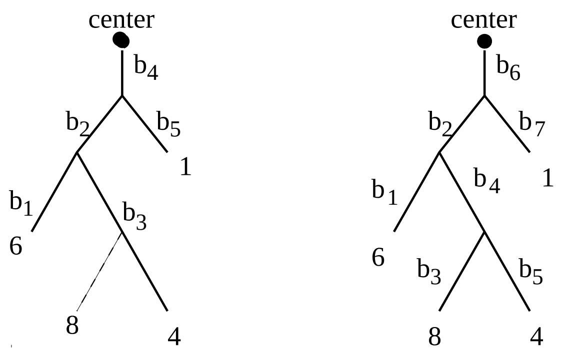


Figure 11. A new tip is added to a tip branch of T_1 on the left and the resulting $\hat{T}_1$ on the right.

Suppose that we have the matrix

$$P_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} b_1 & b_2 & b_3 & b_4 & b_5 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

representing the subtree T_1 on the left of Figure 11 available. We will show how to obtain the matrix corresponding to the the new subtree $\hat{T}_1$ on the right by updating P_1.

Step 1. As described before, when a new branch is added, two branches are inserted. So, by inserting two new columns before column 3 and adding a new row to P_1, we initialize

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{array} \\ \begin{pmatrix} 0 & 0 & \times & \times & 0 & 1 & 1 \\ 0 & 1 & \times & \times & 1 & 1 & 0 \\ 1 & 1 & \times & \times & 0 & 1 & 0 \\ \times & \times & \times & \times & \times & \times & \times \end{pmatrix} \end{array}$$

where "×" indicates an entry to be determined.

Step 2. In the tree $\hat{T}_1$ on the right of Figure 11, ingnoring the the two inserted branches b_3 and b_4, the paths from the center to tip 4 and the new tip 8 have the same branches except b_5. So, we make the last row a copy of the row indexed 4 except its fifth element which is set to zero. This leads to

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{array} \\ \begin{pmatrix} 0 & 0 & \times & \times & 0 & 1 & 1 \\ 0 & 1 & \times & \times & 1 & 1 & 0 \\ 1 & 1 & \times & \times & 0 & 1 & 0 \\ 0 & 1 & \times & \times & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

Step 3. Since the two branches b_3 and b_4 are not on the paths from the center to the tips other than 4 and 8, we set the entries in the columns 3 and 4 of all the rows other than the rows indexed 4 and 8 to zero. We get

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & \times & \times & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & \times & \times & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

Step 4. Now it remains to determine the four entries in the above $\hat{P}_1$ marked by "×". The subtree $\hat{T}_1$ on the right of Figure 11 shows that, after relabeling, branch b_4 is on the paths from the center to tips 4 and 8. Thus we set the entries in column 4 of the rows indexed 4 and 8 to 1 and get

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & \times & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & \times & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

Step 5. Branch b_3 is now a tip branch with the new tip, therefore we set the last element of column 3 to 1 and the rest of column 3 to zero. Finally, we obtain the matrix

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

representing the new subtree $\hat{T}_1$ on the right of Figure 11.

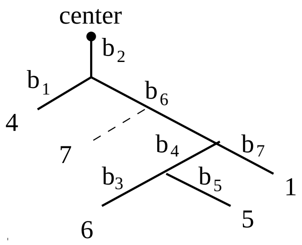
In general, suppose that we have the matrix P_1 representing a subtree T_1 available. When a branch with a new tip is added to a tip branch b_k with tip i in T_1, we can obtain the matrix representing the new subtree by updating P_1 as follows.

1. Initialize a new matrix by inserting two new columns before column k and adding a new row to P_1;

2. Make the last row a copy of the row indexed i and set the $(k+2)$ed element of the last row to zero;
3. Set the entries in columns k and $k+1$ of all the rows other than the row indexed i and the last row to zero;
4. Set the entries in column $k+1$ of the row indexed i and the last row to 1;
5. Set the entry in column k of the last row to 1 and the rest of the elements in column k to zero.

4.3.2. Adding a new branch to an internal branch

Now we consider the case when a branch with a new tip is added to an internal branch (not a tip branch) of a subtree. We illustrate the matrix updating scheme via an example. Suppose that we have the subtree T_1 on the left of Figure 12 available and we add a branch with a new tip $n = 7$ to b_6. The figure on the right of Figure 12 is the subtree $\hat{T}_1$ after relabeling the branches.



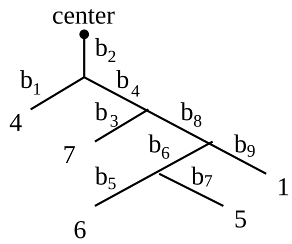


Figure 12. The original subtree T_1 on the left and the resulting tree $\hat{T}_1$ on the right.

The matrix representing T_1 is

$$P_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

The following steps demonstrate how to update P_1 and obtain the matrix representing $\hat{T}_1$.

Step 1. Find all rows whose sixth element is 1. We find that they are the rows indexed 1, 5, and 6. In terms of tree T_1, we locate all tips under b_6.

Step 2. Search for the left most column which has exactly one nonzeros in any of the rows located in Step 1. Column 3, which has exactly one non-zero (in the row indexed 6) is the column. From Figure 12, we can see that this step finds the left-most tip branch under b_6 in T_1.

Step 3. Initialize a matrix $\hat{P}_1$ by inserting two new columns before column 3 found in the previous step and adding a new row to P_1. Specifically,

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 1 & \times & \times & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & \times & \times & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \times & \times & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & \times & \times & 1 & 1 & 0 & 1 & 0 \\ \times & \times & \times & \times & \times & \times & \times & \times & \times \end{pmatrix} \end{array}$$

where "×" marks an entry to be determined. The subtrees in Figure 12 show that when a new branch is added to b_6, branch b_6 becomes b_8 and two branches b_3 and b_4 are inserted. Note that b_3 in the tree on the left of Figure 12 is the left-most branch under b_6 and it was located in Step 2.

Step 4. In this step we determine the elements, except the third and fourth, in the last row. We set its 8th element to 0 and its jth ($j \neq 3$, 4, and 8) element to 1 if and only if the jth element of all the rows (1, 5, and 6) located in Step 1 equals 1. Thus we get

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 1 & \times & \times & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & \times & \times & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \times & \times & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & \times & \times & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & \times & \times & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

In terms of the tree on the right of Figure 12, all branches, ignoring branches 3 and 4 (they will be determined later), common to the paths from the center to tips 1, 5, and 6 (located in Step 1) are on the path to the new tip 7. Note that we exclude b_8 because it is not on the path to the new tip.

Step 5. Since the branches b_3 and b_4 are not on the paths from the center to the new tip and the tips other than those located in Step 1, we set the entries in columns 3 and 4 of the rows other than those indexed 1, 5, 6, and 7 to zero. This leads to

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 1 & \times & \times & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \times & \times & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & \times & \times & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & \times & \times & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Step 6. We set column 4 of the last row and all the rows located in Step 1 to 1, because b_4 is on the paths to the new tip and all the tips located in Step 1 as shown by $\hat{T}_1$ in Figure 12. We now have

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 1 & \times & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \times & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & \times & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & \times & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Step 7. Now it remains four entries in column 3 to be determined. Since b_3 becomes a tip branch with the new tip after relabeling, we set the last row of column 3 to 1 and the rest of column 3 to zero. Finally, we obtain the matrix

$$\hat{P}_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

representing the new tree $\hat{T}_1$ in Figure 12.

In general, suppose that we have the matrix P_1 representing a subtree T_1 available. When a branch with a new tip is added to an internal branch b_k in T_1, we can obtain the matrix representing the new tree by updating P_1 as follows:

1. Locate all the rows whose kth element is 1;
2. Search for the left-most column which has exactly one non-zero in any of the rows located in the previous step, say this is column i;
3. Initialize a new matrix $\hat{P}_1$ by inserting two columns before column i found in the previous step and adding a new row to P_1;
4. Set the $(k+2)$ed element of the last row in $\hat{P}_1$ to 0 and the jth $(j \neq i, i+1, k+2)$ element to 1 if and only if the jth element of all rows located in step 1 equals 1;
5. Set the entries in columns i and $i+1$ of the rows other than the last row and those located in step 1 to 0;
6. Set column $i+1$ of the last row and all rows located in step 1 to 1;
7. Set the last element of column i to 1 and the rest of the column to zero.

4.4. Growing Trees

To generate trees of N tips, we start with the trivial tree of three tips. The three matrices representing its three subtrees are

$$P_1 = \begin{array}{cc} & b_1 \\ 1 & \begin{pmatrix} 1 \end{pmatrix} \end{array}, \quad P_2 = \begin{array}{cc} & b_2 \\ 2 & \begin{pmatrix} 1 \end{pmatrix} \end{array}, \quad \text{and} \quad P_3 = \begin{array}{cc} & b_3 \\ 3 & \begin{pmatrix} 1 \end{pmatrix} \end{array}.$$

To generate a tree with four tips, we add a new tip branch to any of the three subtrees. Thus there are three trees with four tips and each has five branches. To generate a tree of five tips, we add another tip branch to any branch in any of the three subtrees. This process can be repeated until we reach trees of N tips. In general, if there are N_t trees with N tips and $N_b = 2N-3$ branches, then the number of trees with $N+1$ tips is N_bN_t. We also note that in growing trees this way, the number of nodes, *not tips*, in a tree is always $N-2$, that is, two less than the number of tips. This is because we start out with an original tree of $N = 3$ with a single node. Every time we add a branch we also introduce a new node to the tree. The following table shows the relation among N, N_b and N_t, where $(2N-5)!!$ denotes the product $1 \cdot 3 \cdot 5 \cdot \cdots \cdot (2N-5)$.

No. of tips N	3	4	5	6	$\cdots$	n
No. of branches N_b	3	5	7	9	$\cdots$	$2n-3$
No. of trees N_t	1	3	15	105	$\cdots$	$(2n-5)!!$

Table 3.

As shown in Table 3, the number of trees grows faster than N^k for any fixed k. Thus the procedure of growing all trees is not a polynomial-time algorithm [7]. Day [3] has shown that in general the problem is NP-hard.

4.5. Finding the best tree

As pointed out in Introduction, after a tree of N tips is constructed, we need to calculate the distances between tips. In this section, we show how the distances and the best tree can be obtained efficiently using the matrix representation. Consider a problem of $N = 4$ objects. First, we stagger the distances given by the Input Matrix D in a vector

$$d \equiv \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \end{pmatrix} = \begin{pmatrix} D_{12} \\ D_{13} \\ D_{23} \\ D_{14} \\ D_{24} \\ D_{34} \end{pmatrix}$$

of size 6 ($= N(N-1)/2$), where D_{ij} denotes the (i,j)-entry of D, i.e., the given distance between objects i and j. Then we form a 6-by-4 matrix A satisfying:

1. each row of A has exactly two non-zeros;
2. if the kth element d_k of the vector d is D_{ij}, then the two non-zeros in the kth row of A are in the columns i and j. Specifically,

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

Let us consider the tree of four tips in Figure 13.

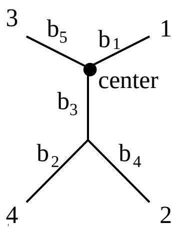


Figure 13. A tree of four tips.

The matrix corresponding to the tree in Figure 13 is

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Using binary (modulo 2) operation, we multiply A and P and get

$$AP = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} b_1 & b_2 & b_3 & b_4 & b_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix} \end{array}.$$

Similar to A, the kth row of AP is associated with the kth element d_k of the vector d. For example, the first row of AP is associated with the first element d_1 $(= D_{12})$ of d. Also, similar to P, the jth column of AP is associated with the branch b_j in the tree. Each row of AP gives the branches connecting a pair of tips in the tree. Let $b = (b_1, b_2, b_3, b_4, b_5)^T$ be a column vector of the branch lengths, then the components of the vector APb are the distances between tips. In this example,

$$APb = \begin{pmatrix} b_1 + b_3 + b_4 \\ b_1 + b_5 \\ b_3 + b_4 + b_5 \\ b_1 + b_2 + b_3 \\ b_2 + b_4 \\ b_2 + b_3 + b_5 \end{pmatrix} = \begin{pmatrix} \hat{D}_{12} \\ \hat{D}_{13} \\ \hat{D}_{23} \\ \hat{D}_{14} \\ \hat{D}_{24} \\ \hat{D}_{34} \end{pmatrix}.$$

Using Figure 13, we can verify that $\hat{D}_{12}$, the distance between tips 1 and 2, is indeed $b_1 + b_3 + b_4$, the first component of the product APb. This shows that the distances can be easily computed by matrix-matrix multiplications. Therefore, we can find the optimal branch lengths b_i $(i = 1, ..., 5)$ by solving the least squares problem $\min_b \|APb - d\|_2$ using the standard matrix techniques. Here we briefly describe the QR decomposition approach to the least squares problem. First, we perform a QR decomposition of AP:

$$AP = A \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where Q is orthogonal, i.e., $Q^T Q = QQ^T = I$, and R is upper triangular. Then we update the right hand side vector d as follows:

$$Q^T d = \tilde{d} = \begin{pmatrix} \tilde{d}_1 \\ \tilde{d}_2 \end{pmatrix},$$

where we partition the updated vector $\tilde{d}$ into $\tilde{d}_1$ whose length is consistent with the size of R and $\tilde{d}_2$. The least squares solution, or the optimal branch length vector, is the solution b for the triangular system $Rb = \tilde{d}_1$. The least squares residual $r = \min_b \|APb - d\|_2$ equals $\|\tilde{d}_2\|_2$. To find the best tree, we generate all trees of N tips in the form of matrix as described before. For each matrix representing a tree, we find the minimum residual r. Then we select the matrix $\hat{P}$ which gives the smallest minimum residual. The best tree can be constructed from the matrix $\hat{P}$ and the optimal branch lengths are given by the least squares solution b minimizing $\|A\hat{P}b - d\|_2$. In summary, we present the matrix method.

Algorithm Matrix

Input Matrix D;

Stagger the distances into a vector d;

Construct the matrix A corresponding to the distances in d;

Generate matrices of all trees of N;

For each matrix P representing a tree,

  Compute AP;

  Find the minimum residual $r = \min_b \|APb - d\|_2$;

  Find the P which gives the smallest minimum residual r;

Construct the best tree from the optimal P;

Assign branch lengths using the least squares solution b;

We note that the above algorithm uses the standard least squares technique and produces the best tree according to Cavalli's criterion [1]. It is not hard to modify it to handle weighted least squares such as Fitch and Margoliash's p.s.d. In particular, we can incorporate a diagonal weighting matrix into to the algorithm.

Example 4.1. Applying this method to the Input Matrix of seven European languages given in Section 1, we get the best fit tree shown in Figure 14.

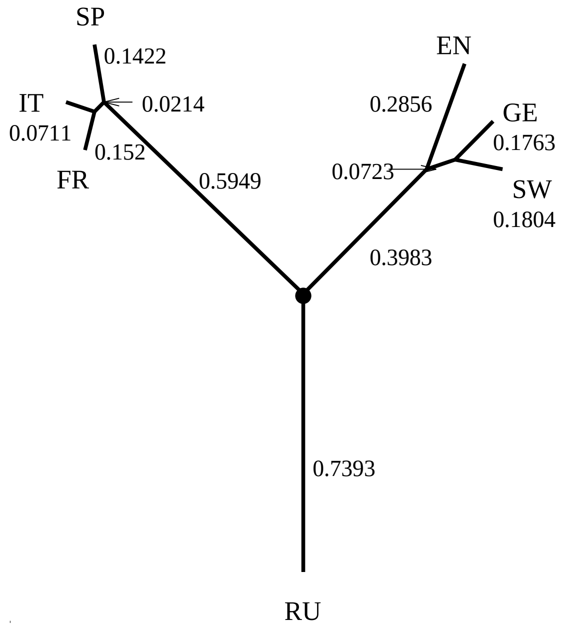


Figure 14. The best tree of seven European languages.

The least squares residuals for the 10 best trees are:

$$0.0463,\ 0.0533,\ 0.0541,\ 0.0936,\ 0.0972,\ 0.0977,\ 0.1088,\ 0.1119,\ 0.1123,\ 0.5684.$$

5. Conclusion

While exhaustive searches of the sort described here are at present difficult to implement when the number of objects is large, this difficulty can be circumvented in the future by exploiting the advantages of parallel computing. Given that the logic of computing implies early branching, it should be relatively easy to divide the task into independent segments and assign the segments to different computers [9].

On the other hand, having the output of the program list the ten best trees (or any number for that matter) gives the investigator a rich body of data upon which to speculate the phylogenetic complexity in the evolution of the objects. In our experiments with language phylogeny, for example, we find that frequently the best several trees in the output contain branches with negative lengths. This could be a consequence of the fact that horizontal transmission [2] plays a very important role in language evolution, more so than in most cases of biological evolution.

Acknowledgement

We thank Joseph Felsenstein for his comments on an earlier draft of this paper. This work was partly supported by grants awarded to W.S-Y. Wang by the City University of Hong Kong and by the Chiang Ching-Kuo Foundation for Scholarly Studies, and a grant awarded to S. Qiao by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Cavalli-Sforsza, L.L. and Edwards, A.W.F. (1967) Phylogenetic Analysis Models and Estimation Procedures. *Evolution*, 32, 550-570 (also published in *Amer. J. Hum. Genet.* 19, 233–257).

[2] Cavalli-Sforsza, L.L. and Wang, W. S-Y. (1986) Spatial Distance and Lexical Replacement. *Language*, 62, 38–55.

[3] Day, W.H.E. (1987) Computational-Complexity of Inferring Phylogenies from Dissimilarity Matrices. *Bulletin of Mathematical Biology*, 49, 46–467.

[4] Dyen, I. et al. (1992) An Indoeuropean Classification. *Transactions of the American Philosophical Society,* 82, Part 5.

[5] Felsenstein, J. An Alternating Least Squares Approach to Inferring Phylogenies from Pairwise Distances, Manuscript.

[6] Fitch, W. and Margoliash, E. (1967) Construction of Phylogenetic Trees. *Science,* 155(3760), 279–284.

[7] Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: a Guide to the Theory of NP-Completeness,* W.H. Freeman, New York.

[8] Olsen, G.J. (1988) Phylogenetic Analysis Using Ribosomal RNA. *Methods Enzymol,* 164:793–873.

[9] Qiao, S. and Tam, S. (1997) A Parallel Implementation of a Matrix Decomposition Method for Evaluatiing Phylogenetic Trees. *The Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97),* Volume II, Editor: H.R. Arabnia, 774–777.

[10] 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.