A Matrix Representation of Phylogenetic Trees

Sanzheng Qiao[1]* and W. S-Y Wang[2]**

[1] Department of Computer Science and Systems, McMaster University, Hamilton, Ontario, Canada L8S 4K1.
[2] Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong.***

Abstract. In this paper we begin by describing two currently used methods for evaluating phylogenetic trees, one proposed by Fitch and Margoliash [5] and the other proposed by Saitou and Nei [7]. Both methods are heuristic in the sense that not all possible trees are tested to ensure that the best solution has been reached. We develop a matrix representation of unrooted binary trees. The problem of evaluating phylogenetic trees is then transformed into the standard linear least squares problem. Then we propose a matrix decomposition method for evaluating phylogenetic trees.

1 Introduction

Suppose we are given N objects, and the distances between all pairs of these objects. We 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.

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.

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.

* This work was partly supported by grants awarded by NSERC Canada.
** This work was partly supported by grants awarded by the City University of Hong Kong and by the Chiang Ching-Kuo Foundation for Scholarly Studies.
*** Also at the University of California at Berkeley, CA, USA.

Now, we describe two currently used methods which solve this problem. The neighbor-joining method proposed by Saitou and Nei [7] constructs the tree by repeatedly joining a pair of objects. At each step, this method selects a pair of objects which are close to each other but far away from the others. Then we merge them into a new node. Consequently, the number of objects is reduced by one. After adjusting the distances, we repeat the procedure until the trivial case of three objects. We refer the details of how to select objects, calculate branch lengths, and adjust the distances to Saitou and Nei [7]. We note that an early decision on pairing objects does not guarantee the final optimal solution.

We describe the Fitch and Margoliash's method based on Felsenstein's implementation [4]. This method starts with a tree of three tips. 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

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

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 is the number of objects. The quantity (1) is called percent standard deviation (p.s.d.) by Fitch and Margoliash [5]. 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 [4] suggests a rearrangement technique to heuristically overcome this problem.

This paper is organized as follows. In Section 2, the matrix representation of trees is illustrated. Section 3 shows how to generate a tree by adding a sequence of branches to the trivial tree, the tree with three tips, and presents techniques for generating trees in the form of matrices and recomposing the tree from a matrix. A matrix decomposition method for evaluating phylogenetic trees is described in Section 4.

2 Representing a Tree by Matrices

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 1 where we have ordered the subtrees clockwise. We then order the branches by traversing the three subtrees clockwise. For each tree, we traverse the branches inorder. That is we traverse recursively the left subtree, the root of the subtree, and then the right subtree. Figure 2 shows an example of an unrooted tree of $N = 6$ tips and its branch labels.

Having labeled the branches, we are ready to represent the tree by an N-by-$(2N-3)$ binary matrix P. We call a matrix binary, if its entries are binary.

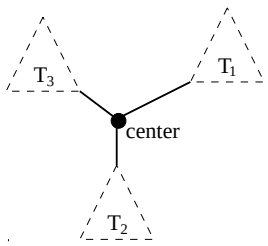

Fig. 1. A tree and its three binary subtrees connected to the center.

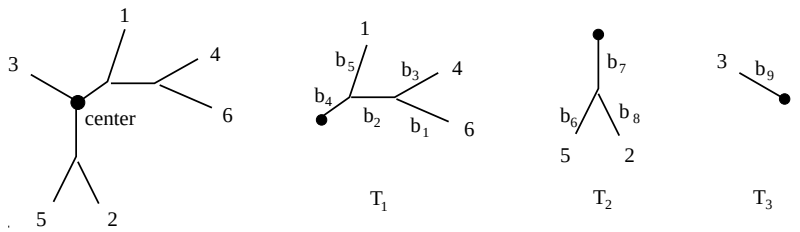

Fig. 2. A tree of six tips and its three subtrees with their branches labeled.

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 2 is

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

To save storage, we propose to extract three submatrices from P to represent the three subtrees. Consider the subtree T_1 in Figure 2, 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 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. Similarly, we extract two submatrices from P to represent subtrees T_2 and T_3. Specifically,

$$P_1 = \begin{array}{c} \\ 1 \\ 4 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} b_1 & b_2 & b_3 & b_4 & b_5 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}, \quad P_2 = \begin{array}{c} \\ 2 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} b_6 & b_7 & b_8 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}, \quad \text{and } P_3 = \begin{array}{c} b_9 \\ 3 \\ \end{array} \begin{array}{c} \\ \begin{pmatrix} 1 \end{pmatrix} \end{array}$$

are the submatrices representing subtrees T_1, T_2 and T_3 respectively. Note that we have bordered the matrices with the tips and branches in their corresponding subtrees.

3 Growing Trees in Matrices

To generate a tree, we start with the trivial tree of three tips and three branches. By adding a new branch to any of the three branches of the three tip tree, we get a tree of four tips. Thus there are three trees with four tips and each has five branches. In general, any tree can be generated by adding a sequence of branches to the trivial tree. In this section, we show how to update the matrix when a new tip branch is added to a tree. Thus, we can generate trees in the form of matrix. We assume that the new branch is always added to the left of an existing branch.

3.1 Adding a New Branch to a Tip Branch

Recalling that we traverse the left branch followed by the root branch and then the right branch. When a branch with a new tip is added to the left of branch b_k of a tree, the new branch is labeled by b_k and the old b_k is broken into branches b_{k+1} and b_{k+2} and all following branches are relabeled accordingly. Figure 3 shows that 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 3.

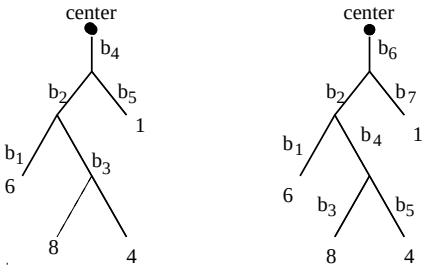

Fig. 3. A new tip branch is added to T_1 on the left and the resulting $\hat{T}_1$ on the right.

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 3 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, we first insert two new columns before column 3 and adding a new

row to the bottom of P_1. Observing the tree $\hat{T}_1$ on the right of Figure 3, the paths from the center to tips 4 and 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 the following matrix where "$\times$" indicates an entry to be determined.

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

Step 2. 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 3. Now it remains to determine the four entries in the above $\hat{P}_1$ marked by "$\times$". The subtree $\hat{T}_1$ on the right of Figure 3 shows that branch b_4 is on the paths from the center to tips 4 and 8 and branch b_3 is a tip branch. Thus we set the entries in column 4 of the rows 4 and 8 to 1 and the last element of column 3 to 1 and the rest of column 3 to zero. Finally, we obtain the following matrix representing the new subtree $\hat{T}_1$ on the right of Figure 3:

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

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. Suppose that we have the subtree T_1 on the left of Figure 4 available and we add a branch with a new tip $n = 7$ to b_6. The figure on the right of Figure 4 is the subtree $\hat{T}_1$ after relabeling the branches.

The matrix representing T_1 is

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

Fig. 4. The original subtree T_1 on the left and the resulting tree $\hat{T}_1$ on the right.

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

Step 1. Allocate all rows whose sixth element is 1 (they are rows 1, 5, and 6) and search for the left most column which has exactly one nonzero in any of the rows just allocated. That is column 3 of P_1. From Figure 4, we can see that this step finds the left-most tip branch under b_6 in T_1.

Step 2. 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 the bottom of P_1. Then we set the 8th element of the last row of $\hat{P}_1$ 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) allocated in Step 1 equals 1. At the end of this step, we have

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

Step 3. We set the entries in columns 3 and 4 of the rows other than those indexed 1, 5, 6, and 7 to zero. Then we set column 4 of the last row and all the rows allocated in Step 1 to 1, because b_4 is on the paths to the new tip and all the tips allocated in Step 1 as shown by $\hat{T}_1$ in Figure 4. 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 4. Now it remains four entries in column 3 to be determined. Since b_3 becomes a new tip branch, we set the last row of column 3 to 1 and the rest of column 3 to zero. Finally, we obtain the following matrix representing the new

tree $\hat{T}_1$ in Figure 4.

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

3.3 Recomposing the Tree From a Matrix

In the previous subsections, we showed how to generate a tree in the form of matrix. In this subsection, we describe the construction of the tree given its corresponding matrices.

Suppose the submatrix representing a subtree T_1 is given by

$$P_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}. \tag{2}$$

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). 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 5.

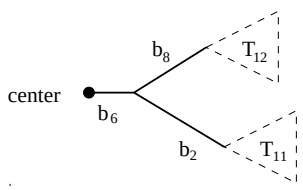

Fig. 5. Two branches b_2 and b_8 are connected to the root branch b_6.

To construct the subtree T_{11} in Figure 5, 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 in (2) and

$$P_{11} = \begin{array}{c} \\ 4 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{3}$$

is the matrix associated with the subtree T_{11} in Figure 5. Recursively applying the above procedure to P_{11} in (3), we get the subtree T_{11} shown in Figure 6. Similarly, we can find the subtree T_{12}. The complete subtree T_1 corresponding to P_1 in (2) is on the right of Figure 6.

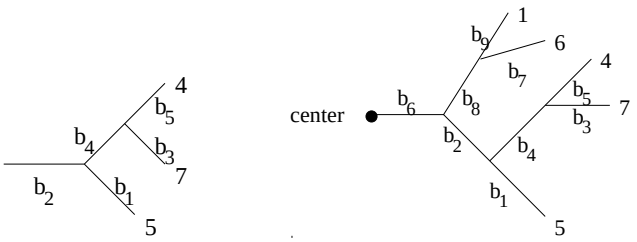

Fig. 6. The subtree T_{11} in Fig. 5 on the left and the complete subtree T_1 on the right.

4 Matrix Decomposition Method

Consider a problem of $N = 4$ objects. First, we stagger the distances given by the Input Matrix D in a vector $d = (D_{12}, D_{13}, D_{23}, D_{14}, D_{24}, D_{34})^T$ 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 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 7. Using binary operation and multiplying A with the matrix P representing the tree in Figure 7, we get

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

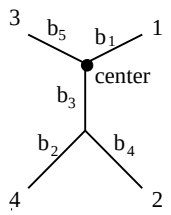

Fig. 7. A tree of four tips.

Each row of AP gives the branches connecting a pair of tips in the tree. For example, there are three non-zeros in the columns 1, 3, and 4 of the first row in AP, which is associated with the first element D_{12} in d. This means that branches b_1, b_3, and b_4 are on the path connecting tips 1 and 2. Thus the distance between tips 1 and 2 in the tree in Figure 7 is $b_1 + b_3 + b_4$. Therefore, we can find the optimal branch lengths b_i $(i = 1, ..., 5)$ by solving the least squares problem $\min_b \|(AP)b - d\|_2$ using the standard matrix techniques, where $b = (b_1, b_2, b_3, b_4, b_5)^T$ is a vector of branch lengths. To find the best tree, we generate all trees of N tips in the form of matrix as described in Section 3. For each matrix representing a tree, we find the minimum residual $r = \min_b \|(AP)b - d\|_2$. 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 $\|A\hat{P}b - d\|_2$.

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.

We note that the method presented above is exhaustive. The number of trees increases exponentially with respect to the number of tips. Thus, finding the best tree by exhaustively inspecting all trees is not a polynomial-time algorithm [6]. Day [3] has shown that in general the problem is NP-hard. However, in applications such as language phylogeny where the number of objects (languages) is small, an exhaustive method can give us rich body of data for investigation.

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.

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.

References

1. Cavalli-Sforsza, L.L. and A.W.F. Edwards. 1967. Phylogenetic Analysis Models and Estimation Procedures. *Evolution*, 32, pp.550-570 (also published in *Amer. J. Hum. Genet.* 19, pp.233-257).
2. Cavalli-Sforsza, L.L. and W. S-Y. Wang. 1986. Spatial Distance and Lexical Replacement. *Language*, 62, pp.38-55.
3. Day, W.H.E. 1987. Computational-Complexity of Inferring Phylogenies from Dissimilarity Matrices. *Bulletin of Mathematical Biology*, 49, pp.46-467.
4. Felsenstein, J. An Alternating Least Squares Approach to Inferring Phylogenies from Pairwise Distances, Manuscript.
5. Fitch, W. and E. Margoliash. 1967. Construction of Phylogenetic Trees. *Science,* Vol. 155, No. 3760, pp.279-284.
6. Garey, M.R. and D.S. Johnson. 1979. *Computers and Intractability: a Guide to the Theory of NP-Completeness*, W.H. Freeman, New York.
7. Saitou, N. and M. Nei. 1987. The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees. *Molecular Biology and Evolution,* 4(4), pp.406-425.

This article was processed using the LaTeX macro package with LLNCS style