Approximation Algorithms for Constructing Evolutionary Trees *

Chia-Mao Huang and Chang-Biau Yang
Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw

Abstract

In this paper, we shall propose heuristic algorithms to construct evolutionary trees under the distance base model. When the distance matrix is metric, the problem is called the triangle minimum ultrametric tree problem (△MUT). For the △MUT, we shall propose an approximation algorithm, with error ratio $\leq \lceil \log_\alpha n \rceil + 1 \cong 1.44\lceil \log n \rceil + 1$, where $\alpha = \frac{\sqrt{5}+1}{2}$. We shall also propose a heuristic algorithm to obtain a good leaf node circular order. The heuristic algorithm is based on the clustering scheme. And then we shall design a dynamic programming algorithm to construct the optimal ultrametric tree under some fixed leaf node circular order. The time complexity of the dynamic programming is $O(n^3)$, if the scoring function is the minimum tree size or L^1-min increment.

Key words: computational biology, evolutionary tree, approximation algorithm, dynamic programming

1 Introduction

An evolutionary tree is an important tool to show branching diagrams and the history of life. And now, we can obtain the DNA (Deoxyribonucleic Acid) sequence from the organisms. The DNA sequence is the most original information of one life. Thus, we shall use the DNA information to infer an evolutionary tree close to the real evolutionary process [15].

Many researchers have studied the construction of evolutionary trees [1, 4, 5, 7]. However, we are not sure what the real evolutionary process is. So, many various construction models for evolutionary trees have been proposed. There are also many various scoring functions to evaluate an evolutionary tree. Most evolutionary tree optimization problems are NP-hard [2, 3, 16], except some very special scoring functions with some special input data [6].

In this paper, we shall use the *distance base model* to construct the evolutionary tree. The model is based on the computing results of the distances between species. First, we use the DNA sequence to compute the distance between every pair of species. Then, we construct the evolutionary tree from these distance data. For examples, the neighbor joining (NJ) method [13] and the unweighted pair group method with maximum (UPGMM) [11] are often used to construct the evolutionary tree from the distance data. The evolution of organisms will change their DNA sequences, and these evolution events can be viewed as insertion, deletion, point mutation, rearrangement or inversion of DNA sequences. Thus, we can compute the number of event occurrences, and accordingly calculate the distance between two species [10, 12, 14]. In this model, the distance between any two species in the evolutionary tree is similar to the original distance.

The organization of this paper is as follows. In Section 2, we shall first present some definitions about the evolutionary tree problem. In Section 3, we shall define the binary splitting tree problem and propose an algorithm to construct a binary splitting tree with height no more than $\lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$ and n is number of leaf nodes. In Section 4, we shall propose an approximation algorithm to construct the minimum ultrametric tree under the metric distance matrix and prove that the error ratio is within $\lceil \log_\alpha n \rceil + 1 \cong 1.44\lceil \log n \rceil + 1$, where $\alpha = \frac{\sqrt{5}+1}{2}$. And in Section 5, we shall propose a heuristic algorithm to construct a leaf node circular order, and also design a dynamic programming algorithm to solve the optimal ultrametric tree problem under a certain

*This research work was partially supported by the National Science Council of the Republic of China under contract NSC-90-2213-E-110-043.

leaf node circular order. Then, in Section 6, we show our experiment results and compare them with the UPGMM method. Finally, we shall give some conclusions in Section 7.

2 Preliminaries

In this section, we shall give some definitions about the evolutionary tree and various scoring functions to estimate the goodness of an evolutionary tree.

Definition 1 *An $n \times n$ distance matrix M is used to represent the distances among n species, where d_{ij}^M denotes the distance between the ith species and the jth specie. Moreover, M is a symmetric matrix, that is, $d_{ij}^M = d_{ji}^M$.*

Definition 2 *An $n \times n$ distance matrix M is metric if the distances among any three points satisfy the triangle inequality, that is, for any three points x,y,z, $d_{xy}^M \le d_{xz}^M + d_{yz}^M$.*

Definition 3 *An $n \times n$ metric M is ultrametric if and only if for any three points i,j,k, $d_{ij}^M \le \max\{d_{ik}^M, d_{jk}^M\}$. In other words, for any triangle, the two longer sides have the same length.*

In the following, $d_{i,j}^T$ is used to denote the distance between the ith species and the jth species in an evolutionary tree T, and $w(T)$ denotes the total weight assigned to the tree edges.

Definition 4 *Given a set S of species, the set of leaves in an evolutionary tree is equal to S. And in the tree, each internal node represents the common ancestor of the species on the leaves of the subtree.*

In a rooted evolutionary tree, each internal node has exactly two children. And, in an unrooted tree, the degree of each internal node is exactly 3.

Fact 1 *[6] Given an ultrametric matrix M, there exists a unique rooted evolutionary tree, called an ultrametric tree, T such that $d_{i,j}^T = d_{i,j}^M$. In addition, for any internal node v, the distances from v to all leaf nodes in the subtree rooted at v are the same.*

Definition 5 *Given an arbitrary distance matrix M, the MUT (minimum ultrametric tree) problem is to construct an ultrametric tree T such that the total weight assigned to the tree edges is minimized and $d_{i,j}^T \ge d_{i,j}^M, \forall i,j$.*

Definition 6 *Given a metric distance matrix M, the △MUT (minimum ultrametric tree for a metric) problem is to construct an ultrametric tree T such that the total weight assigned to the tree edges is minimized and $d_{i,j}^T \ge d_{i,j}^M, \forall i,j$.*

Definition 7 *[19] Given an ultrametric distance matrix M associated with a tree topology T, the MUTT (minimum ultrametric tree with a given topology) problem is to assign each tree edge a weight such that the total weight is minimized and $d_{i,j}^T \ge d_{i,j}^M, \forall i,j$.*

Theorem 1 *[19] The MUTT problem can be solved in $O(n^2)$ time, where n is the number of species.*

There are also many various scoring functions to estimate the goodness of an evolutionary tree. Given an $n \times n$ distance matrix M, several popular scoring functions [6] for measuring an evolutionary tree T are as follows.

- **minimum tree size:** $d_{i,j}^T \ge d_{i,j}^M, \forall i,j$ and the total weight of the tree is minimized. The MUT, △MUT and MUTT problems are based on this scoring function.
- **L^1-min increment:** $d_{i,j}^T \ge d_{i,j}^M, \forall i,j$ and $\sum_{i,j}(d_{i,j}^T - d_{i,j}^M)$ is minimized.
- **L^k-min increment:** $d_{i,j}^T \ge d_{i,j}^M, \forall i,j$ and $\sum_{i,j}(d_{i,j}^T - d_{i,j}^M)^k$ is minimized.
- **L^∞-min increment:** $d_{i,j}^T \ge d_{i,j}^M, \forall i,j$ and $\max_{i,j}(d_{i,j}^T - d_{i,j}^M)$ is minimized.

Because the evolution of organisms repeatedly changes their DNA sequence, the distance may be shorter than the real distance of evolution. Thus, we usually use minimum tree size, L^1-min increment, L^k-min increment and L^∞-min increment scoring functions for measuring an evolutionary tree [17].

3 The Binary Splitting Tree Problem

For solving the △MUT problem, we first have to solve the binary splitting tree problem. In this section, we shall first define the binary splitting tree problem and then propose an algorithm to construct a binary splitting tree with height no more than $\lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$ and n is number of leaf nodes.

Definition 8 *Given a tree $T = (V,E)$, for any two nodes $v_1, v_2 \in V$, the path connecting v_1 and v_2 is denoted as $path_T(v_1,v_2)$. And $E(path_T(v_1,v_2))$ denotes the set of edges contained in $path_T(v_1,v_2)$.*

Definition 9 *Given an unrooted tree $T = (V,E)$, the binary splitting tree $\tau = (V,V_\tau,E_\tau)$ is a rooted binary tree such that and V and V_τ are the set of the leaf nodes and the set of internal nodes, respectively, in τ, E_τ denotes the set of tree edges in τ, and for any two nodes $v_1, v_2 \in V_L$ and any two nodes $v_3, v_4 \in V_R$, where V_L and V_R contain the leaf nodes in the left and right subtrees rooted at node $u \in \tau$, respectively, $E(path_T(v_1,v_2)) \cap E(path_T(v_3,v_4)) = \phi$.*

Definition 10 *For given an unrooted tree T, the binary splitting tree problem is to find a binary splitting tree.*

For example, consider Figure 1. Figure 1 (a) shows an unrooted tree T, and Figure 1 (b) and (c) show two binary splitting trees of T. In Figure 1 (b), the binary splitting tree is built easily. Nodes v_1 and v_2 are connected by edge e_1, nodes v_4, v_3 and v_5 are connected, by edges e_3 and e_4, and $\{e_1\} \cap \{e_3,e_4\} = \phi$. It is similar in the subtree $\{v_4,v_3,v_5\}$. However, Figure 1 (c) is more complicated. $\{v_2,v_1,v_3\}$ are connected by edges $\{e_1,e_2\}$, and $\{v_4,v_5\}$ are connected by edges $\{e_3,e_4\}$. It is clear that $\{e_1,e_2\} \cap \{e_3,e_4\} = \phi$. $\{v_2\}$ and $\{v_1,v_3\}$ have the similar situation. Thus, the tree is a binary splitting tree. Figure 1 (d) is not a binary splitting tree, because edges $\{e_1,e_2,e_3\}$ connect nodes $\{v_1,v_4\}$, and $\{e_2,e_4\}$ connect nodes $\{v_2,v_3,v_5\}$, and $\{e_1,e_2,e_3\} \cap \{e_2,e_4\} \neq \phi$.

Next, we shall propose an algorithm to construct the binary splitting tree and show that the height of the tree is no more than $\lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$ and n is the number of leaf nodes.

Theorem 2 *[18] For any tree $T = (V,E)$, there exists a node $v \in V$ such that the T can be split from v into k, $k \geq 2$, subtrees and the number of nodes in any subtree is no more than $\frac{1}{2}|V|$.*

Definition 11 *An unrooted tree is a k-way tree if the degree of each node is no more than k.*

Before constructing a binary splitting tree from an unrooted (k-way) tree, we have to convert the k-way tree, $k \geq 4$, to a 3-way tree by adding some *virtual nodes*.

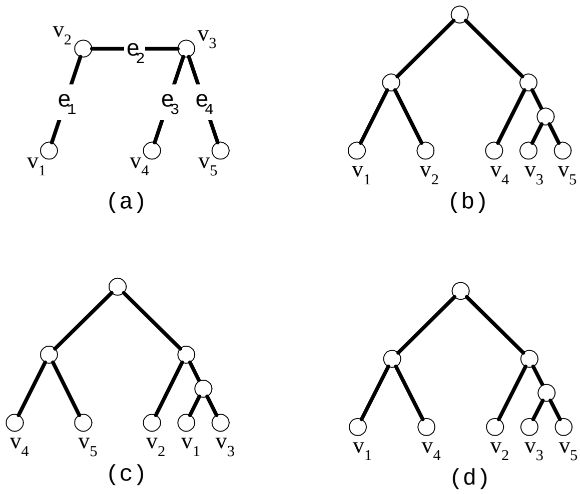

Figure 1: An example for the binary splitting tree. (a) A tree. (b) A binary splitting tree. (c) Another binary splitting tree. (d) Not a binary splitting tree.

Definition 12 *Given a k-way tree $T = (V,E)$, its corresponding 3-way tree $T\prime = (V,V\prime,E\prime)$ is defined as follows. Let $U = \{u_i | u_i \in V, degree(u_i) \geq 4\}$. Suppose $|U| = h$. Let $c_i = degree(u_i)$, and v_{i1}, $v_{i2}, \cdots, v_{i,c_i}$ be adjacent to u_i. $V_i = \{v\prime_{ij} | 3 \leq j \leq c_i - 1\}$, $1 \leq i \leq h$. $E_i = E - \{(u_1,v_{i1}),(u_i,v_{i2})\} \cup \{(u_i,v\prime_{i3}),\ (v\prime_{i3},v_{i3}),\ (v\prime_{i3},v\prime_{i4}),\ (v\prime_{i4},v_{i4}),\ \cdots,\ (v\prime_{i,c_{i-2}},v\prime_{i,c_{i-1}}),\ (v\prime_{i,c_{i-1}},v_{i,c_{i-1}}),\ (v\prime_{i,c_{i-1}},v_{i,c_i})\ \}$, $1 \leq i \leq h$. Then $V\prime = \bigcup_{1 \leq i \leq h} V_i$ and $E\prime = \bigcup_{1 \leq i \leq n} E_i$. The nodes in $V\prime$ are called virtual nodes.*

For example, Figure 2 (a) shows a k-way tree, $k = 4$. By Definition 12, $U = \{g\}$, $V_1 = \{v\prime_{13}\} = \{w\}$. $E_1 = E - \{(g,c),(g,d)\} \cup \{(g,w),(w,c),(w,d)\}$, $V\prime = V_1$ and $E\prime = E_1$, as shown in Figure 2 (b).

Theorem 3 *Given a k-way tree, $k \geq 4$, a 3-way tree $T\prime = (V,V\prime,E\prime)$ can be constructed such that there exists a node $v \in V$ or $v \in V\prime$ to split $T\prime$ into p subtrees, $p = 2$ or $p = 3$, and the number of nonvirtual nodes in any subtree is no more than $\lceil \frac{1}{2}|V| \rceil$.*

Theorem 3 is based on Theorem 2. The only difference between Theorem 2 and Theorem 3 is the latter includes the concept of virtual nodes. Note that in Theorem 3, the splitting node v is included in one of the subtrees.

Our algorithm for constructing a binary splitting tree with height no more than $\lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$, is as follows.

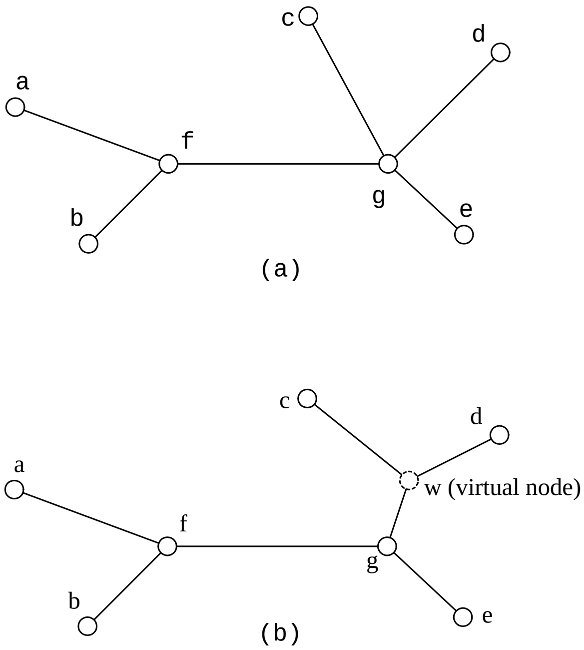

Figure 2: Conversion from a 4-way tree to a 3-way tree. (a) A 4-way tree. (b) A 3-way tree.

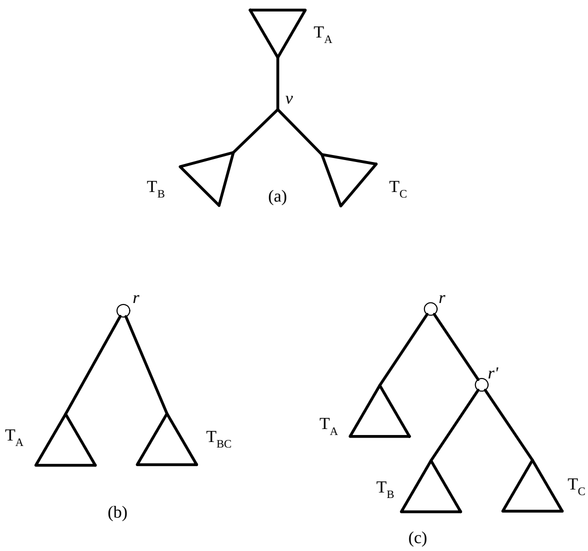

Figure 3: The binary splitting tree.

Algorithm BST (Binary Splitting Tree)

Input: An unrooted tree $T = (V, E)$, $|V| = n$.

Output: A binary splitting tree $\tau = (V, V_\tau, E_\tau)$ with height no more than $\lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$.

Step 1: Convert T to a 3-way tree $T' = (V, V', E')$.

Step 2: If $|V| = 1$, B contains only one node $v \in V$, $V_\tau = \phi$ and $E_\tau = \phi$, and stop.

Step 3: By Theorem 3, find node $v \in V$ to split T' into 3-way subtrees $T_A = (V_A, V'_A, E'_A)$, $T_B = (V_B, V'_B, E_B)$ and $T_C = (V_C, V'_C, E_C)$ such that $|V_C| \le |V_B| \le |V_A| \le \frac{1}{2}|V|$.

Step 4: If $|V_A| \ge \frac{3-\sqrt{5}}{2}|V|$, combine T_B and T_C into T_{BC}. In other words, split T into two subtrees T_A and T_{BC}. If $|V_A| < \frac{3-\sqrt{5}}{2}|V|$, go to Step 7.

Step 5: Let T_A and T_{BC} be T. Recursively apply this algorithm and obtain binary splitting trees $\tau_A = (V_A, V_{\tau_A}, E_{\tau_A})$ and $\tau_{BC} = (V_{BC}, V_{\tau_{BC}}, E_{\tau_{BC}})$, respectively.

Step 6: Create a root r, build a binary splitting tree τ rooted at r with the left and right subtree being τ_A and τ_{BC}, respectively, as shown in Figure 3 (b). In other words, $\tau = (V, V_\tau, E_\tau)$, where $V = V_A \cup V_{BC}$, $V_\tau = V_{\tau_A} \cap V_{\tau_{BC}} \cap \{r\}$ and $E_\tau = E_{\tau_A} \cap E_{\tau_{BC}} \cap \{(r, \text{root of } \tau_A), (r, \text{root of} \tau_{BC})\}$.
Stop.

Step 7: If $|V_A| < \frac{3-\sqrt{5}}{2}|V|$, keep the splitting done in Step 3. In other words, split T into three subtrees T_A, T_B and T_C.

Step 8: Let T_A, T_B and T_C be T. Recursively apply this algorithm and obtain binary splitting $\tau_A = (V_A, V_{\tau_A}, E_{\tau_A})$, $\tau_B = (V_B, V_{\tau_B}, E_{\tau_B})$ and $\tau_C = (V_C, V_{\tau_C}, E_{\tau_C})$, respectively.

Step 9: Create a root r and subroot r', build a binary splitting tree τ rooted at r, as shown in Figure 3 (c). Precisely, $\tau = (V, V_\tau, E_\tau)$, where $V = V_A \cup V_B \cup V_C$, $V_\tau = V_{\tau_A} \cap V_{\tau_B} \cap V_{\tau_C} \cap \{r, r'\}$ and $E_\tau = E_{\tau_A} \cap E_{\tau_B} \cap E_{\tau_C} \cap \{(r, \text{root of } \tau_A), (r, r'), (r', \text{root of } \tau_B), (r', \text{root of } \tau_C)\}$.
Stop.

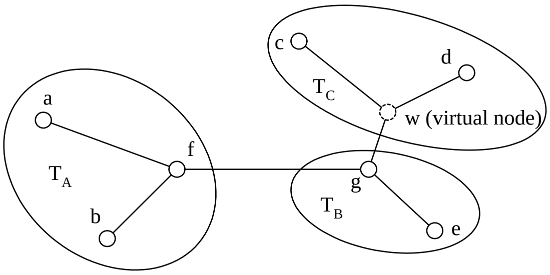

Figure 4: Splitting a tree to three subtrees.

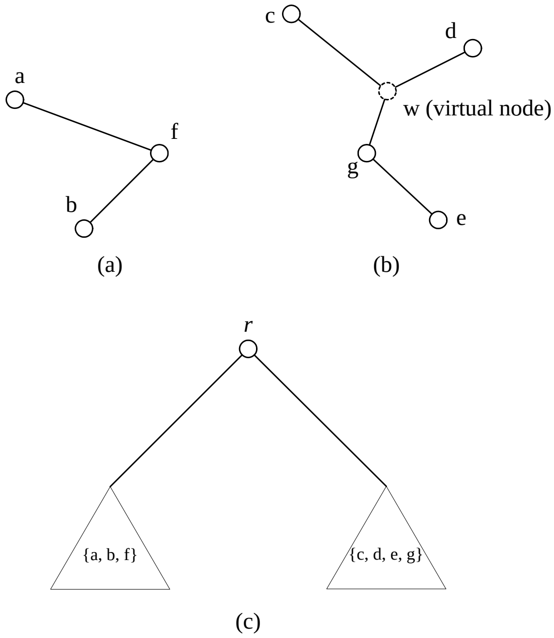

Figure 5: Splitting the tree from node g. (a) Subtree $T_A = \{a,b,f\}$. (b) Subtree $T_{BC} = \{c,d,e,g\}$. (c) The merging procedure.

For example, Figure 2 (a) shows a 4-way tree, in which there are four nodes adjacent to node g. We first convert the tree into a 3-way tree, as shown in Figure 2 (b). Then we further split the 3-way tree from g into three subtrees $T_A = \{a,b,f\}$, $T_B = \{g,e\}$ and $T_C = \{c,d,v\}$, as shown in Figure 4. Since $|V_A| \geq \frac{3-\sqrt{5}}{2}$, the two smaller trees T_B and T_C are merged. Thus, the tree is finally split into two subtrees $T_A = \{a,b,f\}$, $T_{BC} = \{c,d,e,g\}$, as shown in Figure 5 (a) and Figure 5 (b). The merging procedure done in Step 6 is shown in Figure 5 (c). Finally, by Algorithm BST, a binary splitting tree can be constructed, as shown in Figure 6.

Theorem 4 *Given an unrooted tree of n nodes, Algorithm BST constructs a binary splitting tree with height no more than* $\lceil \log_\alpha n \rceil$, *where* $\alpha = \frac{\sqrt{5}+1}{2}$.

Proof: Given an unrooted tree $T = (V,E)$ and $|V| = n$, let $\pi(n)$ denote the number of levels required for splitting the corresponding 3-way tree $T\prime$. By Theorem 3, we can split $T\prime$ into three subtrees $T_A = (V_A, V\prime_A, E_A)$, $T_B = (V_B, V\prime_B, E_B)$ and $T_C = (V_C, V\prime_C, E_C)$ such that $|V_C| \leq |V_B| \leq |V_A| \leq \frac{1}{2}|V|$.

The possible relations between $|V_A|$ and $|V| = n$ are as follows. It is assumed that x is an unknown constant.

Case 1: $nx \leq |V_A| \leq \frac{1}{2}n$.
We combine T_B with T_C to get $T_{BC} = (V_{BC}, E_{BC})$. It is clear that $|V_{BC}| \geq |V_A|$ and $|V_{BC}| = n - |V_A| \leq (1-x)n$. So we split $T\prime$ into T_A and T_{BC} with one level. At the next recursion level, the number of leaf nodes is reduced from n at the current level to no more than $(1-x)n$.

Case 2: $\frac{1}{3}n \leq |V_A| \leq nx$.
Because $|V_B| \leq |V_A|$ and $|V_C| \leq |V_A|$, it

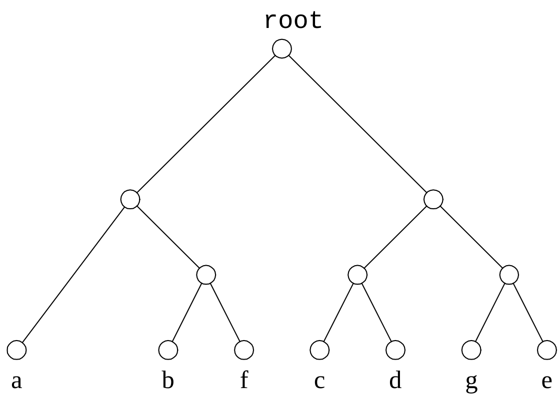

Figure 6: A binary splitting tree constructed from the 4-way tree in Figure 2 (a).

is obvious that $|V_B| \le nx$ and $|V_C| \le nx$. Thus, at the first level, we split T into T_A and $T_B + T_C$. At the next level, we split $T_B + T_C$ into T_B and T_C. Hence, the number of leaf nodes is reduced from n to no more than nx with two levels.

By Case 1 and Case 2, $\pi(n) = \max\{\pi((1-x)n)+1, \pi(xn)+2)\}$.

We claim that if $x = \frac{3-\sqrt{5}}{2}$, then $\pi(n) \le \lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$. We shall prove this claim by induction.

It is clear that $\pi(1) = 0$.

By induction hypothesis, suppose $\pi(k) \le \lceil \log_\alpha k \rceil$, $\forall k < n$.

Let $x = \frac{3-\sqrt{5}}{2}$. It is clear that $\alpha = \frac{1}{1-x}$. We have

$$\begin{aligned}
\pi(n) &= \max\{\pi((1-x)n)+1, \pi(xn)+2)\} \\
&\le \max\{\log_\alpha(1-x)n+1, \log_\alpha xn+2\} \\
&= \max\{\log_\alpha n + \log_\alpha(1-x)+1, \\
&\quad \log_\alpha n + \log_\alpha x + 2\} \\
&= \max\{\log_\alpha n, \log_\alpha n\} \\
&= \log_\alpha n
\end{aligned}$$

Thus, the proof is complete.

□

4 An Approximation Algorithm for △MUT

In this section, we shall propose an approximation algorithm for solving the △MUT problem. Our approximation algorithm uses the minimum spanning tree as the backbone to construct a rooted ultrametric tree under the minimum tree size scoring function with error ratio $\varepsilon \le \lceil \log_\alpha n \rceil + 1$, where $\alpha = \frac{\sqrt{5}+1}{2}$.

Algorithm APP-ULTRA (Approximate Ultrametric Tree)

Input: An $n \times n$ metric distance matrix M.

Output: An ultrametric tree under the minimum tree size scoring function with error ratio $\varepsilon \le \lceil \log_\alpha n \rceil + 1$, where $\alpha = \frac{\sqrt{5}+1}{2}$.

Step 1: Find the minimum spanning tree (MST) T for the distance matrix M.

Step 2: Apply Algorithm BST to construct a binary splitting tree B with input T.

Step 3: Given tree topology B, solve the MUTT problem [19] to construct a weighted evolutionary tree. The tree is the output.

Before giving the following lemma and theorem, we need define some notations. When we are given a metric distance matrix M, we use some notation as follows:

- OPT_{MUT}: the total weight of the optimal solution of the minimum ultrametric tree problem.
- APP_{MUT}: the total weight of the approximation solution obtained from Algorithm APP-ULTRA.
- OPT_{MST}: the total weight of the optimal solution of the minimum spanning tree problem.
- OPT_{TSP}: the total weight of the optimal solution of the traveling salesperson problem.

We shall use the MST (Minimum Spanning Tree) to prove the error ratio of our algorithm.

Lemma 1 $OPT_{MST} \le OPT_{TSP} \le 2OPT_{MUT}$.

$OPT_{MST} \le OPT_{TSP}$ is a clear fact and $OPT_{TSP} \le 2OPT_{MUT}$ has also been proved [8, 19].

Theorem 5 *Given an $n \times n$ metric distance matrix, Algorithm APP-ULTRA builds an ultrametric tree and $APP_{MUT} \le (\lceil \log_\alpha n \rceil + 1)\, OPT_{MUT}$, where $\alpha = \frac{\sqrt{5}+1}{2}$.*

Proof:

The labels of nodes and edges in the ultrametric tree constructed by Algorithm APP-ULTRA are shown in Figure 7. The cost of edge $e_{i,j}$ is denoted as $c_{i,j}$, and the height of node $v_{i,j}$, which is the length from $v_{i,j}$ to any leaf node in the subtree rooted at $v_{i,j}$, is denoted as $Height(v_{i,j})$. Let k denote the number of levels in the tree. By Theorem 4, $k \le \lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$.

Then we have the following inequalities.

$$OPT_{MST} \ge 2Height(v_{1,1}) = \sum_{i=1}^{k}\{c_{i,1} + c_{i,2^{i-1}+1}\} \quad (1)$$

$$\frac{1}{2}OPT_{MST} \ge \sum_{j=1}^{2} Height(v_{2,j}) = \sum_{j=1}^{2}\{c_{2,2j} + \sum_{i=2+1}^{k} c_{i,2^{i-2}+1+(j-1)2^{(i-1)}}\} \quad (2)$$

$$\frac{1}{2}OPT_{MST} \ge \sum_{j=1}^{4} Height(v_{3,j}) = \sum_{j=1}^{4}\{c_{3,2j} + \sum_{i=3+1}^{k} c_{i,2^{i-3}+1+(j-1)2^{(i-2)}}\}$$

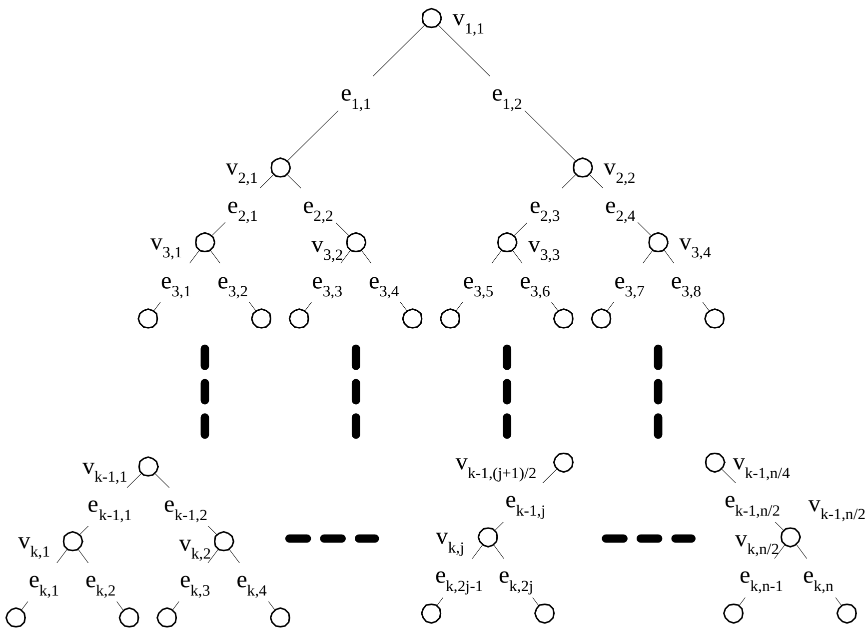

Figure 7: The ultrametric tree with APP-ULTRA.

$$\vdots$$

$$\frac{1}{2} OPT_{MST} \geq \sum_{j=1}^{2^{m-1}} Height(v_{m,j}) = \sum_{j=1}^{2^{m-1}} \{c_{m,2j} + \sum_{i=m+1}^{k} c_{i,2^{i-m}+1+(j-1)2^{(i-m-1)}}\}$$

$$\vdots$$

$$\frac{1}{2} OPT_{MST} \geq \sum_{j=1}^{2^{k-1}} Height(v_{k,j}) = \sum_{j=1}^{2^{k-1}} \{c_{k,2j} + \sum_{i=k+1}^{k} c_{i,2^{i-k}+1+(j-1)2^{(i-k-1)}}\}$$

Let $S_{i,j}$ denote the set of leaf nodes in the subtree rooted at $v_{i,j}$. And let $OPT_{MST_{i,j}}$ denote the total weight of the minimum spanning tree for $S_{i,j}$. We have $Height(v_{i,j}) = \frac{1}{2}\max_{s_1,s_2\in S}\{d(s_1,s_2)\} \leq \frac{1}{2}OPT_{MST_{i,j}}$. Thus, Equation 1 holds. And, because the ultrametric tree is a binary splitting tree (Definition 9), $S_{2,1}$ and $S_{2,2}$ are constructed to two subtrees without edge repetition. $OPT_{MST^{v_{2,1}}} + OPT_{MST^{v_{2,2}}} \leq OPT_{MST}$, so $Height(v_{2,1}) + Height(v_{2,2}) \leq \frac{1}{2}OPT_{MST}$ in Equation 2.

$$\begin{aligned} APP_{MUT} &\leq OPT_{MST} + (\frac{k-1}{2})\,OPT_{MST} \\ &= (\frac{k+1}{2})\,OPT_{MST} \\ &\leq (k+1)\,OPT_{MUT} \\ &= (\lceil\log_\alpha n\rceil + 1)\,OPT_{MUT} \end{aligned}$$

Thus, the approximation algorithm has error ratio $\varepsilon \leq \lceil\log_\alpha n\rceil + 1$, where $\alpha = \frac{\sqrt{5}+1}{2}$. □

For the $\triangle$MUT, there is a previous approximation algorithm [19], with error ratio $\leq 1.5(\lceil\log n\rceil + 1)$. And our algorithm APP-ULTRA, with error ratio $\leq \lceil\log_\alpha n\rceil + 1 \cong 1.44\lceil\log n\rceil + 1$, where $\alpha = \frac{\sqrt{5}+1}{2}$, has a better approximation ratio.

5 A Heuristic Algorithm for MUT

For an evolutionary tree with n different leaves, the order of the leaves from the left to the right is called the *leaf node circular order* [8]. For n different leaves, there are $N(n) = 3*5*\cdots*(2n-5) = \prod_{k=1}^{n-3}(2k+1)$ different unweighted unrooted evolutionary trees [8]. And there are $N(n) = (2n-3)\prod_{k=1}^{n-3}(2k+1)$ unweighted rooted evolutionary trees [9]. However, given a leaf node circular order, only $((2(n-2))!/((n-2)!(n-1)!)$ different unweighted unrooted evolutionary trees can be built with that order and $(2n-3)((2(n-2))!/((n-2)!(n-1)!)$ different unweighted rooted evolutionary trees can be built. Some possible numbers are shown in Table 1.

The number of unweighted rooted evolutionary trees without any circular order grows every fast, so it is very difficult to solve the evolutionary tree optimization problem. In this section, we shall first propose a heuristic algorithm to get a good leaf node circular order. Then, we shall present an algorithm with dynamic programming to construct the optimal ultrametric tree under a certain circular order. The concatenation of these two phases is our heuristic algorithm for solving the MUT (Minimum Ultrametric Tree) problem.

Our heuristic algorithm to obtain a leaf node circular order for given an $n \times n$ distance matrix is follows.

Algorithm Circular-Order

Input: A set S of n species and its distance matrix M.

Output: A node circular order $L = (v_1, v_2, \cdots, v_n)$.

Step 1: Create a new virtual node v_0 and $d^M_{v_0,v_i} = \infty$ for all $v_i \in S$.

Step 2: Find $d(uv) = \max_{i,j \in S}\{d^M_{ij}\}$. Set $L = (v_0, v_1, v_2, v_3)$, where $v_3 = v_0$, $v_1 = u$ and $v_2 = v$. Set $k = 2$. Remove u and v from S

Step 3: Find a node $w \in S$ such that $d^M_{w,v_i} = min_{x \in S, v_j \in L}\{d^M_{x,v_j}\}$.

Step 4: If $d^M_{w,v_{i-1}} \leq d^M_{w,v_{i+1}}$, insert w prior to v_i into L, that is, $L = (v_0, v_1, \cdots, v_{i-1}, w, v_i, \cdots, v_k, v_{k+1})$; otherwise, insert w posterior to v_i, that is, $L = (v_0, v_1, \cdots, v_i, w, v_{i+1}, \cdots, v_k, v_{k+1})$. Reindex L as $(v_0, v_1, \cdots, v_i, \cdots, v_{k+2})$. Set $k = k+1$. Remove w from S.

Step 5: Repeat Step 3 and Step 4, until S becomes empty.

Step 6: Delete v_0 and v_{n+1} from L, and obtain $L = (v_1, v_2, \cdots, v_n)$

In the following, we shall propose a dynamic programming method to construct the optimal ultrametric tree for a certain fixed leaf node circular order. Our algorithm can work on the minimum tree size and L^1-min increment scoring functions and its time complexity is $O(n^3)$. The algorithm is as follows.

Algorithm OPT-ULTRA

Input: An $n \times n$ distance matrix M with its node circular order $(v_1, v_2, \cdots, v_n)$.

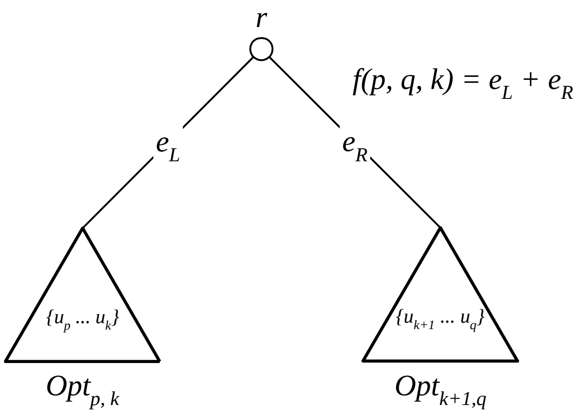

Figure 8: $f(p,q,k)$ for the minimum tree size scoring function.

Output: An optimal ultrametric tree T with respect to the node circular order.

Step 1: Set a sequence $S = (v_1, v_2, \cdots, v_n, v_1, v_2, \cdots, v_{n-1})$. Reindex S as $(u_1, u_2, \cdots, u_n, u_{n+1}, \cdots, u_{2n-1})$.

Step 2: Set $Opt_{i,i} = 0$, $1 \leq i \leq 2n-1$. Set $Opt_{i,i+1}$ = distance of u_i and u_{i+1}, where $1 \leq i \leq 2n-2$.

Step 3: Compute

$$Opt_{i,j} = \min_{i \leq k \leq j-1}\{Opt_{i,k} + Opt_{k+1,j} + f(i,j,k)\}, for 1 \leq i < j \leq 2n-1, 2 \leq j-i \leq n-1$$

where $f(i,j,k)$ is a predefined scoring function.

Step 4: Find the minimum of $Opt_{i,j}$, where $j - i = n-1$, as the cost of the optimal ultrametric tree.

Step 5: Construct the ultrametric tree with the information when we determine the value of $Opt_{i,j}$.

In the following, we shall show how to calculate the predefined scoring function $f(p,q,k)$.

Based on the measurement of the minimum ultrametric tree size, the scoring function f in the above algorithm can be defined as follows:

$$f(p,q,k) = \max_{p \leq i \leq q, p \leq j \leq q}\{d^M_{ij}\} - \left(\max_{p \leq i \leq k, p \leq j \leq k}\{d^M_{ij}\} + \max_{k+1 \leq i \leq q, k+1 \leq j \leq q}\{d^M_{ij}\}\right)/2$$

	without circular order	with circular order
4	15	10
5	105	98
10	34459425	24310
20	8.2×10^{21}	17672631900
n	$(2n-3)\prod_{k=1}^{n-3}(2k+1)$	$(2n-3)((2(n-2))!/((n-2)!(n-1)!)$

Table 1: Number of possible ultrametric trees that can be built without and with a circular order.

where $d^M_{i,j}$ denotes the distance between u_i and u_j. In fact, $f = (p,q,k)$ represents the cost of the root to the subroots of the left subtree $(u_p \cdots u_k)$ and the right subtree $(u_{k+1} \cdots u_q)$, as shown in Figure 8.

And based on the measurement of the L^1-min increment, the scoring function f in the above algorithm can be defined as follows:

$$\begin{aligned} & f(p,q,k) \\ &= \sum_{p\le i\le k, k+1\le j\le q} \left(\max_{p\le i\le q, p\le j\le q} \{d^M_{ij}\} - d^M_{ij} \right) \\ &= (q-k)(k-p+1) \max_{p\le i\le q, p\le j\le q} \{d^M_{ij}\} - \\ &\quad \sum_{p\le i\le k, k+1\le j\le q} \{d^M_{ij}\} \end{aligned}$$

When the scoring function is the minimum ultrametric tree size or L^1-min increment, $f(p,q,k)$ can be calculated in $O(1)$ time. Thus, the time complexity of Algorithm OPT-ULTRA is $O(n^3)$, since

$$\begin{aligned} \max_{p\le i\le q, p\le j\le q} \{d^M_{ij}\} = \max\{ & \max_{p\le i\le q-1, p\le j\le q-1} \{d^M_{ij}\}, \\ & \max_{p+1\le i\le q, p+1\le j\le q} \{d^M_{ij}\}, \\ & d^M_{pq},\ 1 \le p \le n, 1 \le q \le n\} \end{aligned}$$

can be computed by dynamic programming in $O(n^2)$ time.

In addition, based on the measurement of the L^k-min increment, where $k \ge 2$, $f(p,q,k)$ can be defined similarly as follows.

$$f(p,q,k) = \sum_{p\le i\le k, k+1\le j\le q} \left(\max_{p\le i\le q, p\le j\le q} \{d^M_{ij}\} - d^M_{ij} \right)^k$$

Here, $f(p,q,k)$ can be calculated in $O(n^2)$ time. Thus, when scoring functions is L^k-min increment, where $k \ge 2$, Algorithm OPT-ULTRA requires $O(n^5)$ time.

Figure 9 shows the computation dependence of subtrees. For example, $Opt_{1,4}$ needs the results of $Opt_{1,1} + Opt_{2,4}, Opt_{1,2} + Opt_{3,4}$ and $Opt_{1,3} + Opt_{4,4}$.

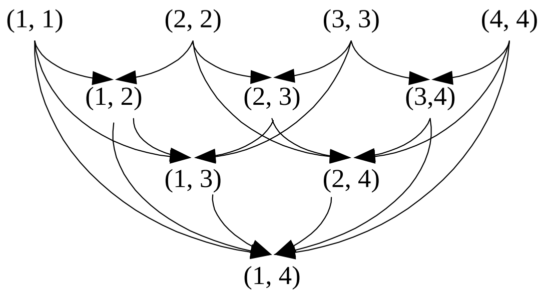

Figure 9: The dependence graph for the dynamic programming.

In the following, we shall present our heuristic algorithm to solve the ultrametric tree problem, which is the combination of Algorithm Circular-Order and Algorithm OPT-ULTRA.

Algorithm HEU-ULTRA

Input: An $n \times n$ distance matrix M and the scoring function for the ultrametric tree problem.

Output: A good ultrametric tree T.

Step 1: Apply Algorithm Circular-Order on the input M to construct a good leaf node circular order L .

Step 2: Apply Algorithm OPT-ULTRA on M and L to construct an ultrametric tree T.

Step 3: The tree T is the solution of this algorithm.

6 Experiment Results

In this section, we shall how our experiment results. In our experiment, we use the random data to test our heuristic algorithm (Circular-Order) and dynamic programming (OPT-ULTRA). Each entry in the distance matrix M is between 2 and

100 and the number of test instances in each test set is 100. We compare our results with the UPGMM method [11]. In addition, we also use the combination of the leaf node circular order of UPGMM with our dynamic programming (UPGMM + OPT-ULTRA) for comparison.

In Table 2 and Table 3, we compare UPGMM, UPGMM + OPT-ULTRA and Circular-Order + OPT-ULTRA with the scoring functions minimum tree size and L^1-min increment, respectively. Each column represents one test set and there are 100 test instances in each test set. Each entry represents the number of occurrences that the performance of the method is superior to those of the other two methods. If both methods or all three methods get the top performance, then the number of occurrences increases one on each method getting top performance. Thus, the total number in each column may be greater than 100. For example, in Table 2, when the number of species is 10, the entry of UPGMM represents that UPGMM method gets the top performance 8 times in 100 test instances. And, the three methods may get the same result, so the sum of 8, 26 and 80 is greater than 100.

In Table 2 and Table 3, we can find that UPGMM + OPT-ULTRA has better performance. We get a conclusion that UPGMM combined with our OPT-ULTRA has significantly improvement. Since UPGMM is based on the minimum tree size scoring function, and Algorithm Circular-Order (our method) is based on neither tree minimum tree size nor L^1-min increment scoring function, in Table 2, we can find that Circular-Order + OPT-ULTRA has worse performance than that of pure UPGMM when n is large. Furthermore, Circular-Order + OPT-ULTRA has better performance than that of pure UPGMM. And, in Table 2 and Table 3, when n is small, Circular-Order + OPT-ULTRA has better performance than other methods. Thus, we get a conclusion that when number of species is smaller, our method can get better leaf node circular order. When the number of species becomes larger, UPGMM shall get better leaf node circular order. Our OPT-ULTRA method is very effective to improve other methods.

7 Conclusion

In this paper, we propose an approximation algorithm, APP-ULTRA, with error ratio $\leq \lceil \log_{\frac{2}{\sqrt{5}-1}} n \rceil + 1 \cong 1.44 \lceil \log n \rceil + 1$, for solving the $\triangle$MUT problem. Our proof of the error ratio is based on MST (Minimum Spanning Tree) and BST (Binary Splitting Tree). And, we define the BST problem and design an $O(n^3)$ algorithm to solve this problem. The algorithm can produce a binary splitting tree with height no more than $\lceil \log_\alpha n \rceil$, where $\alpha = \frac{\sqrt{5}+1}{2}$ and n is the number of leaf nodes.

Besides, we also propose a heuristic algorithm, Algorithm Circular-Order, to construct a leaf node circular order for given an $n \times n$ distance matrix, where n is the number of species. And, we design a dynamic programming algorithm, Algorithm OPT-ULTRA, to solve the optimal ultrametric tree problem under a certain leaf node circular order. Algorithm OPT-ULTRA can work on the minimum tree size and L^1-min increment scoring functions. The time complexity of the dynamic programming is $O(n^3)$. In fact, by our experiment results, we get a clear conclusion that Algorithm OPT-ULTRA significantly improves UPGMM.

References

[1] R. Agarwala, D. Fernandez-Baca, and G. Slutzki, “Fast algorithms for inferring evolutionary trees,” *In Proceedings of the 30th Allerton Conference on Comm., Control, and Comput*, pp. 594–603, 1992.

[2] W. H. E. Day, “Computational complexity of inferring phylogenies by dissimilarity matrices,” *Bulletin of Mathematical Biology*, Vol. 49, No. 4, pp. 461–467, 1987.

[3] W. H. E. Day and D. Sankoff, “Computational complexity of inferring phylogenies by compatibility,” *Systematic Zoology*, Vol. 35, No. 2, pp. 224–229, 1986.

[4] M. Farach and J. Cohen, “Numerical taxonomy on data: Experimental results,” *ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[5] M. Farach, T. Przytycka, and M. Thorup, “On the agreement of many trees,” *Information Processing Letters*, Vol. 55, pp. 297–301, 1995.

[6] M. Farach, S.Kannan, and T.Warnow, “A robust model for finding optimal evolutionary trees,” *Algorithmica*, Vol. 13, No. 1/2, pp. 155–179, 1995.

Method	# of species (n)				
	5	10	20	50	100
UPGMM	0	8	24	9	3
UPGMM + OPT-ULTRA	16	26	90	100	100
Circular-Order + OPT-ULTRA	100	80	10	0	0

Table 2: Experiment results for the minimum tree size scoring function.

Method	# of species (n)				
	5	10	20	50	100
UPGMM	16	5	3	0	0
UPGMM + OPT-ULTRA	16	23	52	98	100
Circular-Order + OPT-ULTRA	100	85	48	3	0

Table 3: Experiment results for the L^1-min increment scoring function.

[7] M. Farach and M. Thorup, "Fast comparison of evolutionary trees," *In Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 481–488, 1994.

[8] C. Korostensky and G. H. Gonnet, "Using traveling salesman problem algorithms for evolutionary tree construction," *Bioinformatics*, Vol. 16, No. 7, pp. 619–627, 2000.

[9] R. C. T. Lee, "Computational biology." http://www.csie.ncnu.edu.tw/˜rctlee/ biology.html, Department of Computer Science and Information Engineering, National Chi-Nan University, 2001.

[10] M. Li, J. H. Badger, X. Chen, S. K. P. Kearney, and H. Zhang, "An information based sequence distance and its application to whole mitochondrial genome phylogeny," *Bioinformatics*, Vol. 17, No. 2, pp. 149–154, 2001.

[11] W. H. Li and D. Graur, *Fundamentals of molecular evolution*. MA: Sinauer Associates, 1991.

[12] W. J. Masek and M. S. Paterson, "How to compute string-edit distances quickly," *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison* (D. Sankoff and J. Kruskal, eds.), Addison-Wesley Reading, 1983.

[13] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees," *Molecular Biology and Evolution*, Vol. 4, pp. 406–424, 1987.

[14] P. H. Sellers, "On the theory and computation of evolutionary distances," *SIAM Journal of Applied Mathematics*, Vol. 26, pp. 787–793, 1974.

[15] D. L. Swofford and G. J. Olsen, "Phylogeny reconstruction," *Molecular Systematics* (D. M. Hillis and C. Moritz, eds.), pp. 411–501, Sinauer Associates, 1990.

[16] H. T. Wareham, "On the computational complexity of inferring evolutionary trees," Tech. Rep. 9301, Department of Computer Science, Memorial University of Newfoundland, 1993. Available by anonymous ftp from ftp.cs.mun.ca in directory pub/techreports.

[17] M. Waterman, T. Smith, M. Singh, and W. Beyer, "Additive evolutionary trees," *Journal of Theoretical Biology*, Vol. 64, pp. 199–213, 1977.

[18] R. Wong, "Worst-case analysis of network design problem heuristics," *SIAM J. Algebraic Descrete Mathematics*, Vol. 1, pp. 51–63, 1980.

[19] B. Y. Wu, K. M. Chao, and C. Y. Tang, "Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices," *Journal of Combinatorial Optimization*, Vol. 3, pp. 199–211, 1999.