Algorithmica (1996) 16: 302–315

Algorithmica
© 1996 Springer-Verlag New York Inc.

Approximation Algorithms for Tree Alignment with a Given Phylogeny

Lusheng Wang,[1] Tao Jiang,[2] and E. L. Lawler[3]

Abstract. We study the following fundamental problem in computational molecular biology: Given a set of DNA sequences representing some species and a phylogenetic tree depicting the ancestral relationship among these species, compute an optimal alignment of the sequences by the means of constructing a minimum-cost evolutionary tree. The problem is an important variant of multiple sequence alignment, and is widely known as *tree alignment*. We design an efficient approximation algorithm with performance ratio 2 for tree alignment. The algorithm is then extended to a polynomial-time approximation scheme. The construction actually works for Steiner trees in any metric space, and thus implies a polynomial-time approximation scheme for planar Steiner trees under a given topology (with any constant degree). To our knowledge, this is the first polynomial-time approximation scheme in the fields of computational biology and Steiner trees. The approximation algorithms may be useful in evolutionary genetics practice as they can provide a good initial alignment for the iterative method in [23].

Key Words. Approximation algorithm, Computational biology, Evolutionary tree, Phylogenetic tree, Steiner tree, Tree alignment.

1. Introduction.

Multiple sequence alignment is one of the fundamental and most challenging problems in computational molecular biology [15], [16]. It plays an essential role in two related areas: finding highly conserved subregions among a set of biological sequences, and inferring the evolutionary history of some species from their associated sequences. A huge number of papers have been written on effective and efficient methods for constructing multiple alignment. For a comprehensive survey, see [4], [5], and [26].

An important approach to multiple sequence alignment is the *tree-alignment* approach [1], [11], [19], [23], [27]. Here, we are given k sequences representing some genetically related species. The ancestral relationship among the species is described by a *phylogenetic tree* or, simply, *phylogeny*. The tree is rooted and has k leaves, each is labeled with a unique input sequence. The internal nodes of the tree correspond to the hypothetical ancestral species and are *unlabeled*. We compute an alignment by reconstructing each ancestral sequence and optimally aligning each pair of sequences induced by the edges of the tree. Hence, the input sequences are compared *indirectly* through

[1] Department of Electrical and Computer Engineering, McMaster University, Hamilton, Ontario, Canada L8S 4K1. lwang@maccs.mcmaster.ca. Supported in part by NSERC Operating Grant OGP0046613.
[2] Department of Computer Science, McMaster University, Hamilton, Ontario, Canada L8S 4K1. jiang@maccs.mcmaster.ca. Supported in part by NSERC Operating Grant OGP0046613 and a Canadian Genome Analysis and Technology Research Grant.
[3] Computer Science Division, University of California, Berkeley, CA 94720, USA. lawler@cs.berkeley.edu. Supported in part by US Department of Energy Grant DE-FG03-90ER6099.

Received August 31, 1993; revised September 16, 1994. Communicated by R. M. Karp.

their common ancestral sequences. To be different from phylogenetic trees, call the resulting fully labeled tree an *evolutionary tree*.[4] Define the cost of an edge (u, v) as the weighted edit distance between the sequence labels of nodes u and v. In this sense, tree alignment is actually the problem of constructing a minimum-cost evolutionary tree for a given phylogeny. We assume that the degree of the given phylogenetic tree is bounded by some small constant d. In fact, usually $d = 2$ (i.e., the tree is binary) in practice [11], [19], [23], [27].

There is a more general and seemingly more difficult variant of tree alignment, which is simply called *generalized tree alignment* in this paper. Here, we are only given a set of DNA sequences and have to construct the hypothetical sequences as well as the phylogeny [9], [18], [24]. While tree alignment can help judge the goodness of a given phylogenetic tree, generalized tree alignment can help create initial phylogenetic trees [9]. It is known that generalized tree alignment is MAX SNP-hard [17], [25]. By the result in [2], this implies that we cannot hope for a polynomial-time approximation scheme (PTAS), unless P = NP.

1.1. *Previous Results*. Sankoff proposed an algorithm to compute an optimal tree alignment using dynamic programming [19], [21]. The time complexity of this algorithm is $O(m(2n)^k)$, where n is the length of an input sequence, m is the number of internal nodes, and k is the number of leaves. The algorithm also assumes that the score between two letters is either 1 or 0. An algorithm that can handle more general score schemes is reported in [11]. Some heuristic algorithms have also been considered in the past. Altschul and Lipman tried to cut down the computation volume required by dynamic programming [1]. Sankoff *et al*. gave an iterative improvement method to speed up the computation [21], [23]. It is claimed that the algorithm usually produces a reasonable alignment in five iterations. Waterman and Perlwitz devised a heuristic method when the sequences are related by a binary tree [27]. Their method computes an "average" sequence for each pair of related input sequences, from bottom to top, and then constructs an overall alignment by aligning each input sequence against the "average" sequence constructed at the root. The running time of this algorithm is $O(kn^2)$. Hein proposed an efficient algorithm based on the concept of a *sequence graph* [11]. Nevertheless, none of these algorithms have a guaranteed performance bound. It has recently been shown that tree alignment is NP-hard even if the given phylogeny is a binary tree [25].

1.2. *Our Results*. In this paper, efficient approximation algorithms with guaranteed performance bound for tree alignment are presented for the first time. We first give a simple algorithm which produces an evolutionary tree by elaborately lifting the input sequences from the leaves to their ancestors. It is shown that the evolutionary tree has a cost at most twice the optimum. The time complexity of this algorithm is $O(k^3 + k^2n^2)$. Augmenting the construction with a local optimization technique, we then extend this algorithm to a PTAS. More precisely, we devise an algorithm which, for each $t > 1$, has a performance ratio $1 + 3/t$ and runs in time $O(k^{d^{t-1}+2}M(d, t-1, n))$, where

[4] The distinction between a phylogeny and an evolutionary tree is nonstandard and is made only for the convenience of presentation.

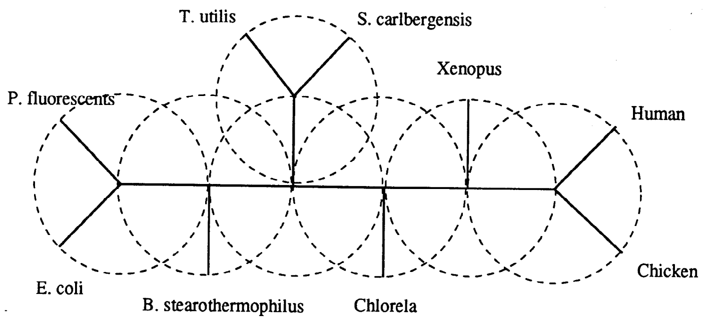

Fig. 1. A phylogeny with nine species, which is divided into seven 3-components.

$M(d, t-1, n)$ is the time needed to align optimally a tree of depth $t-1$ and is upper-bounded by $O(n^{d^{t-1}+1})$. The result is interesting since

(i) To our knowledge, this is the first PTAS in the field of computational biology.
(ii) It shows a great contrast with the MAX SNP-hardness of generalized tree alignment.

1.3. *Applications in the Study of Molecular Evolution.* Our algorithms may also be practical for the analysis of molecular evolution in the following sense: they can help set up a good initial alignment for the iterative method of Sankoff *et al.* [23]. The combined algorithm will always produce an alignment that has a cost not greater than $1+\varepsilon$ times the optimum and is hopefully satisfactory to geneticists.

To illustrate the iterative method in [23], consider the phylogeny in Figure 1, which contains nine given species on its nine leaves. To construct an evolutionary tree, we first assign the given sequences to each internal node (arbitrarily). Then we divide the phylogeny into seven 3-*components* as shown in Figure 1, where a 3-component is a *star* with a *center* and at most three terminals. Local optimization is done for every 3-component as follows. From the labels of the three terminals, we can compute a label of the center using dynamic programming to minimize the cost of the component [19], [10]. The new center label can then be used to update the center label of an overlapping 3-component. The algorithm converges since each local optimization reduces the cost of the tree by at least one. Thus, if the process is repeated long enough, every 3-component will become optimal. However, this does not necessarily result in an optimal evolutionary tree. Nonetheless, it seems the algorithm can produce a reasonably good evolutionary tree after five iterations [23].

1.4. *Applications in Steiner Trees.* Tree alignment can be viewed as a special case of the problem of Steiner trees under a given topology [12], [20]. It is known that a Steiner minimal tree can be computed in polynomial time for a given topology in the rectilinear (i.e., Manhattan) space [7], [20]. This implies that the variant of tree alignment where the space is Σ^n and the distance is Hamming distance is solvable in polynomial time. The same result trivially holds if the space is a graph. (Here the graph is part of the input and the running time is polynomial in the size of the graph.) For the Euclidean plane, the Steiner minimal tree can be found in $O(n^2)$ time if the given topology is a Steiner topology [12], [13]. For arbitrary topology, no exact algorithm is known and an iterative dynamic programming algorithm is given in [20].

The construction of our approximation algorithms in fact works for Steiner trees under a given topology in any metric space. As a corollary, we obtain a PTAS for the problem in the Euclidean plane for arbitary topology. It is worth mentioning that, when the topology is not given, the best we know is that planar Steiner minimal trees can be approximated with ratio $2/\sqrt{3}-\varepsilon$, for some $\varepsilon > 0$ [6]. At the moment, it is open whether the planar Steiner problem has a PTAS. It has been observed that Karp's *probabilistic ε-approximation scheme* for the traveling salesman problem can be modified to work for the planar Steiner problem [12], [14]. We note in passing that the Steiner problem on graphs is MAX SNP-hard [3].

Some standard definitions are given in Section 2. The ratio 2 approximation algorithm and polynomial-time approximation scheme are presented in Sections 3 and 4, respectively.

2. Basic Definitions and Notations.

A *sequence* is a string over some fixed alphabet Σ. For DNA sequences, the alphabet Σ contains four letters A, C, G, and T representing four distinct nucleotides, and for protein sequences, Σ contains 20 letters, each representing a unique amino acid. An *alignment* of two sequences x and y is obtained by inserting spaces into or at either end of x and y such that the two resulting sequences x' and y' are of the same length. That is, every letter in x' is opposite to a unique letter in y'.

Suppose that n is the length of the sequences x' and y'. The value of the alignment is defined as $\sum_{i=1}^{n} \mu(x'(i), y'(i))$, where $x'(i)$ and $y'(i)$ denote the two letters at the ith column of the alignment, and $\mu(x'(i), y'(i))$ denotes the *score* of these two opposing letters under some given *score scheme* μ. There are several popular score schemes for amino acids and for nucleotides [9]. A standard assumption about the score scheme μ is that it is a metric [9], [22]. Namely:

(i) It satisfies *triangle inequality*, i.e., for any three letters a, b, and c, $\mu(a, c) \leq \mu(a, b) + \mu(b, c)$.
(ii) $\mu(a, a) = 0$ for every letter a.

An *optimal alignment* of two sequences is one that *minimizes* the value over all possible alignments. The weighted *edit distance* $dist(x, y)$ between sequences x and y is defined as the minimum alignment value of x and y. The distance is often called the *mutational distance* in the study of molecular evolution [19]. $dist(x, y)$ can be easily computed in time $O(|x| \cdot |y|)$ using dynamic programming [22].

We consider *rooted ordered* trees. The *degree* of a node is its number of children. The degree of a tree is the maximum degree of its nodes. As in all the earlier papers in this area, we only consider trees with degrees bounded by some small constant d. To simplify the presentation, we further assume that each internal node in T has exactly d children in our discussion. The extension of our results to the general case is fairly straightforward.

Throughout this paper we use the following notation. For a tree T, let $r(T)$ be the root of T, $c(T)$ the cost of T, $L(T)$ the set of the leaves of T, and $I(T)$ the set of the internal nodes in T. The parent of a node v is represented as $p(v)$. For each node v in tree T, T_v denotes the subtree of T rooted at v. A leaf that is a descendant of node v is called a *descendant leaf* of v. Define $S(v)$ to be the set of labels of all descendant leaves of v. Note that the label of a leaf never changes during an alignment process.

We end this section with the notion of *polynomial-time approximation schemes*. An *approximation scheme* for a minimization problem is an algorithm A which takes as input both an instance I and an error bound ε, and has the performance guarantee

$$R_A(I, \varepsilon) = \frac{A(I)}{\mathrm{OPT}(I)} \leq 1 + \varepsilon,$$

where $A(I)$ is the solution given by A and $\mathrm{OPT}(I)$ is the optimal solution. Such an algorithm A can be viewed as a family of algorithms $\{A_\varepsilon \mid \varepsilon > 0\}$. A polynomial-time approximation scheme is an approximation scheme $\{A_\varepsilon\}$ where the algorithm A_ε runs in time polynomial in the size of the instance I, for any fixed ε. For more details, see [8].

3. An Approximation Algorithm with Ratio 2. As mentioned earlier, our basic idea is to construct an evolutionary tree by elaborately lifting the given sequences from the leaves to their ancestors. Call an evolutionary tree a *lifted* tree if the label of each internal node equals the label of some child of the node. Below, we first show that there is a lifted tree of small cost and then compute the minimum-cost lifted tree in $O(k^3 + k^2n^2)$ time.

From now on, let $X = \{s_1, \ldots, s_k\}$ be a set of sequences and let T be a phylogeny for X (i.e., the leaves of T are uniquely labeled with the sequences $s_1, \ldots, s_k$). Let $T^{\min}$ denote an optimal evolutionary tree for T. For each node v in $T^{\min}$, the *closest descendant leaf* of v, denoted $l(v)$, is a descendant leaf of v such that the path from v to $l(v)$ is the shortest (i.e., with the minimum-cost) among all descendant leaves of v. For convenience, let $sl(v)$ denote sequence label of $l(v)$. Define an evolutionary tree T^l as follows: for each internal node v in T, assign the sequence $sl(v)$ to v. Clearly, the tree T^l can be made a lifted tree if we break ties carefully while assigning $l(v)$.

LEMMA 1. *$c(T^l) \leq 2(1 - 1/k)c(T^{\min})$, where k is the number of leaves in T.*

PROOF. Consider a counterclockwise walk along the outside of the optimal tree $T^{\min}$ which travels twice, once in each direction, through all the edges except those in the two boundary (leftmost and rightmost) paths. Since the order of the children does not matter, we can choose the two boundary paths such that their cost is the greatest. Therefore, the total cost of this walk is $c(T^l) \leq 2(1 - 1/k)c(T^{\min})$, where k is the number of leaves in T. The walk can be also thought of as a path that links all the leaves into a chain, from left to right. Take an arbitrary node v and consider the subtree $T_v^{\min}$ rooted at v. Let $v_1, \ldots, v_d$ be the children of v. These children induce d subtrees $T_{v_1}^{\min}, \ldots, T_{v_d}^{\min}$. For a node v, denote the rightmost and leftmost descendant leaves of v by $f(v)$ and $g(v)$, respectively. For each $i = 1, \ldots, d - 1$, to connect the two leaves $f(v_i)$ and $g(v_{i+1})$, the walk uses a path

$$P_{v,i}\colon\ f(v_i) \to v_i \to v \to v_{i+1} \to g(v_{i+1}).$$

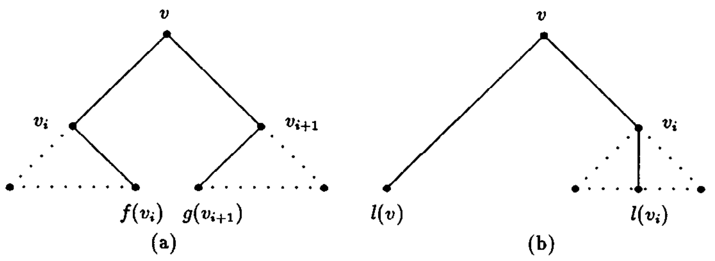

Fig. 2. (a) The path $P_{v,i}$ in the walk. (b) The path $P'_{v,i}$.

(See Figure 2(a).) It is easy to see that

$$\sum_{v \in T} \sum_{i=1}^{d-1} c(P_{v,i}) = c(T^l) \le 2(1 - 1/k)c(T^{\min}).$$

Define $d - 1$ new paths $P'_{v,i}$: $l(v) \to v \to v_i \to l(v_i)$, where $1 \le i \le d$ and $l(v_i) \ne l(v)$. (See Figure 2(b).) Let $P'^{(1)}_{v,i}$ and $P'^{(2)}_{v,i}$ denote the subpaths of $P'_{v,i}$: $l(v) \to v$ and $v \to v_i \to l(v_i)$, respectively. $P^{(1)}_{v,i}$ and $P^{(2)}_{v,i}$ denote the subpaths of $P_{v,i}$: $f(v_i) \to v_i \to v$ and $v \to v_{i+1} \to g(v_{i+1})$, respectively. By the definition of $l(v)$, we have

$$c(P'^{(1)}_{v,i}) \le c(P^{(2)}_{v,i}), \qquad c(P'^{(2)}_{v,i}) \le c(P^{(1)}_{v,i}),$$

and

$$c(P'^{(1)}_{v,i}) \le c(P^{(1)}_{v,i-1}), \qquad c(P'^{(2)}_{v,i}) \le c(P^{(2)}_{v,i-1}).$$

Thus,

$$c(P'_{v,i}) \le c(P_{v,i}) \quad \text{and} \quad c(P'_{v,i}) \le c(P_{v,i-1}).$$

Therefore,

(1) $$\sum_{\substack{1 \le i \le d \\ l(v_i) \ne l(v)}} c(P'_{v,i}) \le \sum_{i=1}^{d-1} c(P_{v,i}).$$

Now consider the tree T^l. The edges between v and its d children cost totally

$$\sum_{i=1}^{d} dist(sl(v), sl(v_i)).$$

By the triangle inequality, we have $dist(sl(v), sl(v_i)) \le c(P'_{v,i})$. It follows from the above that

$$c(T^l) = \sum_{v \in T} \sum_{i=1}^{d} dist(sl(v), sl(v_i))$$

$$\le \sum_{v\in T} \sum_{\substack{1\le i\le d\\ l(v_i)\ne l(v)}} c(P'_{v,i})$$

$$\le \sum_{v\in T} \sum_{i=1}^{d-1} c(P_{v,i}).$$

Therefore, $c(T^l) \le 2(1-1/k)c(T^{\min})$. □

Since T^l is actually a lifted tree, we can immediately conclude

COROLLARY 2. *There exists a lifted tree with cost at most* $2c(1-1/k)T^{\min}$, *where k is the number of leaves in T*.

Computing T^l is not easy since it is derived from the optimal tree $T^{\min}$. However, in the following we describe a simple algorithm that constructs an optimal lifted tree T^*, i.e., one that has the smallest cost among all the lifted trees. From the above corollary,

$$c(T^*) \le c(T^l) \le 2c\left(1-\frac{1}{k}\right)T^{\min}.$$

The idea is to use dynamic programming.

For each $v \in T$, $i = 1, \ldots, k$ such that $s_i \in S(v)$, let $D[v, s_i]$ denote the cost of an optimal lifted tree for T_v with v being assigned the sequence s_i. It is possible to compute $D[v, s_i]$ recursively. For each leaf v, we define $D[v, s_i] = 0$ if the label of v is s_i. Let v be an internal node, and let $v_1, \ldots, v_d$ be its children. Suppose that $s_i \in S(v_p)$, where $1 \le p \le d$. Clearly, p is unique. Then $D[v, s_i]$ can be computed as follows: For each $q = 1, \ldots, d$ and $q \ne p$, find a j_q such that $s_{j_q} \in S(v_q)$ and $D[v_q, s_{j_q}] + dist(s_i, s_{j_q})$ is minimized. Then

$$D[v, s_i] = D[v_p, s_i] + \sum_{\substack{1\le q\le d\\ q\ne p}} (D[v_q, s_{j_q}] + dist(s_i, s_{j_q})).$$

(See Figure 3.) The full algorithm is described in Figure 4.

Let n be the maximum length of any sequence in X. Line 3 of Algorithm 1 takes $O(k^2n^2)$ time. Each execution of line 7 takes $O(k)$ time. Since line 7 is executed $O(k^2)$ times, Algorithm 1 requires $O(k^3 + k^2n^2)$ time in the worst case. Hence, we have the following theorem.

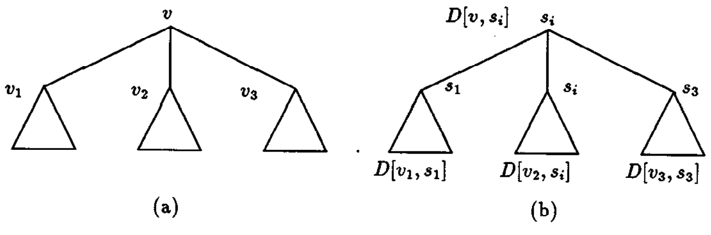

Fig. 3. (a) The subtree T_v. (b) The lifted subtree, where $s_1 \in S(v_1)$, $s_i \in S(v_2)$, and $s_3 \in S(v_3)$.

```
 1. begin
 2. for each pair $(i, j)$, $1 \leq i < j \leq k$, do
 3.    compute $dist(s_i, s_j)$.
 4. for each level of $T$, with the bottom level first, do
 5.    for each node $v$ at the level do
 6.       for $i = 1$ to $k$
 7.          if $s_i \in S(v)$ then compute $D[v, s_i]$.
 8. Select an $s \in X$ such that $D[r(T), s]$ is minimized.
 9. Compute the lifted tree with cost $D[r(T), s]$ by back-tracing.
10. end.
```

Fig. 4. Algorithm 1.

THEOREM 3. *Algorithm* 1 *outputs an evolutionary tree with cost at most* $2(1-1/k)c(T^{\min})$ *in time* $O(k^3 + k^2n^2)$.

4. A Polynomial-Time Approximation Scheme. We extend Algorithm 1 to a PTAS by considering constant-size components of the tree and augmenting the "lifting" technique with local optimization which is also used in the iterative improvement method in Section 1.3. An overiew of the approach is given below.

Recall that, if we label each node v in the tree T with the sequence $sl(v)$, we obtain an evolutionary tree T^l with cost at most twice the cost of the optimal evolutionary tree $T^{\min}$. Let $t > 0$ be an integer. For each $v \in T$, define the *depth-t component* $T_{v,t}$ as the subtree of T_v containing only the top $t + 1$ levels. Clearly, $T_{v,t}$ has at most d^t leaves and at most $(d^t - 1)/(d - 1)$ internal nodes. For a subtree $T_{v,t}$, $v \neq r(T)$, we can obtain an evolutionary subtree $T'_{v,t}$ by assigning to each node $u \in L(T_{v,t}) \cup \{v\}$ the sequence assigned to u in the tree T^l, which is $sl(u)$, and constructing the sequences for the other nodes in $T_{v,t}$ such that the cost of the subtree is minimized by dynamic programming. Obviously, $c(T'_{v,t}) \leq c(T^l_{v,t})$, where $T^l_{v,t}$ is the depth-t subtree of T^l rooted at v. This prompts us to partition the tree T into depth-t components, and construct an optimal evolutionary subtree $T'_{v,t}$ for each component $T_{v,t}$.

We partition T as follows. Let set V_i contain all nodes at level i of T. (The root is at level 0.) We can partition the internal nodes (excluding the root) of T into t groups $G_0, \ldots, G_{t-1}$, where

$$G_i = \bigcup_{j \equiv i \pmod t} V_j.$$

For each $i = 0, \ldots, t-1$, let $T'_{r(T),i}$ denote the evolutionary subtree obtained by assigning to each node $v \in L(T_{r(T),i})$ the sequence $sl(v)$ and each other node in $T_{r(T),i}$ a sequence so that the cost of $T'_{r(T),i}$ is minimized. Clearly, for each i, $0 \leq i \leq t - 1$, the union of subtrees $\bigcup_{v \in G_i} T'_{v,t} \cup T'_{r(T),i}$ forms an evolutionary tree, denoted as T'_i. We first show that the total cost of all these evolutionary subtrees is bounded by $t + 3$ times the cost of the optimal evolutionary tree, i.e.,

$$\sum_{i=0}^{t-1} T_i' = \sum_{v \neq r(T)} c(T_{v,t}') + \sum_{i=0}^{t-1} c(T_{r(T),i}') \leq (t+3)c(T^{\min}). \tag{2}$$

Therefore, there exists an evolutionary tree T_i' with cost $c(T_i') \leq (1 + 3/t)c(T^{\min})$.

Again, computing T_i' is not easy since it relies on the optimal tree $T^{\min}$. Call an evolutionary tree a *depth-t component tree* with respect to some fixed G_i if every node $v \in G_i$ is assigned a sequence which labels some node in $L(T_{v,t})$. In other words, the labels of the nodes in G_i are all lifted from the leaves. Since in the tree T_i' every node $v \in G_i$ is assigned the sequence $sl(v)$, T_i' is a depth-t component tree with respect to G_i. Hence, we can design a dynamic programming algorithm to identify the optimal depth-t component tree with respect to G_i, for each i, and select the best tree as our output which costs at most $(1 + 3/t)c(T^{\min})$.

Now, we first prove inequality (2). For each node v, let $\rho(v)$ be the length of the path from v to $l(v)$, let $h(v)$ be the total length of the edges from v to its children, and let $s(v)$ be the sequence label assigned to v, in the optimal tree $T^{\min}$. In particular, if v is a leaf, then $\rho(v) = 0$ and $h(v) = 0$. The following lemma holds.

LEMMA 4. *Let $v \neq r(T)$ be a node in T and have children $v_1, \ldots, v_d$, then*

$$c(T_{v,t}') \leq \sum_{u \in I(T_{v,t})} h(u) + \sum_{u \in L(T_{v,t})} \rho(u) + h(v) + \sum_{i=1}^{d} \rho(v_i). \tag{3}$$

PROOF. Let $T_{v,t}''$ be the evolutionary subtree obtained by labeling each node $u \in L(T_{v,t}) \cup \{v\}$ with the sequence $sl(u)$ and each other node u in $T_{v,t}$ with the sequence $s(u)$. (See Figure 5.) By the triangle inequality, we have, for each node $u \in L(T_{v,t})$,

$$dist(s(p(u)), sl(u)) \leq dist(s(p(u)), s(u)) + \rho(u).$$

Thus,

$$c(T_{v,t}'') \leq \sum_{i=1}^{d} dist(sl(v), s(v_i)) + \sum_{u \in I(T_{v,t}) - \{v\}} h(u) + \sum_{u \in L(T_{v,t})} \rho(u). \tag{4}$$

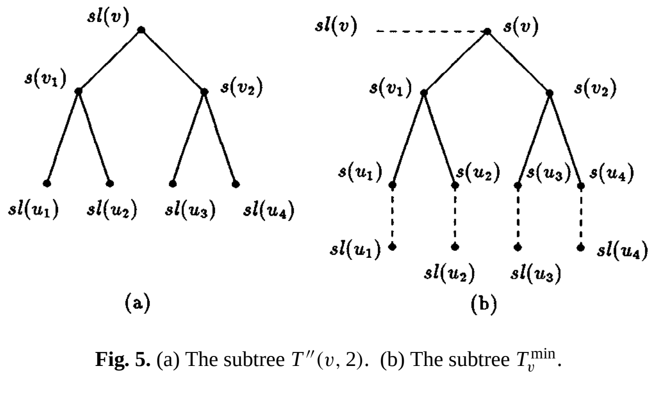

Fig. 5. (a) The subtree $T''(v, 2)$. (b) The subtree $T_v^{\min}$.

An explanation of this inequality is given in Figure 5(b). Again, by the triangle inequality,

$$dist(sl(v), s(v_i)) \leq dist(s(v_i), s(v)) + \rho(v).$$

Moreover,

$$\sum_{i=1}^{d} dist(s(v_i), s(v)) = h(v).$$

Hence, we can conclude

$$c(T''_{v,t}) \leq \sum_{u \in I(T_{v,t})} h(u) + \sum_{u \in L(T_{v,t})} \rho(u) \ + \ d\rho(v).$$

Since $\rho(v) \leq dist(s(v_i), s(v)) + \rho(v_i)$ for each $i = 1, \ldots, d$,

$$\begin{aligned} c(T''_{v,t}) &\leq \sum_{u \in I(T_{v,t})} h(u) \ + \sum_{u \in L(T_{v,t})} \rho(u) \ + \sum_{i=1}^{d} dist(s(v_i), s(v)) + \rho(v_i) \\ &\leq \sum_{u \in I(T_{v,t})} h(u) \ + \sum_{u \in L(T_{v,t})} \rho(u) \ + h(v) \ + \sum_{i=1}^{d} \rho(v_i). \end{aligned}$$

Since the cost of $T'_{v,t}$ is minimized, we have $c(T'_{v,t}) \leq c(T''_{v,t})$. □

Note that, if the node v is near the bottom level of T, the real depth t' of $T_{v,t}$ could be less than t. In this case the above inequality (3) becomes

$$(5) \qquad c(T'_{v,t'}) \leq \sum_{u \in I(T_{v,t'})} h(u) + \sum_{u \in L(T_{v,t'})} \rho(u) \ + h(v) + \sum_{i=1}^{d} \rho(v_i).$$

Similarly, let $T''_{r(T),i}$ denote the evolutionary subtree obtained by assigning to each node $v \in L(T_{r(T),i})$ the sequence $sl(v)$ and to each other node v the sequence $s(v)$. Note that now the root $r(T)$ is labeled with $s(r(T))$ instead of $sl(r(T))$ in the subtree $T''_{r(T),i}$. Then inequality (4) becomes

$$(6) \qquad c(T'_{r(T),i}) \leq c(T''_{r(T),i}) \leq \sum_{u \in I(T_{r(T),i})} h(u) + \sum_{u \in L(T_{r(T),i})} \rho(u).$$

Summing up inequalities (3) or (5) for all $v \in T$, and inequality (6) for $i = 0, \ldots, t-1$, we have

$$(7) \qquad \sum_{v \neq r(T)} c(T'_{v,t}) \ + \sum_{i=0}^{t-1} c(T'_{r(T),i}) \leq (t+1) \sum_{v \in T} h(v) \ + 2 \sum_{v \in T} \rho(v).$$

The coefficients in the above inequality are established from the following observations:

1. The term $h(v)$ in inequality (3) is counted once for every v.
2. The term $\sum_{u \in I(T_{v,t})} h(u)$ in inequalities (3), (5), and (6) totally contributes at most t $h(v)$'s for each v.

3. For each v in the top t levels of T, there is a $\rho(v)$ contribution from (6) but at most a contribution of 1 from inequality (3) or (5). For each v below level t, there is no $\rho(v)$ contribution from inequality (6) but at most two $\rho(v)$'s from inequality (3) or (5). Therefore, each v contributes at most two $\rho(v)$'s.

Now, we want to upper-bound the right-hand side of inequality (7) in terms of $c(T^{\min})$. Obviously,

$$\sum_{v\in T} h(v) = c(T^{\min}). \tag{8}$$

To establish the relation between $\sum_{v\in T} \rho(v)$ and $c(T^{\min})$, we need the following lemma, which is a variation of Lemma 3.2 in [6].

LEMMA 5. *Let U be a tree such that every internal node has at least two children. There exists a one-to-one mapping e from the internal nodes of U to the leaves such that for each internal node v:*

(i) *$e(v)$ is a descendant of v.*
(ii) *The paths from internal nodes v to $e(v)$ are edge-disjoint.*

PROOF. We prove it by induction on the depth of the tree. First, we strengthen the lemma by adding:

(iii) There is another leaf $e'(r(U))$ such that the path from $r(U)$ to $e'(r(U))$ is edge-disjoint from the paths in (ii).

The lemma is trivial for trees with depth 1. Suppose that (i)–(iii) hold for any tree with depth $i > 0$. Consider a tree U with depth $i + 1$. Let the root of U have $d \geq 2$ children: $v_1, \ldots, v_d$. To construct the mapping e for the internal nodes in U, we preserve the mappings for the subtrees $U_{v_1}, \ldots, U_{v_d}$, and assign $e'(v_1)$ as $e(r(U))$ and $e'(v_2)$ as $e'(r(U))$. □

Let e be such a mapping for tree T. For each node v, let $\tau(v)$ denote the length of the path from v to $e(v)$ in $T^{\min}$. ($\tau(v) = 0$ if v is a leaf.) It follows from the above lemma that

$$\sum_{v\in T} \tau(v) \leq c(T^{\min}).$$

By the definition of $\rho(v)$,

$$\sum_{v\in T} \rho(v) \leq \sum_{v\in T} \tau(v) \leq c(T^{\min}). \tag{9}$$

The above inequality is crucial for our result. Inequalities (7)–(9) immediately imply the following lemma:

LEMMA 6.

$$\sum_{i=0}^{t-1} T_i = \sum_{v\neq r(T)} c(T'_{v,t}) + \sum_{i=0}^{t-1} c(T'_{r(T),i}) \leq (t+3)c(T^{\min}).$$

By Lemma 6, there exists an i, $0 \leq i \leq t-1$, such that

$$t \cdot c(T_i') \leq (t+3)c(T^{\min}).$$

Thus, the following lemma holds.

LEMMA 7. *There exists a depth-t component tree with cost at most* $(1+3/t)c(T^{\min})$.

Although T_i' approximates $T^{\min}$ with the desired bound, the trouble is again that it is not easy to compute T_i'. Now we describe an algorithm to construct a *minimum-cost* depth-t component tree $\hat{T}$, i.e., the cost of $\hat{T}$ is the smallest among all the depth-t component trees (with respect to some G_i). The basic idea is to combine dynamic programming with local optimization.

Consider a $T_{v,t}$. Let u be a child of v and let $L(T_{u,t-1}) = \{w_1, \ldots, w_m\}$ be the set of leaves in $T_{u,t-1}$. (Note that the component $T_{v,t}$ may not be full.) For each $i = 1, \ldots, m$, let $s_i \in S(w_i)$ be a sequence. Then, for each sequence s, $\hat{T}(v, t, u, s, s_1, \ldots, s_m)$ denotes the evolutionary subtree obtained from $T_{u,t-1} \cup \{(v,u)\}$ by labeling v with sequence s and each node w_i with sequence s_i, and constructing a sequence for each other node in $T_{u,t-1}$ so that the cost of $\hat{T}(v, t, u, s, s_1, \ldots, s_m)$ is minimized.

The top subtree $T_{r(T),i}$, $0 \leq i \leq t-1$, is treated similarly and the resulting evolutionary subtree is denoted $\hat{T}(r(T), i, u, s, s_1, \ldots, s_m)$.

Let v be a node at level i of T, and $s \in S(v)$. Define $\hat{T}(v, s)$ as the minimum-cost evolutionary subtree obtained from T_v such that the node v is labeled with the sequence s and the evolutionary subtree itself forms a depth-t component tree with respect to G_j, where $j \equiv i \pmod t$. We use $D[v, s]$ to denote the cost of $\hat{T}(v, s)$. Similar to the previous section, $D[v, s]$ can be computed recursively from bottom to top.

If v is a leaf, $D[v, s(v)] = 0$. Let v be an internal node and let $u_1, \ldots, u_d$ be the children of v. For each $i = 1, \ldots, d$, let $L(T_{u_i,t-1}) = \{w_{i,1}, \ldots, w_{i,m}\}$. Suppose that $s \in S(w_{p,q})$, where $1 \leq p \leq d$ and $1 \leq q \leq m$ are unique. Then $D[v, s]$ can be computed as follows: For each $i = 1, \ldots, d$ and each $j = 1, \ldots, m$, find an $s_{i,j}$ such that $s_{i,j} \in S(w_{i,j})$ if $i \neq p$ or $j \neq q$, and $s_{i,j} = s$ otherwise, and moreover

$$c(\hat{T}(v, t, u_i, s, s_{i,1}, \ldots, s_{i,m})) + \sum_{j=1}^{m} D[w_{i,j}, s_{i,j}]$$

is minimized. Then

$$D[v, s] = \sum_{i=1}^{d} c(\hat{T}(v, t, u_i, s, s_{i,1}, \ldots, s_{i,m})) + \sum_{i=1}^{d} \sum_{j=1}^{m} D[w_{i,j}, s_{i,j}].$$

For each $i = 0, \ldots, t-1$, let $D[i]$ be the cost of the optimal depth-t component tree with all the roots of all their depth-t components in G_i. $D[i]$ can be computed from the top subtree $T_{r(T),i}$ and the values $D[v, s]$ of the nodes at level i of T in a way similar to above. Clearly, $\min\{D[i] | 0 \leq i \leq t-1\} = c(\hat{T})$.

The actual algorithm for computing $\hat{T}$ is given in Figure 6. Suppose that computing each $c(\hat{T}(v, t, u, s, s_1, \ldots, s_m))$ or $c(\hat{T}(r(T), i, u, s, s_1, \ldots, s_m))$ (i.e., local optimization) requires at most $M(d, t-1, n)$ time. Clearly, $M(d, t-1, n)$ is upper-bounded by $O(n^{d^{t-1}+1})$ [19]. Thus an execution of line 5 takes $k^{d^{t-1}} M(d, t-1, n)$

```
 1. begin
 2. for each level, with the bottom level first, do
 3.   for each node $v$ at the level do
 4.     for each $s \in S(v)$ do
 5.       compute $D[v, s]$.
 6. for $i = 0$ to $t - 1$ do
 7.   compute $D[i]$.
 8. Select an $i$ to minimize $D[i]$.
 9. Compute the evolutionary tree $\hat{T}$ from $D[i]$ by back-tracing.
10. end.
```

Fig. 6. Algorithm 2.

time. Line 5 is repeated a total of $O(k^2)$ times. Line 7 is executed t times, each taking at most $k^{d^{t-2}+1}M(d, t-2, n)$ time. Therefore, the time complexity of Algorithm 2 is $O(k^{d^{t-1}+2}M(d, t-1, n))$. (In fact, it is easy to show that the algorithm runs in time $O((k/d^t)^{d^{t-1}+2}M(d, t-1, n))$.)

THEOREM 8. *For any t, Algorithm 2 has a performance ratio $R_t \le 1 + 3/t$ and runs in time $O(k^{d^{t-1}+2}M(d, t-1, n))$.*

COROLLARY 9. *Tree alignment has a PTAS.*

5. Remarks.

It is unclear whether Corollary 9 holds for phylogenies with unbounded degrees. Interestingly, we can show that for a score scheme that does not satisfy the triangle inequality, constructing an optimal evolutionary tree is MAX SNP-hard even when the given phylogeny is a star [25].

Also, the high complexity of Algorithm 2 excludes even moderate t from consideration in practice. It would be of great interest to improve the efficiency of the algorithm.

Acknowledgment. We thank the referees for many valuable comments and suggestions.

References

[1] S. Altschul and D. Lipman, Trees, stars, and multiple sequence alignment, *SIAM J. Appl. Math.*, **49** (1989), 197–209.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, Proof verification and hardness of approximation problems, *Proc.* 33*rd IEEE Symp. on Foundations of Computer Science*, 1992, pp. 14–23.

[3] M. Bern and P. Plassmann, The Steiner problem with edge lengths 1 and 2, *Inform. Process. Lett.*, **32** (1989), 171–176.

[4] H. Carrillo and D. Lipman, The multiple sequence alignment problem in biology, *SIAM J. Appl. Math.*, **48** (1988), 1073–1082.

[5] S. C. Chan, A. K. C. Wong, and D. K. T. Chiu, A survey of multiple sequence comparison methods, *Bull. Math. Biol.*, **54**(4) (1992), 563–598.

[6] D. Z. Du, Y. Zhang, and Q. Feng, On better heuristic for Euclidean Steiner minimum trees, *Proc.* 32*nd IEEE Symp. on Foundations of Computer Science*, 1991, pp. 431–439.

[7] J. S. Farris, Methods for computing Wagner trees, *Systematic Zoology*, **19** (1970), 83–92.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[9] D. Gusfield, Efficient methods for multiple sequence alignment with guaranteed error bounds, *Bull. Math. Biol.*, **55** (1993), 141–154.

[10] J. J. Hein, A tree reconstruction method that is economical in the number of pairwise comparisons used, *Mol. Biol. Evol.*, **6**(6) (1989), 669–684.

[11] J. J. Hein, A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given, *Mol. Biol. Evol.*, **6**(6) (1989), 649–668.

[12] F. K. Hwang and D. S. Richards, Steiner tree problems, *Networks*, **22** (1992), 55–89.

[13] F. K. Hwang and J. F. Weng, The shortest network under a given topology, *J. Algorithms*, **13** (1992), 468–488.

[14] R. M. Karp, Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane, *Math. Oper. Res.*, **2** (1977), 209–224.

[15] R. M. Karp, Mapping the genome: some combinatorial problems arising in molecular biology, *Proc. ACM Symp. on Theory of Computing*, 1993, pp. 278–285.

[16] E. S. Lander, R. Langridge, and D. M. Saccocio, Mapping and interpreting biological information, *Comm. ACM*, **34**(11) (1991), 33–39.

[17] C. H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *J. Comput. System Sci.*, **43** (1991), 425–440.

[18] D. Penny, Criteria for optimising phylogenetic trees and the problem of determining the root of a tree, *J. Mol. Evol.*, **8** (1976), 95–116.

[19] D. Sankoff, Minimal mutation trees of sequences, *SIAM J. Appl. Math.*, **28**(1) (1975), 35–42.

[20] D. Sankoff and P. Rousseau, Locating the vertices of a Steiner tree in an arbitrary metric space, *Math. Programming*, **9** (1975), 240–246.

[21] D. Sankoff and R. Cedergren, Simultaneous comparisons of three or more sequences related by a tree, in D. Sankoff and J. Kruskal (eds.), *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, pp. 253–264, Addison-Wesley, Reading, MA, 1983.

[22] D. Sankoff and J. Kruskal (eds.), *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, MA, 1983.

[23] D. Sankoff, R. Cedergren, and G. Lapalme, Frequency of insertion–deletion, transversion, and transition in the evolution of 5S ribosomal RNA, *J. Mol. Evol.*, **7** (1976), 133–149.

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

[25] L. Wang and T. Jiang, On the complexity of multiple sequence alignment, *J. Computat. Biol.*, **1**(4) (1994), 337–348.

[26] M. S. Waterman, Sequence alignments, in M. S. Waterman (ed.), *Mathematical Methods for DNA Sequences*, CRC, Boca Raton, FL, 1989, pp. 53–92.

[27] M. S. Waterman and M. D. Perlwitz, Line geometries for sequence comparisons. *Bull. Math. Biol.*, **46** (1984), 567–577.