JOURNAL OF ALGORITHMS **25,** 255–273 (1997)
ARTICLE NO. AL970882

Improved Approximation Algorithms for Tree Alignment*

Lusheng Wang[†]

Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong

and

Dan Gusfield[‡]

Department of Computer Science, University of California, Davis, California 95616

Received March 26, 1996

Multiple sequence alignment is a task at the heart of much of current computational biology [4]. Several different objective functions have been proposed to formalize the task of multiple sequence alignment, but efficient algorithms are lacking in each case. Thus multiple sequence alignment is one of the most critical, essentially unsolved problems in computational biology. In this paper we consider one of the more compelling objective functions for multiple sequence alignment, formalized as the *tree alignment problem*. Previously in [13], a ratio-two approximation method was developed for tree alignment, which ran in *cubic* time (as a function of the number of fixed length strings to be aligned), along with a polynomial time approximation scheme (PTAS) for the problem. However, the PTAS in [13] had a running time which made it impractical to reduce the performance ratio much below two for small size biological sequences (100 characters long). In this paper we first develop a ratio-two approximation algorithm which runs in *quadratic* time, and then use it to develop a PTAS which has a better performance ratio and a vastly improved worst case running time compared to the scheme in [13] for the case where the given tree is a regular deg-ary tree. With the new approximation scheme, it is now practical to guarantee a ratio of 1.583 for strings of lengths 200 characters or less. © 1997 Academic Press

*Partially supported by Department of Energy Grant DE-FG03-90ER60999.
[†] E-mail: lwang@cs.cityu.edu.hk.
[‡] E-mail: gusfield@cs.ucdavis.edu.

1. INTRODUCTION

Multiple sequence alignment is a very important problem in computational biology [1, 2, 4]. It plays an essential role in finding conserved subregions among a set of sequences and inferring the evolutionary history of some species [4]. Many methods have been proposed, and tree alignment is one of the most interesting [2]. Suppose we are given a set of k sequences and a tree with k leaves, each of which is labeled with a unique sequence. The problem is to determine a sequence for each internal node such that the total cost of the tree is minimized. (The cost of an edge is the edit distance between the two sequences assigned to the two ends of the edge and the cost of the tree is the total cost of edges.) The biological interpretation of the tree alignment problem is that the given tree represents evolutionary history (known from means other than sequence analysis) which has created the molecular sequences (DNA, RNA, or amino acid) written at the leaves of the tree. Thus, in biological applications, the tree is almost always binary. The leaf sequences are ones found in organisms existing today. The sequences to be determined for the interior nodes of the tree represent inferred sequences that may have existed in the ancestral organisms, and the weighted edit distance between two strings models the evolutionary cost of mutating one sequence to the other. Thus the cost of the optimal solution to the tree alignment problem is the minimum total mutation cost needed to explain the evolution of the extant sequences in terms of the given evolutionary tree. There are literally hundreds of papers written each year in the biological literature where such tree alignments are reported on biological sequence data using established or newly deduced evolutionary trees. Given a tree alignment, one can obtain a multiple sequence alignment of the extant (leaf) sequences which is believed to expose evolutionarily significant relationships between the sequences [9, 10]. We will not detail that here.

We assume that the reader is familiar with the notion of the weighted edit distance between two strings. In this paper we need only assume that the scoring scheme satisfies the triangle inequality. That is, if x, y, and z are three strings, then the weighted edit distance between x and y is no more than the weighted edit distance between x and z plus the weighted edit distance between z and y. This is a sensible assumption, since the edit distance reflects the cost of transforming x to y [11]. For clarity, we define some terminology. An *approximation scheme* for the minimization problem is an algorithm A which takes as input both an instance I and a parameter ϵ and has the *performance ratio*

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

where $A(I)$ is the cost of the solution given by A and $OPT(I)$ is the cost of an optimal solution.

Tree alignment was proved to be NP-hard [12]. Many algorithms have been proposed for tree alignment. Sankoff gave an algorithm for computing an optimal solution. The running time is exponential in the number of sequences [8, 9]. 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, Cedergren and Lapalme gave an iterative improvement method to speed up the computation [9, 10]. Waterman and Perlwitz devised a heuristic method when the sequences are related by a binary tree [14]. Hein proposed a heuristic method based on the concept of a *sequence graph* [5]. Ravi and Kececioglu designed an approximation algorithm with performance ratio $\deg + 1/\deg - 1$ when the given tree is a *regular deg-ary* tree (i.e., each internal node has exactly deg children) [7].

The first approximation algorithm with a guaranteed performance ratio was devised by Wang, Jiang, and Lawler [13]. The algorithm lifts chosen sequences associated with leaves to the internal nodes. The performance ratio of the algorithm is 2, and the time complexity is shown to be $O(k^3 + k^2n^2)$ in [13] where n is the length of the sequences. The algorithm was then extended to a polynomial time approximation scheme (PTAS); that is, the performance ratio could arbitrarily approach 1. The PTAS requires computing exact solutions for depth-t subtrees. For a fixed t, the performance ratio was proved to be $1 + 3/t$, and the running time was proved to be $O((k/\deg^t)^{\deg^{t-1}+2}M(2, t-1, n))$ time, where deg is the degree of the given tree, and $M(\deg, t-1, n)$ is the time needed to optimally align a tree with $\deg^{t-1} + 1$ leaves, which is upper-bounded by $O(n^{\deg^{t-1}+1})$. Based on the analysis, to obtain a performance ratio less than 2, exact solutions for depth-4 subtrees must be computed, and thus optimally aligning 9 sequences at a time is required. This is impractical even for sequences of length 100.

In this paper, we propose a new PTAS for the case where the given tree is a regular deg-ary tree. The new algorithm is much faster than the one in [13]. The algorithm also must do local optimizations for depth-t subtrees. For a fixed t, the performance ratio of our new PTAS is $1 + 2/t - 2/t2^t$ and the running time is $O(\min\{2^t, k\}kdM(\deg, t-1, n))$, where d is the depth of the tree. The performance ratios and running time for different t's are listed in Table I. Presently, there are efficient programs [10] to perform local optimizations for three sequences ($t = 2$). In fact, we can expect to obtain optimal solutions for 5 sequences ($t = 3$) of length 200 in practice since there is such a program [3, 6] for SP-score and similar techniques can be used to attack tree alignment problem. Therefore, our

TABLE I
Performance Ratios and Running Time for New and Old Algorithms when the Given Tree Is a Binary Tree. Note That the Running Time of the Old Algorithms Is Based on Theorem 1

t	2	3	4	5	6	7
New ratio	1.75	1.58	1.47	1.39	1.33	1.28
Old ratio	2.5	2.0	1.75	1.60	1.50	1.43
Time of new alg.	$O(kdn^3)$	$O(kdn^5)$	$O(kdn^9)$	$O(kdn^{17})$	$O(kdn^{33})$	$O(kdn^{65})$
Time of old alg.	$O(k^3n^3)$	$O(k^5n^5)$	$O(k^9n^9)$	$O(k^{17}n^{17})$	$O(k^{33}n^{33})$	$O(k^{65}n^{65})$

analysis implies that solutions with a cost at most 1.583 times the optimum can be obtained in practice for strings of length 200. Our new analysis shows that the PTAS in [13] actually has the same performance ratio for any tree with bounded degree. Nevertheless, the new algorithm is much faster. We also propose a ratio-two algorithm running in $O(kd + k\,dn^2)$ time. Similar to our time complexity analyses for the new algorithms, we can reduce the time complexity of the old algorithms in [13] by an order of one. The proof is left to readers. We state the theorem as follows:

THEOREM 1. *The ratio-two algorithm in* [13] *runs in time* $O(k^2 + k^2n^2)$ *and the PTAS in* [13] *runs in time* $O((k)^{\deg^{t-1}+1}M(2, t-1, n))$ *for a fixed* t.

The paper is organized as follows: Section 2 gives the ratio-two algorithm for binary trees. Section 3 discusses the new PTAS and its performance. Section 4 shows that the same ratio also holds for the PTAS in [13] and it works for any bounded degree tree.

2. THE MORE EFFICIENT RATIO-TWO ALGORITHM FOR BINARY TREES

The basic idea of our algorithm is similar to that in [13]. We also lift chosen descendent leaves to the internal nodes. However, we use a more restricted lifting method.

Let T be the given tree, where each of its leaves is labeled with a given sequence. A *loaded* tree T' for T is a tree such that each of its nodes is labeled with some sequence. $C(T')$ denotes the cost of the loaded tree T'. Let v be an internal node of T. T_v denotes the subtree rooted in v. Let $V(T)$ denote the set of internal nodes of T, and $L(T)$ the set of leaves in T. We use $V(T, i)$ to denote the set of nodes of height i in T, and r to denote the root of T. A node is of *height* i if it is i levels above the bottom level.

A *full* binary tree of depth d is a binary tree of depth d having $2^d - 1$ nodes. Given an arbitrary binary tree T, where each of its leaves is labeled with a given sequence, we can pad T to form a full binary tree $\hat{T}$ by extending the leaves in T to subtrees with identical labels on their leaves. Obviously, given a loaded tree for T, there is a loaded tree with the same cost for $\hat{T}$. Thus we assume the given tree is a full binary tree in the analysis. A *lifted alignment* is a tree alignment where each internal node v is assigned one of the sequences that was assigned to the children of v. Thus each node v is assigned a sequence from among the leaf sequences, and if node v is assigned sequence s, then there is a path from v to a leaf in v's subtree where every node on the path is assigned s. Call such a path the *zero-cost* path of v. We say that sequence s has been lifted to node v.

The definition of a lifted alignment allows node v to be assigned any sequence assigned to v's children, without regard to how other nodes at v's level are assigned sequences from their children. We now restrict the kind of lifted alignment permitted. For computational purposes, we assume that the layout of the tree is fixed, so that each node has a fixed left and right child. A *uniform lifted alignment in a binary tree* is one where, at each level of the tree, either every node at that level is assigned the sequence of its right child, or every node at that level is assigned the sequence of its left child. Hence, to specify a uniform lifted alignment, we only specify at each level one bit of information (left or right lift). Therefore, in a binary tree of depth d with fixed layout, there are only 2^d possible uniform lifted alignments that load the tree. Moreover, since there is exactly one leaf sequence s that is lifted all the way to the root in any lifted alignment, and the path from leaf s to the root is unique, the corresponding uniform lifted alignment is completely determined by the sequence assigned (lifted) to the root. The loaded tree obtained in this way is called the *uniform lifted tree* for T, and such a lifting method is called the *uniform lifting method*.

Although uniform lifted alignments (for a fixed layout of the tree) look very restrictive compared to arbitrary lifted alignments, we will show that the optimal uniform lifted alignment has a cost at most twice that of the optimal tree alignment (not required to be lifted).

Let $T^{\min}$ be an optimal loaded tree for T. For each node v, $S(v)$ denotes the set of given sequences assigned to the descendent leaves of v. Let μ_v^s be the cost of the path in $T^{\min}$ from the internal node v to its descendent leaf which is labeled with the given sequence $s \in S(v)$. We use $l(T^s)$ to denote the uniform lifted tree determined by the leaf s. Let v_1 and v_2 be the children of v in T. $l(v^s)$ is the leaf labeled identically to v and $\gamma(v^s)$ is the leaf labeled identically to the label of v's child labeled differently from v. Note that $l(v^s)$ and $\gamma(v^s)$ are in the same position in T_{v_i}'s ($i = 1, 2$). Consider the edge (v, v_i) ($i = 1$ or 2), where the two ends are assigned two different sequences $l(v^s)$ and $\gamma(v^s)$. From the triangle inequality, the cost of edge (v, v_i) in the uniform lifted tree $l(T^s)$ (the edit distance between $\gamma(v^s)$ and $l(v^s)$) is less than or equal to $\mu_v^{l(v^s)} + \mu_v^{\gamma(v^s)}$. Thus the cost of a uniform lifted tree is bounded by the following inequality:

$$C(l(T^s)) \leq \sum_{v \in V(T)} \mu_v^{l(v^s)} + \mu_v^{\gamma(v^s)}. \tag{1}$$

(See Fig. 1.)

Let d be the depth of T. There is a total of 2^d uniform lifted trees. The following lemma bounds the total cost of the 2^d uniform lifted trees.

LEMMA 2.

$$\sum_{s \in S(r)} C(l(T^s)) \leq 2 \sum_{s \in S(r)} \left\{ \sum_{v \in V(T)} \mu_v^{l(v^s)} \right\} = 2 \sum_{s \in S(r)} \left\{ \sum_{v \in V(T)} \mu_v^{\gamma(v^s)} \right\}. \tag{2}$$

Proof. Summing up the costs of all the uniform lifted trees, we have the following inequalities from (1):

$$\begin{aligned} \sum_{s \in S(r)} C(l(T^s)) &\leq \sum_{s \in S(r)} \left\{ \sum_{v \in V(T)} \mu_v^{l(v^s)} + \mu_v^{\gamma(v^s)} \right\} \\ &\leq 2 \sum_{s \in S(r)} \left\{ \sum_{v \in V(T)} \mu_v^{l(v^s)} \right\} = 2 \sum_{s \in S(r)} \left\{ \sum_{v \in V(T)} \mu_v^{\gamma(v^s)} \right\}. \end{aligned} \tag{3}$$

Based on the following observations, the last inequality holds:

(1) In $l(T^s)$, there are two paths from each $v \in V(T)$ to the two of v's descendent leaves, $l(v^s)$ and $\gamma(v^s)$. (See Fig. 1b.)

(2) $C(l(T^s)) = C(l(T^{s'}))$, where $s' = \gamma(r^s)$, and r is the root of T. ∎

Let $C(T^{\min})$ be the cost of $T^{\min}$. To bound (2) in terms of $C(T^{\min})$, we need the following lemma which can be proved by induction on the depth of the tree [13].

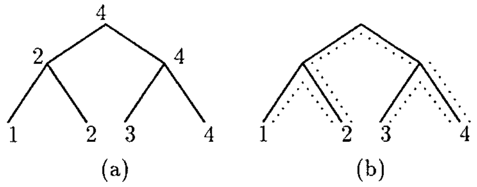

FIG. 1. (a) The uniform lifted tree; (b) the bound of the cost.

LEMMA 3. *Let T be a tree such that every internal node has exactly two children.* (i) *T can be decomposed into a set of edge-disjoint paths, one for each internal node*; (ii) *besides, there is an unused path from the root of T to a leaf* (*called* unused *leaf*) *that is edge-disjoint with all the preceding paths.*

By induction on the depth of the tree, we can show that all the paths $\mu_v^{l(v^s)}$ and $\mu_v^{\gamma(v^s)}$ in (2) can be arranged to form 2^d mappings described in Lemma 3. Thus we can bound the 2^d uniform lifted trees as follows:

LEMMA 4.

$$\sum_{s \in S(r)} C(l(T^s)) \le 2 \sum_{s \in S(r)} \left\{ \sum_{v \in V(T)} \mu_v^{l(v^s)} \right\} \le 2 \times 2^d C(T^{\min}).$$

From Lemma 4, we know that the average cost of the 2^d uniform lifted trees is at most $2C(T^{\min})$. Thus we can immediately conclude:

COROLLARY 5. *There exists a uniform lifted tree with a cost at most twice the optimum.*

Now, let us focus on the computation of an optimal uniform lifted tree. We will first give an algorithm that works for full binary trees and then generalize it to work for an arbitrary tree. Suppose that T is a full binary tree. For each $v \in V(T) \cup L(T)$ and each label s in $S(r)$, $C[v, s]$ denotes the cost of the uniform lifted tree $l(T_v^s)$. We can compute $C[v, s]$ recursively. For each leaf v, we define $C[v, s_i] = 0$ if the label of v is s_i. Let v be an internal node, and v_1, v_2 its children. Suppose $s_i \in S(v_p)$ and $s_i' \in S(v_q)$, where $1 \le p \le 2$, $q \in \{1, 2\} - \{p\}$, and s_i and s_i' are in the same position of T_{v_i}'s ($i = 1, 2$). Then $C[v, s_i]$ can be computed as follows:

$$C[v, s_i] = C[v_p, s_i] + C[v_q, s_i'] + \text{dist}(s_i, s_i'), \tag{4}$$

where $\text{dist}(s_i, s_i')$ is the edit distance between s_i and s_i'. Since the sizes of both $V(T) \cup L(T)$ and $S(r)$ are bounded by $O(k)$, we can compute all the $C[r, s_i]$'s in $O(k^2)$ time if the pairwise edit distances have been computed. A better bound can be obtained by a more careful analysis. A pair of two sequences (s, s') is a *legal pair* if s and s' are assigned at the ends of an edge in a uniform lifted tree. It is easy to see that a sequence s can form at most d legal pairs, where d is the depth of the tree. Thus there are most kd legal pairs in total. Therefore, the running time of our new algorithm is $O(kd + kdn^2)$, where n is the length of the given sequences.

The algorithm we have proposed works only for full binary trees. For an arbitrary binary tree T, Eq. (4) may not make sense. For instance, the given tree T is shown in Fig. 2. To lift the leaf s_i to v, there is no unique s_i'.

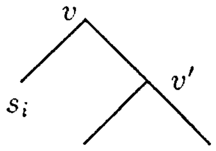

FIG. 2. The case where Eq. (2) does not make sense, since s corresponds to a subtree rooted at v' instead of a leaf s'.

Now we give an efficient method to compute an optimal uniform lifted tree for arbitrary binary trees in $O(kd + kdn^2)$ time. Given an arbitrary binary tree T, consider the internal nodes of T level by level bottom up. Suppose the values $C[v, s]$'s for nodes of height $i - 1$ are known. To compute $C[v, s]$'s for the nodes of height i, we first construct an *extended* tree $TE(v)$ for v as follows (Fig. 3 gives an example):

(1) Overlay the two subtrees T_{v_1} and T_{v_2}, where v_1 and v_2 are the two children of v, such that the roots of the subtrees T_{v_1} and T_{v_2} are matched.

(2) Label the leaves of the obtained tree (supertree) with pairs of sequences. The pairs of sequences are constructed in such a way that if the leaf labeled with the pair of sequences appears in the jth tree T_{v_j}, the jth component of the pair is the corresponding label in T_{v_j}, otherwise, it is the label in T_{v_j} corresponding to the ancestor of the new added leaf.

Note that each pair of sequences contains a label $s \in S(v_j)$ $(j = 1, 2)$, which appears exactly once in $TE(v)$. Thus the total number of leaves in $TE(v)$ is bounded by $|S(v_1)| + |S(v_2)|$. Obviously, the extended tree $TE(v)$ can be computed in $O(|S(v_1)| + |S(v_2)|)$ time. Once we have the extended tree $TE(v)$ for v, we can modify the trees T_{v_1} and T_{v_2} such that they have the same structure as $TE(v)$ and label the new leaves with the labels of their ancestors in T_{v_1} and T_{v_2}. (See Fig. 4.)

Now all the subtrees have the same structure, and we can compute $C[v, s]$'s using Eq. (4). The pairs of sequences in $TE(V)$ provide s_i and s_i'

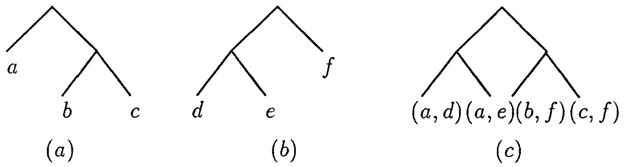

FIG. 3. (a) The tree T_{v_1}; (b) the tree T_{v_2}; (c) the extended tree $TE(v)$.

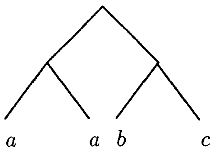

FIG. 4. The modified tree for T_{v_1} in Fig. 3a. The extended tree is in Fig. 3c.

needed in (4). Note that different leaves might be labeled with the same labels in the modified tree. Thus we may have more than one value of $C[v,s]$ for a fixed v and s. In this case, we select the smallest one.

We should emphasize that each $TE(v)$ is constructed directly from T rather than the modified trees. This is crucial to keep the low time complexity $O(kd + kdn^2)$. The algorithm is given in Fig. 5. The algorithm can be generalized to work for regular deg-ary trees. The performance ratio also holds.

THEOREM 6. *Algorithm* 1 *computes a loaded tree with a cost at most twice the optimum in* $O(kd + kdn^2)$ *time for any given regular deg-ary tree.*

3. THE IMPROVED PTAS

In this section, we design a new PTAS by considering constant-size subtrees of the given tree and extending the uniform lifting technique. Our new PTAS works for any regular deg-ary trees, and is much more efficient than the one in [13]. For simplicity, we focus on binary trees.

```
1. begin
2. for each legal pair $(i,j)$, $1 \le i < j \le k$, 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.     begin
7.       Construct the extended tree $TE(v)$
8.         for each label $s$ in the modified tree for $T_{v_j}$ $(j = 1, 2)$,
           where $v_j$'s are the children of $v$
9.           Compute $C[v, s]$ and select the smallest if $C[v, s]$ is not unique.
10.      end
11. Select an $s \in S(r)$ such that $C[r, s]$ is minimized.
12. Compute the lifted tree with the cost $C[r, s]$ by back-tracing.
13. end.
```

FIG. 5. Algorithm 1.

Let v be an internal node of T. $T_{v,t}$ denotes the depth-t subtree of T_v containing only the top $t+1$ levels of nodes. For a binary tree, $T_{v,t}$ contains at most 2^t leaves. Let $l(T^s)$ be the uniform lifted tree uniquely determined by the sequence s. Let $T^s_{v,t}$ be the loaded tree for $T_{v,t}$ such that each node in $T_{v,t}$ is labeled as in $l(T^s)$. Let $Z(v,t)^s$ be the set of nodes on the zero-cost path of v in $T^s_{v,t}$. Let $\hat{T}^s_{v,t}$ be a loaded tree for $T_{v,t}$ obtained by assigning each node u in $L(T_{v,t}) \cup \{v\} \cup Z(v,t)^s$ the label assigned to u in $T^s_{v,t}$ and constructing the sequences for other nodes in $T_{v,t}$ such that the cost of the subtree is minimized. Obviously, the cost of $\hat{T}^s_{v,t}$ is less than or equal to the cost of $T^s_{v,t}$. This hints that we should partition the given tree T into depth-t subtrees and construct an optimal loaded subtree $\hat{T}^s_{v,t}$ for each $T_{v,t}$.

For a fixed t, we construct t different partitions $P_0, P_1, \dots, P_{t-1}$. For a fixed choice of q $(0 \le q < t)$, the top slice of partition P_q contains q levels of edges in T. Each slice below the top contains exactly t levels of edges in T, until less than t levels of edges remain. (See Fig. 6.) The number of edge levels in the bottom slice is denoted p_q. Note that two consecutive slices share a common level of nodes. A subtree $T_{v,t'}$ is *obtained* from partition P_q if v is of height $p_q + jt$ for some integer $j \ge 0$ and $t' = q$ for the top slice, $t' = p_q$ for the bottom slice, and $t' = t$ for other slices.

Let $l(T^s)$ be the uniform lifted tree uniquely determined by the sequence s. A *reserved* node of $l(T^s)$ for partition P_q is defined to be a node in

$$\bigcup_{T_{v,t'}} \left(L(T_{v,t'}) \cup \{v\} \cup Z(v,t')^s \right), \tag{5}$$

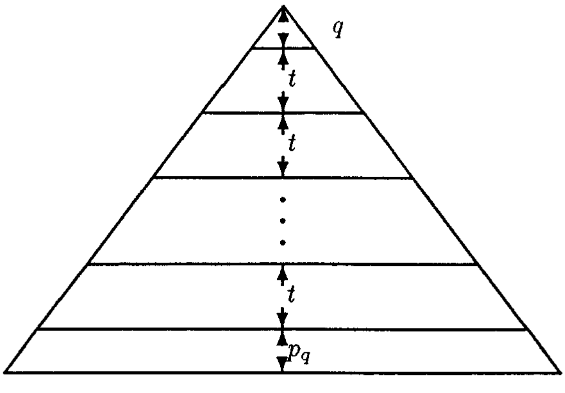

FIG. 6. The partition P_q.

where $T_{v,t'}$'s are the subtrees *obtained* from partition P_q, and $L(T_{v,t'})$ is the set of leaves in tree $T_{v,t'}$. Note that the top subtree $T_{r,q}$ is not included unless $q = 0$. When $q = 0$, the top subtree is actually $T_{r,t}$. Let $T(q, s)$ be the loaded tree for T obtained by assigning each reserved node u of $l(T^s)$ for partition P_q the label assigned to u in $l(T^s)$ and constructing the sequences for other nodes in T such that the cost of the tree is minimized.

Again, we assume that the given tree is a full binary tree for analysis. Let d be the depth of T. For a full binary tree, there are 2^d leaves in $L(T)$, thus 2^d s's. We use $s_1, s_2, \dots, s_{2^d}$ to denote these s's. We will show that the total cost of the $t \times 2^d$ loaded trees $T(q, s)$ is bounded as follows:

$$\sum_{q=0}^{t-1} \sum_{i=1}^{2^d} C(T(q, s_i)) \le \{t \times 2^d + 2^d + 2^{d-t}(2^t - 2)\} C(T_{\min}).$$

Therefore, there is a loaded tree $T(q, s_i)$ with cost at most $1 + 2/t - 2/t2^t$ times the optimum. The loaded tree obtained in this way is called a uniform *lifted* t-tree. To compute an optimal uniform lifted t-tree, it is not necessary to check all the $t \times 2^d$ loaded trees. Similarly, we design an efficient dynamic programming algorithm.

Let $C[v, t, s, s_1, s_2, \dots, s_m]$ be the cost of $\hat{T}^s_{v,t}$, where s_i $(i = 1, 2, \dots, m)$ is the sequence assigned to the leaf $v_i \in L(T_{v,t})$ in $\hat{T}^s_{v,t}$. Let v be a node of height l, where $l = p_q + jt$ for some $j \ge 0$. Let $T(q, s, v)$ be the loaded subtree of $T(q, s)$ rooted at v, and $T(q, s_i, v_i)$ be the loaded subtree of $T(q, s_i)$ rooted at v_i. Note that $T(q, s_i, v_i)$ is identical to $T(q, s, v_i)$ since s_i is the sequence assigned to the leaf $v_i \in L(T_{v,t})$ in $\hat{T}^s_{v,t}$. Let $C[v, s]$ denote the cost of $T(q, s, v)$, and $C[v_i, s_i]$ denote the cost of $T(q, s_i, v_i)$. Similar to Eq. (4), $C[v, s]$ can be computed as follows:

$$C[v, s] = C[v, t, s, s_1, s_2, \dots, s_m] + \sum_{i=1}^{m} C[v_i, s_i]. \tag{6}$$

Again, similar to the ratio-two algorithm in Section 2, the concept of an extended tree $TE(v)$ for v gives an efficient implementation. Here $TE(v)$ is defined differently. To construct $TE(v)$ for a fixed t, we consider the 2^t leaves in subtree $T_{v,t}$ instead of the two children of v. To compute $C[v, s]$ for a v of height i, we (i) pad $T_{v,t}$ to form a full binary tree, that is, a binary tree with 2^t leaves; (ii) construct the *extended* tree $TE(v)$; (iii) modify the subtrees T_{v_i} for all $v_i \in L(T_{v,t})$ so that they have the structure of $TE(v)$ and the leaves are labeled with the corresponding sequences (some leaves may have identical sequences); and (iv) use Eq. (6) to compute $C[v, s]$.

Let $C(T(q, s))$ be the cost of the loaded tree $T(q, s)$. $C(T(q, s))$ can be computed from the top subtree $T_{r,q}$ and the values $C[v, s]$, where $v \in$

$L(T_{r,q})$, in a way similar to $C[v, s]$. Finally, we select a $C(T(q, s))$ with the smallest value among all the $t \times k$ pairs (q, s). The algorithm is given in Fig. 7. It can be shown that the running time of the algorithm is $O(\min\{2^t, k\}kdM(2, t-1, n))$, where $M(2, t-1, n)$ is the time needed to optimally align a tree with $2^{t-1}+1$ leaves which is upper-bounded by $O(n^{2^{t-1}+1})$.

Now let us focus on the performance ratio analysis. Again, we pad T to form a full binary tree. Let $T_{v,t}^{\min}$ be the subtree of $T^{\min}$ rooted at v with a depth t. Let us consider the depth-t subtree $T_{v,t}$. Let s_i $(i = 1, 2, \ldots, 2^t)$ be the sequence assigned to the nodes $v_i \in L(T_{v,t})$ in $T_{v,t}^s$, and $s = s_p$ $(1 \le p \le 2^t)$ the sequence lifted to v as well as the nodes on the zero-cost path of v in $T_{v,t}^s$. If we assign the sequence in $T_{v,t}^{\min}$ to the rest of the nodes in $T_{v,t}$, the cost of the obtained subtree will not be less than $C[v, t, s, s_1, s_2, \ldots, s_{2^t}]$. Using the triangle inequality, it is easy to see that $C[v, t, s, s_1, s_2, \ldots, s_{2^t}]$ is bounded as follows:

$$C[v, t, s, s_1, s_2, \ldots, s_{2^t}] \le C\left(T_{v,t}^{\min}\right) + \sum_{v_i \in L(T_{v,t})} \mu_{v_i}^{s_i} + \sum_{j=1}^{t-1} \mu_{n_j}^{s}, \tag{7}$$

where n_j is the node on the zero-cost path of v in $T_{v,t}^s$. (See Fig. 8.) The thick lines form the $T_{v,t}^{\min}$, the dashed lines stand for the paths from v_i to its descendent leaves, each of which is assigned s_i in $T^{\min}$, and the dotted lines stand for the paths from n_j to its descendent leaf which is assigned $s = s_p$ in $T^{\min}$.

```
1. begin
2. for each level, with the bottom level first, do
3.    for each node $v$ at the level do
3.    begin
4.        Construct the extended tree $TE(v)$
          and extend $T_{v_j}$ according to $TE(v)$ for each $v_j \in L(T_{v,t})$
6.        for each label $s$ in the modified tree for $T_{v_j}$ $(j = 1, 2, \ldots)$,
          where $v_j \in L(T_{v,t})$ is the leaf of tree $T_{v,t}$.
7.          Compute $C[v, s]$ and select the smallest if there are several values for $C[v, s]$.
8.    end
9. for $q = 0$ to $t-1$ do
10.    for $s \in S(r)$ do
11.       Compute $C(T(q, s))$.
12. Select a $C(T(q, s))$ with the smallest value.
13. Compute the loaded tree from $C(T(q, s))$ by back-tracing.
14. end.
```

FIG. 7. Algorithm 2.

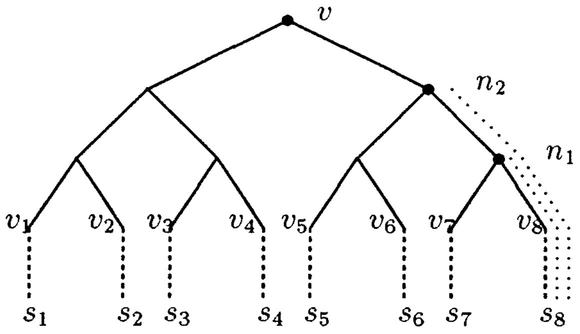

FIG. 8. The bound of the cost for the subtree $T_{v,t}$, where $t = 3$.

We can lift any $s \in \{s_1, s_2, \ldots, s_{2^t}\}$ to v, the root of $T_{v,t}$. The total cost of those 2^t different loaded subtrees is bounded by the following inequality:

LEMMA 7.

$$\sum_{s \in \{s_1, \ldots, s_{2^t}\}} C[v, t, s, s_1, s_2, \ldots, s_{2^t}] \le 2^t C\left(T_{v,t}^{\min}\right) + 2^t \sum_{v_i \in L(T_{v,t})} \mu_{v_i}^{s_i} \tag{8}$$

$$+ \sum_{i=1}^{t-1} \sum_{v' \in V(T_{v,t}, i+1)} \sum_{v_j \in L(T_{v',i})} \mu_{v'}^{s_{j_i}}. \tag{9}$$

Let $d = q + k_q t + p_q$ be the depth of T, where $0 \le p_q, q < t$. The total cost of the 2^d loaded trees is bounded as follows:

LEMMA 8.

$$\sum_{i=1}^{2^d} C(T(q, s_i)) \le 2^d C(T^{\min}) \tag{10}$$

$$+ \sum_{j=0}^{k_q} \sum_{v \in V(T, p_q + (k_q - j)t)} \sum_{s \in S(v)} 2^{jt+q} \mu_v^s \tag{11}$$

$$+ \sum_{j=0}^{k_q} \sum_{v \in V(T, p_q + (k_q - j)t)} \sum_{l=1}^{x} \sum_{v' \in V(T_{v,t}, l)} \sum_{s \in S(v')} 2^{jt+q} \mu_{v'}^s, \tag{12}$$

where $x = p_q - 1$ *if* $j = 0$, *otherwise* $x = t - 1$.

Proof. The term $2^d C(T^{\min})$ in (10) is from the term $2^t C(T_{v,t}^{\min})$ in (8). Equation (11) is obtained from the term

$$2^t \sum_{v_i \in L(T_{v,t})} \mu_{v_i}^{s_i}$$

in (8) based on the following observation: there are $q + jt$ levels of nodes above the nodes of height $p_q + (k_q - j)t$. Thus, once the nodes of height $p_q + (k_q - j)t$ are assigned sequences, there are 2^{jt+q} choices left.

Note that we lift the sequences for every t level. Thus, for nodes of height l, where $p_q + (k_q - j - 1)t < l < p_q + (k_q - j)t$, we have 2^{jt+q} choices left, too. From (9), the total cost of dotted lines in Fig. 8 is bounded by the term in (12). ∎

Lemma 8 considers 2^d loaded trees for a fixed choice of q. Summing up the $t2^d$ loaded trees for all the t choices of q, we have:

LEMMA 9. *The total cost of the $t2^d$ uniform lifted t-trees is bounded by the following inequality*:

$$\sum_{q=0}^{t-1} \sum_{i=1}^{2^d} C(T(q, s_i)) \leq t2^d C(T^{\min}) \tag{13}$$

$$+ \sum_{l=1}^{d-1} \left\{ 2^{d-l} \sum_{v \in V(T,l)} \sum_{s \in S(v)} \mu_v^s \right\} \tag{14}$$

$$+ \sum_{l=1}^{d-1} \left\{ 2^{d-l-t}(2^t - 2) \sum_{v \in V(T,l)} \sum_{s \in S(v)} \mu_v^s \right\}. \tag{15}$$

Proof. The terms in (13) and (14) are from (10) and (11), respectively. The term in (15) comes from (12). Note that there is a coefficient 2^{q+jt} for each $\mu_{v'}^s$ in (12), where v' is of height l, and $p_q + (k_q - j - 1)t < l < p_q + (k_q - j)t$. For a fixed node, when q changes, the coefficient 2^{q+jt} changes. For a node of height l, the total coefficient for all q's should be

$$2^{d-l-1} + 2^{d-l-2} + \cdots + 2^{d-l-t+1} = 2^{d-t-l}(2^t - 2). \tag{16}$$

In (16) we add $t - 1$ terms. The reason is that, in Lemma 8, l takes at most $t - 1$ values in (12). ∎

Each node v of height l has 2^l descendent leaves, each of which corresponds to a path from v to the leaf. Also, there is a coefficient 2^{d-l} for each μ in (14). Thus there is a total of 2^d paths from v to its descendent leaves in (14). (Each path is repeated 2^{d-l} times.) From

Lemma 4, all those paths in (14) can be arranged to form 2^d different mappings of $T^{\min}$ described in Lemma 3. Therefore, the μ's in (14) and (15) can be bounded in terms of $C(T^{\min})$ as follows:

LEMMA 10.

$$\sum_{l=1}^{d-1}\left\{2^{d-l}\sum_{v\in V(T,l)}\sum_{s\in S(v)}\mu_v^s\right\}\le 2^d C(T^{\min})$$

and

$$\sum_{l=1}^{d-1}\left\{2^{d-l-t}(2^t-2)\sum_{v\in V(T,l)}\sum_{s\in S(v)}\mu_v^s\right\}\le 2^{d-t}(2^t-2)C(T^{\min}).$$

LEMMA 11. *There exists a uniform lifted t-tree with a cost at most* $(1+2/t-2/t2^t)C(T^{\min})$.

Proof. Combining Lemmas 9 and 10, we can obtain

$$\sum_{q=0}^{t-1}\sum_{i=1}^{2^d}C(T(q,s_i)\le\{t\times 2^d+2^d+2^{d-t}(2^t-2)\}C(T^{\min}).$$

Thus the average cost of those $t2^d$ $T(q,s_i)$'s is $(1+2/t-2/t\times 2^t)$ $C(T^{\min})$. Therefore, there is a lifted t-tree with a cost at most $(1+2/t-2/t2^t)C(T^{\min})$. ∎

THEOREM 12. *Algorithm* 2 *computes a loaded tree with a cost at most* $1+2/t-2/t2^t$ *times the optimum in* $O(\min\{2^t,k\}kdM(2,t-1,n))$ *time, where k is the number of leaves in the given tree, n is the length of the sequences, and* $M(2,t-1,n)$ *is the time needed to optimally align a tree with* $2^{t-1}+1$ *leaves which is upper-bounded by* $O(n^{2^{t-1}+1})$.

Algorithm 2 can be easily generalized to work for the case where the given tree is a regular deg-ary tree. The same performance ratio can be obtained by considering $t\times\deg^d$ different loaded trees. However, we do not know how to generalize the uniform lifting method for the case where the given tree has a bounded degree (i.e., internal nodes could have a different number of children) since we cannot pad a tree to form a regular deg-ary tree. Nevertheless, we can show that the algorithm in [13] has the same performance ratio.

4. TREES OF BOUNDED DEGREE

Now we show that the old algorithm designed in [13] has the same performance ratio when the degree of the given tree is bounded by a constant deg. The basic idea of the old algorithm in [13] is similar to the algorithm presented in Section 3. The only difference is that the lifting choices are not uniform; that is, we have to try all the descendent leaves of v_i in Eq. (6) instead of the fixed s_i. Thus the computation of each $C[v, s]$ needs $O(k^{2^{t-1}+1})$ time. That is why the old algorithm has a much higher time complexity and works for the bounded degree case.

For analysis, we can pad the given tree T such that all the leaves in T are at the same level. We use $\deg(v)$ to denote the degree of node v. Consider a tree with a degree bounded by a constant deg. Figure 9 shows the bound of the cost for a lifted subtree with a depth t.

Four kinds of edges are considered:

(1) The thick edges form an optimal tree. The cost is obviously bounded by $C(T^{\min})$.

(2) The dashed lines starting at a node v form the rightmost thick lines in the subtree rooted at v. Thus the dashed lines are not counted.

(3) The thin lines are paths from boundary nodes to their descendent leaves. There are $\deg(v) - 1$ such paths for each boundary node.

(4) The dotted edges are paths from internal nodes to their descendent leaves. Only the nodes in the paths from a root of a depth-t subtree to a leaf of the depth-t subtree, which is lifted, have such paths. Each path is repeated exactly deg $- 1$ times.

Recall that we have considered $t2^d$ different loaded trees in Lemma 9, where d is the depth of the given tree and t is the depth of the subtrees for the local optimizations. To obtain the desired performance ratio for the case where the degree is bounded by deg, we also consider $t2^d$ different loaded trees here. Note that an internal node could have more

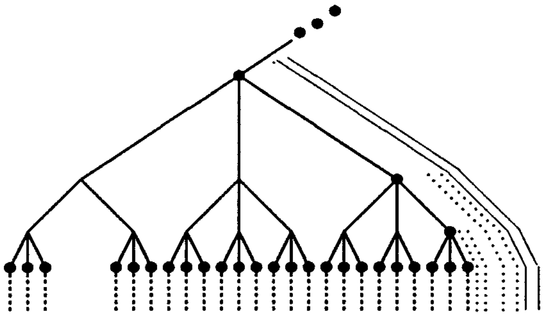

FIG. 9. The bound of the cost for a subtree with a depth t ($t = 3$) and a degree bounded by 3. The circled nodes are labeled with lifted sequences and the rest nodes are labeled with sequences in the optimal tree $T^{\min}$.

than 2 children now. For each internal node v, we select two children among its $\deg(v)$ children in the specific way shown in Fig. 10. A node selected in this way is called a *selected* node.

Figure 11 gives an example of how to select the nodes. The thick edges link the selected nodes. Note that all the selected nodes form a forest instead of a tree. In the forest, there is a depth-d tree which is called the *main* tree. In our analysis, we only count the number of ways to lift the sequences using the uniform lifting method for the main tree. Therefore, we have 2^d different loaded trees for each fixed partition, too. For an internal node v, we use $\mathrm{SS}(v)$ to denote the set of sequences assigned to v's selected descendent leaves. Note that we also have to lift the leaves to the internal nodes not in the main tree. To do this, we can generalize the uniform lifting method. Let $v_1, v_2, \ldots, v_m$ be the internal nodes of the same height, and v_1 one of the selected nodes in the main tree. Suppose i is a selected leaf of T_{v_1} in the main tree. Lifting i to v_1, we also have to lift the selected leaves in the same position of i from subtrees T_{v_j}'s $(j = 2, \ldots, m)$ to v_j's. Similarly, we define the set of reserved nodes for fixed partition P_q and s as in Eq. (5). Define $T(q, s)$ to be the loaded tree for T obtained by assigning each reserved node u for fixed partition P_q and s the label assigned to u in the lifted tree (lift s to the root) and constructing the sequences for the other nodes such that the cost of the tree is minimized.

Similar to Lemma 9, we can bound the total cost of those $t2^d$ loaded trees in terms of $C(T^{\min})$ and μ's.

LEMMA 13.

$$\sum_{q=0}^{t-1} \sum_{i=1}^{2^d} C(T(q, s_i)) \leq t \times 2^d C(T^{\min}) + \sum_{i=1}^{d} \left(2^{d-i} \sum_{v \in V(T, i+1)} \sum_{s \in \mathrm{SS}(v)} (\deg(v) - 1) \mu_v^s \right) + \sum_{i=1}^{d-t} \left(2^{d-i-t}(2^t - 2) \sum_{v \in V(T, i+1)} \sum_{s \in \mathrm{SS}(v)} (\deg(v) - 1) \mu_v^s \right). \quad (17)$$

Lemma 3 can be easily generalized as follows.

LEMMA 14. *Let T be a tree such that every internal node v has* $\deg(v) \geq 2$ *children. $T^{\min}$ can be decomposed into paths such that* (i) *there are exactly* $\deg(v) - 1$ *paths from each internal node v to its descendent leaves*; (ii) *the paths are edge-disjoint*; *and* (iii) *besides, there is an extra path from the root of T to a leaf that is edge-disjoint with all the preceding paths*.

1. Let v be an internal node of height 1 with children v_1, v_2, ..., $v_{deg(v)}$.
   We select v_{i_1} and v_{i_2} such that the costs of edges (v, v_{i_1}) and (v, v_{i_2}) in T^{min} are not greater than any other edges between v and its children.
2. Consider a node of height $j > 1$.
   Suppose all the children v_1, ..., $v_{d(v)}$ of v have two selected children.
   For each node v_i, define ρ_i to be the total cost of paths
   from v_i to all its selected descendent leaves.
   We select two v_i's with the smallest ρ's.

FIG. 10. The way to select the nodes.

From Lemmas 13 and 14, we can immediately conclude the following theorem.

THEOREM 15. *For a given tree with a degree bounded by a constant, there is a loaded tree $T(q, s)$ with a cost at most $1 + 2/t - 2/t2^t$ times the optimum.*

Proof. Using the criterion in the specific way to select the nodes (see Fig. 10), we can bound the cost $(\deg(v) - 1)\mu_v^s$ in (17) with the cost of $\deg(v) - 1$ different paths starting from v. Therefore, in a way similar to Lemma 10, we can prove that the paths in (17) can be decomposed into 2^d and $2^{d-t}(2^t - 2)$ decompositions of $T^{\min}$ described in Lemma 14, respectively. Thus we can obtain the desired performance ratio. ∎

The definition of $T(q, s)$ for a given tree with a degree bounded by a constant deg > 2 depends on the optimal solution $T^{\min}$ which is unknown. However, we know that $T(q, s)$ has the following property: the sequence lifted to v is one of the sequences lifted to its children. Thus we can use the algorithm designed in [13] to compute an optimal $T(q, s)$. Therefore, we have

THEOREM 16. *The performance ratio of the PTAS in* [13] *is* $1 + 2/t - 2/t2^t$ *for a tree with degree bounded by a constant.*

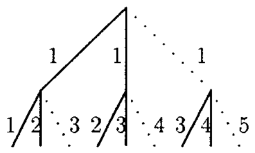

FIG. 11. An example of the specific way to select nodes. The numbers are the costs of edges. The thick lines link the selected nodes.

REFERENCES

1. S. Altschul and D. Lipman, Trees, stars, and multiple sequence alignment, *SIAM J. Appl. Math.* **49** (1989), 197–209.
2. S. C. Chan, A. K. C. Wong, and D. K. T. Chiu, A survey of multiple sequence comparison methods, *Bull. Math. Biol.* **54** (1992), 563–598.
3. S. K. Gupta, J. D. Kececioglu, and A. A. Schaffer, Making the shortest-paths approach to sum-of-pairs multiple sequence alignment more space efficient in practice, *in* ''Combinatorial Pattern Matching 95,'' pp. 128–143.
4. D. Gusfield, Efficient methods for multiple sequence alignment with guaranteed error bounds, *Bull. Math. Biol.* **55** (1993), 141–154.
5. 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** (1989), 649–668.
6. D. J. Lipman, S. F. Altschul, and J. D. Kececioglu, A tool for multiple sequence alignment, *Proc. Nat. Acad. Sci. U.S.A.* **86** (1989), 4412–4415.
7. R. Ravi and J. Kececioglu, Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree, *in* ''Combinatorial Pattern Matching 95,'' pp. 330–339.
8. D. Sankoff, Minimal mutation trees of sequences, *SIAM J. Appl. Math.* **28** (1975), 35–42.
9. D. Sankoff and R. Cedergren, Simultaneous comparisons of three or more sequences related by a tree, *in* ''Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison'' (D. Sankoff and J. Kruskal, Eds.), pp. 253–264, Addison–Wesley, Reading, MA, 1983.
10. D. Sankoff, R. J. 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.
11. D. Sankoff and J. Kruskal, ''Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison,'' Addison–Wesley, Reading, MA, 1983.
12. L. Wang and T. Jiang, On the complexity of multiple sequence alignment, *J. Comput. Biol.* **1** (1994), 337–348.
13. L Wang, T. Jiang, and E. L. Lawler, Approximation algorithms for tree alignment with a given phylogeny, *Algorithmica* **16** (1996), 302–315.
14. M. S. Waterman and M. D. Perlwitz, Line geometries for sequence comparisons, *Bull. Math. Biol.* **46** (1984), 567–577.