SIAM J. Comput.
Vol. 30, No. 1, pp. 283–299

A MORE EFFICIENT APPROXIMATION SCHEME FOR TREE ALIGNMENT*

LUSHENG WANG[†], TAO JIANG[‡], AND DAN GUSFIELD[§]

Abstract. We present a new polynomial time approximation scheme (PTAS) for tree alignment, which is an important variant of multiple sequence alignment. As in the existing PTASs in the literature, the basic approach of our algorithm is to partition the given tree into overlapping components of a constant size and then apply local optimization on each such component. But the new algorithm uses a clever partitioning strategy and achieves a better efficiency for the same performance ratio. For example, to achieve approximation ratios 1.6 and 1.5, the best existing PTAS has to spend time $O(kdn^5)$ and $O(kdn^9)$, respectively, where n is the length of each leaf sequence and d, k are the depth and number of leaves of the tree, while the new PTAS only has to spend time $O(kdn^4)$ and $O(kdn^5)$. Moreover, the performance of the PTAS is more sensitive to the size of the components, which basically determines the running time, and we obtain an improved approximation ratio for each size. Some experiments of the algorithm on simulated and real data are also given.

Key words. approximation algorithm, computational biology, evolutionary tree, multiple sequence alignment, phylogeny

AMS subject classifications. 05C05, 68Q25, 05C90, 81T30, 92B05

PII. S0097539796313507

1. Introduction.

Multiple sequence alignment is one of the fundamental and most challenging problems in computational molecular biology [1, 2, 5, 13]. It plays an essential role in the solution of many problems such as searching for highly conserved subregions among a set of biological sequences and inferring the evolutionary history of a family of species from their molecular sequences. For example, most methods for phylogeny reconstruction based on sequence data assume a given multiple sequence alignment.

An important approach to multiple sequence alignment is the *tree alignment* method. Suppose that we are given k sequences and a *rooted phylogenetic tree* containing k leaves, each of which is labeled with a unique given sequence. The goal is to construct a sequence for each internal node of the tree such that the *cost* of the resulting *fully labeled* tree is minimized. Here, the cost of the fully labeled tree is the total cost of its edges, and the cost of an edge is the *mutational distance* (or *weighted edit distance*) between the two sequences associated with both ends of the edge.

The biological interpretation of the model is that the given tree represents the evolutionary history (known from means other than sequence analysis or postulated

[*] Received by the editors December 13, 1996; accepted for publication (in revised form) August 3, 1999; published electronically May 2, 2000.
http://www.siam.org/journals/sicomp/30-1/31350.html

[†] Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong (lwang@cs.cityu.edu.hk). The work of this author was supported in part by Department of Energy grant DE-FG03-90ER60999 and HK CERG grants 9040444, 9040297, and 9040352. Most of this author's work was completed while the author was at UC Davis.

[‡] Department of Computer Science, University of California, Riverside, CA 92521 (jiang@cs.ucr.edu). The work of this author was supported in part by NSERC research grant OGP0046613, a Canadian Genome Analysis and Technology (CGAT) grant, and a CITO grant. This author's work was completed while the author was visiting the University of Washington and on leave from McMaster University.

[§] Department of Computer Science, University of California, Davis, CA 95616 (gusfield@cs.ucdavis.edu). The work of this author was supported in part by Department of Energy grant DE-FG03-90ER60999.

in a phylogeny reconstruction experiment) which has created the molecular (DNA, RNA, or amino acid) sequence written at the leaves of the tree. The leaf sequences are ones found in organisms existing today, and the sequences to be determined for the internal nodes of the tree represent inferred sequences that may have existed in the ancestral organisms. It should be emphasized that the tree is almost always binary in biological applications.

To see how the above is related to multiple sequence alignment, we demonstrate by an example that the sequences inferred for the internal nodes actually induce a mutiple alignment of the given k sequence. Such a multiple sequence alignment is believed to expose evolutionarily significant relationships among the sequences, according to the maximum parsimony principle.

Example 1. Consider four sequences ACTG, ATCG, GCCA, and GTTA, and the tree shown in Figure 1.1(a) connecting these sequences. Suppose that the internal sequences are reconstructed as in (b). For each edge of the tree in (b) we can construct a pairwise alignment of the two sequences associated with the edge, with a cost equal to the mutational distance of the sequences. Then we can induce a multiple alignment that is *consistent* with all six pairwise alignments by "merging" the pairwise alignments incrementally as in [3], taking the spaces into consideration. For example, from the given pairwise alignments of sequences s_1, h_1 and of sequences s_2, h_1, we can obtain a multiple alignment of s_1, s_2, h_1 as shown in (d). The final induced multiple alignment of the four leaf sequences is shown in (e) (if we ignore everything below the dashed line).

Tree alignment is also a key step in a particular phylogeny reconstruction method called *generalized parsimony* [10, 12]. The basic strategy of generalized parsimony is to start with a tree consisting of two leaves and grow the tree by gradually inserting the rest of the leaves (labeled with sequences) at the edges. To add a leaf, we consider insertion at all possible edges of the current tree and use tree alignment to test which edge will result in a most parsimonious tree (i.e., one with the smallest cost).

Tree alignment is known to be NP-hard [14]. Many heuristic algorithms have been proposed in the literature [1, 7, 8, 10, 11], and some approximation algorithms with guaranteed relative error bounds have been reported recently. In particular, a polynomial time approximation scheme (PTAS) is presented in [15], and an improved version is given in [16]. Both PTASs partition the tree into overlapping constant size components, label the leaves of each such component in some way, and then apply local optimization on each component, i.e., compute an optimal tree alignment for each component. The PTAS in [16] achieves an approximation ratio $1 + \frac{2}{t} - \frac{2}{t \times 2^t}$, i.e., it produces a fully labeled tree with cost at most $1 + \frac{2}{t} - \frac{2}{t \times 2^t}$ times the optimal cost, when the running time is $O(kdn^{2^{t-1}+1})$, where k is the number of the given sequences, n is the length of each given sequence, d is the depth of the given phylogeny, and t is a parameter to control the number of sequences involved in a local optimization as well as the performance ratio. For any fixed t, a local optimization aligns a tree with $2^{t-1} + 1$ leaves (i.e., sequences), which takes $O(n^{2^{t-1}+1})$ time [9]. Thus the more accurate the algorithm is, the more time it consumes. Since in practice n is at least 100, the bottleneck of the time efficiency is the time spent on local optimization. At present, we can expect to optimally align up to eight sequences of length 200 at a time, as demonstrated by the software package MSA for a similar multiple alignment problem [4]. Thus the above PTASs are still far from being practical.

In this paper, we further improve the PTAS in [16]. The new approximation scheme adopts a more clever partitioning strategy and has a better time efficiency for

```
                (h3)                              h3:ACTG
                / \                                  / \
               /   \                                /   \
              /     \                              /     \
             /       \                            /       \
            /         \                          /         \
        (h1)           (h2)                h1:ACTCG     h2:GCTA
         / \           / \                    / \           / \
        /   \         /   \                  /   \         /   \
       /     \       /     \                /     \       /     \
      /       \     /       \              /       \     /       \
s1:ACTG  s2:ATCG  s3:GCCA  s4:GTTA   s1:ACTG  s2:ATCG  s3:GCCA  s4:GTTA
```

(a) The input sequences and tree. (b) The fully labeled tree.

```
 s1: ACT G   s2: A TCG   s3: GCCA   s4: GTTA   h1: ACTCG   h2: GCTA
 h1: ACTCG   h1: ACTCG   h2: GCTA   h2: GCTA   h3: ACT G   h3: ACTG
```

(c) The pairwise alignments.

```
s1: ACT G
h1: ACTCG
s2: A TCG
```

(d) The alignment of s1, s2, and h1.

```
s1                ACT G
s2                A TCG
s3                GCC A
s4                GTT A
----------------------
h1                ACTCG
h2                GCT A
h3                ACT G
```

(e) The induced multiple alignment.

FIG. 1.1. *Tree alignment induces a multiple alignment.*

the same performance ratio. For any fixed r, where $r = 2^{t-1} + 1 - q$ and $0 \leq q \leq 2^{t-2} - 1$, the new PTAS runs in time $O(kdn^r)$ and achieves an approximation ratio of $1 + \frac{2^{t-1}}{2^{t-2}(t+1)-q}$. Here the parameter r represents the "size" of local optimization. In particular, when $r = 2^{t-1}+1$, its approximation ratio is simply $1+\frac{2}{t+1}$. A comparison of the performance of the new PTAS and the PTAS in [16] for small values of t and r is given in Table 1.1. Note that the new PTAS yields an improved approximation ratio for *every* size of local optimization, whereas the previous PTAS does not. This is because the previous partitioning and analysis method works only when the size of local optimization has the form of one plus some power of two, and the new method works for any size. Hence to achieve a ratio 1.5, the new PTAS requires running time $O(kdn^5)$, while the old PTAS would require running time $O(kdn^9)$.

Although the new PTAS is only a small improvement to the previous PTASs in terms of the feasible approximation ratios it can provide, it still represents a concrete

Table 1.1
A comparison of the new PTAS and the best previous PTAS.

r	3	4	5	6	7	8	9
t	2		3				4
Running time	$O(kdn^3)$	$O(kdn^4)$	$O(kdn^5)$	$O(kdn^6)$	$O(kdn^7)$	$O(kdn^8)$	$O(kdn^9)$
Old ratio	1.75		1.58				1.47
New ratio	1.67	1.57	1.50	1.47	1.44	1.42	1.40

step towards constructing a practical approximation algorithm for tree alignment with provable performance bounds, which could be extremely useful in sequence analysis. The new partitioning strategy is also much more flexible and has the potential of leading to even better analytical bounds than what is reported here. We observe that the bounds listed in Table 1.1 do not appear to be tight for either PTAS, and a better analysis technique may also reduce the approximation ratio by a significant amount.

The paper is organized as follows. We review the uniform lifting technique developed in [16] in section 2. Sections 3 and 4 present the new approximation scheme and its analysis, respectively. Finally we describe an implementation of the PTAS and perform some tests on simulated and real data in section 5.

Remark 1.1. There are many ways to define the mutational distance $D(s, s')$ between sequences s and s'. In this paper, we need only assume that the distance is a *metric*. That is, it satisfies

1. *positive definiteness*: $D(s, s') \geq 0$ and $D(s, s) = 0$;
2. *symmetry*: $D(s, s') = D(s', s)$;
3. *triangle inequality*: $D(s, s') \leq D(s, s'') + D(s'', s')$ for any sequence s''.

2. Uniformly lifted trees and the general approach.

Before we discuss the new algorithm, let us introduce some useful concepts and results. Let T be a binary (phylogenetic) tree such that each of its leaves is labeled with a given sequence. For convenience, convert T to an *ordered* tree by specifying the children of each internal as left and right children, arbitrarily. A *loaded* tree for T is a tree in which each internal node is also assigned a sequence label (not necessarily a given sequence). A loaded tree is called a *lifted* tree if the sequence label of every internal node v equals the sequence label of some child of v. Figure 2.1(a) exhibits a lifted tree. A lifted tree is called a *uniformly* lifted tree if, for each level, either every internal node at that level receives its sequence label from its left child or every internal node at that level receives its sequence label from its right child. In other words, the lifting choices for the internal nodes at the same level are uniform. Figure 2.1(b) exhibits a uniformly lifted tree.

Let d denote the depth of the tree T. Then there are 2^d different uniformly lifted trees for T. We use a vector $V = (x_1, \ldots, x_d)$ to denote the uniform lifting choices for all levels, where $x_i = R$ or L corresponds to lifting the sequence of the left child or the right child at ith level. For any vector V, $T(V)$ denotes the uniformly lifted tree for T specified by V. For each internal node v, the path from v to the leaf whose sequence label is lifted to v is called the *lifting path* of v. Observe that, every node on the lifting path of v (including v) is assigned the identical sequence.

The following results are proven in [16]. Let $T^{\min}$ denote an optimal loaded tree for T.

Theorem 2.1. *The average cost of the 2^d uniformly lifted trees for T is at most twice the cost of $T^{\min}$.*

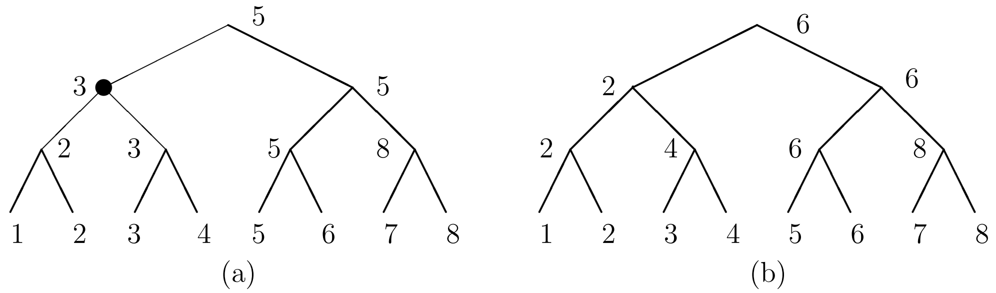

FIG. 2.1. (a) *A lifted tree.* (b) *A uniform lifted tree.*

COROLLARY 2.2. *There exists a uniformly lifted tree for T with a cost at most twice the cost of $T^{\min}$.*

An optimal uniformly lifted tree can be computed in $O(kd + kdn^2)$ time by a straightforward bottom-up dynamic programming, where k is the number of leaves in T and n is the length of each given sequence [16].

Observe that given any lifted tree, we may further reduce its cost by keeping the lifted sequences on some nodes and reconstructing the sequences on the other (internal) nodes to minimize the cost of the edges incident upon these (latter) nodes. For example, based on the lifted sequences 2, 3, 5, we can compute a sequence for the dark circled node in Figure 2.1(a) such that the total cost of the three thin edges is minimized. The new sequence should in general yield a loaded tree with a smaller cost. This suggests the idea of partitioning a (uniformly) lifted tree into a collection of overlapping components, keeping the lifted sequences at the leaves of these components intact, and optimally reconstructing the sequences for the internal nodes in each component, i.e., doing a local optimization on each component. The computation can be done in polynomial time as long as each component has a constant size.

Both PTASs in [15, 16] are based on this idea and partition the tree simply by cutting at levels separated by a fixed constant distance. Our new algorithm also follows the same general approach, but we use a more sophisticated and flexible partitioning strategy.

3. The new partitioning strategy and algorithm. Our algorithm involves partitioning a uniformly lifted tree into many overlapping components, each of which is a binary tree by itself. Let us first describe the structure of the components used in a partition.

A degree-1 node in a tree is called a *terminal.* For a fixed constant r, we consider components of r terminals. If $2^{t-2} + 1 \leq r < 2^{t-1} + 1$ for some positive integer t, a component contains t levels of nodes, where the top level contains only one terminal, called the *head*, the other $t - 1$ levels form a complete binary tree, and the bottom two levels contain $r - 1$ leaves. (See Figure 3.1.) Such a component will be referred to as a *basic component* of a partition, to avoid ambiguity. The terminals (i.e., the head and leaves) of a basic component are called its *boundary nodes.* Note that the child of the head could be either the left or right child. A basic component is of *L-type* (or *R-type*) if it uses the left (or right, respectively) child of the head.

Let T be a phylogeny, V a uniform lifting choice vector, and $T(V)$ the corresponding uniformly lifted tree. Suppose that r is an integer such that $2^{t-2}+1 \leq r \leq 2^{t-1}+1$.

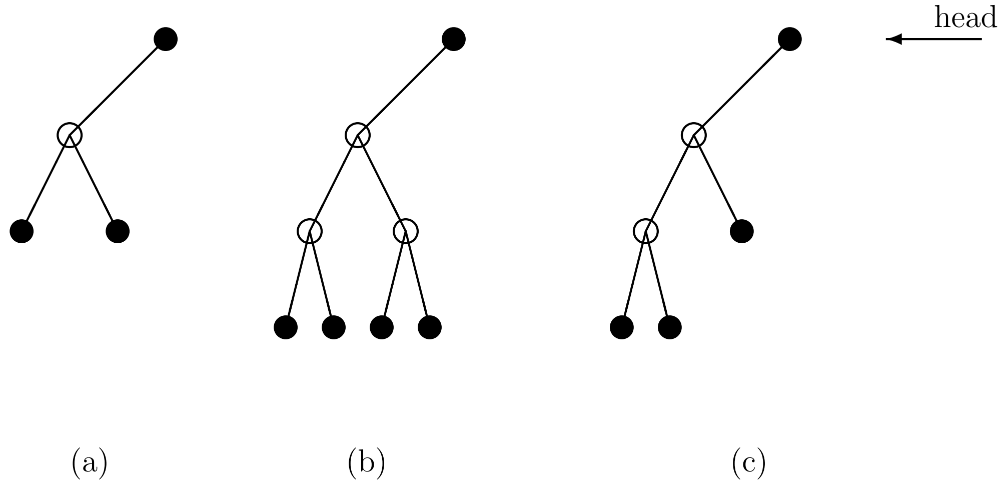

FIG. 3.1. *The L-type basic components for* (a) $r = 3$, (b) $r = 5$, *and* (c) $r = 4$. *The boundary nodes of a basic component are dark circled.*

For every uniform lifting choice vector V, we can obtain t partitions $P_0, P_1, \ldots, P_{t-1}$ of $T(V)$ as follows. Observe that to define a partition, it suffices to specify the heads of the basic components involved and the type of each basic component. First consider $i = 0, 1, \ldots, t-2$.

1. P_i has a (deformed) basic component at the top of $T(V)$, which consists of 2^i leaves at the ith level. (The root of the tree is at level 0, and the ith level is below the $(i-1)$th level. See Figures 3.2 and 3.3.)
2. All nodes on the lifting path of each leaf and each head of every basic component are heads, and the leaves of the basic components are heads.
3. The type of each basic component is identical to the lifting choice at its head as given in V. In other words, the basic component is of L-type (R-type) iff its head receives its sequence from the left child.

The partition P_{t-1} is defined similarly except that its top basic component part is a complete binary tree with $r - 1$ leaves instead. Figures 3.2, 3.3, and 3.4 give an example of the partitions P_0, P_1, P_2 when $r = 4$.

Given a partition P_i, if we preserve the sequences uniformly lifted to all the boundary nodes and optimally reconstruct the sequences on rest of the nodes (i.e., the internal nodes of each basic component) to minimize the cost of each basic component, we obtain a loaded tree, which will simply be called an *(r, i)-tree.* We use $T(V)_{r,i}$ to denote the (r, i)-tree obtained from the uniformly lifted tree $T(V)$. An *optimal r-tree* is some (r, i)-tree $T(V)_{r,i}$ with the smallest cost among all possible i and V. For any loaded tree T_1, $C(T_1)$ denotes its cost. For any tree T_1, denote the set of internal nodes of T_1 as $I(T_1)$ and the set of leaves of T_1 as $L(T_1)$, and for each $a \in L(T_1)$, T_1^a denotes the unique $(r, 0)$-tree for T_1 such that the sequence label of leaf a is lifted to the root of T_1.

Now let us begin to describe our algorithm. We will first assume that T is a full binary tree and then extend the construction to arbitrary binary trees. Let v be a node of T. The subtree of T rooted at v consisting of all descendants of v is denoted as T_v. Note that, since we assume that T is a full binary tree, lifting the sequence of a, for any $a \in L(T_v)$, to v uniquely determines a uniform lifting choice for T_v.

Let T' be the top basic component of T_v^a and v' the child of v that is on the

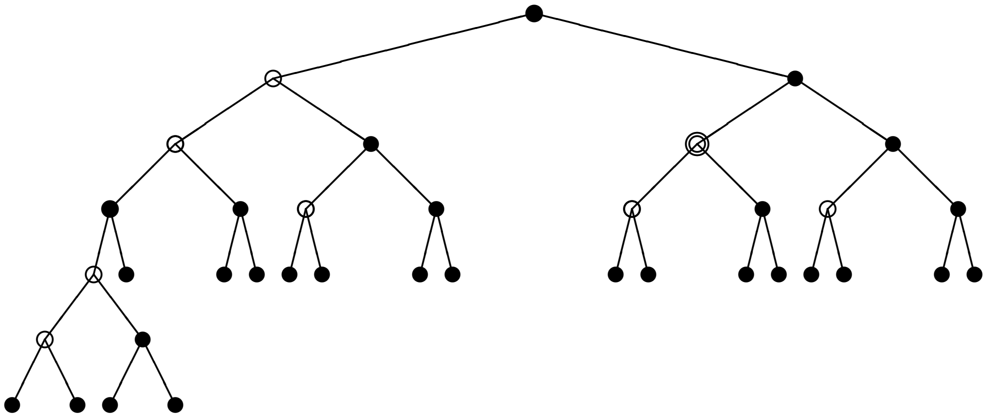

FIG. 3.2. *The partition P_0 when $r = 4$. The uniform lifting choice is $(R, R, \ldots, R)$. The dark circled nodes are boundary nodes of the basic components.*

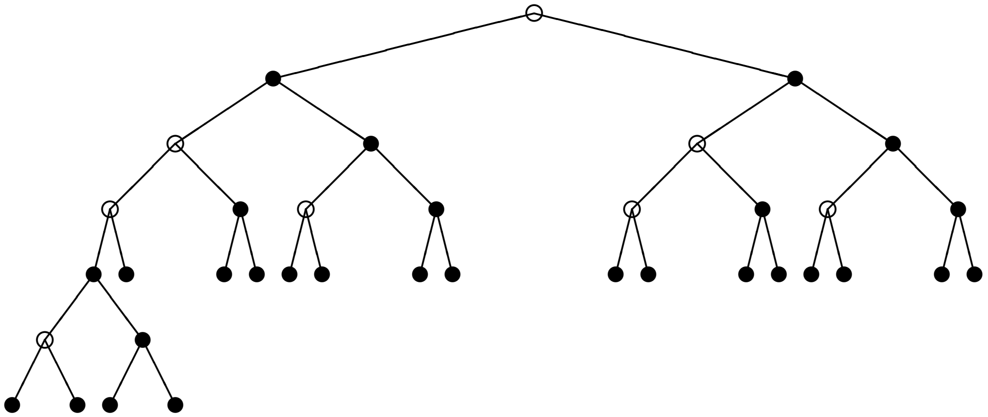

FIG. 3.3. *The partition P_1 when $r = 4$. The uniform lifting choice is $(R, R, \ldots, R)$.*

lifting path of v in T_v^a. Observe that a uniquely determines the sequences lifted to the boundary nodes of T'. The cost of the $(r, 0)$-tree for T_v^a can be computed using the recurrence equation

$$C(T_v^a) = C(T') + \sum_{u \in L(T')} C(T_u^{a(u)}) + C(T_{v'}^a), \tag{3.1}$$

where $a(u)$ is the leaf whose sequence is lifted to u and $a(u)$ is uniquely determined by a. (See Figure 3.5.)

Hence we can compute the values $C(T_v^a)$ for all v and a inductively by traversing the tree bottom-up. Note that, for each pair v and a, computing (3.1) requires $O(r + n^r)$ time, where n is the length of the sequences, given the value of $C(T_u^{a(u)})$ for each $u \in L(T')$ and the value of $C(T_{v'}^a)$. Thus, computing $C(T_v^a)$ for all pairs v and a requires $O((r + n^r) \cdot \sum_{v \in T} |L(T_v)|)$ time. Since each leaf a appears in at most d different $L(T_v)$s, where d is the depth of T, totally we need $O(rkd + kdn^r)$ time. Similarly, the cost of an (r, i)-tree $T_{r,i}^a$ obtained by lifting the sequence label of leaf a

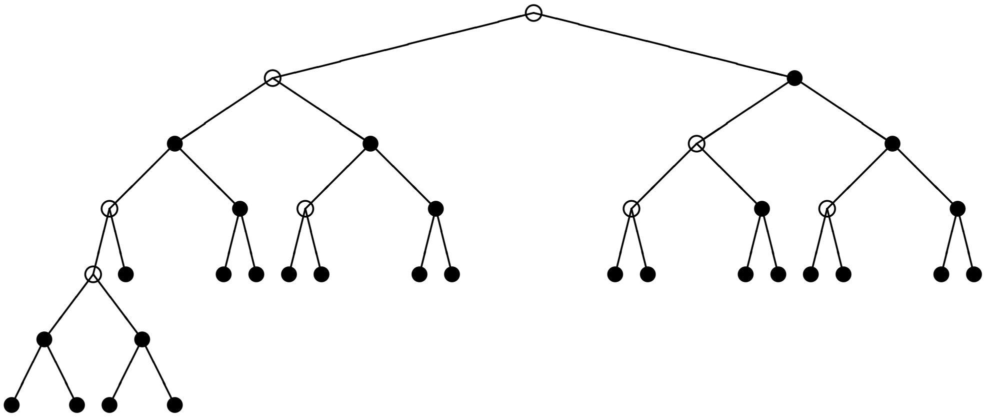

FIG. 3.4. *The partition P_2 when $r = 4$. The uniform lifting choice is $(R, R, \ldots, R)$.*

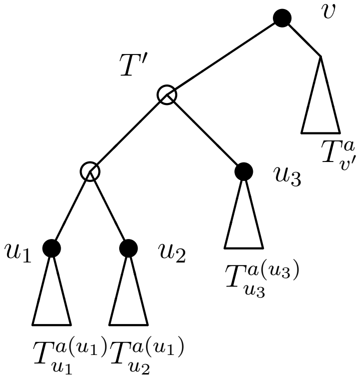

FIG. 3.5. *An illustration for the recurrence equation* (3.1).

to the root of T can be computed as follows:

$$C(T^a_{r,i}) = C(T'_i) + \sum_{v \in L(T'_i)} C(T^{a(v)}_v), \tag{3.2}$$

where T'_i is the top basic component of partition P_i and $a(v)$ is the leaf whose sequence is lifted to v as determined by the choice of a.

The above algorithm only works for full binary trees since, in general, lifting from a leaf a to a node v does not completely determine the uniform lifting choice for the subtree T_v if a is not at the bottom level of T_v. In particular, in arbitrary binary trees, lifting from leaf a to node v does not uniquely determine the leaf $a(u)$ being lifted to the node u for any internal node u that is lower than v, hence invalidating the recurrence equation (3.1).

To extend the algorithm to arbitrary binary trees without losing any time efficiency, we need the notion of an *extension* of a phylogeny. An unlabeled (ordered) tree T_1 *extends* another unlabeled (ordered) tree T_2 if T_2 can be obtained from T_1 by contracting some subtrees to single leaves. Clearly, if T_1 extends T_2, then each leaf of T_2 uniquely corresponds to a subtree of T_1. The *minimal extension* of a set of unlabeled trees is the (unique) unlabeled tree with the fewest edges that extends every tree in

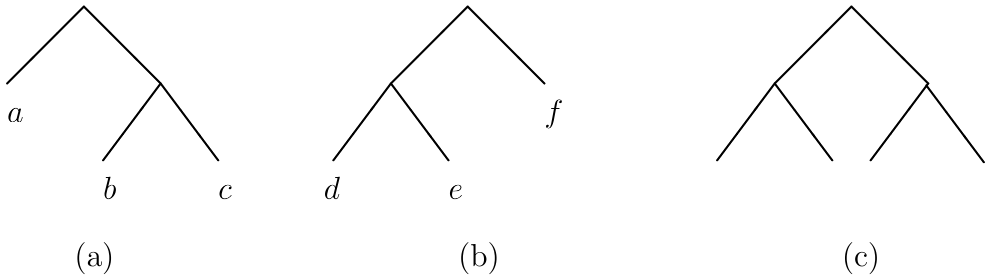

Fig. 3.6. *The minimal extension of the structures of the trees in* (a) *and* (b) *is shown in* (c).

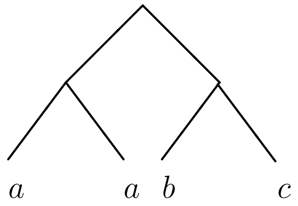

Fig. 3.7. *An extension of the tree in Figure* 3.6(a).

the set. A (partially) leaf-labeled tree T_1 extends another (partially) leaf-labeled tree T_2 if (i) the structure of T_1 extends the structure of T_2; and (ii) for each leaf v of T_2, the subtree of T_1 corresponding to v has all its leaves assigned the same label as that of v. Figure 3.6 gives an example of the minimal extension of two unlabeled trees, and Figure 3.7 shows an extension of the leaf-labeled tree in given Figure 3.6(a).

Now we are ready to generalize the algorithm to an arbitrary binary tree T. Let v be an internal node of T. Denote $T_{v,t-1}$ as the tree consisting of the top t levels of the subtree T_v. To compute $C(T_v^a)$ for each $a \in L(T_v)$, we first extend the tree $T_{v,t-1}$, which is in general a partially leaf-labeled tree, to a full binary (partially leaf-labeled) tree $T'_{v,t-1}$ with 2^{t-1} leaves. Let $v_1, \ldots, v_{2^{t-1}}$ be the leaves of this extended tree. For each v_i, $i = 1, \ldots, 2^{t-1}$, we extend T_{v_i} to obtain a leaf-labeled tree T'_{v_i} such that the structure of T'_{v_i} is the minimal extension of the structures of all $T_{v_1}, \ldots, T_{v_{2^{t-1}}}$. Note that $T'_{v,t-1}$ has 2^{t-1} leaves $v_1, \ldots, v_{2^{t-1}}$ that are the roots of subtrees $T'_{v_1}, \ldots, T'_{v_{2^{t-1}}}$. Denote the tree containing $T_{v,t-1}$ at the top and $T'_{v_1}, \ldots, T'_{v_{2^{t-1}}}$ at the bottom as $ET(v)$. We compute the cost $C(ET(v)^a)$ for each $a \in L(ET(v))$ first.

Suppose that $C(T_u^a)$ has been computed for each $u \in T_{v,t-1}$, where $u \neq v$ and $a \in L(T_u)$. We can easily compute the cost $C(ET(v)_u^a)$ of the $(r, 0)$-tree $ET(v)_u^a$ for each $u \in T_{v,t-1}$, where $u \neq v$ and $a \in L(ET(v)_u)$ as follows:

$$C(ET(v)_u^a) = C(T_u^b), \qquad \text{where } a \in L(ET(v)_b). \tag{3.3}$$

Observe that the subtrees $T'_{v_1}, \ldots, T'_{v_{2^{t-1}}}$ of $ET(v)$ all have the same structure. Thus, lifting the sequence of a, for any $a \in L(ET(v))$, to v uniquely determines from where each node of $ET(v)_{v,t-1}$ receives its lifted sequence. Since $ET(v)_{v,t-1}$ extends the top basic component of $ET(v)^a$, lifting the sequence of leaf a, $a \in L(ET(v))$, to v uniquely determines from where each leaf of the top basic component of $ET(v)^a$ receives its lifted sequence. Therefore, we can compute the cost $C(ET(v)^a)$ for each $a \in L(ET(v))$ using the recurrence equation (3.1).

```
begin
for each level $i$, with the bottom level first
  for each node $v$ at level $i$
  begin
    Construct the extended tree $ET(v)$.
    for each $u \in T_{v,t-1}$
      Compute $C(ET(v)_u^a)$ for each $a \in L(ET(v)_u)$ using equation 3.3.
    for each leaf $a \in L(ET(v))$
      Compute $C(ET_v^a)$ using equation 3.1.
    for each leaf $a \in L(T(v))$
      Compute $C(T_v^a)$ using equation 3.4.
  end
for each $i = 0, 1, \ldots, t-1$
  Compute $C(T_{r,i}^a)$ for every $a \in L(T)$.
Find $i, a$ so that $C(T_{r,i}^a)$ is minimized.
Compute a loaded tree from $C(T_{r,i}^a)$ by back-tracing.
end.
```

FIG. 3.8. *The algorithm to compute an optimal r-tree.*

Once we have the cost $C(ET(v)^a)$ for each $a \in L(ET(v))$, we can compute $C(T_v^a)$ easily for each $a \in L(T_v)$ by "reversing" (3.3) as follows:

$$C(T_v^a) = \min_{b \in L(ET(v)_a)} C(ET(v)^b). \tag{3.4}$$

The time complexity of the algorithm can be analyzed as follows. Since the structure of T_v extends the structure of all subtrees $T_{v_1}, \ldots, T_{v_{2^{t-1}}}$, $|L(T'_{v_i})| \leq |L(T_v)|$ for each i, $1 \leq i \leq 2^{t-1}$. Thus, the extended tree $ET(v)$ can be computed in $O(r|L(T_v)|)$ time for each $v \in T$. Hence totally it requires at most $O(\sum_{v \in T} r|L(T_v)|)$ time to construct all the $ET(v)$s. Since each leaf of T appears in at most d $L(T_v)$s, $O(\sum_{v \in T} r|L(T_v)|) = O(rd|L(T)|) = O(rkd)$. Computing (3.3) and (3.4) takes merely a traversal of $ET(v)$ each. Hence the algorithm still runs in time $O(rkd + kdn^r)$ on arbitrary binary trees. The complete algorithm is summarized in Figure 3.8.

4. The analysis of the algorithm. Given an arbitrary binary phylogeny T, we can extend T into a full binary tree $\hat{T}$. Obviously, any loaded tree for T can be extended to a loaded tree with the same cost for $\hat{T}$. Conversely, given any loaded tree for $\hat{T}$, we can obtain a loaded tree for T with equal or smaller cost by pruning appropriate subtrees and contracting nodes with only one child. The last operation will not increase the cost by the triangle inequality. Thus for the analysis we may assume that the given tree T is a full binary tree. For convenience, we number the levels from top to bottom. That is, the root is at level 0, and the level below level i is level $i+1$.

First, let us find a good upper bound for the cost of an (r,i)-tree $T(V)_{r,i}$ for an arbitrary uniform lifting choice V. Again, let $T^{\min}$ be an optimal loaded tree for T. For each node $v \in T$, let $s(v)$ denote the sequence label of v in $T^{\min}$. We can modify $T(V)_{r,i}$ by replacing the sequence label of each nonterminal node with the sequence $s(v)$ to obtain a loaded tree $T(V)'_{r,i}$. Clearly, the cost of $T(V)'_{r,i}$ is at least that of $T(V)_{r,i}$. So, it suffices to upper bound the cost of $T(V)'_{r,i}$.

There are four types of edges in the loaded tree $T(V)'_{r,i}$:

1. the edges whose both ends are assigned an identical lifted sequence;
2. the edges with ends assigned distinct lifted sequences;
3. the edges whose both ends are assigned sequences from $T^{\min}$;
4. the edges with one end assigned a sequence from $T^{\min}$ and the other assigned a lifted sequence.

(See again Figure 3.2.) Obviously each type 1 edge costs zero and each type 3 edge (u,v) costs $D(s(u),s(v))$. Let (u,v) be a type 4 edge, where u is assigned a sequence from $T^{\min}$ and v is assigned a lifted sequence according to V. The cost of the edge (u,v) is upper bounded by

$$D(s(u),s(v)) + C(P_{v,V}),$$

where $P_{v,V}$ denotes the lifting path of node v in the uniformly lifted tree $T(V)$ and $C(P_{v,V})$ is the cost of the path $P_{v,V}$ in the optimal tree $T^{\min}$. Observe that type 2 edges may only appear at the bottom level of the tree. Hence a type 2 edge can be viewed as a degenerate type 4 edge, with the lower end (which is a leaf) of the edge being the node assigned a sequence from $T^{\min}$, and can be treated similarly to the type 4 edges. For convenience, call $D(s(u),s(v))$ and $C(P_{v,V})$ the *first* and *second* parts of the cost of edge (u,v) in $T(V)$, respectively. For each internal node v in $T(V)_{r,i}$, the path from v to the leaf whose sequence label is lifted to v is called the *lifting path* of v in $T(V)_{r,i}$. Call each internal node on a lifting path in $T(V)_{r,i}$ of some node v a *lifted* node. A lifted node is *heavy* if it is involved in two type 4 edges, and is *light* if it is involved in one type 4 edge. For a light node v, the cost of the lifted path $P_{v,V}$ is charged once in the above upper bound as the second part, whereas for a heavy node v, the cost of the lifted path $P_{v,V}$ is charged twice. A *maximal* lifting path is a lifting path in $T(V)_{r,i}$ which cannot be further extended. Notice that each maximal lifting path contains at most one heavy node at the upper end of the path. Since every edge in a maximal lifting path has type 1 and thus zero cost, we can charge one of the $C(P_{v,V})$s for a heavy node v to $C(T^{\min})$, obtaining the following upper bound for the cost $T(V)_{r,i}$:

$$C(T(V)_{r,i}) \leq C(T^{\min}) + \sum_{v \text{ is a lifted node}} C(P_{v,V}). \tag{4.1}$$

To further bound the total cost of the lifting paths, we need the following lemmas.

Lemma 4.1 (see [15, 16]). *Let T be a binary tree such that every internal node has exactly two children. For each leaf $l \in L(T)$, there exists a mapping π_l from the internal nodes of T to its leaves such that* (i) *for each internal node v, $\pi(v)$ is a descendant leaf of v,* (ii) *the paths from nodes v to their images $\pi(v)$ are edge-disjoint, and* (iii) *moreover, there is an* unused *path from the root of T to the leaf l that is edge-disjoint from all the paths in* (ii).

In other words, Lemma 4.1 says that a binary tree can be decomposed into a set of edge-disjoint paths from internal nodes to leaves, with one path for each nonroot internal node and two paths for the root. Let $C(\pi)$ denote the cost of the paths induced by mapping π in the tree $T^{\min}$.

Lemma 4.2. *There exist mappings $\{\pi_l | l \in L(T)\}$ such that*

$$\sum_{V} \sum_{v \in I(T)} C(P_{v,V}) \leq \sum_{l \in L(T)} C(\pi_l) \leq 2^d C(T^{\min}),$$

where d is the depth of T.

Proof. We prove it by induction on d, the depth of the tree. Recall that T is a full binary tree. The lemma obviously holds for $d = 1$. Suppose that it holds for $d \le q$. Consider a tree T with $d = q + 1$. Let v_0 be the root of T and v_L and v_R the left and right children of v_0. Recall that T_{v_L} and T_{v_R} denote the subtrees rooted at v_L and v_R, respectively. Then the depth of T_{v_j} $(j = L, R)$ is q. Let $\mathcal{V}$ be the set of lifting choices for T, and $\mathcal{V}_j$ the set of lifting choices for T whose first component is j. Clearly $\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_R$. By the inductive hypothesis, for each $j = L, R$, there exist mappings $\{\pi_l | l \in L(T_{v_j})\}$ for the subtree T_{v_j} such that

$$\sum_{V \in \mathcal{V}_j} \sum_{v \in I(T_{v_j})} C(P_{v,V}) \le \sum_{l \in L(T_{v_j})} C(\pi_l).$$

For each mapping π_l, $l \in L(T_{v_j})$, there is an unused path from v_j to the leaf l of T_{v_j}. For any leaf l, let P_l be the path from the root v_0 of T to the leaf l. Number the leaves of T from left to right as $l_1, \ldots, l_{2^{q+1}}$. It is easy to see that for any $h = 1, \ldots, 2^q$, the mappings π_{l_h} and $\pi_{l_{2^q+h}}$ and the path P_{l_h} form a mapping for tree T, denoted $\pi^{h,1}$, with an unused path from v_0 to l_{2^q+h}, and symmetrically the mappings π_{l_h} and $\pi_{l_{2^q+h}}$ and the path $P_{l_{2^q+h}}$ form a mapping for tree T, denoted $\pi_{h,2}$, with an unused path from v_0 to l_h. Thus, we have

$$\begin{aligned}
\sum_{V \in \mathcal{V}} \sum_{v \in I(T)} C(P_{v,V}) &= 2 \cdot \left[\sum_{j=L}^{R} \sum_{V \in \mathcal{V}_j} \sum_{v \in I(T_{v_j})} C(P_{v,V}) \right] + \sum_{V \in \mathcal{V}} C(P_{v_0,V}) \\
&\le \sum_{h=1}^{2^{q+1}} 2 \cdot C(\pi_{l_h}) + C(P_{l_h}) \\
&\le \sum_{h=1}^{2^q} \sum_{j=1}^{2} C(\pi_{l_h}) + C(\pi_{l_{2^q+h}}) + C(P_{l_{(j-1)2^q+h}}) \\
&\le \sum_{h=1}^{2^q} \sum_{j=1}^{2} C(\pi_{h,j}) \\
&\le 2^{q+1} C(T^{\min}). \qquad \square
\end{aligned}$$

Since, there are 2^d distinct uniform lifting choices, the average cost of each $\sum_{v \in I(T)} C(P_{v,V})$ is at most $C(T^{\min})$. Observe that Theorem 2.1 follows immediately from inequality (4.1) and Lemma 4.2, since in a uniformly lifted tree $T(V)$, every internal node is a lifted node. Now we are ready to derive the required upper bound for an optimal r-tree and hence the performance ratio of our approximation algorithm. To simplify the presentation, consider first the case when $r = 2^{t-1} + 1$ for some $t \ge 1$.

4.1. The performance ratio when $r = 2^{t-1}+1$.

Let $y_i(j)$ denote the number of boundary nodes at level j of an (r, i)-tree, where $0 \le i \le t - 1$ and $0 \le j \le d$. It is easy to see that $y_0(j)$ can be computed by the recurrence equation

$$y_0(j) = y_0(j-1) + 2^{t-1} y_0(j-t), \tag{4.2}$$

where $t \le j \le d$. The initial values are $y_0(0) = \cdots = y_0(t-1) = 1$.

Let $x_i(j) = \frac{y_i(j)}{2^j}$ be the fraction of the nodes on level j of an (r, i)-tree that are boundary nodes, where $0 \leq i \leq t-1$ and $0 \leq j \leq d$. From (4.2), we have

$$x_0(j) = \frac{x_0(j-1) + x_0(j-t)}{2}, \tag{4.3}$$

where $t \leq j \leq d$. The initial values are $x_0(j) = 2^{-j}$ for each $j = 0, 1, \ldots, t-1$. Moreover, it is easy to see that

$$x_i(j) = x_{i-1}(j-1) = \cdots = x_0(j-i) \tag{4.4}$$

for any $i \leq j \leq d$ and $x_i(j) = 0$ for any $0 \leq j < i$. The next lemma is a key to our bound. This gives a complete recurrence relation for all $x_i(j)$.

LEMMA 4.3.

$$2x_0(j) + \sum_{i=1}^{t-1} x_i(j) \leq 2.$$

Proof. Define $f(j) = 2x_0(j) + \sum_{i=1}^{t-1} x_i(j)$. From (4.3) and (4.4), we obtain

$$f(j+1) - f(j) = 2x_0(j+1) - x_0(j) - x_0(j-t+1) = 0$$

for $j \geq t-1$. Based on the given initial values, we have

$$f(j) = f(t-1) = 2x_0(t-1) + x_0(t-2) + \cdots + x_0(0) = 2$$

for $j \geq t-1$. It is also easy to see that $f(j) < f(t-1)$ for any $j < t-1$. □

Combining inequality (4.1) and Lemmas 4.2 and 4.3, we have the following lemma.

LEMMA 4.4.

$$2 \cdot \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,0}) + \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,1}) + \cdots + \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,t-1})$$
$$\leq (t+1)2^d C(T^{\min}) + 2 \cdot 2^d C(T^{\min}).$$

The left-hand side of the above inequality consists of $(t+1)2^d$ distinct (r, i)-trees. That is, the average cost of each of these (r, i)-trees is $1 + \frac{2}{t+1} C(T^{\min})$. Thus we can conclude that there exists an (r, i)-tree with cost at most $1 + \frac{2}{t+1} C(T^{\min})$.

THEOREM 4.5. *When* $r = 2^{t-1} + 1$, *the performance ratio of the algorithm in Figure* 3.8 *is* $1 + \frac{2}{t+1}$.

4.2. The performance ratio for an arbitrary r. Assume that $r = 2^{t-1} + 1 - q$ for some integer q, $0 \leq q < 2^{t-2}$. Define variables $y_i(j)$ and $x_i(j)$ as before, and consider $y_0(j)$ and $x_0(j), j \geq t$ first. There are three types of boundary nodes at level j. Recall that each basic component consists of a head and a complete binary tree with $r-1$ leaves.

1. There are $(2^{t-1} - 2q)y_0(j-t)$ boundary nodes which are at the lowest level of some basic components.
2. There are $qy_0(j-t+1)$ boundary nodes which are at the second lowest level of some basic components.
3. There are $y_0(j-1)$ remaining boundary nodes which are on some lifting paths.

The above three types of boundary nodes are disjoint. (Note that each node in classes 1 and 2 is always shared by two adjacent basic components, whereas each node in class 3 is only involved in one basic component.) Therefore, we have recurrence equation

$$y_0(j) = y_0(j-1) + qy_0(j-t+1) + (2^{t-1} - 2q)y_0(j-t)$$

for $j \geq t$. Hence

$$x_0(j) = \frac{1}{2}x_0(j-1) + \frac{2q}{2^t}x_0(j-t+1) + \frac{2^{t-1}-2q}{2^t}x_0(j-t) \tag{4.5}$$

for $j \geq t$. The initial values are

$$\begin{aligned} x_0(j) &= \frac{1}{2^j} \qquad \text{for} \quad j = 0, 1, \ldots, t-2, \\ x_0(t-1) &= \frac{q+1}{2^{t-1}}. \end{aligned}$$

Again, it is easy to see that

$$x_i(j) = x_{i-1}(j-1) \tag{4.6}$$

for any $i = 1, \ldots, t-2$, and

$$y_{t-1}(j) = (2^{t-1} - 2q)y_0(j-t+1) + qy_0(j-t+2).$$

From the last equation, we have

$$x_{t-1}(j) = \frac{2^{t-1}-2q}{2^{t-1}}x_0(j-t+1) + \frac{2q}{2^{t-1}}x_0(j-t+2). \tag{4.7}$$

Observing that $x_i(j) = 0$ for any $j < i$, the above gives a complete recursive definition of all $x_i(j)$.

Lemma 4.6.

$$2^t x_0(j) + 2^{t-1}x_1(j) + \cdots + 2^{t-1}x_{t-3}(j) + (2^{t-1} - 2q)x_{t-2}(j) + 2^{t-1}x_{t-1}(j) \leq 2^t.$$

Proof. Define

$$f(j) = 2^t x_0(j) + 2^{t-1}x_1(j) + \cdots + 2^{t-1}x_{t-3}(j) + (2^{t-1} - 2q)x_{t-2}(j) + 2^{t-1}x_{t-1}(j).$$

From (4.6) and (4.7), we have, for $j \geq t-1$,

$$f(j) = 2^t x_0(j) + 2^{t-1}x_0(j-1) + \cdots + 2^{t-1}x_0(j-t+2) + (2^{t-1} - 2q)x_0(j-t+1). \tag{4.8}$$

From (4.5) and (4.8), we obtain $f(j+1) - f(j) = 0$. Plugging the initial values of $x_0(j)$ in (4.8), we have

$$\begin{aligned} f(j) &= f(t-1) \\ &= 2^t x_0(t-1) + 2^{t-2}x_0(j-1) + \cdots + 2^{t-1}x_0(1) + (2^{t-1} - 2q)x_0(0) \\ &= 2(q+1) + 2 + \cdots + 2^{t-2} + (2^{t-1} - 2q) = 2^t \end{aligned}$$

for any $j \geq t-1$. It is easy to verify that $f(j) < f(t-1)$ for any $j < t-1$. □

Similar to Lemma 4.4, we have the following lemma.

LEMMA 4.7.

$$\begin{aligned} 2^t \sum_{V=(L,\dots,L)}^{(R,\dots,R)} C(T(V)_{r,0}) + 2^{t-1} \sum_{V=(L,\dots,L)}^{(R,\dots,R)} C(T(V)_{r,1}) + \cdots + 2^{t-1} \sum_{V=(L,\dots,L)}^{(R,\dots,R)} C(T(V)_{r,t-3}) \\ + \quad (2^{t-1} - 2q) \sum_{V=(L,\dots,L)}^{(R,\dots,R)} C(T(V)_{r,t-2}) + 2^{t-1} \sum_{V=(L,\dots,L)}^{(R,\dots,R)} C(T(V)_{r,t-1}) \\ \leq \quad (2^{t-1}(t+1) - 2q)2^d C(T^{\min}) + 2^t \cdot 2^d C(T^{\min}). \end{aligned}$$

Noting that the left-hand side of the above inequality consists of $(2^{t-1}(t+1)-2q)2^d$ distinct (r,i)-trees, we have the main theorem of this section.

THEOREM 4.8. *Suppose that* $r = 2^{t-1} + 1 - q$, *where* $0 \leq q < 2^{t-2}$. *The performance ratio of the algorithm in Figure* 3.8 *is* $1 + \frac{2^{t-1}}{2^{t-2}(t+1)-q}$.

5. Discussions.

5.1. Further improvement. Note that an approximate solution produced by the algorithm in Figure 3.8 still uses sequences lifted from the leaves as sequence labels for some internal nodes. Such a solution can be further improved with the iterative method proposed by Sankoff, Cedergren, and Lapalme [10].

To illustrate the iterative method in [10], consider the phylogeny in Figure 5.1, which contains nine given species on its nine leaves. To improve the cost of a loaded tree, we divide the phylogeny into seven 3-components as shown in Figure 5.1, each consisting of a center and three terminals. Local optimization is done for every 3-component based on the labels of its three terminals. 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 often enough, every 3-component will become optimal. Empirical results show that the algorithm produces a reasonably good loaded tree within five iterations [10].

If we are able to handle local optimization of size r, $r \geq 3$, we may extend the above iterative method to components with r terminals. Augmenting the algorithm in Figure 3.8 with the iterative method should result in an improved performance. We do not have an error bound analysis for the combined method at this point that is better than the bound given in section 4. However, there is a good way to estimate the error bound of this combined method for a particular input.

5.2. Estimating the error bound for an instance. Theorem 2.1 says that the average cost of the uniformly lifted trees is at most twice that of an optimal solution. Thus, a half of the average cost is a lower bound of the optimal cost. Let $C_{avg}(I)$ be the cost of the average cost of the uniformly lifted trees for some instance I, and let $X(I)$ be the cost of the solution obtained by the combination of the approximation algorithm in Figure 3.8 and the iterative method above for the input I. Then $X(I)$ is at most $\frac{2X(I)}{C_{avg}(I)}$ times the optimum. This suggests a way to estimate the error bound for a particular instance. In [6], an algorithm to compute the average cost of the uniformly lifted trees is proposed. The running time of the algorithm is $O(kdn^2)$. It is demonstrated that this lower bound is reasonably tight.

5.3. Some experimental results. We have implemented our algorithm in C and run the program on a small example from [10]. The given tree topology is shown in Figure 5.1. Each terminal is labeled with an RNA S_5 sequence. The score scheme

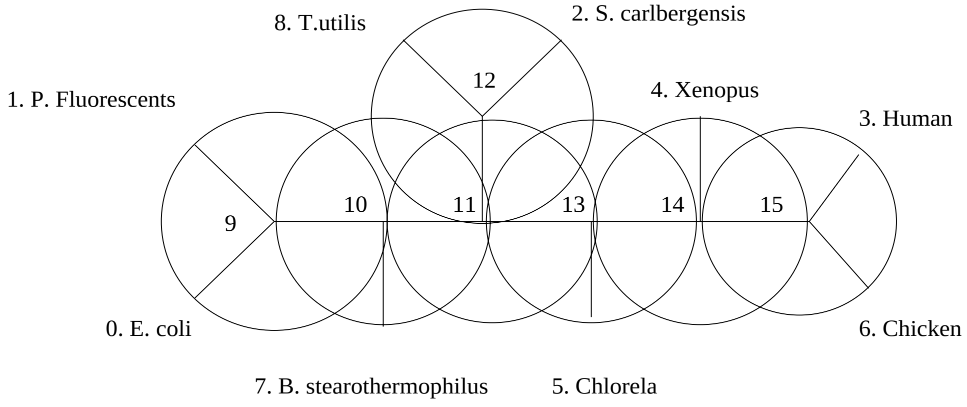

FIG. 5.1. *A phylogeny with nine species, which is divided into seven* 3*-components.*

is the same as in [10]. For $r = 3$, we select the root in the middle of the edge (13,14) and the cost of the corresponding loaded tree is 332.75. For $r = 4$, the same root gives a loaded tree with cost 322.5. Such a cost changes a bit if we try other roots. The average cost of the uniformly lifted trees is 461.6 [6]. Thus we can conclude that the costs of the solutions for $r = 3$ and 4 are at most 1.442 and 1.397 times the optimal for this example. We then use the iterative method of [10] to further improve the solution. The costs are further reduced from 332.75 and 322.5 to 321.25 and 298.25, respectively.

REFERENCES

[1] S. F. ALTSCHUL AND D. J. LIPMAN, *Trees, stars, and multiple biological sequence alignment*, SIAM J. Appl. Math., 49 (1989), pp. 197–209.

[2] S. CHAN, A. WONG, AND D. CHIU, *A survey of multiple sequence comparison methods*, Bull. Math. Biol., 54 (1992), pp. 563–598.

[3] D. FENG AND R. DOOLITTLE, *Progressive sequence alignment as a prerequisite to correct phylogenetic trees*, J. Mol. Evol., 25 (1987), pp. 351–360.

[4] S. GUPTA, J. KECECIOGLU, AND A. SCHAFFER, *Making the shortest-paths approach to sum-of-pairs multiple sequence alignment more space efficient in practice*, in Proceedings of the 6th Combinatorial Pattern Matching Conference, 1995, Springer-Verlag, Berlin, pp. 128–143.

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

[6] D. GUSFIELD AND L. WANG, *New uses for uniform lifted alignment*, in Mathematical Supports for Molecular Biology, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 47, AMS, Providence, RI, 1999, pp. 33–51.

[7] 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), pp. 649–668.

[8] R. RAVI AND J. KECECIOGLU, *Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree*, in Proceedings of the 6th Combinatorial Pattern Matching Conference, 1995, Springer-Verlag, Berlin, pp. 330–339.

[9] D. SANKOFF, *Minimal mutation trees of sequences*, SIAM J. Appl. Math., 28 (1975), pp. 35–42.

[10] D. SANKOFF, R. J. CEDERGREN, AND G. LAPALME, *Frequency of insertion-deletion, transversion, and transition in the evolution of* 5*S ribosomal RNA*, J. Mol. Evol., 7 (1976), pp. 133–149.

[11] 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., Addison-Wesley, Reading, MA, 1983, pp. 253–264.

[12] D. Swofford and G. Olson, *Phylogenetic reconstruction*, in Molecular Systemtics, D. Hillis and C. Moritz, eds., Sinauer Associates, Sunderland, MA, 1990.

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

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

[15] L. Wang, T. Jiang, and E. L. Lawler, *Approximation algorithms for tree alignment with a given phylogeny*, Algorithmica, 16 (1996), pp. 302–315.

[16] L. Wang and D. Gusfield, *Improved approximation algorithms for tree alignment*, J. Algorithms, 25 (1997), pp. 255–273.