Approximation Algorithms for Reconstructing the Duplication History of Tandem Repeats

Zhi-Zhong Chen[*] Lusheng Wang[†] Zhanyong Wang[‡]

Abstract

Tandem repeated regions are closely related to some genetic diseases in human beings. Once a region containing pseudo-periodic repeats is found, it is interesting to study the history of creating the repeats. It is important to reveal the relationship between repeats and genetic diseases. The duplication model has been proposed to describe the history [3, 9, 4]. We design a polynomial-time approximation scheme (PTAS) for the case where the size of the duplication block is 1. Our PTAS is faster than the previously best PTAS in [4]. For example, to achieve a ratio of 1.5, our PTAS takes $O(n^5)$ time while the PTAS in [4] takes $O(n^{11})$ time. We also design a ratio-6 polynomial-time approximation algorithm for the case where the size of each duplication block is at most 2 [1]. This is the first polynomial-time approximation algorithm with a guaranteed ratio for this case.

1 Introduction

The genomes of many species are dominated by short segments repeated consecutively. It is estimated that over 10% of the human genome, the totality of human genetic information, consists of repeated segments. About 10-25% of all known proteins have some form of repeated structures ranging from simple homopolymers to multiple duplications of entire globular domains. In some other species, repeated regions can even dominate the whole genome. For example, in the Kangaroo rat (Dipomys ordii) more than half of the genome consists of three patterns of repeated regions: AAG (2.4 billion repetitions), TTAGG (2.2 billion repetitions) and ACACAGCGGG (1.2 billion repetitions) [11]. Recent studies show that tandem repeats are closely related with human diseases, including neurodegenerative disorders such as fragile X syndrome, Huntington's disease and spinocerebellar ataxia, and some cancers [2, 8]. These tandem repeats may occur in protein coding regions of genes or non-coding regions. Since the initial discovery of tandem repeats [12], many theories on the biological mechanisms that create and extend tandem repeats have been proposed, e.g., slipped-strand mis-paring, unequal sister-chromatid exchange and unequal genetic

[*]Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan. Email: chen@r.dendai.ac.jp. Part of work was done during a visit at City University of Hong Kong.

[†]Supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1196/03E]. Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong. Email: cswangl@cityu.edu.hk.

[‡]Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong. Email: zhyong@cs.cityu.edu.hk.

[1]In the conference version, we wrongly claimed a ratio-2 polynomial-time approximation algorithm.

recombination during meiosis (see [1] for details.) The exact mechanisms responsible for tandem repeat expansion are still controversial. Thus, the study of repeated regions in biological segments has attracted lots of attentions [1, 4, 5, 6, 7, 9, 13].

1.1 The Duplication Model

The model for the duplication history of tandem repeated segments was proposed by Fitch in 1977 [3] and re-proposed by Tang et al. [9] and Jaitly et al. [4]. The model captures both the evolutionary history and the observed order of segments on a chromosome. Let $S = s_1s_2 \ldots s_n$ be an observed string consisting of n segments of the same length m. Let $r_ir_{i+1} \ldots r_{i+k-1}$ be k consecutive segments in an ancestor string of S in the evolutionary history. A *duplication event* generates $2k$ consecutive segments $l_c(r_i)l_c(r_{i+1}) \ldots l_c(r_{i+k-1})r_c(r_i)r_c(r_{i+1}) \ldots r_c(r_{i+k-1})$ by (approximately) copying the k segments $r_ir_{i+1} \ldots r_{i+k-1}$ twice, where both $l_c(r_{i+j})$ and $r_c(r_{i+j})$ are approximate copies of r_{i+j} (see Figure 1). Assume that the n segments s_1, s_2, $\ldots s_n$ were formed from a locus by tandem duplications. Then, the locus had grown from a single copy through a series of duplications. A duplication replaces a stretch of DNA consisting of several segments with two (approximately) identical and adjacent copies of itself. If the stretch contains k segments, the duplication is called a *k-duplication.*

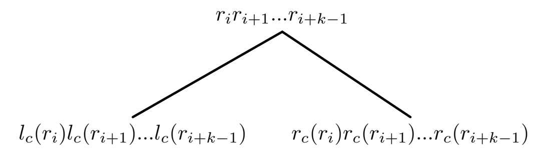
$r_ir_{i+1}...r_{i+k-1}$
$l_c(r_i)l_c(r_{i+1})...l_c(r_{i+k-1})$
$r_c(r_i)r_c(r_{i+1})...r_c(r_{i+k-1})$

Figure 1: A k-duplication, where both $l_c(r_{i+j})$ and $r_c(r_{i+j})$ are approximate copies of r_{i+j}.

A *duplication model* M for a string $S = s_1s_2 \ldots s_n$ of n tandem repeated segments is a directed graph that contains *vertices*, *edges*, and *blocks* as shown in Figure 2. Each vertex in M represents a repeated segment (not necessarily in S), each directed edge (u, v) indicates that vertex v is a child of vertex u, and each block represents a duplication event. Certain edges in the model are allowed to cross each other; this is described using an edge-crossing rule (to be specified later).

Each vertex may have at most one parent and either zero or two children. There is only one vertex, called the *root*, that has no parent. The root represents the original copy at the locus. Those vertices that have two children are called the *internal* vertices, while the others are called the *leaves.* The two children of each internal vertex v are distinguished as the *left child* and the *right child* of v, respectively. Each leaf is labeled with a segment s_i ($1 \leq i \leq n$). Moreover, the left-to-right order of leaves in M is identical to the order of the segments in S. If there is a directed path from a vertex s to another vertex t, then s is an *ancestor* of t and t is a *descendant* of s.

A *block* in M represents a duplication event and hence consists of one or more internal vertices. Each internal vertex appears in a unique block; no vertex in a block is an ancestor (or descendant) of another vertex in the same block. The following rule applies to the children of vertices in each block B: If the left-to-right order of the vertices in B is v_1, v_2, $\ldots$, v_k, then their children are placed (under them) from left to right in the model in the order $l_c(v_1)$, $l_c(v_2)$, $\ldots$, $l_c(v_k)$, $r_c(v_1)$, $r_c(v_2)$, $\ldots$, $r_c(v_k)$, where $l_c(v_i)$ (respectively, $r_c(v_i)$) is the left (respectively, right) child of v_i for each $1 \leq i \leq k$. Hence, for every two integers i and j with $1 \leq i < j \leq k$, the edges $(v_i, r_c(v_i))$

and $(v_j, l_c(v_j))$ cross each other in the model. However, no other edges cross in the model. For simplicity, we will draw a box to depict a block in M only when the block represents a k-duplication event for some $k \geq 2$. We also refer to k as the *size* of the block.

Each edge in M carries a *cost* which is simply the hamming distance between the two segments associated with the two endpoints of the edge. The *cost* of M, denoted by $c(M)$, is the total cost of edges in M. We remark that all our results apply to other distance measures satisfying the triangle inequality.

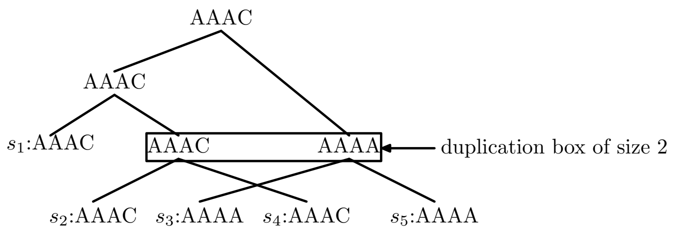

Figure 2: A duplication model with cost 1 for a string $S = s_1s_2s_3s_4s_5$ of five length-4 segments, where $s_1 = AAAC$, $s_2 = AAAC$, $s_3 = AAAA$, $s_4 = AAAC$, and $s_5 = AAAA$.

If we represent only the parent-child relations defined by a duplication model M, then the resulting structure T_M is planar and can be drawn without edge crossings. T_M is a rooted binary tree for the given string $S = s_1s_2 \ldots s_n$ of n segments and is called the *associated phylogeny* for M. Obviously, if every block in M is of size 1, then T_M and M are identical. However, if one or more blocks in M are of size larger than 1, then the left-to-right ordering of the labels assigned to the leaves of T_M is not s_1, s_2, $\ldots$, s_n.

1.2 The Problem and the Results

Now, we are ready to state the problem considered in the paper:

Duplication History Reconstruction (DHR):
- **Input:** An ordered list $\langle s_1, s_2, \ldots, s_n \rangle$ of strings of the same length m.
- **Output:** A duplication model for the string $S = s_1s_2 \ldots s_n$ with the smallest cost.

For each integer $k \geq 1$, let k-DHR denote the special case of DHR where the size of each duplication block is at most k. In this paper, we consider only 1-DHR and 2-DHR. Our first result is a PTAS for 1-DHR; it achieves a ratio of $1 + \frac{2^h}{h2^h - 2q}$ in $O(n^{\ell+1} + mn^{\ell})$ time, where $h = \lceil \log_2 \ell \rceil$ and $q = 2^h - \ell$. Our PTAS is faster than the previously best PTAS for 1-DHR given in [4]. For example, to achieve a ratio of 1.5, our PTAS takes $O(n^5 + mn^4)$ time while the PTAS in [4] takes $O(n^{11} + mn^5)$ time.

Our second result is a polynomial-time approximation algorithm for 2-DHR that achieves a ratio of 6. This is the first polynomial-time approximation algorithm with a guaranteed ratio for 2-DHR. Our algorithm is based on a hidden structure, called the *component tree*, of a duplication model.

2 The PTAS for 1-DHR

A *semiphylogeny* is a rooted tree T satisfying the following conditions (see Figure 3(a) for an example):

- Each vertex of T is assigned a string and all the strings assigned to the vertices of T have the same length.
- Each internal vertex v of T may be *unifurcate* or *bifurcate*. In the former case, v has only one child. In the latter case, v has two children.
- The root of T is bifurcate.
- Both the parent and the child of each unifurcate internal vertex of T are bifurcate.
- The children of each bifurcate internal vertex v of T are distinguished as the left child and the right child of v.
- For every unifurcate internal vertex v in T that is the left child of its parent in T, the string assigned to v in T is the same as the string assigned to $r(v, T)$, where $r(v, T)$ is the rightmost descendant of v in T. (*Comment:* $r(v, T)$ must be a leaf in T because the child of each unifurcate vertex is bifurcate.)
- For every unifurcate internal vertex v in T that is the right child of its parent in T, the string assigned to v in T is the same as the string assigned to $l(v, T)$, where $l(v, T)$ is the leftmost descendant of v in T. (*Comment:* $l(v, T)$ must be a leaf in T because the child of each unifurcate vertex is bifurcate.)

A semiphylogeny is a *phylogeny* if it has no unifurcate internal vertices.

Let T be a semiphylogeny. An edge e of T is *artificial* if e connects a unifurcate internal vertex of T to its (unique) child in T; otherwise, it is *natural*. If we delete all artificial edges from T, then we get a forest in which each connected component is a phylogeny; we use $\mathcal{D}(T)$ to denote the set of these phylogenies. Note that the number of phylogenies in $\mathcal{D}(T)$ is 1 plus the number of artificial edges in T. We call each phylogeny in $\mathcal{D}(T)$ a *subphylogeny* of T.

For an integer $\ell \geq 2$, an *ℓ-semiphylogeny* is a semiphylogeny T such that each subphylogeny of T has at most ℓ leaves. The *cost* of a semiphylogeny T, denoted by $c(T)$, is the total cost of edges in its subphylogenies. In other words, $c(T)$ is the total cost of natural edges in T.

A semiphylogeny for an (ordered) list $\langle s_1, \ldots, s_n \rangle$ of strings of the same length is a semiphylogeny whose leaves are assigned the strings $s_1, \ldots, s_n$ from left to right in this order. Note that 1-DHR can be restated as the problem of constructing a phylogeny for a given list of strings of the same length.

Lemma 2.1 *A semiphylogeny T for a list $\langle s_1, \ldots, s_n \rangle$ of strings of the same length can be easily transformed into a phylogeny T' for the same list with $c(T') = c(T)$.*

Proof. We assemble the phylogenies in $\mathcal{D}(T)$ into a single phylogeny T' for $\langle s_1, \ldots, s_n \rangle$ as follows (see Figure 3):

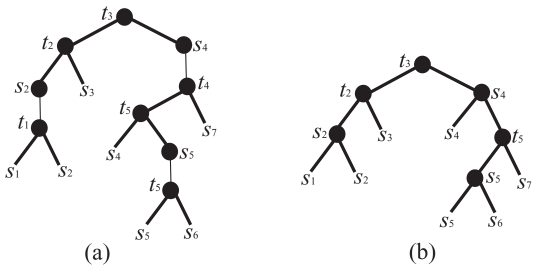

Figure 3: (a) A 3-semiphylogeny T where the bold edges are the natural edges. (b) The phylogeny T' constructed from T.

1. Initialize T' as the forest consisting of the trees in $\mathcal{D}(T)$.
2. For each unifurcate internal vertex v of T that is the left child of its parent in T, add the edge $(v, r(v,T))$ to T', connect a new vertex v' to $r(v,T)$ by a new edge, and assign v' the string that is assigned to $r(v,T)$ in T.
3. For each unifurcate internal vertex v of T that is the right child of its parent in T, add the edge $(v, l(v,T))$ to T', connect a new vertex v' to $l(v,T)$ by a new edge, and assign v' the string that is assigned to $l(v,T)$ in T. (*Comment:* After this step, T' is a tree in which each new vertex is a leaf.)
4. Root T' at r, where r is the root of T.
5. Order the children of each internal vertex in T' accordingly so that the strings $s_1, \ldots, s_n$ appear at the leaves of T' from left to right in this order.
6. While T' has a unifurcate internal vertex, select such a vertex and merge the two edges incident to it into a single edge.

□

The main idea behind our PTAS for 1-DHR is that for any given constant ℓ, we can construct a minimum-cost ℓ-semiphylogeny for a given list of strings of the same length in polynomial time via dynamic programming. The details will be given in Section 2.1.

Now, our PTAS for 1-DHR is as follows:

Input: An ordered list $\langle s_1, \ldots, s_n \rangle$ of strings of the same length and an integer $\ell \geq 2$.

1. Compute a minimum-cost ℓ-semiphylogeny T for $\langle s_1, \ldots, s_n \rangle$.

2. Transform T into a phylogeny T' for $\langle s_1, \ldots, s_n \rangle$ as described in Lemma 2.1.

Output: T'.

In order to prove that the above algorithm is indeed a PTAS for 1-DHR, we also need to show that the minimum cost of an ℓ-semiphylogeny for a list of strings is very close to the minimum cost of a phylogeny for the same list of strings. Sections 2.2 through 2.4 are for this purpose.

2.1 Algorithm for Optimal ℓ-Semiphylogenies

Throughout this subsection, let $\langle s_1, \ldots, s_n \rangle$ be a given list of two or more strings of the same length m, and let ℓ be an integer larger than 1. The goal is to construct a minimum-cost ℓ-semiphylogeny for $\langle s_1, \ldots, s_n \rangle$ via dynamic programming.

First, for each ordered list L of strings among $s_1, \ldots, s_n$ with $2 \le |L| \le \ell$ and for each unlabeled binary tree U with $|L|$ leaves, we compute a minimum-cost phylogeny $T_{L,U}$ for L such that the topology of $T_{L,U}$ is the same as U. Given L and U, $T_{L,U}$ can be computed in $O(m|L|^3)$ time by a simple dynamic programming. Moreover, for each integer $k \ge 2$, there are exactly C_{k-1} unlabeled binary trees with k leaves, where $C_{k-1} = \frac{(2k-2)!}{k!(k-1)!}$ is known as the $(k-1)$-th *Catalan number*. It is also widely known that C_{k-1} is approximately $\frac{4^{k-1}}{\sqrt{\pi}(k-1)^{1.5}}$. Thus, the total time needed for computing all $T_{L,U}$ is $O(mn^{\ell})$ time.

For two integers i and j with $1 \le i \le j \le n$, let $c_{i,j}$ be the cost of a minimum-cost ℓ-semiphylogeny $T_{i,j}$ for $\langle s_i, \ldots, s_j \rangle$. Consider the computation of $c_{i,j}$ with $j - i + 1 > \ell$. Imagine that we already have $T_{i,j}$ and consider $\mathcal{D}(T_{i,j})$. One phylogeny T_{top} in $\mathcal{D}(T_{i,j})$ contains the root of $T_{i,j}$. Let k be the number of leaves in T_{top}. Note that $2 \le k \le \ell$.

Let $v_1, \ldots, v_k$ be the leaves of T_{top} (ordered from left to right). Note that some v_h with $1 \le h \le k$ may also be a leaf of $T_{i,j}$. For each $h \in \{1, \ldots, k\}$, let $T_{i,j}(v_h)$ denote the subtree of $T_{i,j}$ rooted at v_h (i.e., the subtree of $T_{i,j}$ formed by v_h and its descendants in $T_{i,j}$). Note that if v_h is a leaf of $T_{i,j}$, then $T_{i,j}(v_h)$ has only one vertex (namely, v_h) and hence this vertex is both the root and the (unique) leaf of $T_{i,j}(v_h)$. For each $h \in \{1, \ldots, k\}$, let $s_{a_h}, s_{a_h+1}, \ldots, s_{b_h}$ be the strings assigned to the leaves of $T_{i,j}(v_h)$ (from left to right) in $T_{i,j}$. Obviously, $a_1 = i$, $b_k = j$, and $a_{h+1} = b_h + 1$ for each $h \in \{1, \ldots, k-1\}$. Moreover, for each $v_h \in \{v_1, \ldots, v_k\}$, if v_h is a leaf of $T_{i,j}$, then $a_h = b_h$; otherwise, the following hold:

- If v_h is the left child of its parent in T_{top}, then the string assigned to v_h is s_{b_h}.
- If v_h is the right child of its parent in T_{top}, then the string assigned to v_h is s_{a_h}.

We also have $c(T_{i,j}) = c(T_{top}) + \sum_{h=1}^{k} c(T_{i,j}(v_h)) = c(T_{L,U}) + \sum_{h=1}^{k} c(T_{i,j}(v_h))$, where L is the list of strings assigned to the leaves of T_{top} and U is the topology of T_{opt}.

Therefore, when $j - i + 1 > \ell$, we have the following equation:

$$c_{i,j} = \min_{2 \le k \le \ell} \; \min_{i \le b_1 < b_2 < \ldots < b_{k-1} < j} \; \min_{U} \{c(T_{L,U}) + \sum_{h=1}^{k} c_{b_{h-1}+1, b_h}\}, \tag{2.1}$$

where $b_0 = i - 1$, $b_k = j$, U ranges over all unlabeled binary trees with exactly k leaves, and L is the ordered list of k strings defined as follows:

- For each $h \in \{1, \ldots, k\}$ such that $b_h - b_{h-1} = 1$, the h-th string in L is s_{b_h}.
- For each $h \in \{1, \ldots, k\}$ such that $b_h - b_{h-1} \ge 2$, if the h-th leftmost leaf of U is the left child of its parent in U, then the h-th string in L is s_{b_h}; otherwise, the h-th string in L is $s_{b_{h-1}+1}$.

Recall that there are $O(\frac{4^k}{k})$ choices for U in Equation (2.1). Given U and L, $T_{U,L}$ is already available. So, $c_{i,j}$ can be computed in $O(4^\ell (j-i+1)^{\ell-1}\ell)$ time. Obviously, there are $O(n^2)$ pairs (i,j). Thus, $c_{1,n}$ can be computed in $O(n^{\ell+1}+mn^\ell)$ time.

In summary, we have proven:

Theorem 2.2 *For every integer $\ell \geq 2$, we can compute a minimum-cost ℓ-semiphylogeny for a given list of n length-m strings in $O(n^{\ell+1}+mn^\ell)$ time.*

2.2 Transforming Phylogenies into ℓ-Semiphylogenies

To show that the minimum cost of an ℓ-semiphylogeny for a list $\langle s_1,\ldots,s_n\rangle$ of strings of the same length is very close to the minimum cost of a phylogeny for $\langle s_1,\ldots,s_n\rangle$, our idea is to show that a minimum-cost phylogeny for $\langle s_1,\ldots,s_n\rangle$ can be transformed into an ℓ-semiphylogeny for $\langle s_1,\ldots,s_n\rangle$ without increasing its cost significantly.

Let T_{opt} be a minimum-cost phylogeny for $\langle s_1,\ldots,s_n\rangle$. For a path P in T_{opt}, we use $c(P)$ to denote the total cost of edges in P. Moreover, we use $I(T_{opt})$ to denote the set of all internal vertices of T_{opt} other than the root.

For each $v \in I(T_{opt})$, we define a path $P(v)$ as follows: If v is the left child of its parent in T_{opt}, then $P(v)$ is the path in T_{opt} from v to $r(v,T_{opt})$; otherwise, $P(v)$ is the path in T_{opt} from v to $l(v,T_{opt})$ (see Figure 4.)

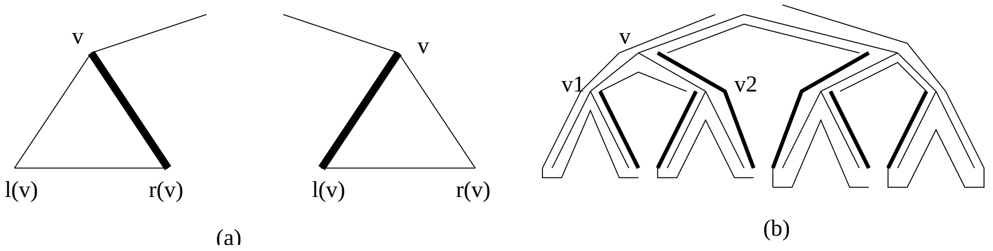

Figure 4: (a) $P(v)$ in two cases. (b) A concrete example.

Lemma 2.3 $\sum_{v\in I(T_{opt})} c(P(v)) \leq c(T_{opt})$.

Proof. To prove the lemma, it suffices to claim that for two distinct vertices u and v in $I(T_{opt})$, $P(u)$ and $P(v)$ are edge-disjoint. The claim is not obvious only when one of u and v is an ancestor of the other in T_{opt}. So, assume that u is an ancestor of v in T_{opt}. Then, $P(u)$ and $P(v)$ are clearly edge-disjoint if v is not a vertex of $P(u)$. So, let us further assume that v is a vertex of $P(u)$. Then, when u is the left child of its parent in T_{opt}, v must be the right child of its parent in T_{opt} and hence $P(v)$ is on the left of $P(u)$. Similarly, when u is the right child of its parent in T_{opt}, v must be the left child of its parent in T_{opt} and hence $P(v)$ is on the right of $P(u)$. Thus, in both cases, $P(u)$ and $P(v)$ are edge-disjoint. □

Since the Hamming distance is a metric, we also have:

Lemma 2.4 *For every $v \in I(T_{out})$, the Hamming distance between the strings assigned to $r(v,T_{opt})$ (respectively, $l(v,T_{opt})$) and the parent u of v in T_{opt} is at most $c(P(v))$ plus the cost of the edge between v and u in T_{opt}.*

For a rooted tree T, we define the *level* of each vertex v in T (denoted by $lev(v,T)$) as follows: The root is at level 0 and its children are at level 1. In general, if a vertex is at level $i \geq 0$, then its children are at level $i+1$.

Fix an integer $\ell \geq 2$. Our goal is to transform T_{opt} into an ℓ-semiphylogeny T for $s_1, \ldots, s_n$ such that $c(T)$ is very close to $c(T_{opt})$. To begin, we let T be a copy of T_{opt}. Then, we find a subset J of $I(T_{opt})$ (in some way to be specified later in Sections 2.3 and 2.4) and modify T by repeating the following five steps until J becomes empty:

1. Find a vertex $v \in J$ whose level in T_{opt} is the minimum among all vertices of J.

2. Split the edge e_v of T between v and its parent v_p in T into two edges by adding a new vertex v' at the middle of e_v.

3. If v' is the left child of v_p in T, then assign v' the string that is assigned to $r(v, T_{opt})$ in T_{opt}. (*Comment:* After this step, v' is a unifurcate internal vertex of semiphylogeny T and the cost of the (natural) edge between v_p and v' in T is at most $c(P(v))$ plus the cost of e_v according to Lemma 2.4. So, this step increases the cost of T by at most $c(P(v))$.)

4. If v' is the right child of v_p in T, then assign v' the string that is assigned to $l(v, T_{opt})$ in T_{opt}. (*Comment:* Same as the comment on the last step.)

5. Remove v from J.

We use $T_{opt}(J)$ to denote the semiphylogeny T obtained from T_{opt} and J as above. Then, we have:

Lemma 2.5 *For every subset J of $I(T_{opt})$, $c(T_{opt}(J)) \leq c(T_{opt}) + \sum_{v \in J} c(P(v))$.*

Unfortunately, depending on J, $T_{opt}(J)$ may not be an ℓ-semiphylogeny. So, we say that a subset J of $I(T_{opt})$ is *ℓ-proper* if $T_{opt}(J)$ is an ℓ-semiphylogeny. What remains to show is that for every integer $\ell \geq 2$, there is always an ℓ-proper subset of $I(T_{opt})$ such that $\sum_{v \in J} c(P(v))$ is relatively small compared to $c(T_{opt})$. The next two subsections are for this purpose.

2.3 The Ratio when ℓ Is a Power of 2

This case is easy as shown in the next lemma:

Lemma 2.6 *For every integer $\ell = 2^h$ with $h \geq 1$, there is always an ℓ-proper subset J of $I(T_{opt})$ such that $\sum_{v \in J} c(P(v)) \leq \frac{1}{h} \cdot c(T_{opt})$.*

Proof. Let r be the root of T_{opt}. For each integer j with $0 \leq j \leq h-1$, let V_j be the set of those vertices $v \in I(T_{opt})$ such that $lev(v, T_{opt}) \equiv j \pmod{h}$. Obviously, V_j is ℓ-proper. Moreover, $V_j \cap V_{j'} = \emptyset$ for $0 \leq j \neq j' \leq h-1$. Hence,

$$\sum_{j=0}^{h-1} \sum_{v \in V_j} c(P(v)) \leq \sum_{v \in I(T_{opt})} c(P(v)) \leq c(T_{opt}),$$

where the last inequality follows from Lemma 2.3. Consequently, there exists an integer $j \in \{0, 1, \ldots, h-1\}$ such that $\sum_{v \in V_j} c(P(v)) \leq \frac{1}{h} \cdot c(T_{opt})$. $\square$

Combining Theorem 2.2 and Lemmas 2.1 and 2.6, we now have:

Theorem 2.7 *Our algorithm is a PTAS for 1-DHR and achieves a ratio of* $1 + \frac{1}{h}$ *in* $O(n^{2^h+1} + mn^{2^h})$ *time for every integer* $h \geq 1$.

For example, to achieve a ratio of 1.5, our PTAS takes $O(n^5 + mn^4)$ time while the PTAS in [4] takes $O(n^{11} + mn^5)$ time.

2.4 The Ratio When ℓ Is not a Power of 2

This case is much more difficult. We start by giving several definitions. The *height* of a binary tree is the maximum level of a vertex in the tree. A binary tree is *complete* if it has 2^k leaves, where k is the height of the tree. Note that for each integer $k \geq 0$, there is a unique unlabeled complete binary tree of height k. Moreover, in a complete binary tree, all leaves appear at the same level. A binary tree T is *balanced* if every internal vertex of T has two children in T and the level of every leaf in T is k or $k-1$, where k is the height of T.

Since ℓ is not a power of 2, $\ell \geq 3$ and ℓ can be written as $\ell = 2^h - q$, where $h = \lceil \log_2 \ell \rceil$ and $q = 2^h - \ell$. Obviously, from a complete binary tree T of height k, we can obtain a balanced binary tree of height h and with ℓ leaves by choosing q vertices at the $(h-1)$-th level in T and deleting their children from T. So, there are only $b = \binom{2^{h-1}}{q}$ unlabeled balanced binary trees of height h and with ℓ leaves. We denote them by $B_1, \ldots, B_b$ (see Figures 5 and 6 for two examples).

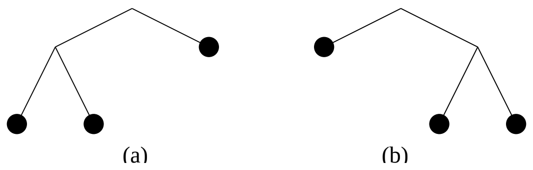

Figure 5: B_1 and B_2 when $\ell = 3$.

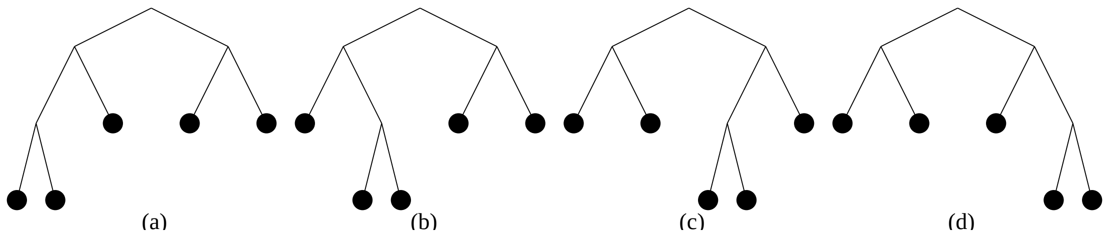

Figure 6: $B_1, \ldots, B_4$ when $\ell = 5$.

We choose a tree B_a among $B_1, \ldots, B_b$ uniformaly at random. For each $i \in \{0, \ldots, h-1\}$, we will use B_a to find an ℓ-proper subset J_i of $I(T_{opt})$. The description of J_i will be much easier if T_{opt} is a complete binary tree. So, we repeat the following steps until T_{opt} becomes a complete binary tree:

1. Select a leaf v whose level in T_{opt} is smaller than the height of T_{opt}.

2. Introduce two new vertices v' and v'' and connect each of them to v by an edge.

3. Assign each of the new leaves v' and v'' the string that is assigned to v.

Obviously, T_{opt} is still a phylogeny but may not be a phylogeny for $\langle s_1, \ldots, s_n \rangle$ because it may have more than n leaves. However, the cost and the height of T_{opt} remain the same. For convenience, we denote the original T_{opt} by T_{opt}^{org}.

We say that a vertex v of T_{opt} is *sufficiently low* in T_{opt} if $lev(v, T_{opt}) \leq h_{opt} - h$, where h_{opt} is the height of T_{opt}. For each sufficiently low vertex v of T_{opt}, we define a subtree $T_{opt}(v, B_a)$ of T_{opt} as follows:

- The root of $T_{opt}(v, B_a)$ is v.
- $T_{opt}(v, B_a)$ is *isomorphic* to B_a, i.e., there is a one-to-one mapping φ from the vertices of $T_{opt}(v, B_a)$ to the vertices of B_a such that
  - $\varphi(v)$ is the root of B_a and
  - for every vertex u of $T_{opt}(v, B_a)$ other than v, if u is the left (respectively, right) child of its parent u' in $T_{opt}(v, B_a)$, then $\varphi(u)$ is the left (respectively, right) child of $\varphi(u')$ in B_a.

Note that $T_{opt}(v, B_a)$ must exist because v is sufficiently low and T_{opt} is a complete binary tree.

We are now ready to use B_a to find an ℓ-proper subset J_i of $I(T_{opt})$ for each $i \in \{0, \ldots, h-1\}$ as follows:

1. Initialize J_i to be the set of all vertices at level i in T_{opt}.

2. For each vertex v of T_{opt}, if $v \in J_i$ and v is sufficiently low in T_{opt}, then color v *black*; otherwise, color v *white*.

3. While T_{opt} has a black vertex, perform the following steps:
   (a) Select an arbitrary black vertex v in T_{opt} and recolor v *white* in T_{opt}.
   (b) Add all leaves of $T_{opt}(v, B_a)$ to J_i.
   (c) Color those leaves of $T_{opt}(v, B_a)$ in T_{opt} *black* that are also sufficiently low in T_{opt}.

Note that J_0 contains the root r of T_{opt} and we delete it from J_0. Then, for each $i \in \{0, \ldots, h-1\}$, J_i is an ℓ-proper subset of $I(T_{opt})$.

Lemma 2.8 *For each $i \in \{0, \ldots, h-1\}$, let $J_i' = J_i \cap I(T_{opt}^{org})$. Then, the following statements hold:*

1. *For each $i \in \{0, \ldots, h-1\}$, J_i' is an ℓ-proper subset of $I(T_{opt}^{org})$ in T_{opt}^{org}.*

2. *If α is a real number (depending on ℓ) such that $\sum_{v \in J_i} c(P(v)) \leq \alpha \cdot c(T_{opt})$, then $\sum_{v \in J_i'} c(P(v)) \leq \alpha \cdot c(T_{opt}^{org})$.*

Proof. The first statement follows from the fact that T_{opt}^{org} is a subtree of T_{opt}. The second statement follows from the fact that $c(P(v)) = 0$ for every $v \in I(T_{opt}) - I(T_{opt}^{org})$. $\square$

By Lemma 2.8, what remains to do is to prove that there is a small number (depending on ℓ) such that $\sum_{v \in J_i} c(P(v)) \leq \alpha \cdot c(T_{opt})$. The proof is not easy because two sets among $J_0, \ldots, J_{h-1}$ may not be disjoint. We give the proof below.

For each $i \in \{0, \ldots, h-1\}$ and for each $j \in \{0, \ldots, h_{opt}\}$, let $y_i(j)$ be the expected number of vertices $v \in J_i$ with $lev(v, T_{opt}) = j$. By the construction of each J_i, we have the following lemma immediately:

Lemma 2.9 *The following statements hold:*

1. $y_0(0) = \cdots = y_0(h-2) = 0$, $y_0(h-1) = q$, $y_0(h) = 2^h - 2q$, *and*

$$y_0(j) = q \cdot y_0(j-h+1) + (2^h - 2q) \cdot y_0(j-h) \text{ for all } j \geq h+1.$$

2. *For every* $i \in \{1, \ldots, h-1\}$, $y_i(0) = \cdots = y_i(i-1) = 0$, $y_i(i) = 2^i$, $y_i(i+1) = \cdots = y_i(i+h-2) = 0$, $y_i(i+h-1) = 2^i q$, *and*

$$y_i(j) = q \cdot y_i(j-h+1) + (2^h - 2q) \cdot y_i(j-h) \text{ for all } j \geq i+h.$$

For each $i \in \{0, \ldots, h-1\}$ and for each $j \in \{0, \ldots, h_{opt}\}$, we define

$$x_i(j) = \frac{y_i(j)}{2^j}.$$

Since B_a is chosen from $B_1, \ldots, B_b$ uniformly at random, each vertex at level j in T_{opt} has the same probability to be included in J_i. So, for each vertex v with $lev(v, T_{opt}) = j$,

$$x_i(j) = \Pr[v \in J_i], \text{ which is the probability that } v \in J_i.$$

Moreover, by Lemma 2.9, we have the following lemma immediately:

Lemma 2.10 *The following statements hold:*

1. $x_0(0) = \cdots = x_0(h-2) = 0$, $x_0(h-1) = \frac{q}{2^{h-1}}$, $x_0(h) = \frac{2^h - 2q}{2^h}$, *and*

$$x_0(j) = \frac{2q}{2^h} \cdot x_0(j-h+1) + \frac{2^h - 2q}{2^h} \cdot x_0(j-h) \text{ for all } j \geq h+1.$$

2. *For every* $i \in \{1, \ldots, h-1\}$, $x_i(0) = \cdots = x_i(i-1) = 0$, $x_i(i) = 1$, $x_i(i+1) = \cdots = x_i(i+h-2) = 0$, $x_i(i+h-1) = \frac{q}{2^{h-1}}$, *and*

$$x_i(j) = \frac{2q}{2^h} \cdot x_i(j-h+1) + \frac{2^h - 2q}{2^h} \cdot x_i(j-h) \text{ for all } j \geq i+h.$$

3. *For each* $i \in \{1, \ldots, h-1\}$ *and for each* $j \in \{1, \ldots, h_{opt}\}$, $x_i(j) = x_{i-1}(j-1)$, *except that* $1 = x_1(1) \neq x_0(0) = 0$.

Note that Statement 3 in Lemma 2.10 follows from Statements 1 and 2 in the same lemma.

It seems very difficult to solve the recurrence relations in Lemma 2.10. We get around the difficulty by considering the following weighted sum of $x_0(j), \ldots, x_{h-1}(j)$ for all $j \geq 0$:

$$X(j) = 2^h \sum_{i=0}^{h-2} x_i(j) + (2^h - 2q) \cdot x_{h-1}(j).$$

Lemma 2.11 *$X(0) = 0$ and $X(j) = 2^h$ for all $j \geq 1$.*

Proof. Using Statements 1 and 2 in Lemma 2.10, it is easy to verify that $X(0) = 0$ and $X(1) = \cdots = X(h) = 2^h$. So, consider an arbitrary integer $j > h$. By Statement 3 in Lemma 2.10,

$$X(j) = 2^h \sum_{i=0}^{h-2} x_0(j-i) + (2^h - 2q) \cdot x_0(j-h+1).$$

Thus,

$$X(j) - X(j-1) = 2^h \cdot x_0(j) - 2q \cdot x_0(j-h+1) - (2^h - 2q) \cdot x_0(j-h).$$

Hence, by the recurrence relation in Statement 1 in Lemma 2.10, $X(j) - X(j-1) = 0$. Therefore, $X(j) = X(j-1) = \cdots = X(h) = 2^h$. □

Next, we consider $\mathcal{E}[\sum_{v \in J_i} c(P(v))]$, which is the expected value of $\sum_{v \in J_i} c(P(v))$. For each $j \geq 0$, let $t(j) = \sum_v c(P(v))$, where v ranges over all vertices v of T_{opt} with $lev(v, T_{opt}) = j$. Then, we have:

$$\mathcal{E}[\sum_{v \in J_i} c(P(v))] = \sum_{j=1}^{h_{opt}} x_i(j) \cdot t(j),$$

because $\Pr[v \in J_i] = x_i(j)$ for each vertex v of T_{opt} with $lev(v, T_{opt}) = j$.

Now, we turn to the following weighted sum of $\mathcal{E}[\sum_{v \in J_0} c(P(v))]$, $\ldots$, $\mathcal{E}[\sum_{v \in J_{h-1}} c(P(v))]$:

$$W = 2^h \cdot \mathcal{E}[\sum_{v \in J_0} c(P(v))] + \cdots + 2^h \cdot \mathcal{E}[\sum_{v \in J_{h-2}} c(P(v))] + (2^h - 2q) \cdot \mathcal{E}[\sum_{v \in J_{h-1}} c(P(v))],$$

which can be rewritten as

$$2^h \cdot \sum_{j=1}^{h_{opt}} x_0(j) \cdot t(j) + \cdots + 2^h \cdot \sum_{j=1}^{h_{opt}} x_{h-2}(j) \cdot t(j) + (2^h - 2q) \cdot \sum_{j=1}^{h_{opt}} x_{h-1}(j) \cdot t(j)$$

and further as

$$\sum_{j=1}^{h_{opt}} t(j) \cdot (2^h \cdot x_0(j) + \cdots + 2^h \cdot x_{h-2}(j) + (2^h - 2q) \cdot x_{h-1}(j)).$$

So, we have

$$W = \sum_{j=1}^{h_{opt}} t(j) \cdot X(j) = 2^h \sum_{j=1}^{h_{opt}} t(j) \leq 2^h \cdot c(T_{opt}),$$

where the last inequality follows from Lemma 2.3. Thus, the weighted average of $\mathcal{E}[\sum_{v \in J_0} c(P(v))]$, $\ldots$, $\mathcal{E}[\sum_{v \in J_{h-1}} c(P(v))]$ is

$$\frac{1}{2^h h - 2q} \cdot W \leq \frac{2^h}{2^h h - 2q} \cdot c(T_{opt}).$$

Hence, there is an $i \in \{0, \ldots, h-1\}$ such that

$$\mathcal{E}[\sum_{v \in J_i} c(P(v))] \leq \frac{2^h}{2^h h - 2q} \cdot c(T_{opt}).$$

Therefore, there is a choice of B_a from $\{B_1, \ldots, B_b\}$ such that the subset J_i constructed from B_a as above satisfies the following inequality:

$$\sum_{v \in J_i} c(P(v)) \leq \frac{2^h}{2^h h - 2q} \cdot c(T_{opt}).$$

In summary, we have proven the following lemma:

Lemma 2.12 *For every integer $\ell \geq 3$ (that is not a power of 2), there is always an ℓ-proper subset J of $I(T_{opt})$ such that $\sum_{v \in J} c(P(v)) \leq \frac{2^h}{2^h h - 2q} \cdot c(T_{opt})$, where $h = \lceil \log_2 \ell \rceil$ and $q = 2^h - \ell$.*

Combining Theorem 2.2 and Lemmas 2.1 and 2.12, we now have:

Theorem 2.13 *Our algorithm is a PTAS for 1-DHR and achieves a ratio of $1 + \frac{2^h}{2^h h - 2q}$ in $O(n^{\ell+1} + mn^{\ell})$ time for every integer $\ell \geq 3$ (that is not a power of 2), where $h = \lceil \log_2 \ell \rceil$ and $q = 2^h - \ell$.*

For example, to achieve a ratio of $\frac{5}{3}$, our PTAS takes $O(n^4 + mn^3)$ time while the PTAS in [4] takes $O(n^7 + mn^3)$ time.

3 A Ratio-6 Approximation Algorithm for 2-DHR

Throughout this section, fix a list $\langle s_1, \ldots, s_n \rangle$ of strings of the same length m, and let M_{opt} be a minimum-cost duplication model for $s_1 s_2 \ldots s_n$.

One problem with M_{opt} is that the label assigned to an internal vertex in M_{opt} may not be in $\{s_1, \ldots, s_n\}$. This makes it hard to compute M_{opt}. So instead, we look for a restricted type of duplication models called *lifted* duplication models. In a lifted duplication model, the label assigned to each internal vertex is a string in $\{s_1, \ldots, s_n\}$. The following lemma shows that there is always a good lifted duplication model:

Lemma 3.1 *There is a lifted duplication model N_{opt} for $s_1 s_2 \ldots s_n$ such that $c(N_{opt}) \leq 2 \cdot c(M_{opt})$.*

PROOF. Recall that $T_{M_{opt}}$ denotes the associated phylogeny for M_{opt}. Note that $c(M_{opt}) = c(T_{M_{opt}})$. Using the *lifting technique* in [10], we can elaborately lift the labels of the leaves in $T_{M_{opt}}$ up to their ancestors so that the resulting phylogeny T has a cost less than or equal to $2 \cdot c(T_{M_{opt}})$. Now, since the vertices of M_{opt} one-to-one correspond to those of T, we can obtain a new model N_{opt} from M_{opt} by simply changing the label of each vertex in M_{opt} to that of the corresponding vertex in T. Obviously, N_{opt} is a lifted duplication model for $s_1 s_2 \ldots s_n$. Moreover, $c(N_{opt}) = c(T) \leq 2 \cdot c(T_{M_{opt}}) = 2 \cdot c(M_{opt})$. □

The rest of this section is organized as follows. In Section 3.1, we give several definitions and prove several lemmas. In Section 3.2, we define the component tree $\mathcal{D}(M)$ of a given lifted duplication model M for $s_1 s_2 \ldots s_n$. In Section 3.3, we show how to construct a new lifted duplication model M' from $\mathcal{D}(M)$. The crucial point is that we can show $c(M') \leq 3 \cdot c(M)$. In Section 3.4, we describe our algorithm for computing a component tree $\mathcal{D}_{opt}$ such that the cost of the new lifted duplication model constructed from $\mathcal{D}_{opt}$ is minimized.

3.1 Preliminaries

Let M be a lifted duplication model for $s_1 s_2 \dots s_n$. An edge in M is *planar* if it is not crossed by another edge in M. A path in M is *planar* if it traverses planar edges only. Two internal vertices u and v in M are *incomparable* if u is neither an ancestor nor a descendant of v in M.

Each size-2 block B in M contains two internal vertices; we distinguish them as the *left vertex* and the *right vertex* in B. For each block B, we define the *leftmost planar path* of B and the *rightmost planar path* of B as follows: If the size of B is 1, then both paths start at the unique internal vertex in B; otherwise, the leftmost planar path of B starts at the left vertex in B and the rightmost planar path of B starts at the right vertex in B. The leftmost (respectively, rightmost) planar path of B then goes down all the way to a leaf by only traversing planar edges based on the following rule:

- When going down from an internal vertex u to one of its children, always choose the left (respectively, right) child v of u in B if the edge (u, v) is planar. (*Comment:* Since M is a duplication model, at least one of the two edges between u and its children is planar. So, the path can always go down to a leaf by traversing planar edges only.)

An *extreme planar path* in M is a path that is the leftmost or rightmost of some block in M. The next lemma is obvious but helps the reader understand the relations between extreme planar paths in M.

Lemma 3.2 *The following statements hold:*

1. *If an internal vertex v of M is contained in a size-1 block, then there are exactly two extreme planar paths starting at v and v is the unique vertex shared by the two paths.*

2. *If an internal vertex v of M is contained in a size-2 block, then there is exactly one extreme planar path starting at v.*

3. *If u and v are two incomparable internal vertices in M, then no extreme planar path starting at u shares a vertex with an extreme planar path starting at v.*

For each integer i with $1 \le i \le n$, the i-th leftmost leaf in M is labeled with the string s_i. For convenience, we also use s_i to denote the i-th leftmost leaf in M. For each internal vertex u of M, we use $a(u)$ (respectively, $b(u)$) to denote the integer i such that s_i is the leftmost (respectively, rightmost) leaf descendant of u in M. Two internal vertices u and v are *unrelated* in M if $b(u) < a(v)$ or $b(v) < a(u)$. An internal vertex u *crosses* another internal vertex v in M if $a(u) < a(v) < b(u) < b(v)$. An internal vertex u *covers* another internal vertex v in M if $a(u) \le a(v) < b(v) \le b(u)$. Note that if u is ancestor of v in M, then u covers v in M. However, an internal vertex may cover another internal vertex in M even if they are incomparable in M. Two internal vertices of M are *unnested* if no one of them covers the other in M.

The next lemma is obvious but helps the reader understand how a vertex crosses another vertex in M.

Lemma 3.3 *If a vertex u crosses another vertex v in M, then the path P_u from u to $s_{b(u)}$ in M and the path P_v from v to $s_{a(v)}$ in M cross each other exactly once and hence there is exactly one size-2 box $B_{u,v}$ in M whose left vertex is on P_u and whose right vertex is on P_v.*

We call the block $B_{u,v}$ in Lemma 3.3 the *block for* (u, v)*-crossing* in M.

The next two lemmas help the reader understand the relations between unnested internal vertices in M.

Lemma 3.4 *There do not exist three pairwise unnested internal vertices u, v, and w in M such that u crosses v in M and so does w.*

PROOF. Suppose that an internal vertex u crosses another internal vertex v in M. Let $\langle x_1, y_1\rangle$ be the block for (u, v)-crossing in M. Suppose that an internal vertex w other than u also crosses v in M. Let $\langle x_2, y_2\rangle$ be the block for (w, v)-crossing in M. Then, both y_1 and y_2 appear on the path P_v from v to $s_{a(v)}$ in M. So, one of y_1 and y_2 is an ancestor of the other in M. Without loss of generality, we may assume that y_1 is an ancestor of y_2 in M. Then, the leftmost planar path starting at x_1 must end at a leaf s_i with $i < a(w)$. So, $a(u) < i < a(w)$ because u covers x_1 in M. Similarly, the rightmost planar path starting at y_2 must end at a leaf s_k with $b(w) < k < b(u)$. Thus, u covers w in M. This establishes the lemma. □

Lemma 3.5 *Suppose that u, v, and w are three pairwise unnested internal vertices in M such that u crosses v in M and v crosses w in M. Then, $a(v) < b(u) < a(w) < b(v)$ and hence u does not cross w in M.*

PROOF. Since u crosses v in M, we have $a(v) < b(u)$ immediately. Similarly, $a(w) < b(v)$. To prove $b(u) < a(w)$, consider the right vertex x in the block for (u, v)-crossing and the left vertex y in the block for (v, w)-crossing. Obviously, the rightmost planar path P_x starting at x must end at a leaf s_i with $b(u) < i$ and the leftmost planar path P_y starting at y must end at a leaf s_j with $j < a(w)$. Moreover, since x is on the left of y in M, P_x is on the left of P_y and so $i < j$. Hence, $b(u) < i < j < a(w)$. □

A triple (L, i, j) is *closed* in M if either $i = j$ and L consists of s_i only, or the following conditions are satisfied:

- i and j are integers with $1 \leq i < j \leq n$.
- L is an ordered list $\langle v_1, v_2, \ldots, v_k\rangle$ of pairwise unnested internal vertices with $1 \leq k \leq 4$.
- The left child of v_1 in M is an ancestor of s_i, and the right child of v_k in M is an ancestor of s_j.
- The path from v_1 to s_i in M is planar and so is the path from v_k to s_j in M.
- For each integer h with $2 \leq h \leq k - 1$, all leaf descendants of v_h in M are among s_i, s_{i+1}, $\ldots$, s_j.
- Every leaf s_ℓ with $i \leq \ell \leq j$ is a descendant of some v_h with $1 \leq h \leq k$ in M.
- For each integer h with $1 \leq h \leq k - 1$, v_h crosses v_{h+1} in M.

For example, both $(\langle t_2\rangle, 1, 4)$ and $(\langle t_5, t_6\rangle, 4, 15)$ are closed triples in the duplication model in Figure 7(a).

For each closed triple (L, i, j) in M, we use $M(L, i, j)$ to denote the subgraph of M induced by the set of those vertices u in M such that u is a vertex in L, u is a leaf s_h with $i \le h \le j$, or u is both an ancestor of some s_h with $i \le h \le j$ and a descendant of some vertex of L in M. For example, $M(\langle r\rangle, 1, n)$ is exactly M, where r is the root of M.

Note that $M(L, i, j)$ is a forest consisting of $|L|$ rooted trees whose roots are the vertices in L and whose leaves are s_i, $\ldots$, s_j. Moreover, $M(L, i, j)$ may have unifurcate vertices, which are those vertices having only one child in $M(L, i, j)$. Obviously, each unifurcate vertex in $M(L, i, j)$ appears on the path from the leaf s_i to the first vertex in L or on the path from the leaf s_j to the last vertex in L.

For each closed triple (L, i, j) in M, we call $M(L, i, j)$ a *component* of M and call the vertices in L the *roots* of the component. Note that the roots of $M(L, i, j)$ are ordered from left to right. For example, the leftmost root is the first root while the rightmost root is the last root.

3.2 The Component Tree of a Model

We inherit the notations in Section 3.1. For each vertex v of M, let $s(v)$ denote the string assigned to v in M. Moreover, for a list L of vertices in M, let $s(L)$ denote the list of strings assigned to the vertices in L. Furthermore, for two strings s' and s'', let $d(s', s'')$ denote the hamming distance between them.

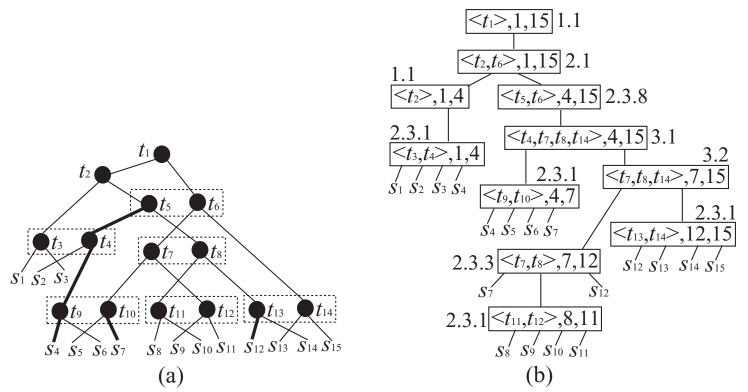

Figure 7: (a) A duplication model M where the three bold paths are the splitting paths for obtaining $\mathcal{D}(M)$. (b) The component tree $\mathcal{D}(M)$ of M where the type of each node is given near the node, the weight of each edge is omitted, and the label $(\langle s_i\rangle, i, i)$ of each leaf is simplified to s_i.

We want to dissect M into components and organize them into a rooted, ordered, edge-weighted, and node-labeled tree $\mathcal{D}(M)$, called the *component tree* of M (see Figure 7 for an example). Since we want to do this recursively, it is more convenient to dissect $M(L, i, j)$ into components and organize them into a rooted, ordered, edge-weighted, and node-labeled tree $\mathcal{D}_M(L, i, j)$ (called the *component tree* of $M(L, i, j)$) for each closed triple (L, i, j) in M.

We next construct $\mathcal{D}_M(L, i, j)$ by induction on $j - i$. In the base case where $j - i = 0$, $\mathcal{D}_M(L, i, j)$ has only one node $\alpha(L, i, j)$; we label the node with $(s(L), i, j)$ and call it a *type-0* node. So, suppose that $j - i \ge 1$. Then, depending on $|L|$, there are three cases. In each case, we first create a root node $\alpha(L,i,j)$ for $\mathcal{D}_M(L,i,j)$ and label it with $(s(L),i,j)$. Then, we proceed to grow $\mathcal{D}_M(L,i,j)$ in each case as follows:

Case 1: $|L|=1$. Let u be the root of $M(L,i,j)$, and let v_1 (respectively, v_2) be the first vertex on the path from u to the leaf s_i (respectively, s_j) that is not a unifurcate internal vertex in $M(L,i,j)$. We further distinguish two subcases as follows:

Case 1.1: v_1 crosses v_2 in M. In this subcase, $(\langle v_1,v_2\rangle,i,j)$ is a closed triple in M. So, we recursively construct $\mathcal{D}_M(\langle v_1,v_2\rangle,i,j)$, then let $\alpha(\langle v_1,v_2\rangle,i,j)$ be the unique child of $\alpha(L,i,j)$, and further let the weight of the edge between $\alpha(L,i,j)$ and its child be $d(s(u),s(v_1))+d(s(u),s(v_2))$. We also call $\alpha(L,i,j)$ a *type-1.1* node.

Case 1.2: v_1 does not cross v_2 in M. In this subcase, both $(\langle v_1\rangle,i,b(v_1))$ and $(\langle v_2\rangle,a(v_2),j)$ are closed triples in M. So, we recursively construct $\mathcal{D}_M(\langle v_1\rangle,i,b(v_1))$ and $\mathcal{D}_M(\langle v_2\rangle,a(v_2),j)$. We then let $\alpha(\langle v_1\rangle,i,b(v_1))$ and $\alpha(\langle v_2\rangle,a(v_2),j)$ be the left and the right child of $\alpha(L,i,j)$, respectively. We further let the weight of the edge between $\alpha(L,i,j)$ and its left (respectively, right) child be $d(s(u),s(v_1))$ (respectively, $d(s(u),s(v_2))$). We also call $\alpha(L,i,j)$ a *type-1.2* node.

Case 2: $|L|=2$. Let u_1 and u_2 be the roots of $M(L,i,j)$. Let $\langle v_1,v_2\rangle$ be the block for (u_1,u_2)-crossing in M. We further distinguish three subcases as follows:

Case 2.1: $u_1\neq v_1$. Let s_k be the leaf at which the leftmost planar path starting at v_1 ends. Then, $(\langle u_1\rangle,i,k)$ and $(\langle v_1,u_2\rangle,k,j)$ are closed triples in M. So, we recursively construct $\mathcal{D}_M(\langle u_1\rangle,i,k)$ and $\mathcal{D}_M(\langle v_1,u_2\rangle,k,j)$. Intuitively speaking, this means that we split $M(L,i,j)$ into two components along the path from v_1 to s_k and then construct their component trees recursively. For convenience, we call the path a *splitting path* and associate it with $\alpha(L,i,j)$. Now, to finish constructing $\mathcal{D}_M(L,i,j)$, we let $\alpha(\langle u_1\rangle,i,k)$ and $\alpha(\langle v_1,u_2\rangle,k,j)$ be the left and the second child of $\alpha(L,i,j)$, respectively. We further let the weight of the edge between $\alpha(L,i,j)$ and each of its children be $\frac{1}{2}d(s_k,s(v_1))$. We also call $\alpha(L,i,j)$ a *type-2.1* node.

Case 2.2: $u_1=v_1$ but $u_2\neq v_2$. Let s_ℓ be the leaf at which the rightmost planar path starting at v_2 ends. Then, $(\langle u_1,v_2\rangle,i,\ell)$ and $(\langle u_2\rangle,\ell,j)$ are closed triples in M. So, we recursively construct $\mathcal{D}_M(\langle u_1,v_2\rangle,i,\ell)$ and $\mathcal{D}_M(\langle u_2\rangle,\ell,j)$. Intuitively speaking, this means that we split $M(L,i,j)$ into two components along the path from v_2 to s_ℓ and then construct their component trees recursively. For convenience, we call the path a *splitting path* and associate it with $\alpha(L,i,j)$. Now, to finish constructing $\mathcal{D}_M(L,i,j)$, we let $\alpha(\langle u_1,v_2\rangle,i,\ell)$ and $\alpha(\langle u_2\rangle,\ell,j)$ be the left and the right child of $\alpha(L,i,j)$, respectively. We further let the weight of the edge between $\alpha(L,i,j)$ and each of its children be $\frac{1}{2}d(s_\ell,s(v_2))$. We also call $\alpha(L,i,j)$ a *type-2.2* node.

Case 2.3: $u_1=v_1$ and $u_2=v_2$. Let w_1 (respectively, w_4) be the first vertex on the path from u_1 (respectively, u_2) to the leaf s_i (respectively, s_j) that is not a unifurcate internal vertex in $M(L,i,j)$. Let w_3 (respectively, w_2) be the right (respectively, left) child of u_1 (respectively, u_2) in $M(L,i,j)$. By Lemma 3.5, one of the following subsubcases occurs:

Case 2.3.1: There do not exist two vertices w_i and w_j with $1\le i<j\le 4$ such that w_i crosses w_j in M. Obviously, $(\langle w_1\rangle,i,b(w_1))$, $(\langle w_2\rangle,a(w_2),b(w_2))$, $(\langle w_3\rangle,a(w_3),b(w_3))$, and $(\langle w_4\rangle,a(w_4),j)$ are closed triples in M. Note that $a(w_h)=b(w_{h-1})+1$ for $2\le h\le 4$. So, we recursively construct $\mathcal{D}_M(\langle w_1\rangle,i,b(w_1))$, $\mathcal{D}_M(\langle w_2\rangle,a(w_2),b(w_2))$, $\mathcal{D}_M(\langle w_3\rangle,a(w_3),b(w_3))$, and $\mathcal{D}_M(\langle w_4\rangle,a(w_4),j)$. We then let $\alpha(\langle w_1\rangle,i,b(w_1))$, $\alpha(\langle w_2\rangle,a(w_2),b(w_2))$, $\alpha(\langle w_3\rangle,a(w_3),b(w_3))$, and $\alpha(\langle w_4\rangle,a(w_4),j)$ be the first (i.e., the leftmost), the second, the third, and the fourth child of $\alpha(L,i,j)$, respectively. We further let the weight of the edge between $\alpha(L,i,j)$ and its first, its second, its third, and its fourth

child be $d(s(u_1), s(w_1))$, $d(s(u_2), s(w_2))$, $d(s(u_1), s(w_3))$, and $d(s(u_2), s(w_4))$, respectively. We also call $\alpha(L, i, j)$ a *type-2.3.1* node.

Case 2.3.2: w_1 crosses w_2 in M but neither w_2 crosses w_3 nor w_3 crosses w_4 in M. Obviously, $(\langle w_1, w_2\rangle, i, b(w_2))$, $(\langle w_3\rangle, a(w_3), b(w_3))$, and $(\langle w_4\rangle, a(w_4), j)$ are closed triples in M. Note that $a(w_3) = b(w_2) + 1$ and $a(w_4) = b(w_3) + 1$. So, we recursively construct $\mathcal{D}_M(\langle w_1, w_2\rangle, i, b(w_2))$, $\mathcal{D}_M(\langle w_3\rangle, a(w_3), b(w_3))$, and $\mathcal{D}_M(\langle w_4\rangle, a(w_4), j)$. We then let $\alpha(\langle w_1, w_2\rangle, i, b(w_2))$, $\alpha(\langle w_3\rangle, a(w_3), b(w_3))$, and $\alpha(\langle w_4\rangle, a(w_4), j)$ be the left, the middle, and the right child of $\alpha(L, i, j)$, respectively. We further let the weight of the edge between $\alpha(L, i, j)$ and its left, its middle, and its right child be $d(s(u_1), s(w_1)) + d(s(u_2), s(w_2))$, $d(s(u_1), s(w_3))$, and $d(s(u_2), s(w_4))$, respectively. We also call $\alpha(L, i, j)$ a *type-2.3.2* node.

Case 2.3.3: w_2 crosses w_3 in M but neither w_1 crosses w_2 nor w_3 crosses w_4 in M. Obviously, $(\langle w_1\rangle, i, b(w_1))$, $(\langle w_2, w_3\rangle, a(w_2), b(w_3))$, and $(\langle w_4\rangle, a(w_4), j)$ are closed triples in M. Note that $a(w_2) = b(w_1) + 1$ and $a(w_4) = b(w_3) + 1$. So, we recursively construct $\mathcal{D}_M(\langle w_1\rangle, i, b(w_1))$, $\mathcal{D}_M(\langle w_2, w_3\rangle, a(w_2), b(w_3))$, and $\mathcal{D}_M(\langle w_4\rangle, a(w_4), j)$. We then let $\alpha(\langle w_1\rangle, i, b(w_1))$, $\alpha(\langle w_2, w_3\rangle, a(w_2), b(w_3))$, and $\alpha(\langle w_4\rangle, a(w_4), j)$ be the left, the middle, and the right child of $\alpha(L, i, j)$, respectively. We further let the weight of the edge between $\alpha(L, i, j)$ and its left, its middle, and its right child be $d(s(u_1), s(w_1))$, $d(s(u_1), s(w_3)) + d(s(u_2), s(w_2))$, and $d(s(u_2), s(w_4))$, respectively. We also call $\alpha(L, i, j)$ a *type-2.3.3* node.

Case 2.3.4: w_3 crosses w_4 in M but neither w_1 crosses w_2 nor w_2 crosses w_3 in M. Obviously, $(\langle w_1\rangle, i, b(w_1))$, $(\langle w_2\rangle, a(w_2), b(w_2))$, and $(\langle w_3, w_4\rangle, a(w_3), j)$ are closed triples in M. Note that $a(w_2) = b(w_1) + 1$ and $a(w_3) = b(w_2) + 1$. So, we recursively construct $\mathcal{D}_M(\langle w_1\rangle, i, b(w_1))$, $\mathcal{D}_M(\langle w_2\rangle, a(w_2), b(w_2))$, and $\mathcal{D}_M(\langle w_3, w_4\rangle, a(w_3), j)$. We then let $\alpha(\langle w_1\rangle, i, b(w_1))$, $\alpha(\langle w_2\rangle, a(w_2), b(w_2))$, and $\alpha(\langle w_3, w_4\rangle, a(w_3), j)$ be the left, the middle, and the right child of $\alpha(L, i, j)$, respectively. We further let the weight of the edge between $\alpha(L, i, j)$ and its left, its middle, and its right child be $d(s(u_1), s(w_1))$, $d(s(u_2), s(w_2))$, and $d(s(u_1), s(w_3)) + d(s(u_2), s(w_4))$, respectively. We also call $\alpha(L, i, j)$ a *type-2.3.4* node.

Case 2.3.5: Both w_1 crosses w_2 and w_2 crosses w_3 but w_3 does not cross w_4 in M. Obviously, $(\langle w_1, w_2, w_3\rangle, i, b(w_3))$ and $(\langle w_4\rangle, a(w_4), j)$ are closed triples in M. Note that $a(w_4) = b(w_3) + 1$. So, we recursively construct $\mathcal{D}_M(\langle w_1, w_2, w_3\rangle, i, b(w_3))$ and $\mathcal{D}_M(\langle w_4\rangle, a(w_4), j)$. We then let $\alpha(\langle w_1, w_2, w_3\rangle, i, b(w_3))$ and $\alpha(\langle w_4\rangle, a(w_4), j)$ be the left and the right child of $\alpha(L, i, j)$, respectively. We further let the weight of the edge between $\alpha(L, i, j)$ and its left (respectively, right) child be $d(s(u_1), s(w_1)) + d(s(u_2), s(w_2)) + d(s(u_1), s(w_3))$ (respectively, $d(s(u_2), s(w_4))$). We also call $\alpha(L, i, j)$ a *type-2.3.5* node.

Case 2.3.6: Both w_1 crosses w_2 and w_3 crosses w_4 but w_2 does not cross w_3 in M. Obviously, $(\langle w_1, w_2\rangle, i, b(w_2))$ and $(\langle w_3, w_4\rangle, a(w_3), j)$ are closed triples in M. Note that $a(w_3) = b(w_2) + 1$. So, we recursively construct $\mathcal{D}_M(\langle w_1, w_2\rangle, i, b(w_2))$ and $\mathcal{D}_M(\langle w_3, w_4\rangle, a(w_3), j)$. We then let $\alpha(\langle w_1, w_2\rangle, i, b(w_2))$ and $\alpha(\langle w_3, w_4\rangle, a(w_3), j)$ be the left and the right child of $\alpha(L, i, j)$, respectively. We further let the weight of the edge between $\alpha(L, i, j)$ and its left (respectively, right) child be $d(s(u_1), s(w_1)) + d(s(u_2), s(w_2))$ (respectively, $d(s(u_1), s(w_3)) + d(s(u_2), s(w_4))$). We also call $\alpha(L, i, j)$ a *type-2.3.6* node.

Case 2.3.7: Both w_2 crosses w_3 and w_3 crosses w_4 but w_1 does not cross w_2 in M. Obviously, $(\langle w_1\rangle, i, b(w_1))$ and $(\langle w_2, w_3, w_4\rangle, a(w_2), j)$ are closed triples in M. Note that $a(w_2) = b(w_1) + 1$. So, we recursively construct $\mathcal{D}_M(\langle w_1\rangle, i, b(w_1))$ and $\mathcal{D}_M(\langle w_2, w_3, w_4\rangle, a(w_2), j)$. We then let

$\alpha(\langle w_1\rangle, i, b(w_1))$ and $\alpha(\langle w_2, w_3, w_4\rangle, a(w_2), j)$ be the left and the right child of $\alpha(L, i, j)$, respectively. We further let the weight of the edge between $\alpha(L, i, j)$ and its left (respectively, right) child be $d(s(u_1), s(w_1))$ (respectively, $d(s(u_2), s(w_2)) + d(s(u_1), s(w_3)) + d(s(u_2), s(w_4))$). We also call $\alpha(L, i, j)$ a *type-2.3.7* node.

Case 2.3.8: For each integer h with $1 \le h \le 3$, w_h crosses w_{h+1} in M. Obviously, $(\langle w_1, w_2, w_3, w_4\rangle, i, j)$ is a closed triple in M. So, we recursively construct $\mathcal{D}_M(\langle w_1, w_2, w_3, w_4\rangle, i, j)$. We then let $\alpha(\langle w_1, w_2, w_3, w_4\rangle, i, j)$ be the unique child of $\alpha(L, i, j)$. We further let the edge between $\alpha(L, i, j)$ and its child be $d(s(u_1), s(w_1)) + d(s(u_2), s(w_2)) + d(s(u_1), s(w_3)) + d(s(u_2), s(w_4))$. We also call $\alpha(L, i, j)$ a *type-2.3.8* node.

Case 3: $3 \le |L| \le 4$. Let $h = |L|$. Let $u_1, \ldots, u_h$ be the roots of $M(L, i, j)$. Let $\langle v_1, v_2\rangle$ be the block for (u_1, u_2)-crossing in M, and let $\langle w_2, w_3\rangle$ be the block for (u_2, u_3)-crossing in M. We further distinguish two subcases as follows:

Case 3.1: $u_2 \ne v_2$. Let s_k be the leaf at which the rightmost planar path starting at v_2 ends. Then, $(\langle u_1, v_2\rangle, i, k)$ and $(\langle u_2, \ldots, u_h\rangle, k, j)$ are closed triples in M. So, we recursively construct $\mathcal{D}_M(\langle u_1, v_2\rangle, i, k)$ and $\mathcal{D}_M(\langle u_2, \ldots, u_h\rangle, k, j)$. Intuitively speaking, this means that we split $M(L, i, j)$ into two components along the path from v_2 to s_k and then construct their component trees recursively. For convenience, we call the path a *splitting path* and associate it with $\alpha(L, i, j)$. Now, to finish constructing $\mathcal{D}_M(L, i, j)$, we just let $\alpha(\langle u_1, v_2\rangle, i, k)$ and $\alpha(\langle u_2, \ldots, u_h\rangle, k, j)$ be the left and the second child of $\alpha(L, i, j)$, respectively. We further let the weight of the edge between $\alpha(L, i, j)$ and each of its child be $\frac{1}{2}d(s_k, s(v_2))$. We also call $\alpha(L, i, j)$ a *type-3.1* node.

Case 3.2: $u_2 = v_2$. Then, $u_2 \ne w_2$. Let s_ℓ be the leaf at which the leftmost planar path starting at w_2 ends. Then, $(\langle u_1, u_2\rangle, i, \ell)$ and $(\langle w_2, u_3, \ldots, u_h\rangle, \ell, j)$ are closed triples in M. So, we recursively construct $\mathcal{D}_M(\langle u_1, u_2\rangle, i, \ell)$ and $\mathcal{D}_M(\langle w_2, u_3, \ldots, u_h\rangle, \ell, j)$. Intuitively speaking, this means that we split $M(L, i, j)$ into two components along the path from w_2 to s_ℓ and then construct their component trees recursively. For convenience, we call the path a *splitting path* and associate it with $\alpha(L, i, j)$. Now, to finish constructing $\mathcal{D}_M(L, i, j)$, we just let $\alpha(\langle u_1, u_2\rangle, i, \ell)$ and $\alpha(\langle w_2, u_3, \ldots, u_h\rangle, \ell, j)$ be the left and the second child of $\alpha(L, i, j)$, respectively. We further let the edge between $\alpha(L, i, j)$ and each of its child be $\frac{1}{2}d(s_\ell, s(w_2))$. We also call $\alpha(L, i, j)$ a *type-3.2* node.

Lemma 3.6 *The component tree $\mathcal{D}(M)$ of M is unique. Moreover, each pair of splitting paths associated with nodes in $\mathcal{D}(M)$ are edge-disjoint.*

Proof. The first assertion is obvious from the construction of $\mathcal{D}(M)$. We next prove the second assertion. For each component $M(L, i, j)$ of M, we define the *left* (respectively, *right*) boundary of $M(L, i, j)$ to be the path from the first (respectively, last) vertex in L to s_i (respectively, s_j) in M. By examining each case in the above construction of $\mathcal{D}_M(L, i, j)$, we can see that if a splitting path is used to split $M(L, i, j)$ into two smaller components, the path does not traverse any edge on the left or right boundary of $M(L, i, j)$ and the splitting path becomes the left or right boundary of each of the smaller components. So, it is impossible for two splitting paths to share an edge. $\square$

We define the *weight* of $\mathcal{D}(M)$ to be the total weight of edges in $\mathcal{D}(M)$.

Lemma 3.7 *The weight of $\mathcal{D}(M)$ is at most $3 \cdot c(M)$.*

Proof. For each closed triple (L, i, j), let $c(M(L, i, j))$ be the total cost of edges in $M(L, i, j)$, and let $c(\mathcal{D}_M(L, i, j))$ be the total weight of edges in $\mathcal{D}_M(L, i, j)$. Moreover, if there are one or more splitting paths associated with $\alpha(L, i, j)$ or its descendants in $\mathcal{D}_M(L, i, j)$, then let $c_p(L, i, j)$ denote the total cost of edges on all the paths; otherwise, let $c_p(L, i, j) = 0$.

To show the lemma, we first claim that for each closed triple (L, i, j), $c(\mathcal{D}_M(L, i, j)) \leq c(M(L, i, j)) + 2 \cdot c_p(L, i, j)$. We prove the claim by induction on $j - i$. The claim is clearly true when $j - i = 0$. So, suppose that $j - i \geq 1$. If no splitting path is associated with $\alpha(L, i, j)$ in $\mathcal{D}_M(L, i, j)$, then by the inductive hypothesis and the definition of the weight(s) between $\alpha(L, i, j)$ and its child(ren) in $\mathcal{D}_M(L, i, j)$, it is clear that $c(\mathcal{D}_M(L, i, j)) \leq c(M(L, i, j)) + 2 \cdot c_p(L, i, j)$. Otherwise, by the definition of the weights between $\alpha(L, i, j)$ and its children in $\mathcal{D}_M(L, i, j)$,

$$c(\mathcal{D}_M(L, i, j)) \leq c(\mathcal{D}_M(L_1, i, k)) + c(\mathcal{D}_M(L_2, k, j)) + c(P),$$

where P is the splitting path associated with $\alpha(L, i, j)$ and $\alpha(L_1, i, k)$ and $\alpha(L_2, k, j))$ are the children of $\alpha(L, i, j)$ in $\mathcal{D}_M(L, i, j)$. Moreover, we have

$$\begin{aligned}
& c(\mathcal{D}_M(L_1, i, k)) + c(\mathcal{D}_M(L_2, k, j)) + c(P) \\
\leq\ & c(M(L_1, i, k)) + 2 \cdot c_p(L_1, i, k) + c(M(L_2, k, j)) + 2 \cdot c_p(L_2, k, j) + c(P) \\
=\ & c(M(L, i, j)) + 2 \cdot c_p(L_1, i, k) + 2 \cdot c_p(L_2, k, j) + 2 \cdot c(P) \\
=\ & c(M(L, i, j)) + 2 \cdot c_p(L, i, j), \qquad (3.1)
\end{aligned}$$

where the inequality follows from the inductive hypothesis and the first equality follows from the fact that the edges shared by $M(L_1, i, k)$ and $M(L_2, k, j)$ are exactly the edges of P. This completes the proof of the claim. The lemma follows from the claim and Lemma 3.6 immediately. □

3.3 Constructing Models from Component Trees

For convenience, we will allow a duplication model to have unifurcate internal vertices from now on. This does not cause any problem because we can obtain a real duplication model by repeating merging the two edges incident to a unifurcate vertex into a single edge until there are no unifurcate vertices in the model.

We inherit the notations in Sections 3.1 and 3.2. We show how to use $\mathcal{D}(M)$ to construct a duplication model M' for $s_1 s_2 \ldots s_n$ such that $c(M')$ equals the total weight of edges in $\mathcal{D}(M)$ (see Figure 8 for an example). In the construction of M', we will only use the label and the type of each node in $\mathcal{D}(M)$, i.e., we will not look at the topology and the vertices of M.

Recall that the label of each node β in $\mathcal{D}(M)$ is a triple $(\mathcal{S}, i, j)$, where $\mathcal{S}$ is an ordered nonempty list of at most four (possibly not distinct) strings among $s_1, \ldots, s_n$ and i and j are two integers with $1 \leq i \leq j \leq n$. For convenience, we call $\mathcal{S}$ the *string list* of β.

The construction of M' indeed involves constructing a rooted ordered forest $M'(\beta)$ for each node β of $\mathcal{D}(M)$. We will maintain the invariant that $M'(\beta)$ has $|\mathcal{S}|$ roots, where $\mathcal{S}$ is the string list of β.

We next detail the construction of M'. We construct M' by processing the nodes of $\mathcal{D}(M)$ in a bottom-up fashion. We first process each leaf β in $\mathcal{D}(M)$ by constructing $M'(\beta)$ as follows: Create a new vertex and assign it the unique string in the string list of β.

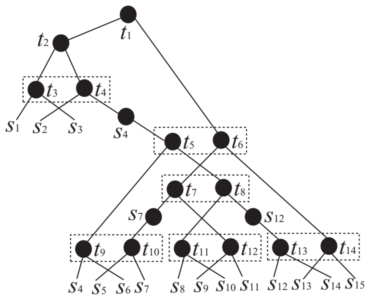
t_1
t_2
t_3
t_4
s_1 s_2 s_3
s_4
t_5
t_6
t_7
t_8
s_7
s_{12}
t_9 t_{10} t_{11} t_{12} t_{13} t_{14}
s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}

Figure 8: The new duplication model M' constructed from the component tree in Figure 7(b).

Now, consider the processing of an internal node β in $\mathcal{D}(M)$. Let $\gamma_1, \ldots, \gamma_h$ be the children of β in $\mathcal{D}(M)$, where the ordering is from left to right. Suppose that $M'(\gamma_1), \ldots, M'(\gamma_h)$ have been constructed by processing $\gamma_1, \ldots, \gamma_h$. The processing of β will depend on its type. Since the possible types of β one-to-one correspond to the cases in Section 3.2, we construct $M'(\beta)$ by distinguishing several cases as follows:

Type 1.1 (cf. Case 1.1 in Section 3.2): We create a root vertex for $M'(\beta)$, assign it the unique string in the string list of β, and connect it to the roots of $M'(\gamma_1)$ by two new edges. Note that the total cost of the two new edges is exactly the weight of the edge between β and γ_1 in $\mathcal{D}(M)$.

Type 1.2 (cf. Case 1.2 in Section 3.2): We create a root vertex u for $M'(\beta)$, assign it the unique string in the string list of β, connect it to the root v_1 of $M'(\gamma_1)$ and to the root v_2 of $M'(\gamma_2)$ so that v_1 and v_2 become the left and the right child of u in $M'(\beta)$, respectively. Note that the total costs of the new edges (u, v_1) and (u, v_2) in $M'(\beta)$ are exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.1 (cf. Case 2.1 in Section 3.2): We connect $M'(\gamma_1)$ and $M'(\gamma_2)$ by making the left root v_2 of $M'(\gamma_2)$ be the unique child of the rightmost leaf v_1 of $M'(\gamma_1)$. Note that the cost of the new edge (v_1, v_2) in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.2 (cf. Case 2.2 in Section 3.2): We connect $M'(\gamma_1)$ and $M'(\gamma_2)$ by making the right root v_1 of $M'(\gamma_1)$ be the unique child of the leftmost leaf v_2 of $M'(\gamma_2)$. Note that the cost of the new edge (v_2, v_1) in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.1 (cf. Case 2.3.1 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in the string list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the root v_1 of $M'(\gamma_1)$ and to the root v_3 of $M'(\gamma_3)$ (so that v_1 and v_3 become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the root v_2 of $M'(\gamma_2)$ and to the root v_4 of $M'(\gamma_4)$ (so that v_2 and v_4 become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges (u_1, v_1), (u_2, v_2), (u_1, v_3), and (u_2, v_4) in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.2 (cf. Case 2.3.2 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in the string list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the left root $v_{1,1}$ of $M'(\gamma_1)$ and to the root v_2 of $M'(\gamma_2)$ (so that $v_{1,1}$ and v_2 become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the right root $v_{1,2}$ of $M'(\gamma_1)$ and to the root v_3 of $M'(\gamma_3)$ (so that $v_{1,2}$ and v_3 become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges $(u_1, v_{1,1})$, $(u_2, v_{1,2})$, (u_1, v_2), and (u_2, v_3) in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.3 (cf. Case 2.3.3 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in the string list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the root v_1 of $M'(\gamma_1)$ and to the right root $v_{2,2}$ of $M'(\gamma_2)$ (so that v_1 and $v_{2,2}$ become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the left root $v_{2,1}$ of $M'(\gamma_2)$ and to the root v_3 of $M'(\gamma_3)$ (so that $v_{2,1}$ and v_3 become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges (u_1, v_1), $(u_2, v_{2,1})$, $(u_1, v_{2,2})$, and (u_2, v_3) in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.4 (cf. Case 2.3.4 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in the strings list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the root v_1 of $M'(\gamma_1)$ and to the left root $v_{3,1}$ of $M'(\gamma_3)$ (so that v_1 and $v_{3,1}$ become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the root v_2 of $M'(\gamma_2)$ and to the right root $v_{3,2}$ of $M'(\gamma_3)$ (so that v_2 and $v_{3,2}$ become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges (u_1, v_1), (u_2, v_2), $(u_1, v_{3,1})$, and $(u_2, v_{3,2})$ in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.5 (cf. Case 2.3.5 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in the string list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the left root $v_{1,1}$ and the right root $v_{1,3}$ of $M'(\gamma_1)$ (so that $v_{1,1}$ and $v_{1,3}$ become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the middle root $v_{1,2}$ of $M'(\gamma_1)$ and to the root v_2 of $M'(\gamma_2)$ (so that $v_{1,2}$ and v_2 become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges $(u_1, v_{1,1})$, $(u_2, v_{1,2})$, $(u_1, v_{1,3})$, and (u_2, v_2) in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.6 (cf. Case 2.3.6 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in string list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the left root $v_{1,1}$ of $M'(\gamma_1)$ and to the left root $v_{2,1}$ of $M'(\gamma_2)$ (so that $v_{1,1}$ and $v_{2,1}$ become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the right root $v_{1,2}$ of $M'(\gamma_1)$ and to the right root $v_{2,2}$ of $M'(\gamma_2)$ (so that $v_{1,2}$ and $v_{2,2}$ become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges $(u_1, v_{1,1})$, $(u_2, v_{1,2})$, $(u_1, v_{2,1})$, and $(u_2, v_{2,2})$ in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.7 (cf. Case 2.3.7 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in the string list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the root v_1 of $M'(\gamma_1)$ and to the middle root $v_{2,2}$ of $M'(\gamma_2)$ (so that v_1 and $v_{2,2}$ become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the left root $v_{2,1}$ and the right root $v_{2,3}$ of $M'(\gamma_2)$ (so that $v_{2,1}$ and $v_{2,3}$ become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges (u_1, v_1), $(u_2, v_{2,1})$, $(u_1, v_{2,2})$, and $(u_2, v_{2,3})$ in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 2.3.8 (cf. Case 2.3.8 in Section 3.2): We create two root vertices u_1 and u_2 for $M'(\beta)$, assign u_1 the first string in the string list $\mathcal{S}$ of β, assign u_2 the second string in $\mathcal{S}$, connect u_1 to the leftmost root $v_{1,1}$ and the third leftmost root $v_{1,3}$ of $M'(\gamma_1)$ (so that $v_{1,1}$ and $v_{1,3}$ become the left and the right child of u_1 in $M'(\beta)$, respectively), and connect u_2 to the second leftmost root $v_{1,2}$ and the rightmost root $v_{1,4}$ of $M'(\gamma_1)$ (so that $v_{1,2}$ and $v_{1,4}$ become the left and the right child of u_2 in $M'(\beta)$, respectively). Note that the total cost of the new edges $(u_1, v_{1,1})$, $(u_2, v_{1,2})$, $(u_1, v_{1,3})$, and $(u_2, v_{1,4})$ in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 3.1 (cf. Case 3.1 in Section 3.2): We connect $M'(\gamma_1)$ and $M'(\gamma_2)$ by making the right root

$v_{1,2}$ of $M'(\gamma_1)$ be the unique child of the leftmost leaf $v_{2,1}$ of $M'(\gamma_2)$. Note that the cost of the new edge $(v_{2,1}, v_{1,2})$ in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Type 3.2 (cf. Case 3.2 in Section 3.2): We connect $M'(\gamma_1)$ and $M'(\gamma_2)$ by making the leftmost root $v_{2,1}$ of $M'(\gamma_2)$ be the unique child of the rightmost leaf $v_{1,2}$ of $M'(\gamma_1)$. Note that the cost of the new edge $(v_{1,2}, v_{2,1})$ in $M'(\beta)$ is exactly the total weight of the edges between β and its children in $\mathcal{D}(M)$.

Lemma 3.8 *The cost of the new mode M' constructed from $\mathcal{D}(M)$ as above equals the total weight of edges in $\mathcal{D}(M)$. Consequently, $c(M') \leq 3 \cdot c(M)$.*

Proof. We claim that for each node β with a label $(\mathcal{S}, i, j)$ in $\mathcal{D}(M)$, the rooted ordered forest $M'(\beta)$ constructed as above satisfies that $c(M'(\beta))$ equals the total weight of edges in $\mathcal{D}_M(\mathcal{S}, i, j)$ and the leaves of $M'(\beta)$ are s_i, s_{i+1}, …, s_j (from left to right). The claim can be easily shown by induction on $j - i$ and so its proof is omitted. The claim implies the first assertion in the lemma immediately. The second assertion follows from the first and Lemma 3.7. □

3.4 Computing an Optimal Component Tree

An obvious but very crucial property in the construction of M' from $\mathcal{D}(M)$ given in Section 3.3 is that the construction of M' only depends on the label and the type of each node in $\mathcal{D}(M)$. This motivates us to define component trees (independently of duplication models) for $\langle s_1, \ldots, s_n \rangle$ and to use dynamic programming to compute a minimum-weight component tree.

In order to define component trees (independently of duplication models) for $\langle s_1, \ldots, s_n \rangle$, we need to define several other terms first. An *abstract quadruple* is a quadruple $(\mathcal{S}, i, j, t)$, where $\mathcal{S}$ is an ordered nonempty list of at most four strings among s_1, …, s_n, i and j are two integers with $1 \leq i \leq j \leq n$, and t satisfies the following conditions:

- $t \in \{0, 1.1, 1.2, 2.1, 2.2, 2.3.1, \ldots, 2.3.8, 3.1, 3.2\}$.
- If $t = 0$, then $i = j$ and $\mathcal{S} = \{s_i\}$.
- If $t \in \{1.1, 1.2\}$, then $|\mathcal{S}| = 1$ and $j - i \geq 1$.
- If $t \in \{2.1, 2.2, 2.3.1, \ldots, 2.3.8\}$, then $|\mathcal{S}| = 2$ and $j - i \geq 3$.
- If $t \in \{3.1, 3.2\}$, then $3 \leq |\mathcal{S}| \leq 4$ and $j - i \geq 2|\mathcal{S}| - 1$.

We call t the *type* of the abstract quadruple $(\mathcal{S}, i, j, t)$. For convenience, if $t \in \{0, 1.1, 1.2\}$, we say that $(\mathcal{S}, i, j, t)$ is of type 1; if $t \in \{2.1, 2.2, 2.3.1, \ldots, 2.3.8\}$, we say that $(\mathcal{S}, i, j, t)$ is of type 2; if $t \in \{3.1, 3.2\}$, we say that $(\mathcal{S}, i, j, t)$ is of type 3.

We define component trees for abstract quadruples $(\mathcal{S}, i, j, t)$ by induction on $j - i$. In the base case where $j - i = 0$, a *component tree* for $(\mathcal{S}, i, j, t)$ is a rooted tree with only one node (the root) which is labeled with $(\mathcal{S}, i, j)$ and is of type t.

Consider the case where $j - i \geq 1$. Suppose that $\mathcal{S} = \langle s_{h_1}, \ldots, s_{h_g} \rangle$. A *component tree* for $(\mathcal{S}, i, j, t)$ is a rooted ordered tree $\mathcal{D}$ such that the root β is labeled with $(\mathcal{S}, i, j)$ and is of type t, and the following conditions are satisfied:

- If $t = 1.1$, then β has only one child in $\mathcal{D}$ and the subtree rooted at the child in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\mathcal{S}', i, j, t')$.

- If $t = 1.2$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_1, i, k, t_1)$ with $k < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_2, k+1, j, t_2)$.

- If $t = 2.1$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\langle s_{h_1} \rangle, i, k, t_1)$ with $k < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\langle s_x, s_{h_2} \rangle, k, j, t_2)$ with $1 \le x \le n$.

- If $t = 2.2$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\langle s_{h_1}, s_x \rangle, i, \ell, t_1)$ with $1 \le x \le n$ and $\ell < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\langle s_{h_2} \rangle, \ell, j, t_2)$.

- If $t = 2.3.1$, then β has four children $\gamma_1, \ldots, \gamma_4$ in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_1, i, k_1, t_1)$ with $k_1 < j$, the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_2, k_1+1, k_2, t_2)$ with $k_2 < j$, the subtree rooted at γ_3 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_3, k_2+1, k_3, t_3)$ with $k_3 < j$, and the subtree rooted at γ_4 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_4, k_3+1, j, t_4)$.

- If $t = 2.3.2$, then β has three children $\gamma_1, \ldots, \gamma_3$ in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\mathcal{S}_1, i, k, t_1)$ with $k < j$, the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_2, k+1, \ell, t_2)$ with $\ell < j$, and the subtree rooted at γ_3 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_3, \ell+1, j, t_3)$.

- If $t = 2.3.3$, then β has three children $\gamma_1, \ldots, \gamma_3$ in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_1, i, k, t_1)$ with $k < j$, the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\mathcal{S}_2, k+1, \ell, t_2)$ with $\ell < j$, and the subtree rooted at γ_3 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_3, \ell+1, j, t_3)$.

- If $t = 2.3.4$, then β has three children $\gamma_1, \ldots, \gamma_3$ in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_1, i, k, t_1)$ with $k < j$, the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_2, k+1, \ell, t_2)$ with $\ell < j$, and the subtree rooted at γ_3 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\mathcal{S}_3, \ell+1, j, t_3)$.

- If $t = 2.3.5$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-3 abstract quadruple $(\mathcal{S}_1, i, k, t_1)$ with $|\mathcal{S}_1| = 3$ and $k < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_2, k+1, j, t_2)$.

- If $t = 2.3.6$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\mathcal{S}_1, i, k, t_1)$ with $k < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\mathcal{S}_2, k+1, j, t_2)$.

- If $t = 2.3.7$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-1 abstract quadruple $(\mathcal{S}_1, i, k, t_1)$ with $k < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-3 abstract quadruple $(\mathcal{S}_2, k+1, j, t_2)$ with $|\mathcal{S}_2| = 3$.

- If $t = 2.3.8$, then β has only one child in $\mathcal{D}$ and the subtree rooted at the child in $\mathcal{D}$ is a component tree for some type-3 abstract quadruple $(\mathcal{S}', i, j, t')$ with $|\mathcal{S}'| = 4$.

- If $t = 3.1$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\langle s_{h_1}, s_x\rangle, i, k, t_1)$ with $1 \le x \le n$ and $k < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-$(g-1)$ abstract quadruple $(\langle s_{h_2}, \ldots, s_{h_g}\rangle, k, j, t_2)$.

- If $t = 3.2$, then β has two children γ_1 and γ_2 in $\mathcal{D}$, the subtree rooted at γ_1 in $\mathcal{D}$ is a component tree for some type-2 abstract quadruple $(\langle s_{h_1}, s_{h_2}\rangle, i, k, t_1)$ with $k < j$, and the subtree rooted at γ_2 in $\mathcal{D}$ is a component tree for some type-$(g-1)$ abstract quadruple $(\langle s_x, s_{h_3}, \ldots, s_{h_g}\rangle, k, j, t_2)$ with $1 \le x \le n$.

Let $(\mathcal{S}, i, j, t)$ be an abstract quadruple, and let $\mathcal{D}$ be a component tree for $(\mathcal{S}, i, j, t)$. Note that each node in $\mathcal{D}$ is labeled with a triple (L, k, ℓ), where L is an ordered nonempty list of strings among $s_1, \ldots, s_n$ and k and ℓ are integers with $1 \le k \le \ell \le n$. Moreover, each node in $\mathcal{D}$ has a type among 0, 1.1, 1.2, 2.1, 2.2, 2.3.1, ..., 2.3.8, 3.1, 3.2. Thus, we can use $\mathcal{D}$ to construct a rooted ordered forest $M'_{\mathcal{D}}$ with $|\mathcal{S}|$ roots as described in Section 3.3. We define the *weight* of $\mathcal{D}$ to be $\sum_e c(e)$, where e ranges over all edges in $M'_{\mathcal{D}}$ and $c(e)$ is the hamming distance between the two strings assigned to the endpoints of e in $M'_{\mathcal{D}}$.

We are now ready to use dynamic programming to compute a minimum-weight component tree for each abstract quadruple $(\mathcal{S}, i, j, t)$. For simplicity, we only explicitly give formulas for computing the minimum weight $W(\mathcal{S}, i, j, t)$ of a component tree for each abstract quadruple $(\mathcal{S}, i, j, t)$ as follows.

- For each abstract quadruple $q = (\mathcal{S}, i, j, t)$ with $t = 0$, $W(q) = 0$.

- For each abstract quadruple $q = (\langle s_h\rangle, i, j, t)$ with $t = 1.1$,

$$W(q) = \min_{q'} \; W(q') + d(s_h, s_x) + d(s_h, s_y),$$

  where q' ranges over all type-2 abstract quadruples $(\langle s_x, s_y\rangle, i, j, t')$.

- For each abstract quadruple $q = (\langle s_h\rangle, i, j, t)$ with $t = 1.2$,

$$W(q) = \min_{i \le k < j} \; \min_{q_1} \; \min_{q_2} \; W(q_1) + W(q_2) + d(s_h, s_x) + d(s_h, s_y),$$

  where q_1 ranges over all type-1 abstract quadruples $(\langle s_x\rangle, i, k, t_1)$ and q_2 ranges over all type-1 abstract quadruples $(\langle s_y\rangle, k+1, j, t_2)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2}\rangle, i, j, t)$ with $t = 2.1$,

$$W(q) = \min_{i<k<j} \min_{q_1} \min_{q_2} W(q_1) + W(q_2) + d(s_k, s_x),$$

  where q_1 ranges over all type-1 abstract quadruples $(\langle s_{h_1}\rangle, i, k, t_1)$ and q_2 ranges over all type-2 abstract quadruples $(\langle s_x, s_{h_2}\rangle, k, j, t_2)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2}\rangle, i, j, t)$ with $t = 2.2$,

$$W(q) = \min_{i<\ell<j} \min_{q_1} \min_{q_2} W(q_1) + W(q_2) + d(s_\ell, s_x),$$

  where q_1 ranges over all type-2 abstract quadruples $(\langle s_{h_1}, s_x\rangle, i, \ell, t_1)$ and q_2 ranges over all type-1 abstract quadruples $(\langle s_{h_2}\rangle, \ell, j, t_2)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2}\rangle, i, j, t)$ with $t = 2.3.1$,

$$\begin{aligned} W(q) &= \min_{i\le k_1<j} \min_{k_1<k_2<j} \min_{k_2<k_3<j} \min_{q_1} \min_{q_2} \min_{q_3} \min_{q_4} W(q_1) + W(q_2) + W(q_3) + W(q_4) \\ &+ d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}), \end{aligned}$$

  where q_1 ranges over all type-1 abstract quadruples $(\langle s_{x_1}\rangle, i, k_1, t_1)$, q_2 ranges over all type-1 abstract quadruples $(\langle s_{x_2}\rangle, k_1 + 1, k_2, t_2)$, q_3 ranges over all type-1 abstract quadruples $(\langle s_{x_3}\rangle, k_2 + 1, k_3, t_3)$, and q_4 ranges over all type-1 abstract quadruples $(\langle s_{x_4}\rangle, k_3 + 1, k_4, t_4)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2}\rangle, i, j, t)$ with $t = 2.3.2$,

$$\begin{aligned} W(q) &= \min_{i+3\le k_1<j} \min_{k_1<k_2<j} \min_{q_1} \min_{q_2} \min_{q_3} W(q_1) + W(q_2) + W(q_3) \\ &+ d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}), \end{aligned}$$

  where q_1 ranges over all type-2 abstract quadruples $(\langle s_{x_1}, s_{x_2}\rangle, i, k_1, t_1)$, q_2 ranges over all type-1 abstract quadruples $(\langle s_{x_3}\rangle, k_1 + 1, k_2, t_2)$, and q_3 ranges over all type-1 abstract quadruples $(\langle s_{x_4}\rangle, k_2 + 1, j, t_3)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2}\rangle, i, j, t)$ with $t = 2.3.3$,

$$\begin{aligned} W(q) &= \min_{i\le k_1<j} \min_{k_1+4\le k_2<j} \min_{q_1} \min_{q_2} \min_{q_3} W(q_1) + W(q_2) + W(q_3) \\ &+ d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}), \end{aligned}$$

  where q_1 ranges over all type-1 abstract quadruples $(\langle s_{x_1}\rangle, i, k_1, t_1)$, q_2 ranges over all type-2 abstract quadruples $(\langle s_{x_2}, s_{x_3}\rangle, k_1+1, k_2, t_2)$, and q_3 ranges over all type-1 abstract quadruples $(\langle s_{x_4}\rangle, k_2 + 1, j, t_3)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2}\rangle, i, j, t)$ with $t = 2.3.4$,

$$\begin{aligned} W(q) &= \min_{i\le k_1<j} \min_{k_1<k_2<j} \min_{q_1} \min_{q_2} \min_{q_3} W(q_1) + W(q_2) + W(q_3) \\ &+ d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}), \end{aligned}$$

  where q_1 ranges over all type-1 abstract quadruples $(\langle s_{x_1}\rangle, i, k_1, t_1)$, q_2 ranges over all type-1 abstract quadruples $(\langle s_{x_2}\rangle, k_1 + 1, k_2, t_2)$, and q_3 ranges over all type-2 abstract quadruples $(\langle s_{x_3}, s_{x_4}\rangle, k_2 + 1, j, t_3)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2} \rangle, i, j, t)$ with $t = 2.3.5$,

$$\begin{aligned} W(q) &= \min_{i+5 \leq k < j} \min_{q_1} \min_{q_2} W(q_1) + W(q_2) \\ &+ d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}), \end{aligned}$$

  where q_1 ranges over all type-3 abstract quadruples $(\langle s_{x_1}, s_{x_2}, s_{x_3} \rangle, i, k, t_1)$ and q_2 ranges over all type-1 abstract quadruples $(\langle s_{x_4} \rangle, k+1, j, t_2)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2} \rangle, i, j, t)$ with $t = 2.3.6$,

$$\begin{aligned} W(q) &= \min_{i+3 \leq k < j} \min_{q_1} \min_{q_2} W(q_1) + W(q_2) \\ &+ d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}), \end{aligned}$$

  where q_1 ranges over all type-2 abstract quadruples $(\langle s_{x_1}, s_{x_2} \rangle, i, k, t_1)$ and q_2 ranges over all type-2 abstract quadruples $(\langle s_{x_3}, s_{x_4} \rangle, k+1, j, t_2)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2} \rangle, i, j, t)$ with $t = 2.3.7$,

$$\begin{aligned} W(q) &= \min_{i \leq k < j} \min_{q_1} \min_{q_2} W(q_1) + W(q_2) \\ &+ d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}), \end{aligned}$$

  where q_1 ranges over all type-1 abstract quadruples $(\langle s_{x_1} \rangle, i, k, t_1)$ and q_2 ranges over all type-3 abstract quadruples $(\langle s_{x_2}, s_{x_3}, s_{x_4} \rangle, k+1, j, t_2)$.

- For each abstract quadruple $q = (\langle s_{h_1}, s_{h_2} \rangle, i, j, t)$ with $t = 2.3.8$,

$$W(q) = \min_{q'} W(q') + d(s_{h_1}, s_{x_1}) + d(s_{h_1}, s_{x_3}) + d(s_{h_2}, s_{x_2}) + d(s_{h_2}, s_{x_4}),$$

  where q' ranges over all type-3 abstract quadruples $(\langle s_{x_1}, \ldots, s_{x_4} \rangle, i, j, t')$.

- For each abstract quadruple $q = (\langle s_{h_1}, \ldots, s_{h_g} \rangle, i, j, t)$ with $t = 3.1$,

$$W(q) = \min_{i+3 \leq k < j} \min_{q_1} \min_{q_2} W(q_1) + W(q_2) + d(s_k, s_x),$$

  where q_1 ranges over all type-2 abstract quadruples $(\langle s_{h_1}, s_x \rangle, i, k, t_1)$ and q_2 ranges over all type-$(g-1)$ abstract quadruples $(\langle s_{h_2}, \ldots, s_{h_g} \rangle, k, j, t_2)$.

- For each abstract quadruple $q = (\langle s_{h_1}, \ldots, s_{h_g} \rangle, i, j, t)$ with $t = 3.2$,

$$W(q) = \min_{i+3 \leq k < j} \min_{q_1} \min_{q_2} W(q_1) + W(q_2) + d(s_k, s_x),$$

  where q_1 ranges over all type-2 abstract quadruples $(\langle s_{h_1}, s_{h_2} \rangle, i, k, t_1)$ and q_2 ranges over all type-$(g-1)$ abstract quadruples $(\langle s_x, s_{h_3}, \ldots, s_{h_g} \rangle, k, j, t_2)$.

Obviously, the minimum weight of a component tree for $\langle s_1, \ldots, s_n \rangle$ is $\min_q W(q)$, where q ranges over all type-1 abstract quadruples $(\mathcal{S}, 1, n, t)$. Moreover, a rough estimate gives a running time of $O(n^{11} + mn^2)$. Indeed, we can lower the time complexity to $O(n^8 + mn^2)$ by modifying the above dynamic programming appropriately (although we omit the details here to save space). Thus, we have the following lemma:

Lemma 3.9 *We can compute a minimum-weight component tree for $\langle s_1, \ldots, s_n \rangle$ in polynomial time.*

Combining Lemmas 3.1, 3.8, and 3.9, we have the following theorem:

Theorem 3.10 *There is a polynomial-time approximation algorithm for 2-DHR that achieves a ratio of 6.*

Acknowledgements

Zhi-Zhong Chen is supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 17500012. Lusheng Wang and Zhanyong Wang are fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 120905].

References

[1] G. Benson and L. Dong. Reconstructing the Duplication History of a Tandem Repeat. *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB-99)*, pp. 44-53, 1999.

[2] T. Boby, A. -M. Patch, and S. J. Aves. TRbase: A Database Relating Tandem Repeats to Disease Genes for the Human Genome, *Bioinformatics*, Advance Access published on October 12, 2004.

[3] W. Fitch. Phylogenies Constrained by Cross-over Process as Illustrated by Human Hemoglobins in a Thirteen Cycle, Eleven Amino-Acid Repeat in Human Apolipoprotein A-I. *Genetics*, Vol. 86, pp. 623-644, 1977.

[4] D. Jaitly, P. E. Kearney, G. Lin, and B. Ma. Methods for Reconstructing the History of Tandem Repeats and Their Application to the Human Genome. *J. Comput. Syst. Sci.*, Vol. 65 , pp. 494-507, 2002.

[5] F. Lillo, S. Basile, and R. N. Mantegna. Comparative Genomics Study of Inverted Repeats in Bacteria. *Bioinformatics*, Vol. 18, pp. 971 - 979, 2002.

[6] J. Macas, T. Mszros, and M. Nouzov. PlantSat: A Specialized Database for Plant Satellite Repeats. *Bioinformatics*, Vol. 18, pp. 28 - 35, 2002.

[7] H. H. Otu and K. Sayood. A New Sequence Distance Measure for Phylogenetic Tree Construction. *Bioinformatics*, Vol. 19, pp. 2122 - 2130, 2004.

[8] S. Subramanian, R. K. Mishra, and L. Singh. Genome-Wide Analysis of Bkm Sequences (GATA Repeats): Predominant Association with Sex Chromosomes and Potential Role in Higher Order Chromatin Organization and Function. *Bioinformatics*, Vol. 19, pp. 681-685, 2003.

[9] M. Tang, M. S. Waterman, and S. Yooseph. Zinc Finger Gene Clusters and Tandem Gene Duplication. *Journal of Computational Biology*, Vol. 9, pp. 429-446, 2002.

[10] L. Wang, T. Jiang, and E. L. Lawler. Approximation Algorithms for Tree Alignment with a Given Phylogeny. *Algorithmica*, Vol. 16, pp. 302-315, 1996.

[11] B. Widegren, U. Arnason, and G. Akusjarvi. Characteristics of a Conserved 1,579-bp Highly Repetitive Component in the Killer Whale, Ornicus Orca. *Mol. Biol. Evol.*, Vol. 2, pp. 411-419, 1985.

[12] A. H. Wyman and R. White. A Highly Polymorphic Locus in Human DNA. *Proc. Natl. Acad. Sci.*, Vol. 77, pp. 6745-6758, 1980.

[13] L. Zhang, B. Ma, L. Wang, and Y. Xu. Greedy Method for Inferring Tandem Duplication History. *Bioinformatics*, Vol. 19, pp. 1497-1504, 2003.