Fast Algorithms for Computing Path-Difference Distances

Biing-Feng Wang and Chih-Yu Li

Abstract—Tree comparison metrics are an important tool for the study of phylogenetic trees. Path-difference distances measure the dissimilarity between two phylogenetic trees (on the same set of taxa) by comparing their path-length vectors. Various norms can be applied to this distance. Three important examples are the l_1-, l_2-, and l_∞-norms. The previous best algorithms for computing path-difference distances all have $O(n^2)$ running time. In this paper, we show how to compute the l_1-norm path-difference distance in $O(n \log^2 n)$ time and how to compute the l_2- and l_∞-norm path-difference distances in $O(n \log n)$ time. By extending the presented algorithms, we also show that the l_p-norm path-difference distance can be computed in $O(pn \log^2 n)$ time for any positive integer p. In addition, when the integer p is even, we show that the distance can be computed in $O(p^2 n \log n)$ time as well.

Index Terms—Algorithms, path-difference distances, phylogenetic trees, tree comparison metrics

1 Introduction

Phylogenetic trees are the basic structures for statistical analysis of differences between species [19]. Many phylogenetic reconstruction methods have been proposed [9], [14], [18], [32]. Using different methods or datasets may result in different trees on the same set of species. To further analyze the used methods, the used datasets, or the constructed trees, it is often necessary to assess how similar or dissimilar these trees are to each other. Many tree comparison metrics have been proposed for this purpose, such as the nearest-neighbor interchange (NNI) distance [35], the Robinson-Foulds (RF) distance [31], the quartet distance [15], and the triplet distance [11]. Since the late 1960s, several researchers introduced metrics based on comparing the path-length vectors of trees, where the path-length vector of a tree is a vector whose coordinates are lengths of the paths between every pair of taxa, in some fixed ordering of the taxon pairs. In this context, the dissimilarity between two trees has been defined as the l_2-norm (or euclidean distance) between their path-length vectors [16], as the l_1-norm (or Manhattan distance) between the vectors [36], as the correlation between the vectors [29], or as the l_∞-norm between the vectors [22]. Different names have been used for these kinds of metrics, including the cladistic difference [16], the topological distance [29], the path-difference distance [34], the nodal distance [4], the k-interval cospeciation distance [22], and the path interval distance [10]. In this paper, the term "path-difference distance" is used, since it seems to be more popular.

Path-difference distances arise from the question whether pairs of taxa are usually close together in a set of trees or, conversely, whether they are usually at opposite ends of the trees [28]. A theoretical basis for these distances is the following classical theorem: two unrooted phylogenetic trees are isomorphic if and only if their path-length vectors are the same [38]. One advantage of path-difference distances is that they can be applied to both unweighted and weighted trees. A branch length of a weighted tree may represent a substitution rate, or an estimate of the amount of change (e.g., number of mutations) [8], [21]. According to Kuhner and Yamato [23], length-using metrics can be used to classify trees which are too similar to be distinguished by topology-only metrics such as the RF and the quartet distances.

The l_1-, l_2-, and l_∞-norm path-difference distances have received considerable attention in the literature. These three distances are the focus of this paper. The l_2-norm path-difference distance was introduced by Farris [16], [17]. This distance has been considered as one of the oldest and most popular tree comparison metrics. Thus, its implementation is usually included in software packages for phylogenetic reconstruction and analysis [3], [5], [23], [30], [33]. In addition to estimating the dissimilarity between phylogenetic trees, it has also been used in other applications, such as inferring correlations between genomic and linguistic diversity [25], reconstructing phylogenetic trees [1], and finding median trees [26]. The l_1-norm path-difference distance was introduced by Williams and Clifford [36]. Similar to the l_2-norm path-difference distance, it has also been used in finding median trees [27]. The l_∞-norm path-difference distance was introduced by Huggins et al. [22]. They revealed that the l_∞-norm path-difference distance may be more useful than the classical NNI distance in the study of concomitantly evolving organisms (or genes), such as host and parasite species. Coons and Rusinko [10] further disclosed several interesting features of this distance. Particularly,

• *The authors are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30013, Republic of China. E-mail: {bfwang, cyuli}@cs.nthu.edu.tw.*

Manuscript received 11 Apr. 2017; revised 15 Nov. 2017; accepted 4 Dec. 2017. Date of publication 8 Jan. 2018; date of current version 29 Mar. 2019. (Corresponding author: Biing-Feng Wang.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCBB.2018.2790957

they showed that it provides a lower bound for the NNI distance. Besides, they demonstrated that it can illuminate global congruence between locally incongruent trees, which cannot be uncovered by classical tree metrics, such as the RF distance. In light of these features, Coons and Rusinko suggested that scientists consider it as an alternative or supplement to classical tree metrics.

Steel and Penny [34] noted that it is important to understand the statistical distribution of a metric, since it provides an indication as to whether or not the measured similarity could have come about by chance. They studied the distribution of the l_2-norm path-difference distance and concluded that the distance generates a wide-range of values and thus is particularly attractive when large trees are studied. They also concluded that the distance may be the method of choice when trees are more dissimilar than expected. Recently, Xi et al. [37] studied the distributions of the l_1-, l_2-,, and l_∞-norm path-difference distances. More specifically, they extended the analyses of the l_2-norm distance in [34] to the l_1- and l_∞-norm distances.

To study large trees, it is essential for a metric to be fast to compute. Therefore, there has been considerable effort devoted to developing efficient algorithms for computing the distances between two phylogenetic trees under a variety of metrics. For example, efficient algorithms have been proposed for the RF distance [12], the quartet distance [6], [7], and the triplet distance [6]. All previous algorithms for computing path-difference distances are based on calculation of the distances between all pairs of leaves in time $O(n^2)$ [5], [34], [37]. In this paper, more efficient algorithms are presented for computing the l_1-,, l_2-,, and l_∞-norm path-difference distances. The algorithm presented for the l_1-norm path-difference distance requires $O(n \log^2 n)$ time and the algorithms presented for the l_2- and l_∞-norm path-difference distances both require $O(n \log n)$ time. It is natural to consider the l_p-norm path-difference distance for any $p \geq 1$ [3], [23], [26], [27], [37]. In this paper, by extending the presented algorithms, we also show that the l_p-norm path-difference distance can be computed in $O(pn \log^2 n)$ time for any positive integer p. In addition, when the integer p is even, we show that the distance can be computed in $O(p^2 n \log n)$ time as well.

The remainder of this paper is organized as follows. Section 2 introduces notation and definitions. Section 3 gives an $O(n \log^2 n)$-time algorithm for computing the l_1-norm path-difference distance. Section 4 shows that the l_p-norm path-difference distance can be computed in $O(pn \log^2 n)$-time for any positive integer p. Section 5 gives an $O(n \log n)$-time algorithm for computing the l_2-norm path-difference distance. Section 6 shows that the l_p-norm path-difference distance can also be computed in $O(p^2 n \log n)$ time when p is a positive even integer. Section 7 gives an $O(n \log n)$-time algorithm for computing the l_∞-norm path-difference distance. The Hamming norm is sometimes referred to as the l_0-norm, which counts the number of nonzero elements of a vector. Theoretically, it would be interesting to consider the path-difference distance problem under the l_0-norm. Section 8 shows how to modify our algorithm for $p = 1$ to compute the l_0-norm path-difference distance. Finally, Section 9 concludes this paper.

2 Notation and Definitions

A *phylogenetic tree* is a tree in which the leaves are uniquely labeled by a set of taxa, and each edge has a positive weight. For convenience, a leaf of a phylogenetic tree is simply identified with its label. A phylogenetic tree can be rooted or unrooted. In this paper, all phylogenetic trees are treated as unrooted. The *distance* between two vertices u, v in a phylogenetic tree T, denoted by $d(T, u, v)$, is the total weight of the unique path between u and v.

Let Σ be a set of n taxa and let T_1 and T_2 be two phylogenetic trees on Σ. The *distance difference* of a pair of leaves i and j, where $i \neq j$ and $i, j \in \Sigma$, on T_1 and T_2 is $|d(T_1, i, j) - d(T_2, i, j)|$. For any integer $p \geq 1$, *the l_p-norm path-difference distance* between T_1 and T_2, denoted by $\mathbb{d}_p(T_1, T_2)$, is obtained by endowing the l_p-norm to the distance differences of all leaf pairs. More specifically,

$$\mathbb{d}_p(T_1, T_2) = \left(\sum_{i \neq j \text{ and } i,j \in \Sigma} |d(T_1, i, j) - d(T_2, i, j)|^p \right)^{1/p}.$$

Given two phylogenetic trees T_1 and T_2 on the same set of taxa, the *l_p-norm path-difference distance problem* is to compute the distance $\mathbb{d}_p(T_1, T_2)$. In this paper, we assume that taking the pth root of a number can be done in $O(1)$ time. Accordingly, the problem is just to compute $\sum_{i \neq j \text{ and } i,j \in \Sigma} |d(T_1, i, j) - d(T_2, i, j)|^p$. We assume that all internal nodes of a phylogenetic tree have degree at least three, so that the the numbers of internal nodes of T_1 and T_2 are both bounded by $n - 1$. (A degree 2 node can be removed by simply merging its two adjacent edges.) We also assume that the two given trees T_1 and T_2 have been preprocessed so that on each of them the distance between any two vertices can be answered in $O(1)$ time. The preprocessing requires $O(n)$ time [13].

In the following, we introduce some notation and definitions that are used throughout this paper. For a sequence X of numbers, $|X|$ denotes its length, $X[i]$, $1 \leq i \leq |X|$, denotes the ith element of X, and $X + w$ denotes the sequence obtained from X by increasing each element by a number w. For a set F, $[F]^2$ denotes the set of unordered pairs $\{i, j\}$, where $i \neq j$ and $i, j \in F$, and a pair $\{i, j\}$ is called an F-F pair if $\{i, j\} \in [F]^2$. For two disjoint sets F and G, $[F \times G]$ denotes the set of unordered pairs $\{i, j\}$, where $i \in F$ and $j \in G$, and a pair $\{i, j\}$ is called an F-G pair if $\{i, j\} \in [F \times G]$. For example, if $F = \{w, x, y, z\}$ and $G = \{a, b\}$, then $\{w, x\}$, $\{w, y\}$, $\{y, z\}$ are F-F pairs and $\{a, w\}$, $\{a, y\}$, $\{b, z\}$ are $F - G$ pairs.

Let T be a phylogenetic tree. The vertex set and leaf set of T are denoted by $V(T)$ and $L(T)$, respectively. For any two vertices u, v of T, let $P(T, u, v)$ be the unique path from u to v. For any subset S of $L(T)$, the *subtree of T induced by S*, denoted by $T|_S$, is the tree whose vertices are either leaves in S, or the meets of triples of these leaves, where the *meet* of three leaves i, j, k is the vertex shared by the simple paths from i to j, j to k, and k to i. The edges of $T|_S$ are obtained as follows: for any two vertices u, v, if $T|_S$ does not contain any internal vertex of $P(T, u, v)$, then add an edge of length $d(T, u, v)$ between u and v. An example of induced subtrees is given in Fig. 1. Note that the distance between any two vertices in $T|_S$ is the same as in T.

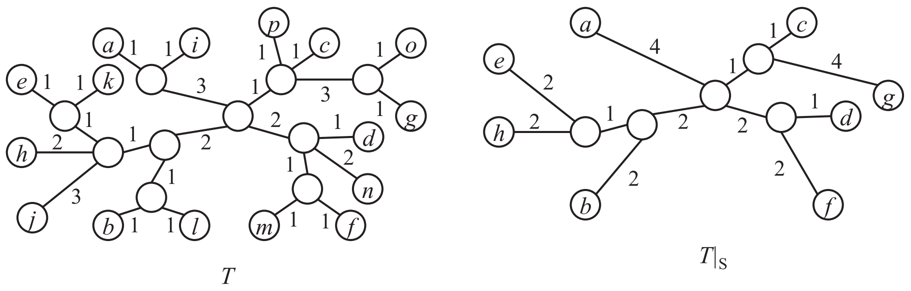


Fig. 1. A tree T and its induced subtree $T|_S$, where $S = \{a, b, c, d, e, f, g, h\}$.

A *decomposition* of a tree T by using a vertex v is a pair of two subtrees (X, Y) such that $V(X) \cap V(Y) = \{v\}$ and T is the union of X and Y. For example, in Fig. 2, $(T_{1,1}, T_{1,2})$ is an decomposition of T_1, in which $V(T_{1,1}) \cap V(T_{1,2}) = \{v_1\}$. Frederickson and Johnson [20] showed that given a tree T of n vertices, in $O(n)$ time we can find a vertex v and use it to decompose T in to two subtrees, each has at most $2n/3$ vertices. This result can be easily modified to obtain the following.

Lemma 2.1. [20]. *Given a phylogenetic tree T of n leaves, in $O(n)$ time we can find a vertex v and use it to decompose T into two subtrees, each has at most $2n/3$ leaves.*

3 An Algorithm for the L_1-Norm Path-Difference Distance Problem

In this section, an $O(n \log^2 n)$-time algorithm is presented for the l_1-norm path-difference distance problem. The previous best upper bound for this problem is $O(n^2)$ [34], [37]. By definition, the distance $\mathbb{d}_1(T_1, T_2)$ is the total distance difference of all leaf pairs. Since there are $O(n^2)$ leaf pairs, if the computation of all distance differences is necessary, the $O(n^2)$ upper bound is optimal. Our intent is to avoid the computation by utilizing the topological structure of a tree.

Our algorithm is designed based upon the divide-and-conquer strategy. It takes a pair of trees T_1 and T_2 on the same set Σ of n taxa as input. If $n < 4$, $\mathbb{d}_1(T_1, T_2)$ is computed in $O(1)$ time by a naive algorithm; otherwise, $\mathbb{d}_1(T_1, T_2)$ is determined by using the divide-and-conquer strategy as follows. By Lemma 2.1, we find a vertex v_1 and use it to decompose T_1 into two subtrees $T_{1,1}$ and $T_{1,2}$. We say that a leaf pair $\{i, j\}$ is *split* by v_1 if i and j are not contained in the same subtree after the decomposition. For example, in Fig. 2, $\{a, d\}$ is split by v_1, but $\{a, b\}$ is not. Also, we find a vertex v_2 and use it to decompose T_2 into two subtrees $T_{2,1}$ and $T_{2,2}$. Similarly, we say that a leaf pair $\{i, j\}$ is *split* by v_2 if i and j are not both contained in any of $T_{2,1}$ and $T_{2,2}$. Note that each of $T_{1,1}$, $T_{1,2}$, $T_{2,1}$, and $T_{2,2}$ has at most $2n/3$ leaves. Depending on whether or not a leaf pair is split by v_1 and by v_2, we partition the set of all leaf pairs into the following subsets:

0-Split $= \{\{i, j\} \mid \{i, j\} \in [\Sigma]^2$, $\{i, j\}$ is neither split by v_1 nor split by $v_2\}$;
1-Split $= \{\{i, j\} \mid \{i, j\} \in [\Sigma]^2$, $\{i, j\}$ is split either by v_1 or by v_2, but not both$\}$; and
2-Split $= \{\{i, j\} \mid \{i, j\} \in [\Sigma]^2$, $\{i, j\}$ is split by both v_1 and $v_2\}$.

For any set $Q \subseteq [\Sigma]^2$, let *Sum*(Q) be the total distance difference of the leaf pairs in Q. Accordingly, our problem is divided into the following three sub-problems: compute *Sum*(0-Split), *Sum*(1-Split), and *Sum*(2-Split).

For any two trees X and Y, let *Shared*(X, Y) denote the set of leaves that are present in both X and Y. For ease of discussion, let $A =$ *Shared*$(T_{1,1}, T_{2,1})$, $B =$ *Shared*$(T_{1,1}, T_{2,2})$, $C =$ *Shared*$(T_{1,2}, T_{2,1})$, and $D =$ *Shared*$(T_{1,2}, T_{2,2})$. For example, in Fig. 2, we have $A = \{b, h, i, j, k\}$, $B = \{a, e, l, q, r\}$, $C = \{m, d\}$, and $D = \{c, f, g, n, o, p\}$. The sets A, B, C, and D, form a partition of Σ. By definition, the set 0-Split contains the leaf pairs that are neither split by v_1 nor split by v_2. Therefore, each pair in 0-Split is an A-A pair, a B-B pair, a C-C pair, or a D-D pair. More specifically, 0-Split is the union of $[A]^2$, $[B]^2$, $[C]^2$, and $[D]^2$. Similarly, by definition, 2-Split is the union of $[A \times D]$ and $[B \times C]$, and 1-Split is the union of $[A \times B]$, $[C \times D]$, $[A \times C]$, and $[B \times D]$.

Since 0-Split is the union of $[A]^2$, $[B]^2$, $[C]^2$, and $[D]^2$, we have *Sum*(0-Split) $=$ *Sum*$([A]^2)$ $+$ *Sum*$([B]^2)$ $+$ *Sum*$([C]^2)$ $+$ *Sum*$([D]^2)$. For each $S \in \{A, B, C, D\}$, *Sum*$([S]^2)$ is equal to the distance $\mathbb{d}_1(T_{1|S}, T_{2|S})$. Therefore, *Sum*(0-Split) can be determined by making four recursive calls, where the inputs for these recursive calls are subtree pairs $(T_1|_A, T_2|_A)$,

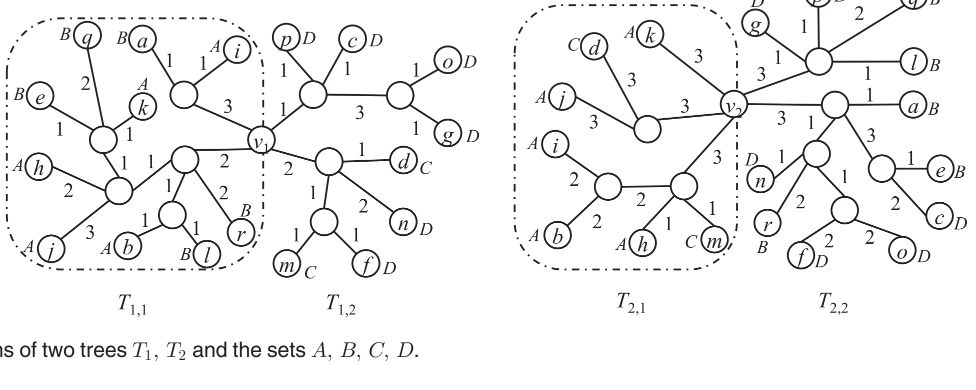


Fig. 2. Decompositions of two trees T_1, T_2 and the sets A, B, C, D.

$(T_1|_B, T_2|_B)$, $(T_1|_C, T_2|_C)$, and $(T_1|_D, T_2|_D)$, respectively. Let $t(n)$ be the worst-case time complexity of our algorithm. Clearly, the construction of subtree pairs $(T_1|_S, T_2|_S)$, where $S \in \{A, B, C, D\}$, can be done in $O(n)$ time. Therefore, the computation of *Sum*(0-SPLIT) requires $t(|A|) + t(|B|) + t(|C|) + t(|D|) + O(n)$ time.

The following two sections show that *Sum*(2-SPLIT) and *Sum*(1-SPLIT) can both be computed in $O(n \log n)$ time.

3.1 Computing *Sum* (2-SPLIT)

The set 2-SPLIT is the union of $[A \times D]$ and $[B \times C]$. We compute $Sum([A \times D])$ and $Sum([B \times C])$ separately. The computations for $[A \times D]$ and $[B \times C]$ are similar. Therefore, only the computation of $Sum([A \times D])$ is described. First, we give some lemmas that are essential to the computation. For any $i \in \Sigma$, $p \in V(T_1)$, and $q \in V(T_2)$, define $\phi(i, p, q)$ as $d(T_1, i, p) - d(T_2, i, q)$. For example, in Fig. 2, $\phi(g, v_1, v_2) = d(T_1, g, v_1) - d(T_2, g, v_2) = 5 - 4 = 1$ and $\phi(i, v_1, v_2) = d(T_1, i, v_1) - d(T_2, i, v_2) = 4 - 7 = -3$. We have the following.

Lemma 3.1. *Let i, j be two leaves and p, q be two vertices such that p is on $P(T_1, i, j)$ and q is on $P(T_2, i, j)$. Then, $d(T_1, i, j) - d(T_2, i, j) = \phi(i, p, q) + \phi(j, p, q)$.*

Proof. Since p is a vertex on $P(T_1, i, j)$, we have $d(T_1, i, j) = d(T_1, i, p) + d(T_1, j, p)$. Similarly, we have $d(T_2, i, j) = d(T_2, i, q) + d(T_2, j, q)$. Therefore, $d(T_1, i, j) - d(T_2, i, j) = d(T_1, i, p) + d(T_1, j, p) - d(T_2, i, q) - d(T_2, j, q) = \phi(i, p, q) + \phi(j, p, q)$. Thus, the lemma holds. □

For each $\{i, j\} \in$ 2-SPLIT, since v_1 is on $P(T_1, i, j)$ and v_2 is on $P(T_2, i, j)$, by Lemma 3.1 we have $d(T_1, i, j) - d(T_2, i, j) = \phi(i, v_1, v_2) + \phi(j, v_1, v_2)$. For example, in Fig. 2, since $\{g, i\} \in$ 2-SPLIT, $\phi(g, v_1, v_2) = 1$, and $\phi(i, v_1, v_2) = -3$, we have $d(T_1, g, i) - d(T_2, g, i) = \phi(g, v_1, v_2) + \phi(i, v_1, v_2) = 1 + (-3) = -2$.

For two sequences X and Y of numbers, we use $X \oplus Y$ to denote $\sum_{1 \le i \le |X|, 1 \le j \le |Y|} |X[i] + Y[j]|$. For example, $(-1, 2, 1) \oplus (2, 3) = |-1 + 2| + |-1 + 3| + |2 + 2| + |2 + 3| + |1 + 2| + |1 + 3|$. For a subset $S \subseteq \Sigma$ and two vertices $p \in V(T_1)$ and $q \in V(T_2)$, let $\Phi(S, p, q)$ denote a sequence consisting of the values $\phi(i, p, q)$, where $i \in S$. Then, we have the following.

Lemma 3.2. *Let S, S' be two disjoint subsets of Σ and p, q be two vertices such that for every $\{i, j\} \in [S \times S']$, vertex p is on $P(T_1, i, j)$ and vertex q is on $P(T_2, i, j)$. Then, $Sum([S \times S']) = \Phi(S, p, q) \oplus \Phi(S', p, q)$.*

Proof. By Lemma 3.1, for each $\{i, j\} \in [S \times S']$, we have $d(T_1, i, j) - d(T_2, i, j) = \phi(i, p, q) + \phi(j, p, q)$. Thus,

$$\begin{aligned} Sum([S \times S']) &= \sum_{\{i,j\} \in [S \times S']} |d(T_1, i, j) - d(T_2, i, j)| \\ &= \sum_{\{i,j\} \in [S \times S']} |\phi(i, p, q) + \phi(j, p, q)| \\ &= \Phi(S, p, q) \oplus \Phi(S', p, q). \end{aligned}$$

Therefore, the lemma holds. □

Note that by letting $(S, S', p, q) = (A, D, v_1, v_2)$ in Lemma 3.2, we have $Sum([A \times D]) = \Phi(A, v_1, v_2) \oplus \Phi(D, v_1, v_2)$.

Lemma 3.3. *Let X and Y be two non-decreasing sequences of numbers. The value of $X \oplus Y$ can be computed in $O(|X| + |Y|)$ time.*

Proof. For each $X[i]$, $1 \le i \le |X|$, let $m(i)$ be the maximum j such that $X[i] + Y[j] < 0$. In other words, $m(i)$ is the number of elements in Y that are smaller than $-X[i]$. Since X and Y are non-decreasing, all $m(i)$ can be computed in $O(|X| + |Y|)$ time by merging X' and Y, where $X' = (-X[|X|], -X[|X| - 1], -X[|X| - 2], \ldots, -X[1])$. For each $Y[j]$, $1 \le j \le |Y|$, let $m'(j)$ be the maximum i such that $X[i] + Y[j] < 0$. Similarly, all $m'(j)$ can be computed in $O(|X| + |Y|)$ time.

Consider the contribution of a fixed $X[i]$ to the value of $X \oplus Y = \sum_{1 \le i \le |X|, 1 \le j \le |Y|} |X[i] + Y[j]|$. For each $j \le m(i)$, since $X[i] + Y[j] < 0$, we have $|X[i] + Y[j]| = -(X[i] + Y[j])$ and thus $X[i]$ contributes $-X[i]$ to the value of $X \oplus Y$; on the contrary, for each $j > m(i)$, we have $|X[i] + Y[j]| = X[i] + Y[j]$ and thus $X[i]$ contributes $X[i]$ to the value of $X \oplus Y$. As a result, we conclude that $X[i]$ contributes a total of

$$-X[i] \times m(i) + X[i] \times (|Y| - m(i)) (= X[i] \times (|Y| - 2m(i)))$$

to the value of $X \oplus Y$. For example, if $X = (-5, -4, -4, 0, 1, 2, 3, 5)$ and $Y = (-5, -4, -2, 0, 0, 2, 3, 4, 4, 5)$, since $X[3] = -4$, $|Y| = 10$, and there are $m(3) = 7$ numbers in Y that are smaller than 4, $X[3]$ contributes a total of $(-4) \times (10 - 2 \times 7) = 16$ to the value of $X \oplus Y$. Similarly, it can be concluded that each $Y[j]$ contributes a total of $Y[j] \times (|X| - 2m'(j))$ to the value of $X \oplus Y$. Therefore, $X \oplus Y$ can be computed as

$$\sum_{1 \le i \le |X|} (X[i] \times (|Y| - 2m(i))) + \sum_{1 \le j \le |Y|} (Y[j] \times (|X| - 2m'(j)))$$

in $O(|X| + |Y|)$ time by using the values of $m(i)$ and $m'(j)$. Consequently, the lemma holds. □

As mentioned, by Lemma 3.2, we have $Sum([A \times D]) = \Phi(A, v_1, v_2) \oplus \Phi(D, v_1, v_2)$. Therefore, $Sum([A \times D])$ can be computed according to Lemma 3.3 as follows. First, we compute X as the sorted sequence of the values in $\Phi(A, v_1, v_2)$ and compute Y as the sorted sequence of the the values in $\Phi(D, v_1, v_2)$. Recall that it has been assumed that the two given trees T_1 and T_2 have been preprocessed so that on each of them the distance between any two vertices can be answered in $O(1)$ time. Thus, $\Phi(A, v_1, v_2)$ and $\Phi(D, v_1, v_2)$ can be found in $O(|A| + |D|) = O(n)$ time. The sequences X and Y are obtained, respectively, by sorting the values in $\Phi(A, v_1, v_2)$ and $\Phi(D, v_1, v_2)$, which requires $O(|A| \log |A| + |D| \log |D|) = O(n \log n)$ time. Next, we determine $Sum([A \times D])$ by computing the value of $X \oplus Y$, which requires $O(|A| + |D|) = O(n)$ time. Consequently, we obtain the following.

Lemma 3.4. *Sum(2-SPLIT) can be found in $O(n \log n)$ time.*

Example 3.1. Consider the example in Fig. 2, in which $\Phi(A, v_1, v_2) = (\phi(b, v_1, v_2), \phi(h, v_1, v_2), \phi(i, v_1, v_2), \phi(j, v_1, v_2), \phi(k, v_1, v_2)) = (-3, 1, -3, 0, 2)$ and $\Phi(D, v_1, v_2) = (\phi(c, v_1, v_2), \phi(f, v_1, v_2), \phi(g, v_1, v_2), \phi(n, v_1, v_2), \phi(o, v_1, v_2), \phi(p, v_1, v_2)) = (-6, -3, 1, -1, -2, -2)$.

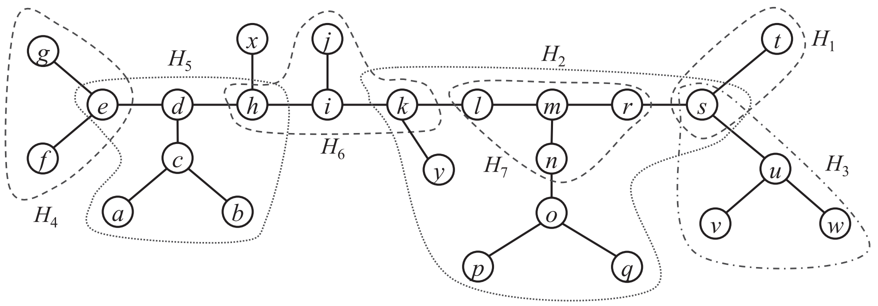


Fig. 3. An illustration of clusters.

First, by sorting, we compute $X = (-3, -3, 0, 1, 2)$ and $Y = (-6, -3, -2, -2, -1, 1)$. Then, we compute $Sum([A \times D]) = X \oplus Y = 97$ according Lemma 3.3.

3.2 Computing *Sum* (1-Split)

The set 1-Split is the union of $[A \times B]$, $[C \times D]$, $[A \times C]$, and $[B \times D]$. We determine $Sum([A \times B])$, $Sum([C \times D])$, $Sum([A \times C])$, and $Sum([B \times D])$ separately. By symmetry, only the computation for $Sum([A \times B])$ is described. Before proceeding, we note that the algorithm in Section 3.1 cannot be applied to compute $Sum([A \times B])$ directly, unless it just so happens that there is a vertex p in T_1 such that vertex p is on $P(T_1, i, j)$ for every $\{i, j\} \in [A \times B]$. Our algorithm for computing $Sum([A \times B])$ is again designed based upon the divide-and-conquer strategy. For convenience, we describe it on a top tree, which was introduced by Alstrup et al. [2] as an interface for designing divide-and-conquer algorithms for tree problems. A top tree of a tree T is defined as follows. For any subtree Z of T, we call a vertex x in Z a *boundary vertex* if it has a neighbor in T outside Z. A subtree Z is a *cluster* of T if it has at most two boundary vertices. Two clusters Z_1 and Z_2 can be *merged* if they share a single vertex and their union is still a cluster. For example, consider the subtrees in Fig. 3. H_1 has one boundary vertex s and H_2 has two boundary vertices k and s. By definition, H_1 and H_2 are clusters. Similarly, H_3, H_4, H_5, and H_6 are clusters. Since H_7 has three boundary vertices l, n, and r, it is not a cluster. We can merge H_1 and H_2, since their union is a subtree having two boundary vertices k and s. Also, we can merge H_1 and H_3, since their union is a subtree having one boundary vertex s. Similarly, we can merge H_2 and H_6, and merge H_4 and H_5. Merging H_5 and H_6 is not allowed, since their union is a subtree having three boundary vertices e, h, and k. From the above examples, it can be observed that if two clusters Z_1 and Z_2 have a common vertex c and can be merged into a cluster Z, then (1) c is a boundary vertex of both Z_1 and Z_2, (2) (Z_1, Z_2) is a decomposition of Z by using c, and (3) the boundary vertices of Z come only from those of Z_1 and Z_2. These properties are used in our algorithm for computing $Sum([A \times B])$.

A *top tree* of T is a rooted binary tree with the following properties.

(1) Each node represents a cluster of T.
(2) The leaves represent the edges of T. (Each edge is regarded as a subtree containing two vertices.)
(3) Each internal node represents the cluster merged from the two clusters represented by its children.
(4) The root represents T.
(5) The height is $O(\log |V(T)|)$.

The cluster represented by a node α of a top tree is denoted by $Z(\alpha)$. Fig. 4 depicts a top tree of $T_1|_{A \cup B}$ for the example in Fig. 2. In a top tree, the cluster represented by an internal node α is just the union of the edges represented by the leaves below α. For example, in Fig. 4, $Z(22)$ is the union of e_4, e_5, e_6, and e_7. Thus, a top tree of T defines a way to repeatedly merge the edges of T into subtrees, until the whole tree T is formed. Alstrup et al. [2] showed that a top tree of T always exists and can be constructed in linear time.

We proceed to describe an algorithm for computing $Sum([A \times B])$. If $|A \cup B| < 3$, we determine $Sum([A \times B])$ naively in $O(1)$ time; otherwise, we compute $Sum([A \times B])$ based on the divide-and-conquer strategy as follows. First, we build a top tree τ of $T_1|_{A \cup B}$. (See Fig. 4.) Note that $L(T_1|_{A \cup B}) = A \cup B$ and the distance between any two vertices in $T_1|_{A \cup B}$ is the same as in T_1. For each node α of τ, let

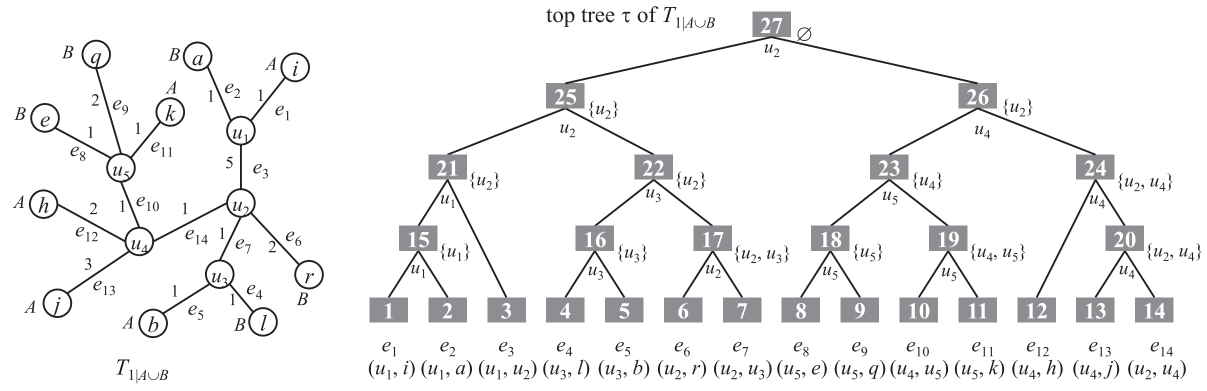


Fig. 4. A top tree τ of $T_1|_{A \cup B}$, in which each leaf α represents edge e_α and the set beside and the node below each internal node α are, respectively, the boundary vertex set of $Z(\alpha)$ and the common vertex of the clusters represented by its children.

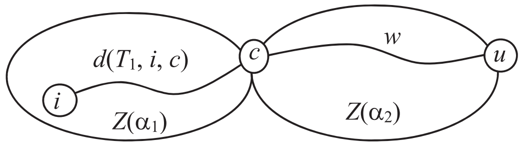


Fig. 5. An illustration of Case 2, in which $u \notin U(\alpha_1)$ and $u \in U(\alpha_2)$.

$U(\alpha)$ denote the set of boundary vertices of $Z(\alpha)$, and $A(\alpha)$ and $B(\alpha)$ denote, respectively, the intersections $L(Z(\alpha)) \cap A$ and $L(Z(\alpha)) \cap B$. Next, in a bottom-up fashion, we compute $Sum([A(\alpha) \times B(\alpha)])$ for all nodes α of τ. Since the root of τ represents the tree $T_1|_{A\cup B}$, the computation for the root produces the desired value, $Sum([A \times B])$. To reduce the time complexity, during the bottom-up computation, some additional information is maintained for each top tree node. Specifically, for each node α of τ, two sequences $X(\alpha, u)$ and $Y(\alpha, u)$ are computed for every $u \in U(\alpha)$, where $X(\alpha, u)$ is the sorted sequence of the values in $\Phi(A(\alpha), u, v_2)$ and $Y(\alpha, u)$ is the sorted sequence of the values in $\Phi(B(\alpha), u, v_2)$. Recall that v_2 is the vertex used to decompose T_2, and note that every pair in $[A \times B]$ is split by v_2.

In summary, for each node α of τ, we compute $Sum([A(\alpha) \times B(\alpha)])$ and compute $X(\alpha, u)$ and $Y(\alpha, u)$ for every $u \in U(\alpha)$. We proceed to describe the details. First, consider the case that α is a leaf. In this case, $Z(\alpha)$ is an edge, say (s, t), of $T_1|_{A\cup B}$. Since $|A \cup B| \geq 3$, at most one of s and t is a leaf node of $T_1|_{A\cup B}$. Thus, $|A(\alpha) \cup B(\alpha)| \leq 1$ and we simply compute $Sum([A(\alpha) \times B(\alpha)])$ as 0. The computation of $X(\alpha, u)$ and $Y(\alpha, u)$ for each $u \in U(\alpha)$ is done as follows.

Case 1. s and t are both internal nodes of $T_1|_{A\cup B}$. In this case, $U(\alpha) = \{s, t\}$ and $A(\alpha) = B(\alpha) = \emptyset$. We compute $X(\alpha, s)$, $Y(\alpha, s)$, $X(\alpha, t)$, and $Y(\alpha, t)$ as empty sequences.

Case 2. One of s and t is an internal node of $T_1|_{A\cup B}$ and the other is in $A \cup B$. By symmetry, assume that s is an internal node of $T_1|_{A\cup B}$. In this case, $U(\alpha) = \{s\}$. If $t \in A$, since $A(\alpha) = \{t\}$ and $B(\alpha) = \emptyset$, we compute $X(\alpha, s) = (\phi(t, s, v_2))$ and compute $Y(\alpha, s)$ as an empty sequence; otherwise, since $A(\alpha) = \emptyset$ and $B(\alpha) = \{t\}$, we compute $X(\alpha, s)$ as an empty sequence and compute $Y(\alpha, s) = (\phi(t, s, v_2))$.

The above computation for a leaf α requires $O(1)$ time. We proceed to describe the computation for an internal node α. Let α_1 and α_2 be the children of α, and let c be the common vertex of $Z(\alpha_1)$ and $Z(\alpha_2)$. Note that by the definition of a top tree, c is a boundary vertex of both $Z(\alpha_1)$ and $Z(\alpha_2)$. (See Fig. 3.) Also note that $Z(\alpha_1)$ and $Z(\alpha_2)$ form a decomposition of $Z(\alpha)$ by using the vertex c. First, we show that $Sum([A(\alpha) \times B(\alpha)])$ can be computed in $O(|L(Z(\alpha))|)$ time. The sets $A(\alpha_1)$ and $A(\alpha_2)$ are a partition of $A(\alpha)$, and the sets $B(\alpha_1)$ and $B(\alpha_2)$ are a partition of $B(\alpha)$. Thus, $Sum([A(\alpha) \times B(\alpha)])$ can be computed as the total of

$$Sum([A(\alpha_1) \times B(\alpha_1)]), Sum([A(\alpha_1) \times B(\alpha_2)]),$$
$$Sum([A(\alpha_2) \times B(\alpha_1)]), \text{ and } Sum([A(\alpha_2) \times B(\alpha_2)]).$$

Since the values of $Sum([A(\alpha_1) \times B(\alpha_1)])$ and $Sum([A(\alpha_2) \times B(\alpha_2)])$ have been determined, respectively, for α_1 and α_2, we only need to compute the values of $Sum([A(\alpha_1) \times B(\alpha_2)])$ and $Sum([A(\alpha_2) \times B(\alpha_1)])$. By symmetry, only the computation for $Sum([A(\alpha_1) \times B(\alpha_2)])$ is described.

Since $[A(\alpha_1) \times B(\alpha_2)] \subseteq [A \times B]$, every pair in $[A(\alpha_1) \times B(\alpha_2)]$ is split by v_2. As mentioned, $Z(\alpha_1)$ and $Z(\alpha_2)$ form a decomposition of $Z(\alpha)$ by using the vertex c. Thus, every pair in $[A(\alpha_1) \times B(\alpha_2)]$ is also split by c. Consequently, $Sum([A(\alpha_1) \times B(\alpha_2)])$ can be determined by using Lemmas 3.2 and 3.3 as follows. By Lemma 3.2, $Sum([A(\alpha_1) \times B(\alpha_2)]) = \Phi(A(\alpha_1), c, v_2) \oplus \Phi(B(\alpha_2), c, v_2)$. Since c is a boundary vertex of both $Z(\alpha_1)$ and $Z(\alpha_2)$, the sequences $X(\alpha_1, c)$ and $Y(\alpha_2, c)$ have been determined during the computation for α_1 and α_2, respectively. Recall that $X(\alpha_1, c)$ is the sorted sequence of the values in $\Phi(A(\alpha_1), c, v_2)$ and $Y(\alpha_2, c)$ is the sorted sequence of the values in $\Phi(B(\alpha_2), c, v_2)$. Therefore, $Sum([A(\alpha_1) \times B(\alpha_2)])$ is computed as $X(\alpha_1, c) \oplus Y(\alpha_2, c)$, which by Lemma 3.3 requires $O(|A(\alpha_1)| + |B(\alpha_2)|) = O(|L(Z(\alpha))|)$ time.

We proceed to describe the computation of the sequences $X(\alpha, u)$ and $Y(\alpha, u)$ for each $u \in U(\alpha)$. Consider a fixed $u \in U(\alpha)$. Only a boundary vertex of $Z(\alpha_1)$ or $Z(\alpha_2)$ can be a boundary vertex of $Z(\alpha)$. (See Fig. 3.) Thus, at least one of $U(\alpha_1)$ and $U(\alpha_2)$ contains u. The following three cases are considered.

Case 1. $u \in U(\alpha_1)$ and $u \in U(\alpha_2)$. In this case, $u = c$. Since $A(\alpha) = A(\alpha_1) \cup A(\alpha_2)$, we obtain $X(\alpha, c)$ in $O(|A(\alpha)|)$ time by merging $X(\alpha_1, c)$ and $X(\alpha_2, c)$. Similarly, we obtain $Y(\alpha, c)$ in $O(|B(\alpha)|)$ time by merging $Y(\alpha_1, c)$ and $Y(\alpha_2, c)$.

Case 2. $u \notin U(\alpha_1)$ and $u \in U(\alpha_2)$. Recall that c is a vertex in $U(\alpha_1)$. In this case, we obtain $X(\alpha, u)$ by using $X(\alpha_1, c)$ and $X(\alpha_2, u)$ as follows. Let $w = d(T_1, c, u)$ and X' be the sequence $X(\alpha_1, c) + w$. For each $i \in A(\alpha_1)$, since c is on the path $P(T_1, i, u)$, we have

$$\begin{aligned} d(T_1, i, u) &= d(T_1, i, c) + d(T_1, c, u) \\ &= d(T_1, i, c) + w. \qquad \text{(See Fig. 5.)} \end{aligned}$$

Therefore, for each $i \in A(\alpha_1)$, we have

$$\begin{aligned} \phi(i, u, v_2) &= d(T_1, i, u) - d(T_2, i, v_2) \\ &= d(T_1, i, c) + w - d(T_2, i, v_2) \\ &= \phi(i, c, v_2) + w. \end{aligned}$$

As a result, $\Phi(A(\alpha_1), u, v_2)$ can be obtained from $\Phi(A(\alpha_1), c, v_2)$ by adding w to each number, from which we conclude that X' is the sorted sequence of the values in $\Phi(A(\alpha_1), u, v_2)$. Consequently, $X(\alpha, u)$ can be obtained in $O(|A(\alpha)|)$ time by merging X' and $X(\alpha_2, u)$. Similarly, $Y(\alpha, u)$ can be obtained in $O(|B(\alpha)|)$ time by using $Y(\alpha_1, c)$ and $Y(\alpha_2, u)$.

Case 3. $u \in U(\alpha_1)$ and $u \notin U(\alpha_2)$. Similar to Case 2.

The above computation of $X(\alpha, u)$ and $Y(\alpha, u)$ for each $u \in U(\alpha)$ requires $O(|A(\alpha)| + |B(\alpha)|) = O(|L(Z(\alpha))|)$ time.

Our computation of $Sum([A \times B])$ is summarized in the following procedure, in which MERGE(L, L') is an auxiliary procedure that returns the merge of its two sorted input lists L and L', and $Sum([A(\alpha) \times B(\alpha)])$ is simply denoted by $Sum(\alpha)$.

```
Procedure COMPUTING_AB_1
begin
  $\tau \leftarrow$ a top tree of $T_1|_{AB}$
  for each leaf $\alpha$ of $\tau$ do begin
    $Sum(\alpha) \leftarrow 0$
    $(s, t) \leftarrow$ the edge $Z(\alpha)$
    if $s$ and $t$ are both internal nodes of $T_1|_{A\cup B}$ then
      $(X(\alpha, s), Y(\alpha, s), X(\alpha, t), Y(\alpha, t)) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset)$
    else begin //either $s$ or $t$ is an internal node
      if $s$ is not an internal node of $T_1|_{A\cup B}$ then
        exchange $s$ and $t$          //set $s$ as the internal node
      if $t \in A$ then $(X(\alpha, s), Y(\alpha, s)) \leftarrow (\phi(t, s, v_2), \varnothing)$
      else $(X(\alpha, s), Y(\alpha, s)) \leftarrow (\emptyset, (\phi(t, s, v_2))$
    end
  end
  for each internal node $\alpha$ of $\tau$, in a bottom-up fashion, do
  begin
    $(\alpha_1, \alpha_2) \leftarrow$ the children of $\alpha$
    $c \leftarrow$ the common vertex of $Z(\alpha_1)$ and $Z(\alpha_2)$
    $S_{12} \leftarrow X(\alpha_1, c) \oplus Y(\alpha_2, c)$          //using Lemma 3.3
    $S_{21} \leftarrow X(\alpha_2, c) \oplus Y(\alpha_1, c)$          //using Lemma 3.3
    $Sum(\alpha) \leftarrow Sum(\alpha_1) + S_{12} + S_{21} + Sum(\alpha_2)$
    for each $u \in U(\alpha)$ do begin     //find $X(\alpha, u)$ and $Y(\alpha, u)$
      $w \leftarrow d(T_1, c, u)$
      if $u = c$ then               // $u \in U(\alpha_1)$ and $u \in U(\alpha_2)$
        FIND_XY$(\alpha, u, \alpha_1, \alpha_2, c, c, 0, 0)$
      else if $u \notin U(\alpha_1)$ then                  //$u \in U(\alpha_2)$
        FIND_XY$(\alpha, u, \alpha_1, \alpha_2, c, u, w, 0)$
      else FIND_XY$(\alpha, u, \alpha_1, \alpha_2, u, c, 0, w)$
    end
  end
  return$(Sum(r))$, where $r$ is the root of $\tau$
end
```

```
Procedure FIND_XY$(\alpha, u, \alpha_1, \alpha_2, u_1, u_2, w_1, w_2)$
// find $X(\alpha, u)$ and $Y(\alpha, u)$
begin
  $X(\alpha, u) \leftarrow$ MERGE$(X(\alpha_1, u_1) + w_1, X(\alpha_2, u_2) + w_2)$
  $Y(\alpha, u) \leftarrow$ MERGE$(Y(\alpha_1, u_1) + w_1, Y(\alpha_2, u_2) + w_2)$
end
```

The time complexity for the computation of all top tree nodes is analyzed as follows. For each node α, the computation time is $O(|L(Z(\alpha))|)$. No two clusters of the same layer of τ share a common leaf. Therefore, the computation for all nodes in a layer requires $O(n)$ time. Since there are $O(\log n)$ layers, we obtain the following.

Lemma 3.5. *Sum(1-SPLIT) can be found in $O(n \log n)$ time.*

Example 3.2. Fig. 6 illustrates the execution of COMPUTING_AB_1 for the example in Fig. 2, in which the top tree in Fig. 4 is used. The computations for top tree nodes 1, 2, 3, 15, 16, 27 are explained briefly as follows. The computation for the other nodes are similarly.

$\alpha = 1$: Since 1 is a leaf, $Sum(1) = 0$. Since u_1 is an internal node and $i \in A$, $X(1, u_1) = (\phi(i, u_1, v_2)) = (-6)$ and $Y(1, u_1) = \emptyset$.

$\alpha = 2$: Since 2 is a leaf, $Sum(2) = 0$. Since u_1 is an internal node and $a \in B$, $X(2, u_1) = \emptyset$ and $Y(2, u_1) = (\phi(a, u_1, v_2)) = (-3)$.

$\alpha = 3$: Since 3 is a leaf, $Sum(3) = 0$. Since both u_1 and u_2 are internal nodes, $X(3, u_1) = Y(3, u_1) = X(3, u_2) = Y(3, u_2) = \emptyset$.

$\alpha = 15$: Since $c = u_1$, we have $S_{12} = X(1, u_1) \oplus Y(2, u_1) = (-6) \oplus (-3) = 9$ and $S_{21} = X(2, u_1) \oplus Y(1, u_1) = \emptyset \oplus \emptyset = 0$. Thus, $Sum(15) = Sum(1) + Sum(2) + 9 + 0 = 9$. Since $u_1 = c$, we compute $X(15, u_1) = \text{MERGE}(X(1, u_1), X(2, u_1)) = (-6)$ and $Y(15, u_1) = \text{MERGE}(Y(1, u_1), Y(2, u_1)) = (-3)$.

$\alpha = 26$: Since $c = u_4$, we have $S_{12} = X(23, u_4) \oplus Y(24, u_4) = (-1) \oplus \emptyset = 0$ and $S_{21} = X(24, u_4) \oplus Y(23, u_4) = (-3, -2) \oplus (-5, -2) = 24$. Thus, $Sum(26) = Sum(23) + Sum(24) + 0 + 24 = 37$. Since $u_2 \notin U(23)$, $u_2 \in U(24)$, and $w = d(T_1, u_4, u_2) = 1$, we compute $X(26, u_2) = \text{MERGE}(X(23, u_4) + 1, X(24, u_2)) = (-2, -1, 0)$ and $Y(26, u_2) = \text{MERGE}(Y(23, u_4) + 1, Y(24, u_2)) = (-4, -1)$.

$\alpha = 27$: Since $c = u_2$, we have $S_{12} = X(25, u_2) \oplus Y(26, u_2) = (-5, -1) \oplus (-4, -1) = 22$ and $S_{21} = X(26, u_2) \oplus Y(25, u_2) = (-2, -1, 0) \oplus (-4, -2, 2) = 27$. Thus, $Sum(26) = Sum(23) + Sum(24) + 22 + 27 = 124$.

Since node 27 is the root of τ, $Sum([A \times B]) = Sum(27) = 124$.

3.3 Time Complexity

Let $t(n)$ be the worst-case time complexity of our algorithm. The computation of Sum(0-SPLIT) requires $t(|A|) + t(|B|) + t(|C|) + t(|D|) + O(n)$ time, where $|A| + |B| + |C| + |D| = n$ and $0 \le |A|, |B|, |C|, D| \le 2n/3$. The computation of Sum(1-SPLIT) and Sum(2-SPLIT) requires $O(n \log n)$ time. Hence, we have the following recurrence for the running time: if $n < 4$, $t(n) = O(1)$, and otherwise

$$t(n) = \max\{t(a) + t(b) + t(c) + t(d) | a + b + c + d = n, \\ 0 \le a, b, c, d \le 2n/3\} + O(n \log n).$$

In the following, we prove that the solution of $t(n)$ is $O(n \log^2 n)$. Without loss of generality, we simply write the above recurrence as: if $n < 4$, $t(n) = 1$, and otherwise

$$t(n) = \max\{t(a) + t(b) + t(c) + t(d) | a + b + c + d = n, \\ 0 \le a, b, c, d \le 2n/3\} + n \lg n,$$

where $\lg n$ stands for $\log_2 n$. The following lemma shows that $t(n) = O(n \log^2 n)$.

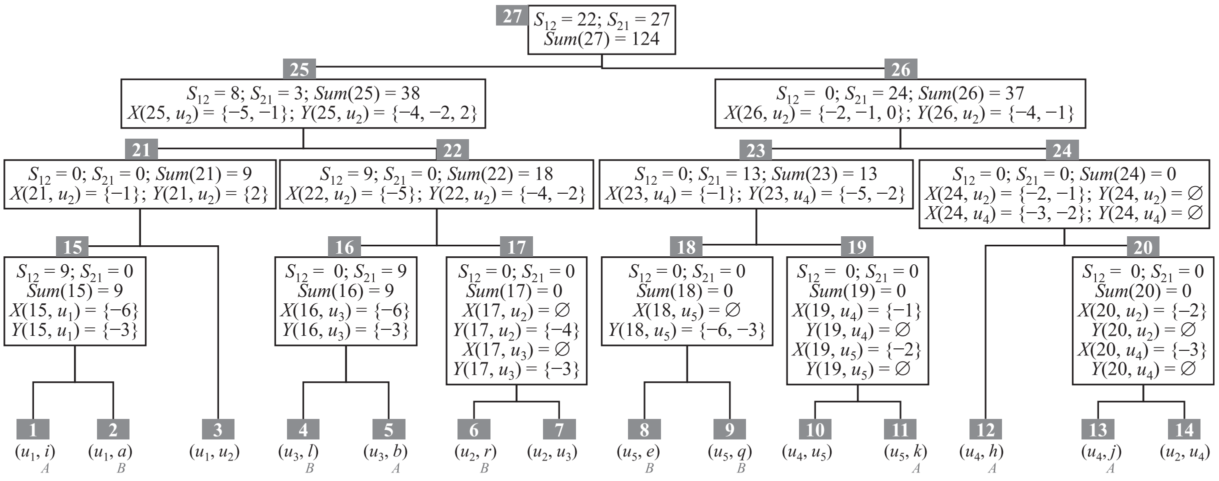
Lemma 3.6. $t(n) \le 3 n \lg^2 n + 1$ *for any* $n \ge 0$.

Proof. We prove this lemma by induction on n. For $n < 4$, since $t(n) = 1$, the lemma holds. Assume that $n \ge 4$ and suppose, by induction, that this lemma is true for all values less than n; we will show that this lemma holds for n as well. By the induction hypothesis, for any a, b, c, d such that $a + b + c + d = n$ and $0 \le a, b, c, d \le 2n/3$, we have

$$\begin{aligned} & t(a) + t(b) + t(c) + t(d) \\ & \le 3\, a \lg^2 a + 3\, b \lg^2 b + 3\, c \lg^2 c + 3\, d \lg^2 d + 4 \\ & \le 3(a + b + c + d) \lg^2(2n/3) + 4 \\ & \le 3\, n \lg^2(2n/3) + 4. \end{aligned}$$

α	$Sum(\alpha)$	$X(\alpha, u)$	$Y(\alpha, u)$
1	0	$X(1, u_1) = \{-6\}$	$Y(1, u_1) = \varnothing$
2	0	$X(2, u_1) = \varnothing$	$Y(2, u_1) = \{-3\}$
3	0	$X(3, u_1) = \varnothing$ $X(3, u_2) = \varnothing$	$Y(3, u_1) = \varnothing$ $Y(3, u_2) = \varnothing$
4	0	$X(4, u_3) = \varnothing$	$Y(4, u_3) = \{-3\}$
5	0	$X(5, u_3) = \{-6\}$	$Y(5, u_1) = \varnothing$
6	0	$X(6, u_2) = \varnothing$	$Y(6, u_2) = \{-4\}$
7	0	$X(7, u_2) = \varnothing$ $X(7, u_3) = \varnothing$	$Y(7, u_2) = \varnothing$ $Y(7, u_3) = \varnothing$
8	0	$X(8, u_5) = \varnothing$	$Y(8, u_5) = \{-6\}$
9	0	$X(9, u_5) = \varnothing$	$Y(9, u_5) = \{-3\}$
10	0	$X(10, u_4) = \varnothing$ $X(10, u_5) = \varnothing$	$Y(10, u_4) = \varnothing$ $Y(10, u_5) = \varnothing$
11	0	$X(11, u_5) = \{-2\}$	$Y(11, u_5) = \varnothing$
12	0	$X(12, u_4) = \{-2\}$	$Y(12, u_4) = \varnothing$
13	0	$X(13, u_4) = \{-3\}$	$Y(13, u_4) = \varnothing$
14	0	$X(14, u_2) = \varnothing$ $X(14, u_4) = \varnothing$	$Y(14, u_2) = \varnothing$ $Y(14, u_4) = \varnothing$

(a) computation for the leaves of τ.



(b) computation for the internal nodes of τ.

Fig. 6. Computation of $Sum([A \times B])$.

Therefore,

$$
\begin{aligned}
& t(n) \\
&= \max\{t(a) + t(b) + t(c) + t(d) | a + b + c + d = n, \\
&\qquad 0 \le a, b, c, d \le 2n/3\} + n \lg n \\
&\le 3\, n \lg^2(2n/3) + 4 + n \lg n \\
&= 3\, n\big(\lg^2 n - 2 \times \lg n \times \lg(3/2) + \lg^2(3/2)\big) + 4 + n \lg n \\
&\qquad (\text{since } \lg^2(2n/3) = (\lg n - \lg 3/2)^2) \\
&\le 3\, n\big(\lg^2 n - \lg n \times \lg(3/2)\big) + 4 + n \lg n \\
&\qquad (\text{since } \lg^2(3/2) - \lg n \times \lg(3/2) < 0) \\
&= 3\, n \lg^2 n - 3\, n \lg n \times \lg(3/2) + 4 + n \lg n \\
&\le 3\, n \lg^2 n - n \lg n \times \lg(3/2) + 4 \\
&\qquad (\text{since } n \lg n - 2\, n \lg n \times \lg(3/2) < 0) \\
&\le 3\, n \lg^2 n + 1, \qquad (\text{since } 3 - n \lg n \times \lg(3/2) < 0)
\end{aligned}
$$

which completes the proof of this lemma. □

Consequently, we obtain the following.

Theorem 3.1. *The l_1-norm path-difference distance problem can be solved in $O(n \log^2 n)$ time.*

4 Computing the L_P-Norm Path-Difference Distance for Any Positive Integer P

Let $p \ge 1$ be an integer. This section extends the algorithm in Section 3 to solve the l_p-norm path-difference problem in $O(pn \log^2 n)$ time.

To start with, we extend the notations *Sum* and $\oplus$ as follows. For any set $Q \subseteq [\Sigma]^2$, define $Sum_p(Q)$ to be $\sum_{\{i,j\} \in Q} |d(T_1, i, j) - d(T_2, i, j)|^p$. Recall that it has been assumed that taking the pth root of a number can be done in constant time. Accordingly, our problem is to compute $Sum_p([\Sigma]^2)$. For two sequences X and Y of numbers, define $X \oplus_p Y$ to be $\sum_{1 \le i \le |X|,\, 1 \le j \le |Y|} |X[i] + Y[j]|^p$.

Lemma 4.1. *Let S, S' be two disjoint subsets of Σ and p, q be two vertices such that for every $\{i, j\} \in [S \times S']$, vertex p is on $P(T_1, i, j)$ and vertex q is on $P(T_2, i, j)$. Then, $Sum_p([S \times S']) = \Phi(S, p, q) \oplus_p \Phi(S', p, q)$.*

Proof. According to Lemma 3.1, $Sum_p([S \times S']) = \sum_{\{i,j\} \in [S \times S']} |d(T_1, i, j) - d(T_2, i, j)|^p = \sum_{\{i,\ j\} \in [S \times_{S'}]} |\phi(i, p, q) + \phi(j, p, q)|^p = \Phi(S, p, q) \oplus_p \Phi(S', p, q)$. Thus, the lemma holds. □

Lemma 4.2. *Let X and Y be two non-decreasing sequences of numbers. The value of $X \oplus_p Y$ can be computed in $O(p \times (|X| + |Y|))$ time.*

Proof. For each $X[i]$, $1 \le i \le |X|$, let $m(i)$ be the maximum j such that $Y[j] + X[i] < 0$. Then, we have

$$
X \oplus_p Y = \sum_{1 \le i \le |X|} \left\{ \sum_{1 \le j \le m(i)} (-X[i] - Y[j])^p + \sum_{m(i) < j \le |Y|} (X[i] + Y[j])^p \right\}.
$$

Applying the binomial theorem, we obtain

$$
\begin{aligned}
& X \oplus_p Y \\
&= \sum_{1 \le i \le |X|} \left\{ \sum_{1 \le j \le m(i)} \sum_{0 \le k \le p} (-1)^p C(p, k) \times X[i]^{p-k} \times Y[j]^k \right. \\
&\quad \left. + \sum_{m(i) < j \le |Y|} \sum_{0 \le k \le p} C(p, k) \times X[i]^{p-k} \times Y[j]^k \right\}
\end{aligned}
$$

$$= \sum_{0\le k\le p}\sum_{1\le i\le |X|}\left\{C(p,k)X[i]^{p-k}\times\left((-1)^p\sum_{1\le j\le m(i)}Y[j]^k + \sum_{m(i)<j\le|Y|}Y[j]^k\right)\right\},$$

where 0^0 is defined as 1. In the following, we complete the proof by showing that after an $O(p \times (|X|+|Y|))$-time preprocessing, for any k and i, where $0 \le k \le p$ and $1 \le i \le |X|$, the value of

$$C(p,k)X[i]^{p-k}\times\left((-1)^p\sum_{1\le j\le m(i)}Y[j]^k + \sum_{m(i)<j\le|Y|}Y[j]^k\right)$$

can be computed in $O(1)$ time. We proceed to describe the preprocessing. First, we compute $m(i)$ for each $X[i]$, $1 \le i \le |X|$, which requires $O(|X|+|Y|)$ time as mentioned in the proof of Lemma 3.3. Next, for each k, $0 \le k \le p$, we compute the following:

(1) $C(p, k)$,
(2) a sequence $X^{(k)} = (X[1]^k, X[2]^k, \ldots, X[|X|]^k)$, and
(3) a sequence $PS^{(k)}$, which is the prefix sum of the sequence $(Y[1]^k, Y[2]^k, \ldots, Y[|Y|]^k)$ (i.e., $PS^{(k)}[i] = Y[1]^k + Y[2]^k + \ldots + Y[i]^k$ for $1 \le i \le |Y|$).

Clearly, this step requires $O(p + p|X| + p|Y|) = O(p \times (|X| + |Y|))$ time.

After the above preprocessing, for any given k and i, where $0 \le k \le p$ and $1 \le i \le |X|$, $C(p, k)\, X[i]^{p-k}\times ((-1)^p \sum_{1\le j\le m(i)} Y[j]^k + \sum_{m(i)<j\le|Y|} Y[j]^k)$ can be computed as

$$C(p,k)X^{(p-k)}[i]\times((-1)^pPS^{(k)}[m(i)] + PS^{(k)}[|Y|] - PS^{(k)}[m(i)])$$

in $O(1)$ time. Therefore, the lemma holds. □

Based upon Lemmas 4.1 and 4.2, the algorithm in Section 3 is extended to compute $Sum_p([\Sigma]^2)$ as follows. Let v_1, v_2, 0-SPLIT, 1-SPLIT, 2-SPLIT, A, B, C, and D be defined the same as in Section 3. Similarly, the computation of Sum_p(0-SPLIT) is done by four recursive calls. In the following, we show how to compute Sum_p(2-SPLIT) and Sum_p(1-SPLIT). For brevity, similar to Section 3, only the computation of $Sum_p([A \times D])$ and $Sum_p([A \times B])$ is described.

By Lemma 4.1, $Sum_p([A \times D]) = \Phi(A, v_1, v_2) \oplus_p \Phi(D, v_1, v_2)$. Thus, we compute $Sum_p([A \times D])$ as follows. First, in $O(n \log n)$ time we compute X and Y as the sorted sequences of the values in $\Phi(A, v_1, v_2)$ and the values in $\Phi(D, v_1, v_2)$, respectively. Then, we determine $Sum_p([A\times D])$ by computing $X \oplus_p Y$, which by Lemma 4.2 requires $O(pn)$ time.

Lemma 4.3. *Sum_p(2-SPLIT) can be computed in $O(n \log n + pn)$ time for any integer $p \ge 1$.*

Lemma 4.3 is an extension of the result in Section 3.1. Similarly, the procedure COMPUTING_AB_1 in Section 3.2 can be extended as well to compute $Sum_p([A \times B])$ by replacing Lines 18 and 19 with the following.

```
18.    $S_{12} \leftarrow X(\alpha_1, c)\oplus_p Y(\alpha_2, c)$
19.    $S_{21} \leftarrow X(\alpha_2, c)\oplus_p Y(\alpha_1, c)$
```

With the above modification, the computation for each top node α takes a total of $O(p|L(\alpha)|)$ time. Since τ consists of $O(\log n)$ layers and no two clusters of the same layer share a common leaf, the time complexity for the computation of all top tree nodes is $O(pn \log n)$. Consequently, we obtain the following.

Lemma 4.4. *$Sum_p(1 - Split)$ can be computed in $O(pn\ log\, n)$ time for any integer $p \ge 1$.*

Let $t_p(n)$ be the worst-case time complexity of our algorithm. Then, we have the following recurrence: if $n < 4$, $t_p(n) = O(1)$, and otherwise

$$t_p(n) = \max\{t_p(a) + t_p(b) + t_p(c) + t_p(d) | a + b + c + d = n, \\ 0 \le a, b, c, d \le 2n/3\} + O(pn \log n).$$

It is easy to conclude the following by adapting the proof of Lemma 3.6.

Theorem 4.1. *For any integer $p \ge 1$, the l_p-norm path-difference distance problem can be solved in $O(pn\ log^2\ n)$ time.*

5 AN ALGORITHM FOR THE L_2-NORM PATH-DIFFERENCE DISTANCE PROBLEM

For the l_2-norm path-difference distance problem, the algorithm in Section 4 has improved the previous upper bound from $O(n^2)$ to $O(n \log^2 n)$. In this section, we further reduce the time complexity to $O(n \log n)$. Our problem is to compute $Sum_2([\Sigma]^2)$. We solve the problem by a modified version of the algorithm in Section 3. For brevity, only the computation of $Sum_2([A \times D])$ and $Sum_2([A \times B])$ is described.

5.1 Computing $Sum_2([A \times D])$

For an integer $k \ge 1$ and a sequence X of numbers, define the kth *power sum* of X to be $\sigma_k(X) = X[1]^k + X[2]^k + \ldots + X[|X|]^k$. Then, we have the following.

Lemma 5.1. *Let X and Y be two sequences of numbers. Given $|X|$, $|Y|$, $\sigma_1(X)$, $\sigma_1(Y)$, $\sigma_2(X)$, and $\sigma_2(Y)$, the value of $X \oplus_2 Y$ can be computed in $O(1)$ time.*

Proof. By definition,

$$\begin{aligned} &X\oplus_2 Y \\ &= \sum_{1\le i\le|X|,1\le j\le|Y|}|X[i]+Y[j]|^2 \\ &= \sum_{1\le i\le|X|,1\le j\le|Y|}(X[i]^2 + 2X[i]Y[j] + Y[j]^2) \\ &= |Y|\times\sum_{1\le i\le|X|}X[i]^2 + 2\left(\sum_{1\le i\le|X|}X[i]\right)\left(\sum_{1\le j\le|Y|}Y[j]\right) \\ &\quad + |X|\times\sum_{1\le j\le|Y|}Y[j]^2 \\ &= |Y|\times\sigma_2(X) + 2\times\sigma_1(X)\times\sigma_1(Y) + |X|\times\sigma_2(Y). \end{aligned}$$

Therefore, the lemma holds. □

By Lemma 4.1, $Sum_2([A \times D]) = \Phi(A, v_1, v_2) \oplus_2 \Phi(D, v_1, v_2)$. By Lemma 5.1, $\Phi(A, v_1, v_2) \oplus_2 \Phi(B, v_1, v_2)$ can be determined in $O(1)$ time if the following values are given: $|A|$, $|D|$, $\sigma_1(\Phi(A, v_1, v_2))$, $\sigma_1(\Phi(D, v_1, v_2))$, σ_2

$(\Phi(A,\ v_1,\ v_2))$, and $\sigma_2(\Phi(D,\ v_1,\ v_2))$. Since these values can be found in $O(n)$ time, we obtain the following. (An example can be found at [24].)

Lemma 5.2. *Sum_2(2-SPLIT) can be computed in $O(n)$ time.*

5.2 Computing $Sum_2([A \times B])$

Similar to Section 3.2, we build a top tree τ of $T_1|_{A\cup B}$ and compute $Sum_2([A(\alpha) \times B(\alpha)])$ for all nodes α of τ in a bottom-up fashion. Our intent is to compute $Sum_2([A(\alpha)\times B(\alpha)])$ for each node α in $O(1)$ time. To achieve the desired time bound, for each node α of τ, we maintain $|A(\alpha)|$, $|B(\alpha)|$, and the following four auxiliary values for each $u \in U(\alpha)$: $\sigma_k(\Phi(A(\alpha),\ u,\ v_2))$ and $\sigma_k(\Phi(B(\alpha),\ u,\ v_2))$, where $k = 1$ and 2.

In the following, we show that the computation for each top tree node can be done in $O(1)$ time. For brevity, the computation for each leaf is omitted. Let α be an internal node of τ. Let α_1 and α_2 be the children of α, and let c be the common vertex of $Z(\alpha_1)$ and $Z(\alpha_2)$. Similar to Section 3.2, for the computation of $Sum_2([A(\alpha) \times B(\alpha)])$, we only show how to determine $Sum_2([A(\alpha_1) \times B(\alpha_2)])$. By Lemma 4.1, $Sum_2([A(\alpha_1) \times B(\alpha_2)])$ can be computed as $\Phi(A(\alpha_1),\ c,\ v_2) \oplus_2 \Phi(B(\alpha_2),\ c,\ v_2)$. By Lemma 5.1, this computation can be done in $O(1)$ time if the following values are given: $|A(\alpha_1)|$, $|B(\alpha_2)|$, and $\sigma_k(\Phi(A(\alpha_1),\ c,\ v_2))$ and $\sigma_k(\Phi(B(\alpha_2),\ c,\ v_2))$, where $k = 1$ and 2. Since c is a boundary vertex of both α_1 and α_2, these values have been determined during the computation for α_1 and α_2. Therefore, $Sum_2([A(\alpha_1)\times B(\alpha_2)])$ can be determined in $O(1)$ time.

The values of $|A(\alpha)|$ and $|B(\alpha)|$ are computed, respectively, as $|A(\alpha_1)| + |A(\alpha_2)|$ and $|B(\alpha_1)| + |B(\alpha_2)|$. In the following, we show that the four auxiliary values for each $u \in U(\alpha)$ can also be computed in $O(1)$ time. We need the following.

Lemma 5.3. Let $p \geq 2$ be an even integer. *Let w be a number and $X,\ X'$ be two sequences of numbers such that $X' = X + w$. Given $|X|$, $\sigma_1(X)$, $\sigma_2(X), \ldots,$ and $\sigma_p(X)$, the values $\sigma_1(X')$, $\sigma_2(X'), \ldots,$ and $\sigma_p(X')$ can be computed in $O(p^2)$ time.*

Proof. Consider the computation of $\sigma_k(X')$ for a fixed k, where $1 \leq k \leq p$. Applying the binomial theorem, we have

$$\begin{aligned}\sigma_k(X') &= \sum_{1\le i\le |X|} (X[i] + w)^k \\ &= \sum_{1\le i\le |X|}\sum_{0\le l\le k} (C(k,l)X[i]^{k-l} \times w^l) \\ &= \sum_{0\le l\le k}\left(C(k,l) \times w^l \times \sum_{1\le i\le |X|} X[i]^{k-l}\right) \\ &= \sum_{0\le l\le k} (C(k,l) \times w^l \times \sigma_{k-l}(X)).\end{aligned}$$

Therefore, each $\sigma_k(X')$ can be computed in $O(k)$ time by using the given values. Consequently, the lemma holds.□

For brevity, only the computation of $\sigma_k(\Phi(A(\alpha),\ u,\ v_2))$, where $k = 1$ and 2, is described. The following three cases are considered.

Case 1. $u \in U(\alpha_1)$ and $u \in U(\alpha_2)$. In this case, $u = c$ and we compute

$$\begin{aligned}\sigma_k(\Phi(A(\alpha), c, v_2)) = {} & \sigma_k(\Phi(A(\alpha_1), c, v_2)) \\ & + \sigma_k(\Phi(A(\alpha_2), c, v_2))\end{aligned}$$

for $k = 1$ and 2.

Case 2. $u \notin U(\alpha_1)$ and $u \in U(\alpha_2)$. Let $w = d(T_1,\ c,\ u)$. As discussed in Section 3.2, for each $i \in A(\alpha_1)$, we have $\phi(i,\ u,\ v_2) = \phi(i,\ c,\ v_2) + w$. (See Fig. 5.) That is, $\Phi(A(\alpha_1),\ u,\ v_2) = \Phi(A(\alpha_1),\ c,\ v_2) + w$. First, we obtain the values $\sigma_k(\Phi(A(\alpha_1),\ u,\ v_2))$, $k = 1$ and 2, from the values $\sigma_k(\Phi(A(\alpha_1),\ c,\ v_2))$, $k = 1$ and 2. By applying Lemma 5.3 with $p = 2$, this step requires $O(1)$ time. Then, we compute

$$\begin{aligned}\sigma_k(\Phi(A(\alpha), u, v_2)) = {} & \sigma_k(\Phi(A(\alpha_1), u, v_2)) \\ & + \sigma_k(\Phi(A(\alpha_2), u, v_2))\end{aligned}$$

for $k = 1$ and 2.

Case 3. $u \in U(\alpha_1)$ and $u \notin U(\alpha_2)$. Similar to Case 2.

Consequently, the computation for each top tree node requires $O(1)$ time.

Our computation of $Sum_2([A \times B])$ is summarized in procedure COMPUTING_AB_2, in which $Sum_2([A(\alpha) \times B(\alpha)])$, $\Phi(A(\alpha),\ u,\ v_2)$, and $\Phi(B(\alpha),\ u,\ v_2)$ are simply denoted by $Sum(\alpha)$, $\Phi(A(\alpha),\ u)$, and $\Phi(B(\alpha),\ u)$, respectively.

Since the size of τ is linear to the size of $T_1|_{A\cup B}$ [2], we obtain the following. (An example can be found at [24].)

Lemma 5.4. *Sum_2(1-SPLIT) can be computed in $O(n)$ time.*

According to Lemmas 5.2 and 5.4, we obtain the following recurrence for the time complexity:

$$\begin{aligned}t_2(n) = \max\{&t_2(a) + t_2(b) + t_2(c) + t_2(d) | a + b + c + d = n, \\ &0 \le a, b, c, d \le 2n/3\} + O(n).\end{aligned}$$

It is not difficult to prove by induction that the solution to this recurrence is $O(n \log n)$. Hence, we obtain the following.

Theorem 5.1. *The l_2-norm path-difference distance problem can be solved in $O(n \log n)$ time.*

6 COMPUTING THE L_P-NORM PATH-DIFFERENCE DISTANCE FOR A POSITIVE EVEN INTEGER P

Let $p \geq 2$ be an even integer. This section extends the algorithm in Section 5 to solve the l_p-norm path-difference distance problem in $O(p^2 n \log n)$ time.

Lemma 6.1. *Let X and Y be two sequences of numbers. Given $|X|$, $|Y|$, $\sigma_1(X)$, $\sigma_1(Y)$, $\sigma_2(X)$, $\sigma_2(Y)$, $\ldots$, $\sigma_p(X)$, and $\sigma_p(Y)$, the value of $X \oplus_p Y$ can be computed in $O(p)$ time, where $p \geq 2$ is an even integer.*

Proof. Since p is even, $X \oplus_p Y = \sum_{1\le i\le |X|,\ 1\le j\le |Y|} (X[i]+Y[j])^p$. Applying the binomial theorem, we obtain

$$\begin{aligned}X\oplus_p Y &= \sum_{1\le i\le |X|, 1\le j\le |Y|}\ \sum_{0\le k\le p} C(p,k) \times X[i]^{p-k} \times Y[j]^k \\ &= \sum_{0\le k\le p}\left(C(p,k) \sum_{1\le i\le |X|} X[i]^{p-k} \sum_{1\le j\le |Y|} Y[j]^k\right) \\ &= \sum_{0\le k\le p} C(p,k)\sigma_{p-k}(X)\sigma_k(Y).\end{aligned}$$

Note that 0^0 is defined as 1 and thus, $|X| = \sigma_0(X)$ and $|Y| = \sigma_0(Y)$. In $O(p)$ time, all $C(p,\ k)$, $0 \leq k \leq p$, are precomputed. After that, the value of $C(p,\ k)\ \sigma_{p-k}(X)$

$\sigma_k(Y)$ can be obtained in $O(1)$ time for any k, where $0 \le k \le p$. Therefore, $X \oplus_p Y$ can be computed in $O(p)$ time. Thus, the lemma holds. □

```
Procedure COMPUTING_AB_2
begin
1  $\tau \leftarrow$ a top tree of $T_1|_{AB}$
2  for each leaf $\alpha$ of $\tau$ do begin
3    $Sum(\alpha) \leftarrow 0$
4    $(s, t) \leftarrow$ the edge $Z(\alpha)$
5    if $s$ and $t$ are both internal nodes of $T_1|_{A \cup B}$ then begin
6      $(|A(\alpha)|, |B(\alpha)|) \leftarrow (0, 0)$
7      for ($k \in \{1, 2\}$) and ($u \in \{s, t\}$) do
8        $(\sigma_k(\Phi(A(\alpha), u)), \sigma_k(\Phi(B(\alpha), u))) \leftarrow (0, 0)$
9    end
10   else begin //either $s$ or $t$ is an internal node
11     if $s$ is not an internal node of $T_1|_{A \cup B}$ then
12       exchange $s$ and $t$    //set $s$ as the internal node
13     if $t \in A$ then begin
14       $(|A(\alpha)|, |B(\alpha)|) \leftarrow (1, 0)$
15       for $k \in \{1, 2\}$ do
16         $(\sigma_k(\Phi(A(\alpha), s)), \sigma_k(\Phi(B(\alpha), s))) \leftarrow$
             $(\phi(t, s, v_2)^k, 0)$
17     end
18     else begin // $t \in B$
19       $(|A(\alpha)|, |B(\alpha)|) \leftarrow (0, 1)$
20       for $k \in \{1, 2\}$ do
21         $(\sigma_k(\Phi(A(\alpha), s)), \sigma_k(\Phi(B(\alpha), s))) \leftarrow$
             $(0, \phi(t, s, v_2)^k)$
22     end
23   end
24 end
25 for each internal node $\alpha$ of $\tau$, in a bottom-up fashion, do
26 begin
27   $(\alpha_1, \alpha_2) \leftarrow$ the children of $\alpha$
28   $c \leftarrow$ the common vertex of $Z(\alpha_1)$ and $Z(\alpha_2)$
29   $S_{12} \leftarrow \Phi(A(\alpha_1), c) \oplus_2 \Phi(B(\alpha_2), c)$    //using Lemma 5.1
30   $S_{21} \leftarrow \Phi(A(\alpha_2), c) \oplus_2 \Phi(B(\alpha_1), c)$    //using Lemma 5.1
31   $Sum(\alpha) \leftarrow Sum(\alpha_1) + S_{12} + S_{21} + Sum(\alpha_2)$
32   $(|A(\alpha)|, |B(\alpha)|) \leftarrow (|A(\alpha_1)| + |A(\alpha_2)|, |B(\alpha_1)| + |B(\alpha_2)|)$
33   for each $u \in U(\alpha)$ do begin
34     $w \leftarrow d(T_1, c, u)$
35     if $u = c$ then              // $u \in U(\alpha_1)$ and $u \in U(\alpha_2)$
36       FIND_SIGMA($\alpha, u, \alpha_1, \alpha_2, c, c, 0, 0$)
37     else if $u \notin U(\alpha_1)$ then           // $u \in U(\alpha_2)$
38       FIND_SIGMA($\alpha, u, \alpha_1, \alpha_2, c, u, w, 0$)
39     else FIND_SIGMA($\alpha, u, \alpha_1, \alpha_2, u, c, 0, w$)
40   end
41 end
42 return($Sum(r)$), where $r$ is the root of $\tau$
end
```

```
Procedure FIND_SIGMA($\alpha, u, \alpha_1, \alpha_2, u_1, u_2, w_1, w_2$)
//find $\sigma_k(\Phi(A(\alpha), u))$ and $\sigma_k(\Phi(B(\alpha), u))$
begin
1 for $k \in \{1, 2\}$ do begin          //using Lemma 5.3
2   $\sigma_k(\Phi(A(\alpha), u)) \leftarrow \sigma_k(\Phi(A(\alpha_1), u_1) + w_1)$
                          $+ \sigma_k(\Phi(A(\alpha_2), u_2) + w_2)$
3   $\sigma_k(\Phi(B(\alpha), u)) \leftarrow \sigma_k(\Phi(B(\alpha_1), u_1) + w_1)$
                          $+ \sigma_k(\Phi(B(\alpha_2), u_2) + w_2)$
4 end
end
```

We proceed to present an extended algorithm. For brevity, only the computation of $Sum_p([A \times D])$ and $Sum_p([A \times B])$ is described. According to Lemma 6.1, $Sum_p([A \times D]) = \Phi(A, v_1, v_2) \oplus_p \Phi(D, v_1, v_2)$ can be computed in $O(p)$ time if the following values are given: $|\Phi(A, v_1, v_2)|$, $|\Phi(D, v_1, v_2)|$, and $\sigma_k(\Phi(A, v_1, v_2))$ and $\sigma_k(\Phi(D, v_1, v_2))$, where $k = 1, 2, \ldots, p$. Since these values can be computed in $O(pn)$ time, we obtain the following.

Lemma 6.2. *Sum_p(2-SPLIT) can be computed in $O(pn)$ time for any even integer $p \ge 2$.*

Next, the algorithm in Section 5.2 is extended to compute $Sum_p([A \times B])$ as follows: for each node α of the top tree τ, $Sum_p([A(\alpha) \times B(\alpha)])$ is computed; in addition, we maintain $|A(\alpha)|$, $|B(\alpha)|$ and the following auxiliary values for every $u \in U(\alpha)$: $\sigma_k(\Phi(A(\alpha), u, v_2))$ and $\sigma_k(\Phi(B(\alpha), u, v_2))$, where $k = 1, 2, \ldots, p$. We proceed to describe the computation for each top tree node. For brevity, only the computation for each internal node α is described. Let α_1 and α_2 be the children of α, and let c be the common vertex of $Z(\alpha_1)$ and $Z(\alpha_2)$. For the computation of $Sum_p([A(\alpha) \times B(\alpha)])$, similar to Section 5.2, we only show how to determine the value of $Sum_p([A(\alpha_1) \times B(\alpha_2)])$. According to Lemmas 4.1 and 6.1, this value is computed as $\Phi(A(\alpha_1), c, v_2) \oplus_p \Phi(B(\alpha_2), c, v_2)$ in $O(p)$ time by using $|A(\alpha_1)|$, $|B(\alpha_2)|$, and $\sigma_k(\Phi(A(\alpha_1), c, v_2))$ and $\sigma_k(\Phi(B(\alpha_2), c, v_2))$, where $k = 1, 2, \ldots, p$.

The values of $|A(\alpha)|$ and $|B(\alpha)|$ are computed as $|A(\alpha_1)| + |A(\alpha_2)|$ and $|B(\alpha_1)| + |B(\alpha_2)|$, respectively. We proceed to describe the computation of the auxiliary values for each $u \in U(\alpha)$. For brevity, only the computation of $\sigma_k(\Phi(A(\alpha), u, v_2))$, where $k = 1, 2, \ldots, p$, is given. The following three cases are considered.

Case 1. $u \in U(\alpha_1)$ and $u \in U(\alpha_2)$. In this case, $u = c$ and we compute

$$\sigma_k(\Phi(A(\alpha), c, v_2)) = \sigma_k(\Phi(A(\alpha_1), c, v_2)) + \sigma_k(\Phi(A(\alpha_2), c, v_2))$$

for each k, $1 \le k \le p$.

Case 2. $u \notin U(\alpha_1)$ and $u \in U(\alpha_2)$. Recall that in this case we have $\phi(i, u, v_2) = \phi(i, c, v_2) + w$ for each $i \in A(\alpha_1)$, where $w = d(T_1, c, u)$. First, we obtain the values $\sigma_k(\Phi(A(\alpha_1), u, v_2))$, $k = 1, 2, \ldots, p$, from the values $\sigma_k(\Phi(A(\alpha_1), c, v_2))$, $k = 1, 2, \ldots, p$. By Lemma 5.3 this step requires $O(p^2)$ time. Then, we compute

$$\sigma_k(\Phi(A(\alpha), u, v_2)) = \sigma_k(\Phi(A(\alpha_1), u, v_2)) + \sigma_k(\Phi(A(\alpha_2), u, v_2))$$

for each k, $1 \le k \le p$.

Case 3. $u \in U(\alpha_1)$ and $u \notin U(\alpha_2)$. Similar to Case 2.

Based upon the above discussion, it is easy to see that for any even integer p, the procedure COMPUTING_AB_2 can be extended to find $Sum_p([A \times B])$ by replacing all "$k \in \{1, 2\}$" with "$k \in \{1, 2, 3, \ldots, p\}$" and replacing all $\oplus_2$ with $\oplus_p$.

In summary, the computation for each node of τ requires $O(p^2)$ time. Therefore, we have the following.

Lemma 6.3. *Sum_p(1-SPLIT) can be computed in $O(p^2 n)$ time for any even integer $p \ge 2$.*

According to Lemmas 6.2 and 6.3, the following recurrence is obtained:

$$t_p(n) = \max\{t_p(a) + t_p(b) + t_p(c) + t_p(d) | a + b + c + d = n, \\ 0 \leq a, b, c, d \leq 2n/3\} + O(p^2 n).$$

It is not difficult to prove by induction that the solution to this recurrence is $O(p^2 n \log n)$. Hence, we obtain the following.

Theorem 6.1. *For any even integer $p \geq 2$, the l_p-norm path-difference distance problem can be solved in $O(p^2 n \log n)$ time.*

7 An Algorithm for the L_∞-Norm Path-Difference Distance Problem

This section modifies the algorithm in Section 5 to solve the l_∞-norm path-difference distance problem in $O(n \log n)$ time. For any set $Q \subseteq [\Sigma]^2$, define $Sum_\infty(Q)$ to be $\max_{\{i,j\} \in Q} |d(T_1, i, j) - d(T_2, i, j)|$. Then, our problem is to compute $Sum_\infty([\Sigma]^2)$. For a sequence X of numbers, let $\min(X)$ and $\max(X)$ be, respectively, the smallest and largest numbers in X. For two sequences X and Y of numbers, define $X \oplus_\infty Y$ to be $\max_{1 \leq i \leq |X|,\, 1 \leq j \leq |Y|} |X[i] + Y[j]|$. Then, we have the following.

Lemma 7.1. *Let X and Y be two sequences of numbers. Given $min(X)$, $min(Y)$, $max(X)$, and $max(Y)$, the value of $X \oplus_\infty Y$ can be computed in $O(1)$ time.*

Proof. Let a and b be two indices such that $X \oplus_\infty Y = |X[a] + Y[b]|$. Assume first that $X[a] + Y[b] \geq 0$. By definition, $|X[a] + Y[b]| \geq |\max(X) + \max(Y)|$. On the other hand, since $\max(X) + \max(Y) \geq X[a] + Y[b] \geq 0$, we have $|\max(X) + \max(Y)| \geq |X[a] + Y[b]|$. By combining these two statements, we conclude that if $X[a] + Y[b] \geq 0$, $|\max(X) + \max(Y)| = |X[a] + Y[b]|$. Similarly, we can conclude that if $X[a] + Y[b] < 0$, $|\min(X) + \min(Y)| = |X[a] + Y[b]|$. Consequently, $X \oplus_\infty Y$ can be computed as

$$\max\{|\max(X) + \max(Y)|, |\min(X) + \min(Y)|\}.$$

Thus, the lemma holds. □

We proceed to show how to modify the algorithm in Section 5 to compute $Sum_\infty([\Sigma]^2)$. For brevity, only the computation of $Sum_\infty([A \times D])$ and $Sum_\infty([A \times B])$ is described. According to Lemma 3.1, $Sum_\infty([A \times D])$ can be computed as $\Phi(A, v_1, v_2) \oplus_\infty \Phi(D, v_1, v_2)$. By Lemma 7.1, this computation requires $O(1)$ time if $\min(X)$, $\min(Y)$, $\max(X)$, and $\max(Y)$ are given, where X and Y denote $\Phi(A, v_1, v_2)$ and $\Phi(D, v_1, v_2)$, respectively. Since these values can be obtained in $O(n)$ time, the computation of $Sum_\infty([A \times D])$ requires $O(n)$ time.

Next, the algorithm in Section 5.2 is modified to compute $Sum_\infty([A \times B])$ as follows: for each node α of the top tree τ, $Sum_\infty([A(\alpha) \times B(\alpha)])$ is computed; in addition, we maintain the following auxiliary values for every $u \in U(\alpha)$: $\min_A(\alpha, u) = \min(X)$, $\min_B(\alpha, u) = \min(Y)$, $\max_A(\alpha, u) = \max(X)$, and $\max_B(\alpha, u) = \max(Y)$, where $X = \Phi(A(\alpha), u, v_2)$ and $Y = \Phi(B(\alpha), u, v_2)$. We proceed to describe the computation for each top tree node. For brevity, only the computation for each internal node α of τ is described. Let α_1 and α_2 be the children of α, and let c be the common vertex of $Z(\alpha_1)$ and $Z(\alpha_2)$. Consider the computation of $Sum_\infty([A(\alpha) \times B(\alpha)])$. Let

$$m_{1,1} = Sum_\infty([A(\alpha_1) \times B(\alpha_1)]),$$
$$m_{1,2} = Sum_\infty([A(\alpha_1) \times B(\alpha_2)]),$$
$$m_{2,1} = Sum_\infty([A(\alpha_2) \times B(\alpha_1)]), \text{ and}$$
$$m_{2,2} = Sum_\infty([A(\alpha_2) \times B(\alpha_2)].$$

Then, we have $Sum_\infty([A(\alpha) \times B(\alpha)]) = \max\{m_{1,1}, m_{1,2}, m_{2,1}, m_{2,2}\}$. The values of $m_{1,1}$ and $m_{2,2}$ have been found during the computation for α_1 and α_2, respectively. According to Lemmas 3.1 and 7.1, $m_{1,2}$ can be computed as $\Phi(A(\alpha_1), c, v_2) \oplus_\infty \Phi(B(\alpha_2), c, v_2)$ in $O(1)$ time by using $\min_A(\alpha_1, c)$, $\min_B(\alpha_2, c)$, $\max_A(\alpha_1, c)$, and $\max_B(\alpha_2, c)$. Similarly, $m_{2,1}$ can be computed in $O(1)$ time. Therefore, the computation of $Sum_\infty([A(\alpha) \times B(\alpha)])$ requires $O(1)$ time.

We proceed to describe the computation of the auxiliary values for each $u \in U(\alpha)$. For brevity, only the computation of $\min_A(\alpha, u)$ and $\max_A(\alpha, u)$ is given.

Case 1. $u \in U(\alpha_1)$ and $u \in U(\alpha_2)$. In this case, $u = c$ and we compute

$$\begin{aligned}\min_A(\alpha, c) &= \min(\Phi(A(\alpha), c, v_2)) \\ &= \min\{\min(\Phi(A(\alpha_1), c, v_2)), \min(\Phi(A(\alpha_2), c, v_2))\} \\ &= \min\left\{\min_A(\alpha_1, c), \min_A(\alpha_2, c)\right\}\end{aligned}$$

and

$$\begin{aligned}\max_A(\alpha, c) &= \max(\Phi(A(\alpha), c, v_2)) \\ &= \max\{\max(\Phi(A(\alpha_1), c, v_2)), \max(\Phi(A(\alpha_2), c, v_2))\} \\ &= \max\left\{\max_A(\alpha_1, c), \max_A(\alpha_2, c)\right\}.\end{aligned}$$

Case 2. $u \notin U(\alpha_1)$ and $u \in U(\alpha_2)$. Let $w = d(T_1, c, u)$. Since $\phi(i, u, v_2) = \phi(i, c, v_2) + w$ for each $i \in A(\alpha_1)$, we compute

$$\min_A(\alpha, u) = \min\left\{\min_A(\alpha_1, c) + w, \min_A(\alpha_2, c)\right\}$$

and

$$\max_A(\alpha, u) = \max\left\{\max_A(\alpha_1, c) + w, \max_A(\alpha_2, c)\right\}.$$

Case 3. $u \in U(\alpha_1)$ and $u \notin U(\alpha_2)$. Similar to Case 2.

The computation for each top tree node takes $O(1)$ time. Therefore, the computation of $Sum_\infty([A \times B])$ requires $O(n)$ time. Our computation of $Sum_\infty([A \times B])$ is summarized in procedure COMPUTING_AB_INF, in which $Sum_\infty([A(\alpha) \times B(\alpha)])$, $\Phi(A(\alpha), u, v_2)$, and $\Phi(B(\alpha), u, v_2)$ are simply denoted by $Sum(\alpha)$, $\Phi(A(\alpha), u)$, and $\Phi(B(\alpha), u)$, respectively.

Based upon the above discussion, the time complexity can be expressed as the same recurrence in Section 5. Therefore, we obtain the following.

Theorem 7.1. *The l_∞-norm path-difference distance problem can be solved in $O(n \log n)$ time.*

8 An Algorithm for the L_0-Norm Path-Difference Distance Problem

In this section, we briefly describe how to modify the algorithm in Section 3 to compute the l_0-norm path-difference distance in $O(n \log^2 n)$ time. For any real number x, define $h(x)$ as follows: $h(x)$ is 0 if $x = 0$, and is 1 otherwise. For any $Q \subseteq [\Sigma]^2$, define $Sum_0(Q)$ to be $\sum_{\{i,j\} \in Q} h(d(T_1, i, j) - d(T_2, i, j))$. Then, our problem is to compute $Sum_0([\Sigma]^2)$.

```
Procedure COMPUTING_AB_INF
begin
 1  $\tau \leftarrow$ a top tree of $T_1|_{AB}$
 2  for each leaf $\alpha$ of $\tau$ do begin
 3    $Sum(\alpha) \leftarrow 0$
 4    $(s, t) \leftarrow$ the edge $Z(\alpha)$
 5    if $s$ and $t$ are both internal nodes of $T_1|_{A\cup B}$ then begin
 6      SET_MM($\alpha, s, \infty, 0, \infty, 0$);
        SET_MM($\alpha, t, \infty, 0, \infty, 0$)
 7    end
 8    else begin                 //either $s$ or $t$ is an internal node
 9      if $s$ is not an internal node of $T_1|_{A\cup B}$ then
10        exchange $s$ and $t$          //set $s$ as the internal node
11      if $t \in A$ then SET_MM($\alpha, s, \phi(t, s, v_2), \phi(t, s, v_2)$,
         $\infty, 0$)
12      else SET_MM($\alpha, s, \infty, 0, \phi(t, s, v_2), \phi(t, s, v_2)$)
13    end
14  end
15  for each internal node $\alpha$ of $\tau$, in a bottom-up fashion, do
16  begin
17    $(\alpha_1, \alpha_2) \leftarrow$ the children of $\alpha$
18    $c \leftarrow$ the common vertex of $Z(\alpha_1)$ and $Z(\alpha_2)$
19    $m_{12} \leftarrow \Phi(A(\alpha_1), c) \oplus_\infty \Phi(B(\alpha_2), c)$
                                              // using Lemma 7.1
20    $m_{21} \leftarrow \Phi(A(\alpha_2), c) \oplus_\infty \Phi(B(\alpha_1), c)$
                                              // using Lemma 7.1
21    $Sum(\alpha) \leftarrow \max\{Sum(\alpha_1), m_{12}, m_{21}, Sum(\alpha_2)\}$
22    for each $u \in U(\alpha)$ do begin
23      $w \leftarrow d(T_1, c, u)$
24      if $u = c$ then              // $u \in U(\alpha_1)$ and $u \in U(\alpha_2)$
25        FIND_MM($\alpha, u, \alpha_1, \alpha_2, c, c, 0, 0$)
26      else if $u \notin U(\alpha_1)$ then              //$u \in U(\alpha_2)$
27        FIND_MM($\alpha, u, \alpha_1, \alpha_2, c, u, w, 0$)
28      else FIND_MM($\alpha, u, \alpha_1, \alpha_2, u, c, 0, w$)
29    end
30  end
31  return($Sum(r)$), where $r$ is the root of $\tau$
end
```

```
Procedure SET_MM($\alpha, u, a_{\min}, a_{\max}, b_{\min}, b_{\max}$)
//set $\min_A(\alpha, u)$, $\max_A(\alpha, u)$, $\min_B(\alpha, u)$, and $\max_B(\alpha, u)$
begin
1 $(\min_A(\alpha, u), \max_A(\alpha, u)) \leftarrow (a_{\min}, a_{\max})$
2 $(\min_B(\alpha, u), \max_B(\alpha, u)) \leftarrow (b_{\min}, b_{\max})$
end
```

```
Procedure FIND_MM($\alpha, u, \alpha_1, \alpha_2, u_1, u_2, w_1, w_2$)
//find $\min_A(\alpha, u)$, $\min_B(\alpha, u)$, $\max_A(\alpha, u)$, and $\max_B(\alpha, u)$
begin
1 $\min_A(\alpha, u) \leftarrow \min\{\min_A(\alpha_1, u_1) + w_1, \min_A(\alpha_2, u_2) + w_2\}$
2 $\max_A(\alpha, u) \leftarrow \max\{\max_A(\alpha_1, u_1) + w_1, \max_A(\alpha_2, u_2) + w_2\}$
3 $\min_B(\alpha, u) \leftarrow \min\{\min_B(\alpha_1, u_1) + w_1, \min_B(\alpha_2, u_2) + w_2\}$
4 $\max_B(\alpha, u) \leftarrow \max\{\max_B(\alpha_1, u_1) + w_1, \max_B(\alpha_2, u_2) + w_2\}$
end
```

We proceed to describe how to compute $Sum_0([A \times D])$. For two sequences X and Y of numbers, define $X \oplus_0 Y$ to be $\sum_{1 \le i \le |X|, 1 \le j \le |Y|} h(X[i] + Y[j])$. Given two sorted sequences X and Y, $X \oplus_0 Y$ can be computed by modifying the algorithm in the proof of Lemma 3.3 as follows. For each $X[i]$, $1 \le i \le |X|$, let $m(i)$ be the number of elements $Y[j]$ in Y satisfying $Y[j] = -X[i]$. Then, it is not difficult to see that $X \oplus_0 Y$ can be computed as $|X||Y| - \sum_{1 \le i \le |X|} m(i)$. Since X and Y are non-decreasing, all $m(i)$ can be computed by a procedure similar to merge sort. Therefore, $X \oplus_0 Y$ can be determined in $O(|X| + |Y|)$ time.

According to Lemma 3.1, we have $Sum_0([A \times D]) = \Phi(A, v_1, v_2) \oplus_0 \Phi(D, v_1, v_2)$. Thus, the algorithm in Section 3.1 can be modified to compute $Sum_0([A \times D])$ in $O(n \log n)$ time by replacing $\oplus$ with $\oplus_0$. Similarly, the procedure COMPUTING_AB_1 in Section 3.2 can be modified to compute $Sum_0([A \times B])$ in $O(n \log n)$ time by replacing $\oplus$ with $\oplus_0$ in Lines 18 and 19. Consequently, the l_0-norm path-difference distance problem is solved in $O(n \log^2 n)$ time.

9 CONCLUSION

The focus of this paper was the l_p-norm path-difference distance problem, where $p \ge 1$ is an integer. In this paper, the l_1-norm path-difference distance problem was solved in $O(n \log^2 n)$-time and the l_2- and l_∞-norm path-difference distance problems were both solved in $O(n \log n)$ time. By extending the presented algorithms, we also solved the l_p-norm path-difference distance problem in $O(pn \log^2 n)$ time for any positive integer p. In addition, if the integer p is even, we showed that the problem can be solved in $O(p^2 n \log n)$ time as well.

An implementation of the presented algorithms for $p = 1$, 2, and ∞ is available at [24]. Experiments showed that less than 1 second is needed for $n = 10^4$.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to the anonymous referees for their careful reading and helpful suggestions which have greatly improved the quality of the paper. This research is supported by the National Science Council of the Republic of China under grant MOST-106-2221-E- 007-022-MY3.

REFERENCES

[1] E. Albright, J. Hessel, N. Hiranuma, C. Wang, and S. Goings, "A comparative analysis of popular phylogenetic reconstruction algorithms," in *Proc. Midwest Instruction Comput. Symp.*, 2014.

[2] S. Alstrup, J. Holm, K. D. Lichtenberg, and M. Thorup, "Maintaining information in fully dynamic trees with top trees," *ACM Trans. Algorithms*, vol. 1, no. 2, pp. 243–264, 2005.

[3] E. Arnaoudova, et al., "Statistical phylogenetic tree analysis using differences of means," *Frontiers Neuroscience*, vol. 4, art. 47, 2010.

[4] J. Bluis and D.-G. Shin, "Nodal distance algorithm: Calculating a phylogenetic tree comparison metric," in *Proc. 3rd IEEE Symp. Bioinf. Bioengineering*, 2003, pp. 87–94.

[5] D. Bogdanowicz, K. Giaro, and B. Wróbel, "TreeCmp: Comparison of trees in polynomial time," *Evol. Bioinf. Online*, vol. 8, pp. 475–487, 2012.

[6] G. S. Brodal, R. Fagerberg, T. Mailund, C. N. S. Pedersen, and A. Sand, "Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree," in *Proc. Annu. ACM-SIAM Symp. Discr. Algorithms*, 2013, pp. 1814–1832.

[7] G. S. Brodal, R. Fagerberg, and C. N. S. Pedersen, "Computing the quartet distance between evolutionary trees in time *O*(*n* log *n*)," *Algorithmica*, vol. 38, no. 2, pp. 377–395, 2004.

[8] D. Bryant, “A classification of consensus methods for phylogenetics,” *DIMACS Series Discr. Math. Theoretical Comput. Sci.*, vol. 61, pp. 163–184, 2003.

[9] L. L. Cavallip-Sforza and A. W. F. Edwards, “Phylogenetic analysis: Models and estimation procedures,” *Am. J. Human Genetics*, vol. 19, pp. 233–257, 1967.

[10] J. I. Coons and J. Rusinko, “A note on the path interval distance,” *J. Theoretical Biol.*, vol. 398, pp. 145–149, 2016.

[11] D. E. Critchlow, D. K. Pearl, and C. Qian, “The triples distance for rooted bifurcating phylogenetic trees,” *Systematic Biol.*, vol. 45, pp. 323–334, 1996.

[12] W. H. E. Day, “Optimal algorithm for comparing trees with labeled leaves,” *J. Classification*, vol. 2, pp. 7–28, 1985.

[13] H. N. Djidjev, G. E. Pantziou, and C. D. Zaroliagis, “Computing shortest paths and distances in planar graphs,” in *Proc. 18th Int. Colloq. Automata, Languages Program.*, 1991, pp. 327–338.

[14] R. V. Eck and M. O. Dayoff, *Atlas of Protein Sequence and Structure*. Washington, DC, USA: National Biomedical Research Foundation, 1966.

[15] G. F. Estabrook, F. R. McMorris, and C. A. Meacham, “Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units,” *Systematic Biol.*, vol. 34, no. 2, pp. 193–200, 1985.

[16] J. S. Farris, “A successive approximations approach to character weighting,” *Systematic Biol.*, vol. 18, no. 4, pp. 374–385, 1969.

[17] J. S. Farris, “On comparing the shapes of taxonomic trees,” *Systematic Zoology*, vol. 22, pp. 50–54, 1973.

[18] J. Felsenstein, “Evolutionary trees from DNA sequences: A maximum likelihood approach,” *J. Molecular Evol.*, vol. 17, pp. 368–376, 1981.

[19] J. Felsenstein, *Inferring Phylogenies*. Sunderland, Massachusetts, USA: Sinauer Associates, Inc., 2004.

[20] G. N. Frederickson and D. B. Johnson, “Finding *k*th paths and *p*-centers by generating and searching good data structures,” *J. Algorithms*, vol. 4, pp. 61–80, 1983.

[21] J. P. Huelsenbeck and M. A. Suchard, “A nonparametric method for accommodating and testing across-site rate variation,” *Systematic Biol.*, vol. 56, no. 6, pp. 975–987, 2007.

[22] P. Huggins, M. Owen, and R. Yushida, “First steps toward the geometry of cophylogeny,” in *Proc. 2nd CREST-SBM Int. Conf. “Harmony Gröbner Bases Modern Ind. Soc.*,” 2012, pp. 99–116.

[23] M. K. Kuhner and J. Yamato, “Practical performance of tree comparison metrics,” *Systematic Biol.*, vol. 64, no. 2, pp. 205–214, 2015.

[24] C.-Y. Li, FastPDD, 2017. [Online]. Available: http://venus.cs.nthu.edu.tw/~daniel/PDD.html

[25] G. Longobardi, et al., “Across language families: Genome diversity mirrors linguistic variation within Europe,” *Am. J. Phys. Anthropology*, vol. 157, pp. 630–640, 2015.

[26] A. Markin and O. Eulenstein, “Path-difference median trees,” in *Proc. 12th Int. Symp. Bioinf. Res. Appl.*, 2016, pp. 211–223.

[27] A. Markin and O. Eulenstein, “Manhattan path-difference median trees,” in *Proc. 7th ACM Int. Conf. Bioinf., Comput. Biol. Health Inform.*, 2016, pp. 404–413.

[28] D. Penny and M. D. Hendy, “The use of tree comparison metrics,” *Systematic Zoology*, vol. 34, no. 1, pp. 75–82, 1985.

[29] J. B. Phipps, “Dendrogram topology,” *Systematic Zoology*, vol. 20, pp. 306–308, 1971.

[30] P. Puigbò, S. Garcia-Vallvé, and J. O. McInerney, “TOPD/FMTS: A new software to compare phylogenetic trees,” *Bioinf.*, vol. 23, no. 12, pp. 1556–1558, 2007.

[31] D. F. Robinson and L. R. Foulds, “Comparison of phylogenetic trees,” *Math. Biosci.*, vol. 53, no. 1–2, pp. 131–147, 1981.

[32] N. Saitou and M. Nei, “The neighbor-joining method: A new method for reconstructing phylogenetic trees,” *Mol. Biol. Evol.*, vol. 4, pp. 406–425, 1987.

[33] K. P. Schliep, “Phangorn: Phylogenetic analysis in R,” *Bioinf.*, vol. 27, no. 4, pp. 592–593, 2010.

[34] M. Steel and D. Penny, “Distribution of tree comparison metrics–some new results,” *Systematic Biol.*, vol. 42, no. 2, pp. 126–141, 1993.

[35] M. S. Waterman and T. F. Smith, “On the similarity of dendrograms,” *J. Theoretical Biol.*, vol. 73, no. 4, pp. 789–800, 1978.

[36] W. T. Williams and H. T. Clifford, “On the comparison of two classifications of the same set of elements,” *Taxon*, vol. 20, no. 4, pp. 519–522, 1971.

[37] J. Xi, J. Xie, and R. Yoshida, “Distributions of topological tree metrics between a species tree and a gene tree,” *Ann. Inst. Statistical Math.*, pp. 1–25, 2015.

[38] K. A. Zaretskii, “Postroenie dereva po naboru rasstoianii mezhdu visiacimi,” *Uspekhi Matematicheskikh Nauk*, vol. 20, pp. 94–96, 1965.

Biing-Feng Wang received the BS degree in computer science from National Chiao Tung University, Taiwan, in June 1988, and the PhD degree in computer science from National Taiwan University, in June 1991. He joined the Faculty of National Tsing Hua University, in August 1993, where he is a professor in the Department of Computer Science. His current research interests include design and analysis of algorithms and parallel computation. He received the Academia Sinica’s Young Scholar Paper Award and the K.T. Li Young Researcher Award, in 1999, the ISI Citation Classic Award, in 2001, and the National Science Council’s Outstanding Research Award, in 2002. He served as the chairman of the Department of Computer Science from 2003 to 2006. He was awarded as a Tsing Hua Distinguished Professor in 2006.

Chih-Yu Li received the BS degree in computer science from National Tsing Hua University, Taiwan, in June 2014. Since September 2016, he has been working toward the PhD degree in computer science at National Tsing Hua University. His current research interests include design and analysis of algorithms and computational biology.