FPT-Algorithms for computing Gromov-Hausdorff and interleaving distances between trees

Elena Farahbkhsh Touli[*] Yusu Wang[†]

July 16, 2019

Abstract

The Gromov-Hausdorff distance is a natural way to measure the distortion between two metric spaces. However, there has been only limited algorithmic development to compute or approximate this distance. We focus on computing the Gromov-Hausdorff distance between two metric trees. Roughly speaking, a metric tree is a metric space that can be realized by the shortest path metric on a tree. Any finite tree with positive edge weight can be viewed as a metric tree where the weight is treated as edge length and the metric is the induced shortest path metric in the tree. Previously, Agarwal et al. showed that even for trees with unit edge length, it is NP-hard to approximate the Gromov-Hausdorff distance between them within a factor of 3. In this paper, we present a fixed-parameter tractable (FPT) algorithm that can approximate the Gromov-Hausdorff distance between two general metric trees within a *multiplicative factor* of 14.

Interestingly, the development of our algorithm is made possible by a connection between the Gromov-Hausdorff distance for metric trees and the interleaving distance for the so-called merge trees. The merge trees arise in practice naturally as a simple yet meaningful topological summary (it is a variant of the Reeb graphs and contour trees), and are of independent interest. It turns out that an exact or approximation algorithm for the interleaving distance leads to an approximation algorithm for the Gromov-Hausdorff distance. One of the key contributions of our work is that we re-define the interleaving distance in a way that makes it easier to develop dynamic programming approaches to compute it. We then present a fixed-parameter tractable algorithm to compute the interleaving distance between two merge trees **exactly**, which ultimately leads to an FPT-algorithm to approximate the Gromov-Hausdorff distance between two metric trees. This exact FPT-algorithm to compute the interleaving distance between merge trees is of interest itself, as it is known that it is NP-hard to approximate it within a factor of 3, and previously the best known algorithm has an approximation factor of $O(\sqrt{n})$ even for trees with unit edge length.

[*]Department of Mathematics, Stockholm University

[†]Department of Computer Science, Ohio State University, email: yusu@cse.ohio-state.edu

1 Introduction

Given two metric spaces (X, d_X) and (Y, d_Y), a natural way to measure their distance is via the *Gromov-Hausdorff distance* $\delta_{\mathcal{GH}}(X, Y)$ between them [15], which intuitively describes how much *additive* distance distortion is needed to make the two metric spaces isometric.

We are interested in computing the Gromov-Hausdorff distance between *metric trees*. Roughly speaking, a metric tree (X, d) is a geodesic-metric space that can be realized by the shortest path metric on a tree. Any finite tree $T = (V, E)$ with positive edge weights $w : E \to \mathbb{R}$ can be naturally viewed as a metric tree $\mathcal{T} = (|T|, d)$: the space is the underlying space $|T|$ of T, each edge e can be viewed as a segment of length $w(e)$, and the distance d is the induced shortest path metric. See Figure 1 (a) for an example. Metric trees occur commonly in practical applications: e.g., a neuron cell has a tree morphology, and can be modeled as an embedded metric tree in $\mathbb{R}^3$. It also represents an important family of metric spaces that has for example attracted much attention in the literature of metric embedding and recovery of hierarchical structures, e.g., [2, 4, 3, 10, 11, 13, 23].

Unfortunately, it is shown in [1, 22] that it is not only NP-hard to compute the Gromov-Hausdorff distance between two trees, but also NP-hard to approximate it within a factor of 3 even for trees with unit edge length. A polynomial-time approximation algorithm is given in [1]; however, the approximation factor is high: it is $O(\sqrt{n})$ even for unit-edge weight trees.

Another family of tree structures that is of practical interest is the so-called *merge tree*. Intuitively, a merge tree is a rooted tree T associated with a real-valued function $f : T \to \mathbb{R}$ such that the function value is monotonically decreasing along any root-to-leaf path – We can think of a merge tree to be a tree with height function associated to it where all nodes with degree > 2 are down-forks (merging nodes); see Figure 1 (b). The merge tree is a loop-free variant of the so-called Reeb graph, which is a simple yet meaningful topological summary for a scalar field $g : X \to \mathbb{R}$ defined on a domain X, and has been widely used in many applications in graphics and visualization e.g., [7, 14, 17, 24]. Morozov et al. introduced the *interleaving distance* to compare merge trees [20], based on a natural "interleaving idea" which has recently become fundamental in comparing various topological objects. Also, several distance measures have been proposed for the Reeb graphs [5, 6, 12]. When applying them to merge trees, it turns out that two of these distance measures are equivalent to the interleaving distance. However, the same reduction in [1] to show the hardness of approximating the Gromov-Hausdorff distance can also be used to show that it is NP-hard to approximate the interleaving distance between two merge trees within a factor 3.

New work. Although the Gromov-Hausdorff distance is a natural way to measure the degree of near-isometry between metric spaces [15, 19], the algorithmic development for it has been very limited so far [1, 9, 21, 22]. In [22], Schmiedl gave an FPT algorithm for approximating the Gromov-Hausdorff distance between two *finite metrics*, where the approximation contains *both an additive and multiplicative terms*; see more discussion in *Remarks* after Theorem 6. In this paper, we present the first FPT algorithm to approximate the Gromov-Hausdorff distance for metric trees *within a constant multiplicative factor*.

Interestingly, the development of our approximation algorithm is made possible via a connection between the Gromov-Hausdorff distance between metric trees and the interleaving distance between certain merge trees (which has already been observed previously in [1]). This connection implies that any exact or approximation algorithm for the interleaving distance will lead to an approximation algorithm for the Gromov-Hausdorff distance for metric trees of similar time complexity. Hence we can focus on developing algorithms for the interleaving distance. The original interleaving distance definition requires considering a pair of maps between the two input merge trees and their interaction. One of the key insights of our work is that we can in fact develop an equivalent definition for the interleaving distance that relies on only *a single map* from one tree to the other. This, together with the height functions equipped with merge trees (which give rises to natural ordering between points in the two trees), essentially allows us to develop a dynamic programming algorithm to check whether the interleaving distance between two merge trees is smaller than a given threshold or not: In particular in Section 4, we first give a simpler DP algorithm with slower time complexity to illustrate the main idea. We then show how we can modify this DP algorithm to improve the time complexity. Finally, we solve the optimization problem for computing the interleaving distance[1] in Section 5, which leads to a constant-factor (a multiplicative factor of 14) approximation FPT algorithm for the Gromov-Hausdorff distance between metric trees.

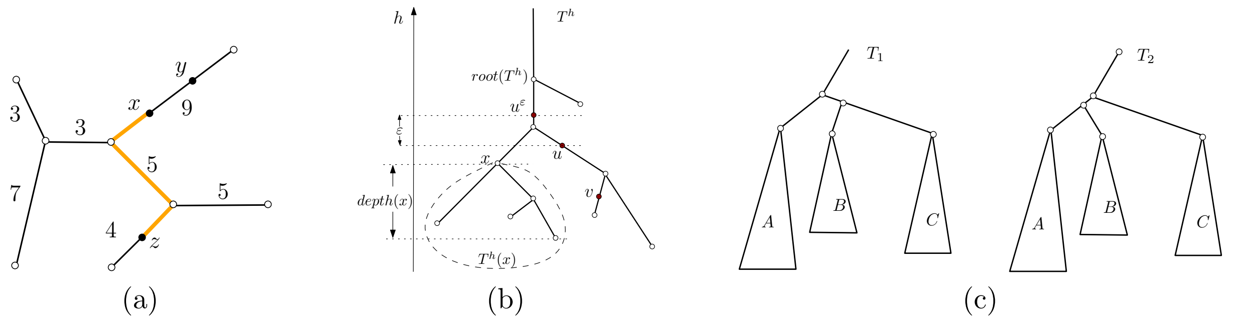


Figure 1: (a) A metric tree (T, d_T) with edge length marked. Tree nodes are white dots. $d_T(x, z) = 3 + 5 + 2 = 10$ is the length of the thickened path $\pi(x, z)$. (b) A merge tree T^h, with examples of $u \succ v$, u^ε, $T^h(x)$ and $depth(x)$ marked. (c) Tree alignment distance between T_1 and T_2 arbitrarily large, while $\delta_{\mathcal{GH}}(T_1, T_2)$ is roughly bounded by the pairwise distance difference which is small.

More on related work. There have been several tree distances proposed in the literature. Two most well-known ones are the tree edit and tree alignment distances [8], primarily developed to compare labeled trees. Unfortunately, both distances are MAX SNP-hard to compute for unordered trees [18, 25]. For tree edit distance, it is MAX SNP-hard even for trees with bounded degree. For the tree alignment distance, it can be computed in polynomial time for trees with bounded degree. However the tree alignment distance requires that parent-child relation to be strictly preserved, and thus the small local configuration change shown in Figure 1 (c) will incur a large tree alignment distance.

We will not survey the large literature in metric embedding which typically minimizes the metric distortion in a *mulplicative* manner. However, we mention the work of Hall and Papadimitriou [16], where, given two equal-sized point sets, they propose to find the best *bijection* under which the *additive distortion* is minimized. They show it is NP-hard to approximate this optimal additive distortion within a factor of 3 even for points in $\mathbb{R}^3$. In contrast, the Gromov-Hausdorff distance is also *additive*, but allows for many-to-many correspondence (instead of bijection) between points from two input metric spaces. We also note that our metric trees consist of all points in the underlying space of input trees (i.e, including points in the interior of a tree edge). This makes the distance robust against adding extra nodes and short "hairs" (branches). Nevertheless, we can

[1]We note that the final time complexity for the optimization problem presented in Theorem 5 is based on an argument by Kyle Fox. His argument improves our previous n^4 factor (as in Theorem 4) by an almost n^2 factor, by performing a double-binary search, instead of a sequence search we originally used.

also consider discrete metric trees, where we only aim to align nodes of input trees (instead of all points in the underlying space of the trees). Our algorithms hold for the discrete case as well.

2 Preliminaries

Metric space, metric trees. A metric space is a pair (X, d) where X is a set and $d : X \times X \to \mathbb{R}_{\geq 0}$ satisfies: (i) for any $x, y \in X$, $d(x, y) \geq 0$ and $d(x, y) = 0$ holds only when $x = y$; (ii) $d(x, y) = d(y, x)$, and (iii) for any x, y, z, $d(x, z) \leq d(x, y) + d(y, z)$. We call d a metric on the space X. A metric space (X, d) is a finite metric tree if it is a length metric space[2] and X is homeomorphic to the underlying space $|T|$ of some finite tree $T = (V, E)$.

Equivalently, suppose we are given a finite tree $T = (V, E)$ where each edge $e \in E$ has a positive weight $\ell(e) > 0$. View the underlying space $|e|$ of e as a segment with length $\ell(e)$ (i.e, it is isometric to $[0, \ell(e)]$), and we can thus define the distance $d_T(x, y)$ between any two points $x, y \in |e|$ as the length of the sub-segment $e[x, y]$. The underlying space $|T|$ of T is the union of all these segments (and thus includes points in the interior of each edge as well). For any $x, z \in |T|$, there is a unique simple path $\pi(x, z) \subset |T|$ connecting them. The (shortest path) distance $d_T(x, z)$ equal to the length of this path, which is simply the sum of the lengths of the restrictions of this path to edges in T. See Figure 1 (a). The space $|T|$ equipped with d_T is a metric tree $(|T|, d_T)$.

Given a tree $T = (V, E)$, we use the term *tree nodes* to refer to points in V, and an arbitrary $x \in |T|$ potentially from the interior of some tree edge is referred to as a *point*. Given T, we also use $V(T)$ and $E(T)$ to denote its node-set and edge-set, respectively. To emphasize the combinatorial structure behind a metric tree, in the paper we will write a metric tree (T, d_T), with the understanding that the space is in fact the underlying space $|T|$ of T.

Note that if we restrict this metric space to only the tree nodes, we obtain a *discrete metric tree* $(V(T), d_T)$, and the distance between two tree nodes is simply the standard shortest path distance between them in a weighted graph (tree T in this case). Our algorithms developed in this paper can be made to work for the discrete metric trees as well.

Gromov-Hausdorff distance. Given two metric spaces $\mathcal{X} = (X, d_X)$ and $\mathcal{Y} = (Y, d_Y)$, a *correspondence* between them is a relation $C : X \times Y$ whose projection on X and on Y are both surjective; i.e, for any $x \in X$, there is at least one $(x, y) \in C$, and for any $y' \in Y$, there is at least one $(x', y') \in C$. If $(x, y) \in C$, then we say y (resp. x) is a pairing partner for x (resp. y); note that x (resp. y) could have multiple pairing partners. The cost of this correspondence is defined as:

$$\mathrm{cost}(C) = \max_{(x,y),(x',y') \in C} |d_X(x, x') - d_Y(y, y')|,$$

which measures the maximum metric distortion (difference in pairwise distances) under this correspondence. The *Gromov-Hausdorff distance* between them is:

$$\delta_{\mathcal{GH}}(\mathcal{X}, \mathcal{Y}) = \frac{1}{2} \inf_{C \in \Pi(X,Y)} \mathrm{cost}(C), \quad \text{where } \Pi(X, Y) = \text{set of correspondences between } X \text{ and } Y.$$

Merge trees. A *merge tree* is a pair (T, h) where T is a rooted tree, and the continuous function $h : |T| \to \mathbb{R}$ is *monotone* in the sense the value of h is decreasing along any root-to-leaf path. See

[2] (X, d) is a length metric space if d is the same as the shortest path (i.e, intrinsic) metric it induces on X.

Figure 1 (b) for an example. For simplicity, we often write the merge tree as T^h, and refer to h as the *height function*, and $h(x)$ *the height* of a point $x \in |T|$. The merge tree is an natural object: e.g., it can be used to model a hierarchical clustering tree, where the height of a tree node indicates the parameter when the cluster (corresponding to the subtree rooted at this node) is formed. It also arises as a simple topological summary of a scalar function $\tilde{h} : M \to \mathbb{R}$ on a domain M, which tracks the connected component information of the sub-level sets $\tilde{h}^{-1}(-\infty, a]$ as $a \in \mathbb{R}$ increases.

To define the interleaving distance, we modify a merge tree T^h slightly by extending a ray from $root(T^h)$ upwards with function value h goes to $+\infty$. All merge trees from now on refer to this modified version. Given a merge tree T^h and a point $x \in |T|$, $T^h(x)$ is the subtree of T^h rooted at x, and the *depth of x (or of $T^h(x)$)*, denoted by $depth(x)$, is the largest function value difference between x and any node in its subtree; that is, the height of the entire subtree $T^h(x)$ w.r.t. function h. Given any two points $u, v \in |T|$, we use $u \succeq v$ to denote that u is an ancestor of v; $u \succ v$ if u is an ancestor of v and $u \neq v$. Similarly, $v \preceq u$ means that v is a descendant of u. Also, the degree of a node in a merge tree is defined as the downward degree of the node. We use $LCA(u, v)$ to represent the lowest common ancestor of u and v in $|T|$. For any non-negative value $\varepsilon \geq 0$, u^ε represents the unique ancestor of u in T such that $h(u^\varepsilon) - h(u) = \varepsilon$. See Figure 1 (b).

Interleaving distance. We now define the interleaving distance between two merge trees T_1^f and T_2^g, associated with functions $f : |T_1^f| \to \mathbb{R}$ and $g : |T_2^g| \to \mathbb{R}$, respectively.

Definition 1 (ε-Compatible maps [20])**.** *A pair of continuous maps $\alpha : |T_1^f| \to |T_2^g|$ and $\beta : |T_2^g| \to |T_1^f|$ is* ε-compatible w.r.t T_1^f and T_2^g *if the following four conditions hold:*

$$(\mathsf{C1}).\ g(\alpha(u)) = f(u) + \varepsilon \ \ \textit{and} \ \ (\mathsf{C2}).\ \beta \circ \alpha(u) = u^{2\varepsilon} \ \ \textit{for any } u \in |T_1^f|;$$
$$(\mathsf{C3}).\ f(\beta(w)) = g(w) + \varepsilon \ \ \textit{and} \ \ (\mathsf{C4}).\ \alpha \circ \beta(w) = w^{2\varepsilon} \ \ \textit{for any } w \in |T_2^g|.$$

To provide some intuition for this definition, note that if $\varepsilon = 0$, then $\alpha = \beta^{-1}$: In this case, the two trees T_1 and T_2 are not only isomorphic, but also the function values associated to them are preserved under the isomorphism. In general for $\varepsilon > 0$, this quantity measures how far a pair of maps are away from forming a function-preserving isomorphism between T_1^f and T_2^g. In particular, β is no longer the inverse of α. However, the two maps relate to each other in the sense that if we send a point $u \in |T_1^f|$ to $|T_2^g|$ through $\alpha : |T_1^f| \to |T_2^g|$, then bring it back via $\beta : |T_2^g| \to |T_1^f|$, we come back at an ancestor of u in $|T_1^f|$ (i.e, property (C2)). This ancestor must be at height $f(u) + 2\varepsilon$ due to properties (C1) and (C3).

Definition 2 (Interleaving distance [20])**.** *The* interleaving distance *between two merge trees T_1^f and T_2^g is defined as:*

$$d_I(T_1^f, T_2^f) = \inf\{\ \varepsilon \ \mid \ \textit{there exist a pair of } \varepsilon\textit{-compatible maps w.r.t } T_1^f \textit{ and } T_2^g\}.$$

Interestingly, it is shown in [1] that the Gromov-Hausdorff distance between two metric trees is related to the interleaving distance between two specific merge trees.

Claim 1 ([1])**.** *Given two metric trees $\mathcal{T}_1 = (T_1, d_1)$ and $\mathcal{T}_2 = (T_2, d_2)$ with node sets $V_1 = V(T_1)$ and $V_2 = V(T_2)$, respectively, let $f_u : |T_1| \to \mathbb{R}$ (resp. $g_w : |T_2| \to \mathbb{R}$) denote the geodesic distance function to the base point $u \in V_1$ (resp. $v \in V_2$) defined by $f_u(x) = -d_1(x, u)$ for any $x \in |T_1|$ (resp. $g_w(y) = -d_2(y, w)$ for any $y \in |T_2|$). Set $\mu := \min_{u \in V_1, w \in V_2} d_I(T_1^{f_u}, T_2^{g_w})$. We then have that*

$$\frac{\mu}{14} \leq \delta_{\mathcal{GH}}(\mathcal{T}_1, \mathcal{T}_2) \leq 2\mu.$$

Note that to compute the quantity μ, we only need to check all pairs of *tree nodes* of T_1 and T_2, instead of all pairs of points from $|T_1|$ and $|T_2|$.

We say a quantity A is a *c-approximation* for a quantity B if $\frac{A}{c} \leq B \leq cA$; obviously, $c \geq 1$ and $c = 1$ means that $A = B$. The above claim immediately suggests the following:

Corollary 1. *If there is an algorithm to c-approximate the interleaving distance between any two merge trees in $T(n)$ time, where n is the total complexity of input trees, then there is an algorithm to $O(c)$-approximate the Gromov-Hausdorff distance between two metric trees in $n^2T(n)$ time.*

In the remainder of this paper, we will focus on developing an algorithm to compute the interleaving distance between two merge trees T_1^f and T_2^g. In particular, in Section 3 we will first show an equivalent definition for interleaving distance, which has a nice structure that helps us to develop a fixed-parameter tractable algorithm for the decision problem of "Is $d_I(T_1^f, T_2^g) \geq \varepsilon$?" in Section 4. We show how this ultimately leads to FPT algorithms to compute the interleaving distance **exactly** and to **approximate** the Gromov-Hausdorff distance in Section 5.

3 A New Definition for Interleaving Distance

Given two merge trees T_1^f and T_2^g and $\delta > 0$, to answer the question "Is $d_I(T_1^f, T_2^g) \leq \delta$?", a natural idea is to scan the two trees bottom up w.r.t the "height" values (i.e, f and g), while checking for possible ε-compatible maps between the partial forests of T_1^f and T_2^g already scanned. However, the interaction between the pair maps α and β makes it complicated to maintain potential maps. We now show that in fact, we only need to check for the existence of a *single* map from T_1^f to T_2^g, which we will call the ε-good map. We believe that this result is of independent interest.

Definition 3 (ε-good map)**.** *A continuous map $\alpha : |T_1^f| \to |T_2^g|$ is ε-good if and only if:*

(P1) *for any $u \in |T_1^f|$, we have $g(\alpha(u)) = f(u) + \varepsilon$;*

(P2) *if $\alpha(u_1) \succeq \alpha(u_2)$, then we have $u_1^{2\varepsilon} \succeq u_2^{2\varepsilon}$, (note $u_1 \succeq u_2$ may not be true); and*

(P3) *if $w \in |T_2^g| \setminus \mathrm{Im}(\alpha)$, then we have $|g(w^F) - g(w)| \leq 2\varepsilon$, where $\mathrm{Im}(\alpha) \subseteq |T_2^g|$ is the image of α, and w^F is the lowest ancestor of w in $\mathrm{Im}(\alpha)$.*

A map $\rho : |T_1^{h_1}| \to |T_2^{h_2}|$ between two arbitrary merge trees $T_1^{h_1}$ and $T_2^{h_2}$ is *monotone* if for any $u \in |T_1^{h_1}|$, we have that $h_2(\rho(u)) \geq h_1(u)$. In other word, ρ carries any point u from $T_1^{h_1}$ to a point higher than it in $T_2^{h_2}$. If ρ is continuous, then it will map an ancestor of u in $T_1^{h_1}$ to an ancestor of $\rho(u)$ in $T_2^{h_2}$ as stated below (but the converse is not necessarily true):

Observation 1. *Given a* continuous and monotone *map $\rho : |T_1^{h_1}| \to |T_2^{h_2}|$ between two merge trees $T_1^{h_1}$ and $T_2^{h_2}$, we have that if $u_1 \succeq u_2$ in $T_1^{h_1}$, then $\rho(u_1) \succeq \rho(u_2)$ in $T_2^{h_2}$.*

This implies that if $w = \rho(u)$ for $u \in |T_1^{h_1}|$, then ρ maps the subtree $T_1^{h_1}(u)$ rooted at u into the subtree $T_2^{h_2}(w)$ rooted at w. This also implies that if $w \notin \mathrm{Im}(\rho)$, neither does any of its descendant.

Note that an ε-good map, or a pair of ε-compatible maps, are all monotone and continuous. Hence the above observations are applicable to all these maps.

The main result of this section is as follows. Its proof is in Appendix A.

Theorem 1. *Given any two merge trees T_1^f and T_2^g, then $d_I(T_1^f, T_2^g) \leq \varepsilon$ if and only if there exists an ε-good map $\alpha : |T_1^f| \to |T_2^g|$.*

4 Decision Problem for Interleaving Distance

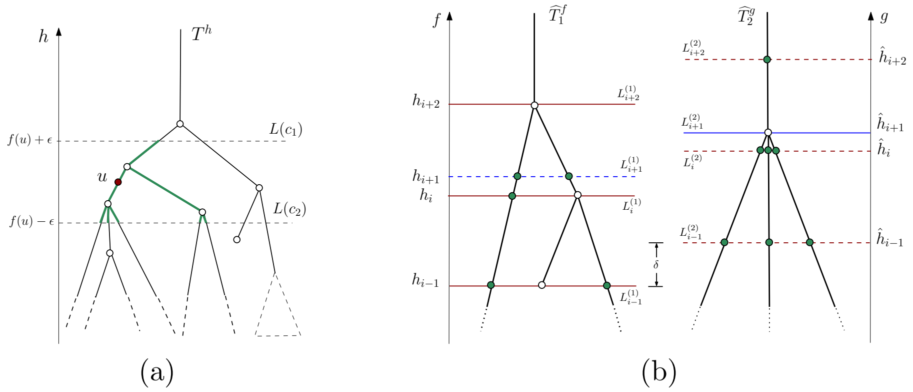


Figure 2: (a) Green component within the slab is $B_\varepsilon(u, T_1^f)$. The sum of degrees for nodes within this ε-ball is 13. The ε-degree bound $\tau_\varepsilon(T_1^f, T_2^g)$ is the largest value of this sum for any ε-ball in T_1^f or in T_2^g. (b) White dots are tree nodes of T_1^f and T_2^g. Green dots are newly augmented tree nodes in $\widehat{T}_1^f$ and $\widehat{T}_2^g$.

In this section, given two merge trees T_1^f and T_2^g as well as a positive value $\delta > 0$, we aim to develop a fixed-parameter tractable algorithm for the decision problem "Is $d_I(T_1^f, T_2^g) \leq \delta$?". The specific parameter our algorithm uses is the following: Given a merge tree T^h and any point $u \in T^h$, let $B_\varepsilon(u; T^h)$ denote the ε-ball

$$B_\varepsilon(u; T^h) = \{x \in |T| \mid \forall y \in \pi_T(u, x), |h(y) - h(u)| \leq \varepsilon\},$$

where $\pi_T(u, x)$ is the unique path from u to x in T^h. In other words, $B_\varepsilon(u; T^h)$ contains all points reachable from u via a path whose function value is completely contained with the range $[f(u) - \varepsilon, f(u) + \varepsilon]$. See Figure 2 (a) for an example: in particular, consider the restriction of T^h within the height interval $[f(u) - \varepsilon, f(u) + \varepsilon]$. There could be multiple components within this slab, and $B_\varepsilon(u; T^h)$ is the component containing u.

Parameter τ_δ : Let $\tau_\varepsilon(T_1^f, T_2^g)$ denote the largest sum of degrees of all tree nodes contained in any ε-ball in T_1^f or T_2^g, which we also refer to as the *ε-degree-bound* of T_1^f and T_2^g. The parameter for our algorithm for the decision problem will be $\tau_\delta = \tau_\delta(T_1^f, T_2^g)$.

4.1 A Slower FPT-Algorithm

Augmented trees. We now develop an algorithm for the decision problem "Is $d_I(T_1^f, T_2^g) \leq \delta$?" via a dynamic programming type approach. First, we will show that, even though a δ-good map is defined for all (infinite number of) points from T_1^f and T_2^g, we can check for its existence by inspecting only a finite number of points from T_1^f and T_2^g. In particular, we will augment the input merge trees T_1^f and T_2^g with extra tree nodes, and our algorithm later only needs to consider the discrete nodes in these augmented trees to answer the decision problem.

The set of points from tree T_1^f or T_2^g at a certain height value c is called a *level (at height c)*, denoted by $L(c)$. For example, in Figure 2 (a), the level $L(c_1)$ for $c_1 = f(u) + \varepsilon$, contains 2 points, while $L(c_2)$ with $c_2 = f(u) - \varepsilon$ contains 7 points. The function value of a level L is called its *height*, denoted by *height*(L); so *height*$(L(c)) = c$.

Definition 4 (Critical-heights and Super-levels)**.** *For the tree* T_1^f*, the* set of critical-heights C_1 *consists of the function values of all tree nodes of* T_1^f*; similarly, define* C_2 *for* T_2^g*. That is,*

$$\mathrm{C}_1 := \{f(x) \mid x \textit{ is a tree node of } T_1^f\}; \textit{ and } \mathrm{C}_2 := \{g(y) \mid y \textit{ is a tree node of } T_2^g\}.$$

The set of super-levels $\mathcal{L}_1$ w.r.t. δ *for* T_1^f *and the* set of super-levels $\mathcal{L}_2$ *for* T_2^g *are:*

$$\mathcal{L}_1 := \{L(c) \mid c \in \mathrm{C}_1\} \cup \{L(c-\delta) \mid c \in \mathrm{C}_2\} \textit{ while}$$
$$\mathcal{L}_2 := \{L(c+\delta) \mid c \in \mathrm{C}_1\} \cup \{L(c) \mid c \in \mathrm{C}_2\}.$$

Now sort all levels in $\mathcal{L}_i$ in increasing order of their heights, denoted by $\mathcal{L}_1 = \{\mathrm{L}_1^{(1)}, \mathrm{L}_2^{(1)}, \dots, \mathrm{L}_m^{(1)}\}$ and $\mathcal{L}_2 = \{\mathrm{L}_1^{(2)}, \dots, \mathrm{L}_m^{(2)}\}$, respectively. The *child-level* of super-level $\mathrm{L}_i^{(1)}$ (resp. $\mathrm{L}_i^{(2)}$) is $\mathrm{L}_{i-1}^{(1)}$ (resp. $\mathrm{L}_{i-1}^{(2)}$) for any $i \in [2, m]$; symmetrically, $\mathrm{L}_i^{(1)}$ (resp. $\mathrm{L}_i^{(2)}$) is the *parent-level* of $\mathrm{L}_{i-1}^{(1)}$ (resp. $\mathrm{L}_{i-1}^{(2)}$). Let $h_1, \dots, h_m$ be the sequence of height values for $\mathrm{L}_1^{(1)}, \mathrm{L}_2^{(1)}, \dots, \mathrm{L}_m^{(1)}$; that is, $h_i = \mathit{height}(\mathrm{L}_i^{(1)})$. Similarly, let $\widehat{h}_1, \widehat{h}_2, \dots, \widehat{h}_m$ be the corresponding sequence for $\mathrm{L}_i^{(2)}$'s.

Note that there is a one-to-one correspondence between super-levels in $\mathcal{L}_1$ and $\mathcal{L}_2$: specifically, for any $i \in [1, m]$, we have $\widehat{h}_i = h_i + \delta$. From now on, when we refer to the i-th super-levels of $\widehat{T}_1^f$ and $\widehat{T}_2^g$, we mean super-levels $\mathrm{L}_i^{(1)}$ and $\mathrm{L}_i^{(2)}$. Also observe that there is no tree node in between any two consecutive super-levels in either T_1^f or in T_2^g (all tree nodes are from some super-levels). See Figure 2 (b) for an illustration.

Next, we augment the tree T_1^f (resp. T_2^g) to add points from all super-levels from $\mathcal{L}_1$ (resp. from $\mathcal{L}_2$) also as *tree nodes*. The resulting *augmented trees* are denoted by $\widehat{T}_1^f$ and $\widehat{T}_2^g$ respectively; obviously, $\widehat{T}_1^f$ (resp. $\widehat{T}_2^g$) has isomorphic underlying space as T_1^f (resp. T_2^g), just with additional degree-2 tree nodes. In particular, $V(\widehat{T}_1^f)$ (resp. $V(\widehat{T}_2^g)$) is formed by all points from all super-levels in $\mathcal{L}_1$ (resp. $\mathcal{L}_2$). See Figure 2 (b): In this figure, solid horizontal lines indicate levels passing through critical heights, while dashed ones are induced by critical height from the other tree. In what follows, given a super-level L, we use $V(L)$ to denote the set of nodes from this level. Note that $V(\mathrm{L}_m^{(1)})$ and $V(\mathrm{L}_m^{(2)})$ each contain only one node, which is $\mathit{root}(\widehat{T}_1^f)$ and $\mathit{root}(\widehat{T}_2^g)$ respectively. Given a node v from $\mathrm{L}_i^{(1)}$ (resp. $\mathrm{L}_i^{(2)}$), let $\mathrm{Ch}(v)$ denote its children nodes in the augmented tree. Each child node of v must be from level $\mathrm{L}_{i-1}^{(1)}$ (resp. $\mathrm{L}_{i-1}^{(2)}$), as there are no tree-nodes between two consecutive super-levels.

Definition 5 (Valid pair)**.** *Given a node* $w \in V(\widehat{T}_2^g)$ *and a collection of nodes* $S \subseteq V(\widehat{T}_1^f)$*, we say that* (S, w) *form a* valid pair *if there exists an index* $j \in [1, m]$ *such that (1)* $S \subseteq V(\mathrm{L}_j^{(1)})$ *and* $w \in V(\mathrm{L}_j^{(2)})$ *(which implies that nodes in* S *at height* h_j *while* w *has height* $g(w) = \widehat{h}_j$*); and (2) all nodes in* S *have the same ancestor at height* $h_j + 2\delta$ *(which also equals* $\widehat{h}_j + \delta$*). Intuitively, it indicates that* S *has the basic condition to be mapped to* w *under some* ε*-good maps.*

We say that S *is* valid *if it participates some valid pair (and thus condition (2) above holds).*

A first (slower) dynamic programming algorithm. We now describe our dynamic programming algorithm. To illustrate the main idea, we first describe a much cleaner but also slower dynamic programming algorithm DPgoodmap() below. Later in Section 4.2 we modify this algorithm to improve its time complexity (which requires significant additional technical details).

Our algorithm maintains a certain quantity, called *feasibility* $F(S,w)$ for valid pairs in a bottom-up manner. Recall that we have defined the depth of a node $u \in T^h$ in a merge tree T^h as the height of the subtree $T^h(u)$ rooted at u; or equivalently $depth(u) = \max_{x \preceq u} |h(u) - h(x)|$.

Algorithm $\mathsf{DPgoodmap}(T_1^f, T_2^g, \delta)$**:**

Base case ($i = 1$)**:** For each valid-pair (S,w) from level-1, set $F(S,w) = 1$ ("true") if and only if $depth(w) \le 2\delta$; otherwise, set $F(S,w) = 0$ ("false").

When $i > 1$**:** Suppose we have already computed the feasibility values for all valid-pairs from level-$(i-1)$ or lower. Now for any valid-pair (S,w) from level-i, we set $F(S,w) = 1$ if and only if the following holds: Consider the set of children $\mathrm{Ch}(S) \subseteq \mathrm{L}_{i-1}^{(1)}$ of nodes in S, and w's children $\mathrm{Ch}(w) = \{w_1, \ldots, w_k\}$ in $\mathrm{L}_{i-1}^{(2)}$.

If $\mathrm{Ch}(w)$ is empty, then $F(S,w) = 1$ *only if* $\mathrm{Ch}(S)$ is also empty; otherwise $F(S,w) = 0$.

If $\mathrm{Ch}(w)$ is not empty, then we set $F(S,w)$=1 if there exists a partition of $\mathrm{Ch}(S) = S_1 \cup S_2 \cup \ldots \cup S_k$ (where $S_i \cap S_j = \emptyset$ for $i \neq j$, and it is possible that $S_i = \emptyset$) such that for each $j \in [1,k]$,

(F-1) if $S_j \neq \emptyset$, then $F(S_j, w_j) = 1$; and

(F-2) if $S_j = \emptyset$, then $depth(w_j) \le 2\delta - (\widehat{h}_i - \widehat{h}_{i-1})$; note that this implies that $\widehat{h}_i - \widehat{h}_{i-1} \le 2\delta$ in this case.

Output: $\mathsf{DPgoodmap}(T_1^f, T_2^g, \delta)$ returns "yes" if and only if $F(root(\widehat{T}_1^f), root(\widehat{T}_2^g)) = 1$.

Recall that $root(\widehat{T}_1^f)$ (resp. $root(\widehat{T}_2^g)$) is the only node in $V(\mathrm{L}_m^{(1)})$ (resp. $V(\mathrm{L}_m^{(2)})$).

We will first prove the following theorem for this slower. In Section 4.2 we show that time complexity can be reduced by almost a factor of n.

Theorem 2. *(i) Algorithm* $\mathsf{DPgoodmap}$*(T_1^f, T_2^g, δ) returns "yes" if and only if $d_I(T_1^f, T_2^g) \le \delta$.*

(ii) Algorithm $\mathsf{DPgoodmap}$*(T_1^f, T_2^g, δ) can be implemented to run in $O(n^3 2^\tau \tau^{\tau+1})$ time, where n is the total size of T_1^f, T_2^g, and $\tau = \tau_\delta(T_1^f, T_2^g)$ is the δ-degree-bound w.r.t. T_1^f and T_2^g.*

Note that if τ is constant, then the time complexity is $O(n^3)$.

In the remainder of this section, we sketch the proof of Theorem 2.

Part (i) of Theorem 2: correctness. We first show the correctness of algorithm $\mathsf{DPgoodmap}()$. Give a subset of nodes S' from some super-level of $\widehat{T}_1^f$, let $\mathcal{F}_1(S')$ denote the forest consisting of all subtrees rooted at nodes in S'. For a node $w' \in T_2^g$, let $T_2(w')$ denote the subtree of T_2^g rooted at w'. We will now argue that $F(S,w) = 1$ if and only if there is a "partial" δ-good map from $\mathcal{F}_1(S) \to T_2(w)$.

More precisely: a continuous map $\alpha : \mathcal{F}_1(S) \to T_2(w)$ with (S,w) being valid is a *partial-ε-good*map, if properties (P1), (P2), and (P3) from Definition 3 hold (with T_1^f replaced by $\mathcal{F}_1(S)$ and T_2^g replaced by $T_2(w)$). Note that in the case of (P2), the condition in (P2) only needs to hold for $u_1, u_2 \in \mathcal{F}_1(S)$ (implying that $\alpha(u_1), \alpha(u_2) \in T_2(w)$); that is, if $\alpha(u_1) \succeq \alpha(u_2)$ for $u_1, u_2 \in \mathcal{F}_1(S)$, then, we have $u_1^{2\varepsilon} \succeq u_2^{2\varepsilon}$. Note that while u_1 and u_2 are from $\mathcal{F}_1(S)$, $u_1^{2\varepsilon}$ and $u_2^{2\varepsilon}$ may not be in $\mathcal{F}_1(S)$ as it is possible that $f(u_1^{2\varepsilon}) = f(u_1) + 2\varepsilon \ge height(S)$. First, we observe the following:

Claim 2. *At the top level where* $\mathrm{L}_m^{(1)} = \{u = root(\widehat{T}_1^f)\}$ *and* $\mathrm{L}_m^{(2)} = \{w = root(\widehat{T}_2^g)\}$ *both contain only one node, if there is a partial-δ-good map from* $\mathcal{F}_1(\{u\}) \to T_2(w)$*, then there is a δ-good map from* $|T_1^f| \to |T_2^g|$.

The correctness of our dynamic programming algorithm (part (ii) of Theorem 2) will follow from Claim 2 and Lemma 1 below. Lemma 1 is one of our key techincal results, and its proof can be found in Appendix B.

Lemma 1. *For any valid pair* (S, w)*,* $F(S, w) = 1$ *if and only if there is a partial-δ-good map* $\alpha : \mathcal{F}_1(S) \to T_2(w)$*.*

Part (ii) of Theorem 2: time complexity. We now show that Algorithm DPgoodmap() can be implemented to run in the claimed time. Note that the augmented-tree nodes contain tree nodes of T_1^f and T_2^g, as well as the intersection points between tree arcs of T_1^f (resp. T_2^g) and super-levels. As there are at most $m = 2n$ number of super-levels in $\mathcal{L}_1$ and $\mathcal{L}_2$, it follows that the total number of tree nodes in the augmented trees $\widehat{T}_1^f$ and $\widehat{T}_2^g$ is bounded by $O(nm) = O(n^2)$. In what follows, in order to distinguish between the tree nodes for the augmented trees ($\widehat{T}_1^f$ and $\widehat{T}_2^g$) from the tree nodes of the original trees (T_1^f and T_2^f), we refer to nodes of the former as *augmented-tree nodes*, while the latter simply as *tree nodes*. It is important to note that the δ-degree-bound is defined with respect to the original tree nodes in T_1^f and T_2^g, not for the augmented trees (the one for the augmented trees can be significantly higher).

Our DP-algorithm essentially checks for the feasibility $F(S, w)$ of valid-pairs (S, w)S. The following two lemmas bound the size of valid pairs, and their numbers. Their proofs are in Appendix C and D, respectively.

Lemma 2. *For any valid pair* (S, w)*, we have* $|S| \le \tau$ *and* $|\mathrm{Ch}(S)| \le \tau$*, where* $\tau = \tau_\delta(T_1^f, T_2^g)$ *is the δ-degree-bound w.r.t.* T_1^f *and* T_2^g*.*

Lemma 3. *Let* $\tau = \tau_\delta(T_1^f, T_2^g)$ *be the δ-degree-bound w.r.t.* T_1^f *and* T_2^g*. The total number of valid pairs that Algorithm* DPgoodmap*(*T_1^f, T_2^g, δ*) will inspect is bounded by* $O(n^3 2^\tau)$*, and they can be computed in the same time.*

To obtain the final time complexity for Algorithm DPgoodmap, consider computing $F(S, w)$ for a fixed valid pair (S, w). This takes $O(1)$ time in the base case (the super-level index $i = 1$). Otherwise for the case $i > 1$, observe that $k = |\mathrm{Ch}(w)| = degree(w) \le \tau$, and $|\mathrm{Ch}(S)| \le \tau$ by Lemma 2. Hence the number of partitioning of $\mathrm{Ch}(S)$ is bounded by $O(|\mathrm{Ch}(S)|^k) = O(\tau^\tau)$. For each partition, checking conditions (F-1) and (F-2) takes $O(k)$ time; thus the total time needed to compute $F(S, w)$ is $O(k\tau^\tau) = O(\tau^{\tau+1})$. Combining this with Lemma 3, we have that the time complexity of Algorithm DPgoodmap() is bounded from above by $O(n^3 2^\tau \tau^{\tau+1})$, as claimed.

4.2 A Faster Algorithm

It turns out that we do not need to inspect all the $O(n^3 2^\tau)$ number of valid pairs as claimed in Lemma 3. We can consider only what we call sensible-pairs, which we define now.

Definition 6. *Given a valid-pair* (S, w)*, suppose* S *is from super-level* $\mathrm{L}_i^{(1)}$ *and thus* w *is from super-level* $\mathrm{L}_i^{(2)}$*. Then,* (S, w) *is a* sensiblepair *if either of the following two conditions hold:*

(C-1) *S contains a tree node from $V(T_1^f)$, or its children* $\mathrm{Ch}(S) \subseteq \mathrm{L}_{i-1}^{(1)}$ *in the augmented tree $\widehat{T}_1^f$ contains some tree node from $V(T_1^f)$, or the parents of nodes of S in the augmented tree $\widehat{T}_1^f$ (which are necessarily from super-level $\mathrm{L}_{i+1}^{(1)}$) contains some tree node from $V(T_1^f)$; or*

(C-2) *w is a tree node of T_2^g, or* $\mathrm{Ch}(w) \subseteq \mathrm{L}_{i-1}^{(2)}$ *contains a tree node of T_2^g; or the parent of w from super-level $\mathrm{L}_{i+1}^{(2)}$ in the augmented tree $\widehat{T}_2^g$ is a tree node of T_2^g.*

Algorithm DPgoodmap() can be modified to Algorithm modified-DP() so that it only inspects sensible-pairs. The modification is non-trivial, and the reduction in the bound on number of sensible-pairs is by relating sensible-pairs to certain appropriately defined edge-list pairs ($A \subseteq E(T_1^f), \alpha \in E(T_2^g)$). The rather technical details can be found in Appendix E. We only summarize the main theorem below.

Theorem 3. *(i) Algorithm* modified-DP*(T_1^f, T_2^g, δ) returns "yes" if and only if $d_I(T_1^f, T_2^g) \leq \delta$.*
(ii) Algorithm modified-DP*(T_1^f, T_2^g, δ) can be implemented to run in $O(n^2 2^\tau \tau^{\tau+2} \log n)$ time, where n is the total complexity of input trees T_1^f and T_2^g, and $\tau = \tau_\delta(T_1^f, T_2^g)$ is the δ-degree-bound w.r.t. T_1^f and T_2^g.*
Note that if τ is constant, then the time complexity is $O(n^2 \log n)$.

5 Algorithms for Interleaving and Gromov-Hausdorff Distances

5.1 FPT Algorithm to Compute Interleaving Distance

In the previous section, we show how to solve the decision problem for interleaving distance between two merge trees T_1^f and T_2^g. We now show how to compute the interleaving distance δ^*, which is the smallest δ value such that $d_I(T_1^f, T_2^g) \leq \delta$ holds.

The main observation is that there exists a set Π of $O(n^2)$ number of *candidate values* such that δ^* is necessarily one of them. Specifically, let $\Pi_1 = \{|f(u) - g(w)| \mid u \in V(T_1^f), w \in V(T_2^g)\}$, $\Pi_2 = \{|f(u) - f(u')|/2 \mid u, u' \in V(T_1^f)\}$, and $\Pi_3 = \{|g(w) - g(w')|/2 \mid w, w' \in V(T_2^g)\}$. Set $\Pi = \Pi_1 \cup \Pi_2 \cup \Pi_3$. The proof of the following lemma can be found in Appendix F.

Lemma 4. *The interleaving distance $\delta^* = d_I(T_1^f, T_2^g)$ satisfies that $\delta^* \in \Pi$.*

Finally, compute and sort all candidate values in Π where by construction, $|\Pi| = O(n^2)$. Then, starting with δ being the smallest candidate value in Π, we perform algorithm DPgoodmap(T_1^f, T_2^g, δ) for each δ in Π in increasing order, till the first time the answer is 'yes'. The corresponding δ value at the time is $d_I(T_1^f, T_2^g)$. Furthermore, note that for the degree-bound parameter, $\tau_\delta(T_1^f, T_2^g) \leq \tau_{\delta'}(T_1^f, T_2^g)$ for $\delta \leq \delta'$. Combining with Theorem 3, we can easily obtain the following trivial bound:

Theorem 4. *Let $\delta^* = d_I(T_1^f, T_2^g)$ and $\tau^* = \tau_{\delta^*}(T_1^f, T_2^g)$ be the degree-bound parameter of T_1^f and T_2^g w.r.t. δ^*. Then we can compute δ^* in $O(n^4 2^{\tau^*} (\tau^*)^{\tau^*+2} \log n)$ time.*

However, it turns out that one can remove almost an $O(n^2)$ factor by using a double-binary search like procedure, as discovered by Kyle Fox. We include this improved result below and his argument below for completeness. See Appendix G for the proof.

Theorem 5. *Let $\delta^* = d_I(T_1^f, T_2^g)$ and $\tau^* = \tau_{\delta^*}(T_1^f, T_2^g)$ be the degree-bound parameter of T_1^f and T_2^g w.r.t. δ^*. Then we can compute δ^* in $O(n^2 2^{2\tau^*} (2\tau^*)^{2\tau^*+2} \log^3 n)$ time.*

5.2 FPT-Algorithm for Gromov-Hausdorff Distance

Finally, we develop a FPT-algorithm to approximate the Gromov-Hausdorff distance between two input trees (T_1, d_1) and (T_2, d_2). To approximate the Gromov-Hausdorff distance between two metric trees, we need to modify our parameter slightly (as there is no function defined on input trees any more). Specifically, now given a metric tree (T, d), a ε-geodesic ball at $u \in |T|$ is simply $\widehat{B}_\varepsilon(u, T) = \{x \in |T| \mid d(x, u) \leq \varepsilon\}$.

Parameter τ: Given $\mathcal{T}_1 = (T_1, d_1)$ and $\mathcal{T}_2 = (T_2, d_2)$, define the *$\varepsilon$-metric-degree-bound* parameter $\widehat{\tau}_\varepsilon(T_1, T_2)$ to be the largest sum of degrees of all tree nodes within any ε-geodesic ball in T_1 (w.r.t. metric d_1) or in T_2 (w.r.t. d_2).

We obtain our main result for approximating the Gromov-Hausdorff distance between two metric trees within a factor of 14. We note that to obtain this result, we need to also relate the ε-metric-degree-bound parameter for metric trees with the ε-degree-bound parameter used for interleaving distance for the special geodesic functions we use (in fact, we will show that $\widehat{\tau}_\delta \leq \tau_\delta \leq \widehat{\tau}_{2\delta}$). The proof of the following main theorem of this section can be found in Appendix H.

Theorem 6. *Given two metric trees $\mathcal{T}_1 = (T_1, d_1)$ and $\mathcal{T}_2 = (T_2, d_2)$ where the total number of vertices of T_1 and T_2 is n, we can 14-approximate the Gromov-Hausdorff distance $\hat{\delta^*} = \delta_{\mathcal{GH}}(\mathcal{T}_1, \mathcal{T}_2)$ in $O(n^4 \log n + n^2 2^{\widehat{\tau}} \widehat{\tau}^{\widehat{\tau}+2} \log^3 n)$ time, where $\widehat{\tau} = 2\widehat{\tau}_{28\hat{\delta^*}}(T_1, T_2)$ is twice the metric-degree-bound parameter w.r.t. $28\hat{\delta^*}$.*

Remarks: We remark that the time complexity of the FPT approximation algorithm of [22] contains terms n^k, where k is the parameter and could be large in general – Indeed, k is the cardinality of an ε-net of one of the input metric spaces, and ε also appears as an additive approximation term for algorithm. In contrast, the dependency of our algorithm on the parameter $\hat{\tau}$ is roughly $O(2^{O(\hat{\tau})})$, and our algorithm has only constant multiplicative approximation factor. On the other hand, note that the algorithm of [22] works for general finite metric spaces. We also remark that the Gromov-Hausdorff distance between two metric spaces (X, d_X) and (Y, d_Y) measures their *additive distortion*, and thus is not invariant under scaling. In particular, suppose the input two metric spaces $\mathcal{T}_1 = (T_1, d_1)$, $\mathcal{T}_2 = (T_2, d_2)$ scale by the same amount to a new pair of input trees $\mathcal{T}_1' = (T_1', d_1' = c \cdot d_1)$, $\mathcal{T}_2' = (T_2', d_2' = c \cdot d_2)$. Then the new Gromove-Hausdorff distance between them $\delta_{GH}(\mathcal{T}_1', \mathcal{T}_2') = c \cdot \delta_{GH}(\mathcal{T}_1, \mathcal{T}_2)$. However, note that the metric-degree-bound parameter for the new trees satisfies $\widehat{\tau}_{c\delta}(\mathcal{T}_1', \mathcal{T}_2') = \widehat{\tau}_\delta(\mathcal{T}_1, \mathcal{T}_2)$. Hence the time complexity of our algorithm to approximate the Gromov-Hausdorff distance $\delta_{GH}(\mathcal{T}_1', \mathcal{T}_2')$ for scaled metric-trees $\mathcal{T}_1'$ and $\mathcal{T}_2'$ **remains the same** as that for approximating the Gromov-Hausdorff distance $\delta_{GH}(\mathcal{T}_1, \mathcal{T}_2)$.

6 Concluding Remarks

In this paper, by re-formulating the interleaving distance, we developed the first FPT algorithm to compute the interleaving distance *exactly* for two merge trees, which in turn leads to an FPT algorithm to approximate the Gromov-Hausdorff distance between two metric trees.

We remark that the connection between the Gromov-Hausdorff distance and the interleaving distance is essential, as the interleaving distance has more structure behind it, as well as certain "order" (along the function associated to the merge tree), which helps to develop dynamic-programming type of approach. For more general metric graphs (which represent much more general metric spaces than trees), it would be interesting to see whether there is a similar relation between the Gromov-Hausdorff distance of metric graphs and the interleaving distance between the so-called Reeb graphs (generalization of merge trees).

Acknowledgment: We thank reviewers for helpful comments. We would like to thank Kyle Fox, for suggesting an elegant double-binary search procedure, to improve the time complexity of the optimal interleaving distance by almost a factor of n^2 (see Theorem 5), which further leads to a similar improvement for approximating the Gromov-Hausdorff distance (Theorem 6). This work is partially supported by National Science Foundation (NSF) under grants CCF-1740761, IIS-1815697 and CCF-1618247, as well as by National Institute of Health (NIH) under grant R01EB022899.

References

[1] P. K. Agarwal, K. Fox, A. Nath, A. Sidiropoulos, and Y. Wang. Computing the Gromov-Hausdorff distance for metric trees. *ACM Trans. Algorithms*, 14(2):24:1–24:20, 2018.

[2] R. Agarwala, V. Bafna, M. Farach, M. Paterson, and M. Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM Journal on Computing*, 28(3):1073–1085, 1998.

[3] N. Ailon and M. Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 73–82. IEEE, 2005.

[4] N. Alon, M. Bădoiu, E. D. Demaine, M. Farach-Colton, M. Hajiaghayi, and A. Sidiropoulos. Ordinal embeddings of minimum relaxation: general properties, trees, and ultrametrics. *ACM Transactions on Algorithms (TALG)*, 4(4):46, 2008.

[5] U. Bauer, X. Ge, and Y. Wang. Measuring distance between reeb graphs. In *30th Annual Sympos. on Comput. Geom.*, page 464, 2014.

[6] U. Bauer, C. Landi, and F. Mémoli. The reeb graph edit distance is universal. *CoRR*, abs/1801.01866, 2018.

[7] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theor. Comput. Sci.*, 392(1-3):5–22, 2008.

[8] P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, June 2005.

[9] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Efficient computation of isometry-invariant distances between surfaces. *SIAM J. on Sci. Comput.*, 28(5):1812–1836, 2006.

[10] M. Bădoiu, P. Indyk, and A. Sidiropoulos. Approximation algorithms for embedding general metrics into trees. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 512–521. Society for Industrial and Applied Mathematics, 2007.

[11] V. Chepoi, F. F. Dragan, I. Newman, Y. Rabinovich, and Y. Vaxes. Constant approximation algorithms for embedding graph metrics into trees and outerplanar graphs. *Discrete & Computational Geometry*, 47(1):187–214, 2012.

[12] V. de Silva, E. Munch, and A. Patel. Categorified reeb graphs. *Discrete & Computational Geometry*, 55(4):854–906, Jun 2016.

[13] M. Fellows, F. Fomin, D. Lokshtanov, E. Losievskaja, F. A. Rosamond, and S. Saurabh. Parameterized low-distortion embeddings-graph metrics into lines and trees. *arXiv preprint arXiv:0804.3028*, 2008.

[14] X. Ge, I. Safa, M. Belkin, and Y. Wang. Data skeletonization via Reeb graphs. In *Proc. 25th Annu. Conf. Neural Information Processing Systems (NIPS)*, pages 837–845, 2011.

[15] M. Gromov. *Metric Structures for Riemannian and Non-Riemannian Spaces.* Birkhäuser Basel, 2007.

[16] A. Hall and C. Papadimitriou. Approximating the distortion. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, volume 3624 of *Lecture Notes in Computer Science*, pages 111–122. Springer Berlin Heidelberg, 2005.

[17] F. Hétroy and D. Attali. Topological quadrangulations of closed triangulated surfaces using the Reeb graph. *Graph. Models*, 65(1-3):131–148, 2003.

[18] T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995.

[19] F. Mémoli and G. Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Found. of Comput. Math.*, 5(3):313–347, 2005.

[20] D. Morozov, K. Beketayev, and G. H. Weber. Interleaving distance between merge trees. In *Workshop on Topological Methods in Data Analysis and Visualization: Theory, Algorithms and Applications*, 2013.

[21] F. Mmoli. Some properties of gromovhausdorff distances. *Discrete & Computational Geometry*, pages 1–25, 2012. 10.1007/s00454-012-9406-8.

[22] F. Schmiedl. Computational aspects of the gromov–hausdorff distance and its application in non-rigid shape matching. *Discrete & Computational Geometry*, 57(4):854–880, Jun 2017.

[23] A. Sidiropoulos, D. Wang, and Y. Wang. Metric embeddings with outliers. In *Proc. 28th ACM-SIAM Sympos. Discrete Algorithms (SoDA)*, pages 670–689, 2017.

[24] J. Tierny. *Reeb graph based 3D shape modeling and applications.* PhD thesis, Universite des Sciences et Technologies de Lille, 2008.

[25] K. Zhang and T. Jiang. Some max snp-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249 – 254, 1994.

A Proof of Theorem 1

Theorem 1 follows from Lemma 5 and 6 below.

Lemma 5. *If $d_I(T_1^f, T_2^g) \leq \varepsilon$, then there exists an ε-good map $\alpha : |T_1^f| \to |T_2^g|$.*

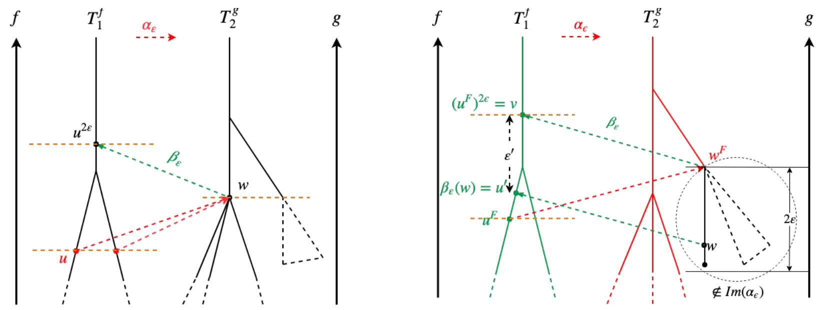
Proof. By definition, if $d_I(T_1^f, T_2^g) \leq \varepsilon$, then there exists a pair of ε-compatible maps $\alpha_\varepsilon : |T_1^f| \to |T_2^g|$ and $\beta_\varepsilon : |T_2^g| \to |T_1^f|$. We simply set $\alpha := \alpha_\varepsilon$, and argue that the three properties in Definition 3 will all hold for α.

Indeed, property (P1) follows trivially from condition (C1) of Definition 1. To see that property (P2) holds, set $w_i = \alpha(u_i)$, for $i = 1, 2$, and note that $u_i^{2\varepsilon} = \beta_\varepsilon(w_i)$. Since the map β_ε is continuous and $f(\beta_\varepsilon(w)) = g(w) + \varepsilon$ for any $w \in |T_2^g|$ (therefore monotone), it then follows from Observation 1 that $u_1^{2\varepsilon} = \beta_\varepsilon(w_1) \succeq \beta_\varepsilon(w_2) = u_2^{2\varepsilon}$, establishing property (P2).

We now show that property (P3) also holds. Specifically, consider any $w \in |T_2^g| \setminus \mathrm{Im}(\alpha)$, and let w^F be its lowest ancestor from $\mathrm{Im}(\alpha)$. Assume on the contrary that $g(w^F) - g(w) > 2\varepsilon$, then it must be that $w^F \succ w^{2\varepsilon} \succ w$. On the other hand, consider $u = \beta_\varepsilon(w)$. As $\alpha (= \alpha_\varepsilon)$ and β_ε are ε-compatible, we have that $\alpha(u) = w^{2\varepsilon}$, meaning that $w^{2\varepsilon} \in \mathrm{Im}(\alpha)$. This however contradicts our assumption that w^F is the lowest ancestor of w from $\mathrm{Im}(\alpha)$ as $w^F \succ w^{2\varepsilon}$. Hence it is not possible that $g(w^F) - g(w) > 2\varepsilon$, and property (P3) holds. $\square$

Lemma 6. *If there is an ε-good map $\alpha : |T_1^f| \to |T_2^g|$, then $d_I(T_1^f, T_2^g) \leq \varepsilon$.*

Proof. Given an ε-good map $\alpha : |T_1^f| \to |T_2^g|$, we will show that we can construct a pair of continuous maps $\alpha_\varepsilon : |T_1^f| \to |T_2^g|$ and $\beta_\varepsilon : |T_2^g| \to |T_1^f|$ that are ε-compatible, which will then prove the lemma. Specifically, set $\alpha_\varepsilon = \alpha$. We construct β_ε as follows:



(Case-1) (Case-2)

Figure 3: (Case-1). Construction of β_ϵ for the points of T_2^g which are in $\mathrm{Im}(\alpha)$. (Case-2). Construction of β_ϵ for the points of T_1^f which are not in $\mathrm{Im}(\alpha)$. w is not in the $\mathrm{Im}(\alpha)$, and w^F is its lowest ancestor in the $\mathrm{Im}(\alpha)$. $v = \beta_\varepsilon(w^F)$ is a point in 2ε higher than u^F in T_1^f. $u' = \beta_\varepsilon(w)$ is an ancestor of u^F that we map w by β_ε.

(Case-1): For any point $w \in \mathrm{Im}(\alpha) (\subseteq |T_2^g|)$, we simply set $\beta_\varepsilon(w) = u^{2\varepsilon}$, where $u \in |T_1^f|$ is any point from $\alpha^{-1}(w)$. Note that the choice of u does not matter. This is because that as $\alpha(u_1) = \alpha(u_2) = w$ (thus $\alpha(u_1) \succeq \alpha(u_2)$), it then follows from property (P2) of the ε-good map α that $u_1^{2\varepsilon} \succeq u_2^{2\varepsilon}$. Since $f(u_1^{2\varepsilon}) = f(u_2^{2\varepsilon}) = f(u_1) + 2\varepsilon$, it then must be that $u_1^{2\varepsilon} = u_2^{2\varepsilon}$. Hence $\beta_\varepsilon(w)$ is well-defined (independent of the choice of u). See the right figure in Figure 3.

(Case-2): For any point $w \in |T_2^g| \setminus \mathrm{Im}(\alpha)$ (i.e, $w \notin \mathrm{Im}(\alpha)$), let w^F be its lowest ancestor in $\mathrm{Im}(\alpha)$. There could be multiple points in T_1^f from $\alpha^{-1}(w^F)$. Consider an arbitrary but fixed choice $u^F \in \alpha^{-1}(w^F)$: For example, assume that all nodes and edges from T_1^f have a unique integer id, and we choose u^F to be the point from the node or edge with lowest id. Set $v = (u^F)^{2\varepsilon}$ to be the ancestor of u^F at height 2ε above u^F. From (Case-1), we know that we have already set $\beta_\varepsilon(w^F) = v$. See the above figure for an illustration.

Let $\varepsilon' = g(w^F) - g(w)$; by property (P3), $\varepsilon' \leq 2\varepsilon$. We simply set $\beta_\varepsilon(w)$ to be the unique point u' from the path connecting u^F to its ancestor v such that $f(v) - f(u') = \varepsilon'$; that is, $v \succeq u' \succeq u^F$. Easy to verify that under this construction, $f(\beta_\varepsilon(w)) = g(w) + \varepsilon$.

We now show that α_ε and β_ε as constructed above form a pair of ε-compatible maps for T_1^f and T_2^g. First, we claim that β_ε is continuous (Note that α_ε is continuous as $\alpha_\varepsilon = \alpha$.) Next we show that all conditions in Definition 1 are satisfied.

(Claim-1) The map $\beta_\varepsilon : |T_2^g| \to |T_1^f|$ is continuous. To this end, we first put a function-induced metric d_f (resp. d_g) on T_1^f (resp. on T_2^g), defined as follows: for any $u, u' \in |T_1^f|$, let $\pi(u, u')$ be the unique tree path connecting u to u'. Set $d_f(u, u') = \max_{x \in \pi(u,u')} f(x) - \min_{y \in \pi(u,u')} f(y)$. In other words, $d_f(u, u')$ is the maximum variation of the f-function value along the unique path $\pi(u, u')$ from u to u'. Define the function-induced metric d_g for $|T_2^g|$ in a symmetric manner.

Let $B_r(x, T_1^f) = \{y \in |T_1^f| \mid d_f(x, y) < r\}$ denote the open ball around a point $x \in T_1^f$ under the function-induced metric d_f; and define $B_r(z, T_2^g)$ symmetrically. (Note that this ball is in fact the same as the ε-ball we will use at the beginning of Section 4 to introduce the degree-bound parameter.) For simplicity, we sometimes omit the reference to the tree in these open balls when its choice is clear. To show that $\beta_\varepsilon : |T_2^g| \to |T_1^f|$ is continuous, we just need to show that for any $w \in |T_2^g|$ and $u = \beta_\varepsilon(w)$, given any radius $r > 0$, there always exists $r' > 0$ such that $\beta_\varepsilon(B_{r'}(w, T_2^g)) \subseteq B_r(u, T_1^f)$.

Fix $w \in |T_2^g|$, $u = \beta_\varepsilon(w)$, and radius $r > 0$. Let $0 < r' < r$ be a sufficiently small value so that $B_{r'}(w, T_2^g)$ contains no tree nodes other than potentially w. We will prove that $\beta_\varepsilon(w_0) \in B_{r'}(u, T_1^f)$ (and thus in $B_r(u, T_1^f)$) for any $w_0 \in \beta_\varepsilon(B_{r'}(w, T_2^g))$. Note first, by our choice of r', the unique path π connecting w_0 to w is monotone in g-function values, as the ball $B_{r'}(w, T_2^g)$ does not contain any tree node other than potentially w. If $\pi \subset \mathrm{Im}(\alpha)$, then, by our construction of β_ε, $\beta_\varepsilon(\pi)$ is a monotone path and thus lies in $B_{r'}(u, T_1^f)$. Hence $\beta_\varepsilon(w_0) \in B_{r'}(u, T_1^f)$ for any $w_0 \in B_{r'}(w, T_2^g)$.

Now assume that $\pi \setminus \mathrm{Im}(\alpha) \neq \emptyset$. W.o.l.g assume that $g(w_0) < g(w)$; otherwise, we simply switch the role of the two in our argument below. Let w^F denote the lowest ancestor of w_0 from $\mathrm{Im}(\alpha)$. First, assume that $w \notin \mathrm{Im}(\alpha)$, which means that the entire path $\pi \in |T_2^g| \setminus \mathrm{Im}(\alpha)$, and $w^F \succeq w \succeq w_0$. In this case, let $u^F \in \alpha^{-1}(w^F)$ be the preimage of w^F under α used in our procedure to construct $\beta_\varepsilon(w)$. Set $v = (u^F)^{2\varepsilon} = \beta_\varepsilon(w^F)$, and let $\pi'(u^F, v)$ denote the unique path between them (note that the path $\pi'(u^F, v)$ is monotone.) Based on our procedure to construct β_ε, we know that both $\beta_\varepsilon(w)$ and $\beta_\varepsilon(w_0)$ are in $\pi'(u^F, v)$, and $\beta_\varepsilon(\pi)$ is in fact a subpath of $\pi'(u^F, v)$ and thus also monotone. It then follows that $\beta_\varepsilon(\pi) \subseteq B_{r'}(u, T_1^f)$.

The only remaining case is when $w \succeq w^F \succeq w_0$. In this case, we consider the two sub-path $\pi_1 = \pi(w_0, w^F)$ and $\pi_2 = \pi(w^F, w)$ of π. Using similar arguments above, we can show that $\beta_\varepsilon(\pi)$ is still a monotone continuous path in $|T_1^f|$ and thus $\beta_\varepsilon(\pi) \in B_{r'}(u, T_1^f)$. Hence $\beta_\varepsilon(w_0) \in B_{r'}(u, T_1^f)$ for any $w_0 \in B_{r'}(w, T_2^g)$.

Putting everything together, it then follows that the map β_ε constructed is continuous.

(Claim-2) All conditions in Definition 1 are satisfied for the map $\beta_\varepsilon : |T_2^g| \to |T_1^f|$. Conditions (C1), (C2) and (C3) follow from the constructions of α_ε and β_ε. What remains is to prove that condition (C4) holds. Consider any $w \in |T_2^g|$.

If $w = \alpha(u) \in \mathrm{Im}(\alpha)$, then by construction $\beta_\varepsilon(w) = u^{2\varepsilon} \succ u$. Then Observation 1 implies that $\alpha_\varepsilon(\beta_\varepsilon(w))$ is necessarily an ancestor of $\alpha(u) = w$. Furthermore, since $g(\alpha_\varepsilon(\beta_\varepsilon(w)) - g(w) = 2\varepsilon$ (by condition (C3)), it then must be that $\alpha_\varepsilon(\beta_\varepsilon(w)) = w^{2\varepsilon}$, establishing condition (C4).

Otherwise, $w \notin \mathrm{Im}(\alpha)$: let w^F be its lowest ancestor from $\mathrm{Im}(\alpha)$ with u^F being the point from $\alpha^{-1}(w^F)$ as used in (Case-2) of the construction above. Recall that we set $\beta_\varepsilon(w)$ such that $(u^F)^{2\varepsilon} \succeq \beta_\varepsilon(w) \succeq u^F$. Hence $\alpha_\varepsilon(\beta_\varepsilon(w))$ must be an ancestor of $\alpha(u^F) = w^F$. It then follows that $\alpha_\varepsilon(\beta_\varepsilon(w)) \succeq w^F \succeq w$. Furthermore, since $g(\alpha_\varepsilon \circ \beta_\varepsilon(w)) = g(w) + 2\varepsilon$, we have that $\alpha_\varepsilon \circ \beta_\varepsilon(w) = w^{2\varepsilon}$. Putting everything together, we thus have that α_ε and β_ε form a pair of ε-compatible maps for T_1^f and T_2^g, implying that $d_I(T_1^f, T_2^g) \le \varepsilon$.

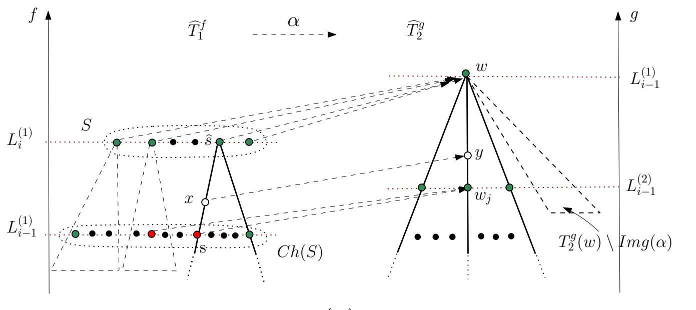
This finishes the proof of Lemma 6. □

B Proof of Lemma 1

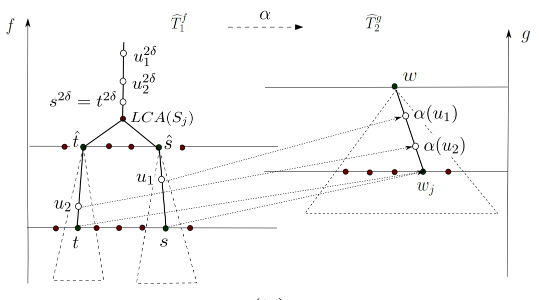
Recall that $h_i = height(\mathrm{L}_i^{(1)})$ and $\widehat{h}_i = height(\mathrm{L}_i^{(2)}) = h_i + \delta$, for any $i \in [1, m]$. We prove this lemma by induction on the indices of super-levels.

For the base case when $i = 1$, it is easy to verify that the lemma holds for any valid pair (S, w) from level-1. Indeed, at this first level, $\mathcal{F}(S) = S$ and $T_2(w) = w$, and thus nodes in S and w are all leaf-nodes of T_1^f or T_2^g. Hence if $F(S, w) = 1$, properties (P1) and (P3) hold trivially for the map $\alpha : \mathcal{F}(S) \to T_2(w)$ defined as $\alpha(s) = w$ for any $s \in \mathcal{F}(s) = S$. Property (P2) follows from the fact that (S, w) is valid, thus $s_1^{2\delta} = s_2^{2\delta}$ for any two $s_1, s_2 \in S$. Similarly, if there is a partial-δ-good map $\alpha : \mathcal{F}(S) \to T_2(w) = w$ for a valid pair (S, w), it has to be that $\alpha(s) = w$ for any $s \in \mathcal{F}(S)$. Since $T_2(w) = w$, w is a leaf and thus $depth(w) = 0 < 2\delta$, meaning that $F(S, w) = 1$.

Now consider a generic level $i > 1$, and assume the claim holds for all valid pairs from level $j < i$. We prove that the claim holds for any valid pair (S, w) from level-i as well:



(a)



(b)

Figure 4: (a). Assume that $S_j \subset \mathrm{Ch}(S)$ consists the two red points s and s'; note $\alpha(S_j) = w_j$. Under extension of α, x (from edge $(s, \hat{s})$) is mapped to $y \in (w_j, w)$. (b). Illustration for case-(a): $s, t \in S_j$ and $\alpha(S_j) = w_j$, hence S_j must be valid, meaning that $s^{2\delta} = t^{2\delta} \succeq LCA(S_j)$, which ulitmately implies that $u_1^{2\delta} \succeq u_2^{2\delta}$.

⇒: Suppose $F(S,w)=1$. In this case, we will show that we can construct a partial-δ-good map $\alpha : \mathcal{F}_1(S) \to T_2(w)$. We assume that w is not a leaf-node; as otherwise, $F(S,w)$ means that all nodes in S are also necessarily leaf-nodes for $\widehat{T}_1^f$ and thus $\mathcal{F}(S)=S$ and $T_2(w)=\{w\}$. We then simply set $\alpha : \mathcal{F}(S) \to T_2(w)$ as $\alpha(s)=w$ for each $s \in S$ and easy to see that this α is partial-δ-good (by using the same argument as for the base case). Now suppose $\mathrm{Ch}(w)=\{w_1,\ldots,w_k\}$ and let $S_1,\ldots,S_k$ be the partition of $\mathrm{Ch}(S)$ that make $F(S,w)=1$. Assume w.o.l.g that $S_1,\ldots,S_a$ are non-empty, while $S_{a+1},\ldots,S_k$ are empty. Since $F(S_j,w_j)=1$ for $j \in [1,a]$, there exists an partial-δ-good map $\alpha_j : \mathcal{F}_1(S_j) \to T_2(w_j)$ by induction hypothesis. The restriction of α to each $\mathcal{F}_1(S_j)$ is simply α_j. Then, we "extend" these α_j's into a map $\alpha : \mathcal{F}_1(S) \to T_2(w)$ as follows:

For each child $s \in \mathrm{Ch}(S)$, suppose $s \in S_j$ and the parent of s is $\hat{s} \in S$. Under the partial-δ-good map $\alpha_j : \mathcal{F}_1(S_j) \to T_2(w_j)$ (from induction hypothesis), we know that $\alpha_j(s)=w_j$. We extend α_j to all points within segment $(s,\hat{s})$. Specifically, for any point $x \in \mathit{edge}(s,\hat{s})$, we simply set $\alpha(x)$ to be the corresponding point $y \in \mathit{edge}(w_j,w)$ at height $f(x)+\delta$ (i.e, $g(y)=f(x)+\delta$). See Figure 4 (a) for an illustration.

Easy to verify that this extended map is continuous: as first, the extension along each edge $(s,\hat{s})$ is continuous. The only place where discontinuity may happen is at points $\hat{s}$ from S. However, all points in S will be mapped to w under this extension. Hence α constructed above is continuous.

Furthermore, by construction, α satisfies property (P1). To prove that property (P3) holds, consider any point $z \in T_2(w) \setminus \mathrm{Im}(\alpha)$. First, observe that by construction, all edge segments $[w_\ell, w]$, for $\ell \in [1,a]$, are contained in $\mathrm{Im}(\alpha)$. Now suppose $z \in T_2(w_j)$ for some $j > a$; that is, $S_j=\emptyset$. Then, the lowest ancestor of z from $\mathrm{Im}(\alpha)$, denoted by z^F, is necessarily $z^F=w$. By (F-2) of our procedure, we have $\mathit{depth}(w_j) \le 2\delta - (\hat{h}_i - \hat{h}_{i-1})$. It then follows that:

$$g(z^F)-g(z)=g(w)-g(z)=g(w)-g(w_j)+g(w_j)-g(z) \le g(w)-g(w_j)+\mathit{depth}(w_j)$$
$$\le \hat{h}_i - \hat{h}_{i-1} + \mathit{depth}(w_j) \le 2\delta.$$

The only case left is that $z \in \mathit{edge}(w_j,w)$ for some $j > a$. Again, its lowest ancestor from $\mathrm{Im}(\alpha)$ is $z^F=w$, and

$$g(z^F)-g(z)=g(w)-g(z) \le g(w)-g(w_j) \le 2\delta.$$

(Again, the last inequality follows from (F-2) of our procedure.) Hence property (P3) holds for the constructed map α.

What remains is to show that (P2) also holds for α. In particular, we need to show that for any $u_1,u_2 \in \mathcal{F}(S)$ such that $\alpha(u_1)\succeq\alpha(u_2)$, we have that $u_1^{2\delta}\succeq u_2^{2\delta}$.

First, if $\alpha(u_1)=w$ (implying that $u_1 \in S$), then this claim follows from the fact that (S,w) is valid – Indeed, as $f(LCA(S)) \le h_i + 2\delta = f(u_1)+2\delta$, the node $u_1^{2\delta}$ thus is the ancestor of $u^{2\delta}$ for any point $u \in \mathcal{F}(S)$. So from now on we assume that $\alpha(u_1)\prec w$. If $\alpha(u_1) \in T_2(w_j)$ for some $j \in [1,a]$, then by construction of the map α, we have $\alpha(u_1)=\alpha_j(u_1)$ and $\alpha(u_2)=\alpha_j(u_2)$ for the partial-δ-good map $\alpha_j : \mathcal{F}_1(S_j) \to T_2(w_j)$. Thus $u_1^{2\delta}\succeq u_2^{2\delta}$ holds as α_j is partial-δ-good.

Now assume otherwise, which means that there exists some $j \in [1,a]$ such that $\alpha(u_1) \in (w_j,w)$ where (w_j,w) denote the interior of the edge connecting w_j and its parent w, implying that

$\alpha(u_2) \in T_2(w_j) \cup (w_j, w)$. Since $f(u_1) = g(\alpha(u_1)) - \delta$, there exists some $s \in S_j$ such that u_1 is from edge $(s, \hat{s})$ with $\hat{s}$ being the parent of s from S. There are two cases:

(a) Suppose $\alpha(u_2) \in (w_j, w)$. By construction of α, there must be $t \in S_j$ and $\hat{t} \in S$ such that $u_2 \in (t, \hat{t})$. Since $F(S_j, w_j) = 1$, the pair (S_j, w_j) must be valid, meaning that $f(LCA(S_j)) \leq height(S_j) + 2\delta$. As $s, t \in S_j$, it then follows that $s^{2\delta} = t^{2\delta} \succeq LCA(S_j)$. See Figure 4 (b). Since in the tree T_1^f, $u_1^{2\delta} \succeq s^{2\delta}$, $u_2^{2\delta} \succeq t^{2\delta}$, and $f(u_1^{2\delta}) \geq f(u_2^{2\delta})$, it must be that $u_1^{2\delta} \succeq u_2^{2\delta} \succeq LCA(S_j)$. Hence property (P2) for holds for u_1 and u_2.

(b) The second case is that $\alpha(u_2) \in T_2(w_j)$. In this case, note that $\alpha(u_1) \succeq \alpha(s) \succeq \alpha(u_2)$ as $\alpha(s) = w_j$ is the only child node of $\alpha(u_1)$ in $\widehat{T}_2^g$. We thus obtain that $s^{2\delta} \succeq u_2^{2\delta}$ by applying property (P2) w.r.t. the partial-δ-good map $\alpha_j : \mathcal{F}_1(S_j) \to T_2(w_j)$ to the pair of points s and u_2. As $u_1^{2\delta} \succeq s^{2\delta}$, it then follows that $u_1^{2\delta} \succeq u_2^{2\delta}$.

This finishes the proof that property (P2) also holds for the newly constructed map $\alpha : \mathcal{F}_1(S) \to T_2(w)$. Putting everything together, we have that α is an partial-δ-good map.

$\Leftarrow$: Now suppose there is an partial-δ-good map $\alpha : \mathcal{F}_1(S) \to T_2(w)$ for a valid pair (S, w). We aim to show that $F(S, w) = 1$ in this case. As α is monotonically continuous, we know $\alpha(\mathrm{Ch}(S)) \subseteq \mathrm{Ch}(w) = \{w_1, \ldots, w_k\}$. For each $i \in [1, k]$, set $S_i = \alpha^{-1}(w_i)$ (we set $S_i = \emptyset$ if $w_i \notin \mathrm{Im}(\alpha)$). Obviously, $S_1, \ldots, S_k$ obtained this way form a partition of $\mathrm{Ch}(S)$; that is, $\cup_i S_i = \mathrm{Ch}(S)$ and $S_i \cap S_j = \emptyset$. Similar to above, assume w.o.l.g. that $S_1, \ldots, S_a$ are non-empty, and $S_{a+1}, \ldots, S_k$ are empty. It is easy to see that the restriction of α to each $\alpha_j : \mathcal{F}(S_j) \to T_2(w_j)$, for $j \in [1, a]$, gives rise to an partial-δ-good map. Furthermore, we claim that each (S_j, w_j), $j \in [1, a]$, is a valid pair. In particular, we need to show that $LCA(S_j) \leq h + 2\delta$ where $h = h_{i-1}$ is the height (f-value) of nodes in S_j (from super-level $\mathrm{L}_{i-1}^{(1)}$). This follows from property (P2) of map α as for any two $u_1, u_2 \in S_j$, $\alpha(u_1) = \alpha(u_2) = w_j$, meaning that $u_1^{2\delta} = u_2^{2\delta}$. Hence $LCA(S_j)$ must be at height at most 2δ above $f(u_1) = h_{i-1}$. Thus (S_j, w_j) is a valid pair for any $j \in [1, a]$. Since (S_j, w_j) is valid, and it is from level $i - 1$, it then follows from the induction hypothesis that $F(S_j, w_j) = 1$ for $j \in [1, a]$ as there is a partial-δ-good map $\alpha_j : \mathcal{F}(S_j) \to T_2(w_j)$. This establishes condition (F-1) in our algorithm DPgoodmap().

Finally, consider any $S_j = \emptyset$ (i.e, $j > a$). This means that $w_j \in T_2(w) \setminus \mathrm{Im}(\alpha)$, and thus $T_2(w_j) \subseteq T_2(w) \setminus \mathrm{Im}(\alpha)$. On the other hand, since there is no tree nodes of $\widehat{T}_2^g$ between $\mathrm{L}_{i-1}^{(2)}$ and $\mathrm{L}_i^{(2)}$ (and thus between w_j and its ancestor w), the lowest ancestor w_j^F of w_j from $\mathrm{Im}(\alpha)$ must be w. This implies that for any $y \in |T_2(w_j)|$, $y^F = w$ as well. It then follows that for any $y \in |T_2(w_j)|$, $g(w) - g(y) \leq 2\delta$ by property (P3) of α. We thus have:

$$\begin{aligned} & g(w_j) - g(y) = (g(w) - g(y)) - (g(w) - g(w_j)) \leq 2\delta - (g(w) - g(w_j)) \\ \Rightarrow \quad & depth(w_j) = \max_{y \in |T_2(w_j)|} (g(w_j) - g(y)) \leq 2\delta - (g(w) - g(w_j)) = 2\delta - (\widehat{h}_i - \widehat{h}_{i-1}). \end{aligned}$$

This shows that condition (F-2) also holds. Hence $F(S, w) = 1$ as (S, w) is valid.

Lemma 1 then follows from the above two directions.

C Proof of Lemma 2

First, we need the following simple result.

Claim 3. *Given $v \in |T_1^f|$ (not necessarily a tree node of T_1^f), Set $T'_v := \{x \in |T_1^f| \mid x \preceq v, \text{ and } f(v) - f(x) \geq 2\delta\}$. Then there exists $u \in |T_1^f|$ such that $T'_v \subseteq B_\delta(u, T_1^f)$. This implies that the sum of degrees of all tree nodes from T'_v is bounded by the δ-degree-bound w.r.t. T_1^f and T_2^g.*

Proof. See Figure 5 for an illustration: We simply choose u as any descendant $u \preceq v$ such that $f(v) - f(u) = \delta$. (If there is no point at this height, just take u to be the lowest descendant of v.) It is easy to see that that for each $x \in |T'_v|$, the path $\pi(u, x)$ is contained inside T'_v and thus all points in this path is within δ height difference from $f(u)$, that is, for any $y \in \pi(u, x)$, $|f(y) - f(u)| \leq \delta$. Hence $T'_v \subseteq B_\delta(u; T_1^f)$. The bound on the sum of degrees for all tree nodes in T'_v follows from the definition of δ-degree-bound. □

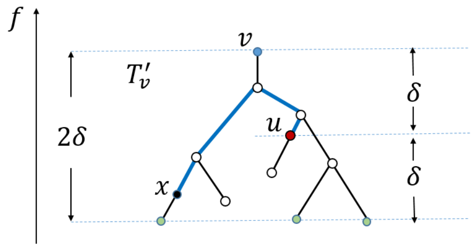


Figure 5: The thickened path is $\pi(u, x)$.

Next, to prove Lemma 2, note that points in S are augmented-tree nodes from the augmented tree $\widehat{T}_1^f$ (not necessarily from T_1^f), while the degree-bound parameter τ is defined w.r.t. the original trees T_1^f and T_2^g.

Given a valid-pair (S, w), if $|S| = 1$, then the claim holds easily. So we now assume that $|S| > 1$. In this case, (S, w) being valid means that $v = LCA(S)$ is at most 2δ height above nodes in S. Let $T'_v = \{x \preceq v \mid f(v) - f(x) \leq 2\delta\}$ denote the subset of the subtree $T_1^f(v)$ consisting of all descendant of v in $|T_1^f|$ whose height is within 2δ difference to the height $f(v)$ of v. See Figure 6 for an illustration. Points in S *may not* be tree nodes from T'_v. However, as all nodes in S are coming from the same height (f-function value), each tree arc in T'_v can give rise to at most one point in S. Hence $|S|$ is bounded by the total number of edges in T'_v, which in turn is at most the sum of degrees of all tree nodes in T'_v. It then follows from Claim 3 that $|S| \leq \tau$ as claimed.

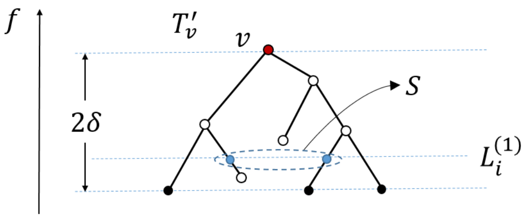


Figure 6: Blue dots are S from super-level $\mathrm{L}_i^{(1)}$.

Next, we now bound $|\mathrm{Ch}(S)|$, the number of child-nodes of S. Indeed, first, suppose a point $s \in S$ is an augmented tree node of $\widehat{T}_1^f$ but not a tree node of T_1^f. Then s can give rise to only one child-node in $\mathrm{Ch}(S)$, and we can charge this node to the tree arc s lies in. Otherwise, suppose s is also a tree node in T_1^f. Then the number of its child-nodes is already counted when we compute the sum of degrees of all tree nodes in T'_v. Putting these two together, we have that $|\mathrm{Ch}(S)|$ is also bounded from above by the sum of degrees of all tree nodes within T'_v, which is further bounded by $\tau_\delta(T_1^f, T_2^g) = \tau$ by Claim 3. This finishes the proof for Lemma 2.

D Proof of Lemma 3

In what follows we will separate valid pairs to two classes: (i) a *singleton-pair* (S, w) is a valid pair with $|S| = 1$, or (ii) a *non-singleton-pair* is a valid pair (S, w) with $|S| > 1$.

First, consider singleton-pairs, which have the form (s, w) with $s \in V(\widehat{T}_1^f)$ and $w \in V(\widehat{T}_2^g)$. The number of augmented tree nodes in each augmented tree $\widehat{T}_1^f$ or $\widehat{T}_2^g$ is $O(n^2)$. Hence there are $O(n^2)$

choices of w. For each $w \in \mathrm{L}_i^{(2)}$, it can be paired with $O(n)$ potential augmented-tree nodes from the super-level $\mathrm{L}_i^{(1)}$ of $\widehat{T}_1^f$. Therefore the total number of singleton-pairs is bounded by $O(n^3)$.

Next we bound the number of non-singleton-pairs (S, w), with $|S| > 1$, that Algorithm DP-goodmap() may inspect. Given such a set $S \subset V(\widehat{T}_1^f)$ from the super-level $\mathrm{L}_i^{(1)}$, its common ancestor $v = LCA(S)$ has to be a *tree node* in $V(T_1^f)$ whose height (f-function value) is at most 2δ above points in S; that is, $f(v) \le h_i + 2\delta$ where recall that h_i is the height (f-value) of super-level $\mathrm{L}_i^{(1)}$. As $v \in V(T_1^f)$, there are $|V(T_1^f)| \le n$ choices for v. We now count how many possible sets of S a fixed choice $v \in V(T_1^f)$ can produce.

To this end, set $T'_v = \{x \preceq v \mid f(v) - f(x) \le 2\delta\}$ as in Claim 3, by which we know that the sum of degrees of all nodes within T'_v is τ. Hence the total number tree edges (from T_1^f) contained in T'_v is at most τ. On the other hand, there can be $O(n)$ number of super-levels intersecting T'_v. For each such super-level, the number of augmented tree nodes contained in $|T'_v|$ is bounded by the number of tree edges of T_1^f in T'_v and thus by τ. It then follows that for each super-level intersecting T'_v, the number of potential subset S's it can produce is at most 2^τ. Hence all super-levels from T'_v can produce at most $O(n2^\tau)$ number of potential sets of S. Overall, considering all $O(n)$ choices of v's, there can be at most $O(n^2 2^\tau)$ potential S's that the algorithm will never need to inspect.

For each potential S, say from $\mathrm{L}_i^{(1)}$, there are n choices for w (as it must be an augmented-tree node from the super-level $\mathrm{L}_i^{(2)}$ of $\widehat{T}_2^g$). Putting everything together, we have that there are at most $O(n^3 2^\tau)$ number of valid-pairs (S, w) with $|S| > 1$, and they can also be enumerated within the same amount of time.

E A Faster FPT Algorithm for Deciding "Is $d_I(T_1^f, T_2^g) \le \delta$"

In what follows, we first bound the number of sensible-pairs in Lemma 8. We then show that Algorithm DPgoodmap() can be modified to consider ony sensible-pairs, and achieves the claimed time complexity.

Edge-list pairs. Consider any pair (S, w) with $S \subseteq \mathrm{L}_i^{(1)}$ and $w \in \mathrm{L}_i^{(2)}$. Suppose $S = \{s_1, \ldots, s_\ell\}$; each s_j is an augmented-tree node from some tree arc, say e_j of T_1^f. We call $A = \{e_1, \ldots, e_\ell\} \subseteq E(T_1^f)$ the *edge-list supporting* S. Let $\alpha \in E(T_2^g)$ be the tree edge in T_2^g such that $w \in \alpha$. We say that the *edge-list pair* (A, α) *supports* (S, w). Two different pairs (S, w) and (S', w') could be supported by the same edge-list pair. However, we claim that each edge-list pair can support at most 4 sensible-sets. Recall that $E(T)$ stands for the edge set of a tree T. Given an arbitrary tree edge $e = (u, u') \in E(T_1^f)$, we refer to the endpoint, say u, with smaller f-value as the *lower-endpoint of* e, while the other one with higher f-value as the *upper-endpoint of* e. Similarly define the lower/upper-endpoints for edges in T_2^g.

Lemma 7. *Each edge-list (A, α), with $A \subseteq E(T_1^f)$ and $\alpha \in E(T_2^g)$, can support at most 4 sensible-pairs.*

Proof. Let u_H be the lowest upper-end point of all edges in the edge-list $A = \{e_1, \ldots, e_\ell\}$ and u_L be the highest lower-endpoint of all edges in it. Suppose $\alpha = (z_L, z_H)$ with $g(z_L) < g(z_H)$. See Figure 7 for an illustration. Let i_H be the *smaller* index of the super-levels supporting u_H from $\mathcal{L}_1$ and supporting z_H from $\mathcal{L}_2$, respectively. Similarly, let i_L be the *larger* index of the super-levels

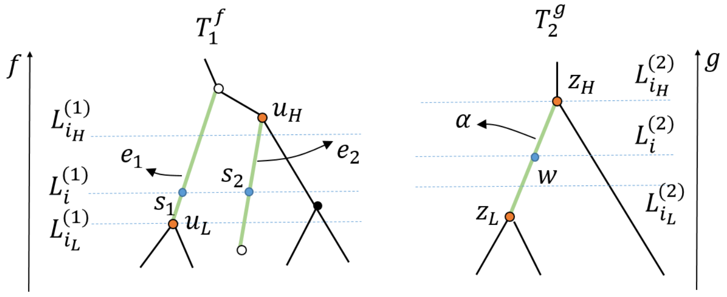


Figure 7: Green edges are $A = \{e_1, e_2\}$ and α. In this example, i_L is the index of the super-level passing through u_L, while i_H is the index of the super-level passing through z_H.

supporting u_L from $\mathcal{L}_1$ and supporting z_L from $\mathcal{L}_2$. Then any valid pair (S, w) supported by (A, w) can only come from the i-th super-level with $i \in [i_L, i_H]$. Furthermore, any **sensible-pair** (S, w) supported by (A, w) must be from the i-th super-levels with $i \in \{i_L, i_L + 1, i_H - 1, i_H\}$, as for any other choice of $i \in [i_L, i_H]$, no point from S, w, their children or parents in the augmented trees $\widehat{T}_1^f$ and $\widehat{T}_2^g$, can be a tree node. This proves the lemma. □

Lemma 8. *There are $O(n^2 2^\tau)$ distinct edge-list pairs supporting sensible-pairs. Hence there are $O(n^2 2^\tau)$ sensible-pairs, and they can be computed in $O(n^2 2^\tau)$ time.*

Proof. First, it is easy to see that there are $O(n^2)$ singleton sensible-pairs. Indeed, each singleton pair $(\{s\}, w)$ is necessarily supported by a singleton edge-list pair of the form (e, α) with $s \in e$ and $w \in \alpha$. By Lemma 7, each edge-list pair can support at most 4 sensible-pairs (thus at most 4 singleton sensible-pairs). Since there are $O(n^2)$ number of singleton edge-list pairs, it then follows that there can be $O(n^2)$ singleton sensible-pairs.

We now focus on non-singleton sensible-pairs. To this end, we will distinguish two types of sensible-pairs and bound them separately. A sensible-pair (S, w) is *type-1* if condition (C-1) in Definition 6 holds; and *type-2* if condition (C-2) holds.

First, we bound type-1 sensible-pairs. We say that a set S from some super-level $\mathrm{L}_i^{(1)}$ is a *sensible-set* if S satisfies condition (C-1). (In other words, if (S, w) is type-1 sensible-pair, then S must be a sensible-set.) We will now bound the number of sensible-sets. Given a non-singleton type-1 sensible-pair (S, w) (thus $|S| > 1$), similar to the argument in the proof of Lemma 3, the lowest common ancestor $v = LCA(S)$ of augmented-tree nodes in S must be a *tree node* from $V(T_1^f)$ less than 2δ heighb above S. Set $T'_v = \{x \preceq v \mid f(v) - f(x) \leq 2\delta\}$: we know that S is contained within T'_v. By Claim 3, both the number of tree nodes and the number of tree arcs within T'_v are bounded from above by τ. Since there are only $O(\tau)$ number of tree-arcs of T_1^f contained in T'_v, T'_v can produce at most 2^τ distinct edge-lists. On the other hand, a similar argument as the one used for Lemma 7 can also show that each edge-list $A \subset E(T_1^f)$ can only support at most 4 sensible-sets. Hence there are at most $O(2^\tau)$ sensible-sets possible from T'_v. Ranging over all $O(n)$ choices of v's, there can be $O(n2^\tau)$ possible sensible-sets.

For each sensible-set S, there can be at most n choices of w forming a potential sensible-pair (S, w) with it (where w has to be an augmented tree-nodes from a specific super-level in $\widehat{T}_2^g$). This leads to $O(n^2 2^\tau)$ type-1 sensible-pairs in total, and they can be computed within the same time complexity.

Next, we bound the type-2 sensible-pairs (S, w). Note that there are only $O(n)$ choices of w participating type-2 sensible-pairs (as each tree edge in T_2^g can give rise to at most 4 choices of w's satisfying condition (C-2) in Definition 6). For each fixed choice $w \in \alpha_w$, by an argument similar to the one used for the type-1 case, we can argue that there are only $O(n2^\tau)$ number of edge-lists from $E(T_1^f)$ supporting some sensible-pair of the form (S, w). Ranging over $O(n)$ choices for w, this gives rise to $O(n^2 2^\tau)$ total edge-list pairs supporting at most $O(n^2 2^\tau)$ type-2 sensible-pairs. This finishes the proof of the claim. $\square$

We now modify Algorithm $\mathsf{DPgoodmap}(T_1^f, T_2^g, \delta)$ so that it will only compute feasibility $F(S, w)$ for sensible-pair (S, w)'s.

Description of $\mathsf{modified\text{-}DP}(T_1^f, T_2^g, \delta)$

(1) Compute all edge-list pairs Ξ that support sensible-sets. For each edge-list $(A, \alpha) \in \Xi$, associate to it all (at most 4) sensible-sets that (A, α) supports. From Ξ, compute all sensible-pair and associate, to each super-level $\mathrm{L}_i^{(1)}$, the collection of sensible-pairs (S, w) such that $S \subseteq \mathrm{L}_i^{(1)}$.

(2) Compute the feasibility $F(S, w)$ in a bottom-up manner (i.e, in increasing order of their super-level indices) only for sensible-pairs (S, w). Consider a sensible-pair (S, w) from the i-th super-levels, that is, $S \subseteq \mathrm{L}_i^{(1)}$ and $w \in \mathrm{L}_i^{(2)}$.

Base case $i = 1$: All valid-pairs from the first super-level are sensible-pairs. The computation of $F(S, w)$ is the same as in the Base-case of Algorithm $\mathsf{DPgoodmap}()$.

When $i > 1$: We have already computed the feasibility for all sensible-pairs from level $(i-1)$ or lower. Now given a valid pair (S, w) from level-i, we will use a similar strategy as in Algorithm $\mathsf{DPgoodmap}()$. Specifically, let $\mathrm{Ch}(S) \subseteq \mathrm{L}_{i-1}^{(1)}$ denote the set of children of S from super-level $\mathrm{L}_{i-1}^{(1)}$, and $\mathrm{Ch}(w) = \{w_1, \ldots, w_k\} \subseteq \mathrm{L}_{i-1}^{(2)}$ the set of children of augmented-tree node w.

If $\mathrm{Ch}(w)$ is empty, the $F(S, w) = 1$ if and only if $\mathrm{Ch}(S) = \emptyset$.

If $\mathrm{Ch}(w)$ is not empty, then we check whether there exists a partition of $\mathrm{Ch}(S) = S_1 \cup \cdots \cup S_k$ such that conditions (F-1) and (F-2) in Algorithm $\mathsf{DPgoodmap}()$ hold.

However, we need to modify condition (F-1), as it is possible that (S_j, w_j) is valid but not a sensible-pair and thus $F(S_j, w_j)$ has not yet being computed.

More precisely, we do the following: if S_j is not valid, then obviously $F(S_j, w_j) = 0$. Otherwise, (S_j, w_j) is a valid-pair. If it is also a sensible-pair, then $F(S_j, w_j)$ is already computed.

The remaining case is that (S_j, w_j) is valid but not a sensible-pair. Let A be the edge-list supporting S_j, and $\alpha \in E(T_2^g)$ the edge containing w_j. Let (S', w') be the highest sensible-pair supported by (A, α) but from a super-level below that of (S_j, w_j). If such a (S', w') does not exist, then we set $F(S_j, w_j) = 0$. Otherwise, set $F(S_j, w_j) = F(S', w')$.

The $\mathsf{(modified\ F\text{-}1)}$ is: If $S_j \neq 0$, the $F(S_j, w_j)$ as setup above should equal to 1.

Output: The algorithm returns "yes" if and only if $F(root(\widehat{T}_1^f), root(\widehat{T}_2^g)) = 1$.

Proof of Part (i) of Theorem 3. We now show that Algorithm $\mathsf{modified\text{-}DP}(T_1^f, T_2^g, \delta)$ returns "yes" if and only if $d_I(T_1^f, T_2^g) \leq \delta$.

First, note that $(root(\widehat{T}_1^f), root(\widehat{T}_2^g))$ must be a sensible-pair, and thus will be computed by our algorithm. We now show that for any sensible-pair (S, w), the feasibility $F(S, w)$ computed by modified-DP() is the same as that computed by DPgoodmap(). To differentiate the two feasibility values, we use $F_{new}(S, w)$ and $F_{old}(S, w)$ to denote the feasibility computed by modified-DP() and by DPgoodmap(), respectively.

We prove this by induction w.r.t. sensible-pairs from super-levels of increasing indices. At the base case when the index of super-level $i = 1$, it is easy to check that a valid pair has to be sensible – in fact, S and w are either empty or contains only tree-nodes from T_1^f and T_2^g.

Now consider a sensible-pair (S, w) from the i-th super-level. Let $\mathrm{Ch}(S)$ and $\mathrm{Ch}(w)$ be as defined in modified-DP() (which is the same as in DPgoodmap()). For any partition $\mathrm{Ch}(S) = S_1 \cup \cdots S_k$ of S, we only need to show that condition (F-1) holds if and only if condition (modified F-1) holds. Furthermore, the only case we need to consider is when S_j is valid but the pair (S_j, w_j) is not sensible. Note that in this case, $F_{new}(S_j, w_j)$ could be set to $F_{new}(S', w')$ as described in modified-DP(). As in the algorithm, let A denote the edge-list supporting S_j, and $\alpha = (z_L, z_H)$, with $g(z_L) \leq g(z_H)$, is the edge from T_2^g containing w_j. We now prove the following two claims:

(Claim-1) First we show that if $F_{new}(S_j, w_j) = 1$; then it must be that $F_{old}(S_j, w_j) = 1$. In this case, (S', w') must exist when running Algorithm modified-DP(), and it is the highest sensible-pair supported by edge-list pair (A, α) but below S_j; with $F_{new}(S', w') = 1$. We claim that all augmented-tree nodes in S' and augmented-tree node w' must all be of degree-2 (i.e, they are all in the interior of some tree arcs of the original trees T_1^f and T_2^g, and none of them is a tree node for T_1^f or T_2).

Indeed, suppose this is not the case and some augmented-tree nodes in S' or w' is in fact a tree node. Then, as (S', w') is supported by (A, α), by the proof of Lemma 7, the only possibility is that $w' = z_L$ or S' contains u_L, where u_L is the **highest** lower-endpoint of all edges in A. Suppose S' is from super-level $\mathrm{L}_c^{(1)}$. Then consider its parents S'' from super-level $\mathrm{L}_{c+1}^{(1)}$ as well as the parent w'' of w' from $\mathrm{L}_{c+1}^{(2)}$. Since (S', w') (coming from c-th super-level) is below (S_j, w_j) which comes from the $(i-1)$-th super-level, we have that $c + 1 \leq i - 1$. It then follows that S'' must be supported by the same edge-list A, and similarly, $w' \prec w'' \preceq w_j$ along edge $\alpha = (z_L, z_H)$. As S' is valid, S'' must be valid. As S' and w' contain a tree node of T_1^f or T_2^g, (S'', w'') must be sensible (satisfying either condition (C-1) or (C-2) in Definition 6). This however contradicts that (S', w') is the highest sensible-pair below (S_j, w_j) supported by edge-list pair (A, α). Hence the assumption is wrong and all points in S' and w' must be in the interior of some tree arcs.

Since (S', w') is a sensible-pair and is from a lower super-level than (S, w), by the induction hypothesis, we already have that $F_{old}(S', w') = F_{new}(S', w')$, meaning that $F_{old}(S', w') = 1$. Now during the execution of Algorithm DPgoodmap(), it will inspect a sequence of **valid pairs**

$$(S', w') = (S_j^{(0)}, w_j^{(0)}), (S_j^{(1)}, w_j^{(1)}), \ldots, (S_j^{(t)}, w_j^{(t)}) = (S_j, w_j),$$

listed in increasing heights, supported by edge-list pair (A, α). As S' is valid, and all $S_j^{(b)}$'s, for $b \in [1, t]$, are supported on the same edge-list, each $(S_j^{(b)}, w_j^{(b)})$ must be valid as well. Furthermore, as $w_j^{(b)}$ is the only child of $w_j^{(b+1)}$ for any $b \in [0, t-1]$ (they are both contained in the interior of edge α), it then follows that during the execution of Algorithm DPgoodmap(), $F_{old}(S_j^{(b+1)}, w_j^{(b+1)})$

is set to be $F_{old}(S_j^{(b)}, w_j^{(b)})$ for each $b \in [0, t-1]$. Hence

$$(F_{old}(S_j, w_j) =)\ F_{old}(S_j^{(t)}, w_j^{(t)}) = \cdots = F_{old}(S_j^{(1)}, w_j^{(1)}) = F_{old}(S_j^{(0)}, w_j^{(0)})\ (= F_{old}(S', w')).$$

Since $F_{old}(S', w') = 1$, it thus follows that $F_{old}(S_j, w_j) = 1$, establising (Claim-1).

(Claim-2) Next we show that if $F_{old}(S_j, w_j) = 1$, then $F_{new}(S_j, w_j) = 1$. In this case, let $(S_j^{(0)}, w_j^{(0)}), (S_j^{(1)}, w_j^{(1)}), \ldots, (S_j^{(t)}, w_j^{(t)}) = (S_j, w_j)$ denote the sequence of all pairs (not necessarily valid) which are: (i) supported by the edge-list pair (A, α), (ii) at or below (S_j, w_j); and (iii) strictly higher than the super-levels containing u_L and z_L. We also assume that this sequence is listed in increasing heights.

Assume that the higher super-level containing either u_L or z_L is the c-th super-level. Then (iii) above implies that (1) all (augmented-tree) nodes in $(S_j^{(b)}, w_j^{(b)})$, for all $b \in [0, t]$, are degree-2 nodes (i.e, not tree nodes); (2) $(S_j^{(0)}, w_j^{(0)})$ must come from the $(c+1)$-th super-levels $\mathrm{L}_{c+1}^{(1)}$ and $\mathrm{L}_{c+1}^{(2)}$; and (3) no $(S_j^{(b)}, w_j^{(b)})$, for $b \in [1, t-1]$, can be sensible. Note that (2) implies that $(S_j^{(0)}, w_j^{(0)})$ satisfies either condition (C-1) or (C-2) of Definition 6 (although we have not yet shown it is valid, and thus we do not know whether it is sensible or not yet).

We argue that if $F_{old}(S_j, w_j) = 1$, then $F_{old}(S_j^{(b)}, w_j^{(b)}) = 1$ for every $b \in [0, t)$. This is because in Algorithm DPgoodmap(), $F_{old}(S_j^{(b)}, w_j^{(b)})$ is set to be $F_{old}(S_j^{(b-1)}, w_j^{(b-1)})$ for each $b \in [1, t]$, as $w_j^{(b-1)}$ is the only child of $w_j^{(b)}$ for $b \in [1, t]$. It then follows that $F_{old}(S_j^{(0)}, w_j^{(0)}) = 1$, meaning $(S_j^{(0)}, w_j^{(0)})$ must be a valid pair. Combined with the fact that $(S_j^{(0)}, w_j^{(0)})$ satisfies either condition (C-1) or (C-2) of Definition 6 as shown in the previous paragraph, we have that $(S_j^{(0)}, w_j^{(0)})$ is a sensible-pair. Furthermore, by (3) from the previous paragraph, $(S_j^{(0)}, w_j^{(0)})$ has to be the highest sensible-pair supported by edge-list pair (A, α). Hence Alorithm modified-DP() will set $(S', w') = (S_j^{(0)}, w_j^{(0)})$. By our induction hypothesis, we have $F_{new}(S', w') = F_{old}(S', w')$, implying that the modified algorithm will set $F_{new}(S_j, w_j) = F_{new}(S', w') = F_{old}(S', w') = 1$. This proves (Claim-2).

Combining (Claim-1) and (Claim-2) above, we have that for any sensible-pair (S, w), $F_{new}(S, w) = F_{old}(S, w)$. It then follows that Algorithm modified-DP() reutnrs the same answer as Algorithm DP-goodmap(), which, combined with part (i) of Theorem 2, establishes the correctness of Algorithm modified-DP().

Proof of Part (ii) of Theorem 3 We now show that Algorithm modified-DP() can be implemented to run in the claimed time complexity.

First, Lemma 8 shows that we can compute all sensible-pairs, as well as the set of edge-list pairs Ξ supporting them, in time $O(n^2 2^\tau)$ time. We store all sensible-pairs in increasing order of their super-levels, and process it one by one. For each edge-list pair (A, α) with $A \subseteq E(T_1^f)$ and $\alpha \in E(T_2^g)$, we link to it the four sensible-pairs it supports. For each sensible-pair, it also stores a pointer, linking to the edge-list pair supporting it.

In step (2) when computing $F(S, w)$, we need to be able to search whether a pair (S_j, w_j), with $S_j \subseteq \mathrm{Ch}(S)$ and $w_j \in \mathrm{Ch}(w)$, is sensible or not, and identify (S', w') through the edge-list pair supporting (S_j, w_j) when necessary. This can be done by storing all edge-list pairs in Ξ in some data structure that supports search efficiently. To this end, we view each edge-list pair (A, α) as a set of ordered edge indices $[id_1, \ldots, id_t]$, where $id_1, \ldots, id_{t-1}$ are indices for edges in A from T_1^f

in increasing order, while id_t is index for edge α in T_2^g. Given the collection of all edge-list pairs Ξ that support some sensible-pairs, we first use a standard balanced binary search tree Π to store all indices occured in the first position of the ordered index-sets representation of edge-list pair in Ξ. Next, for each index, say id_1 stored in this tree Π, we then associate to it a secondary data structure (another balanced binary search tree Π_{id_1}) to store the set of second indices from edge-list pairs of the form $[id_1, \ldots]$. We then repeat, and build a third-level tree for each index in the secondary tree Π_{id_1}, say id_2, to store all indices in the 3rd position from edge-list pairs of the form $[id_1, id_2, \ldots]$. Let ℓ be the largest cardinality of any edge-list pair. Then this process is repeated at most ℓ times to build auxiliary trees of at most ℓ levels. Each auxiliary tree has size at most n as the choice of each position is bounded by the total number of edges in each input tree.

By Lemma 2, $|S| \leq \tau$ for any valid pair (S, w). Hence $\ell \leq \tau + 1$. This data structure takes $O(\ell \cdot |\Xi|) = O(n^2 \tau 2^\tau)$ space, since each index of every edge-list pair will only be stored at most once. The data structure can be built in $O(n^2 \tau 2^\tau \log n)$ time. Finally, to search for a specific edge-list pair (represented by the ordered-index set) $[id_1, \ldots, id_t]$, we need to perform t searches in t balanced binary search trees each of which is of size at most n. Hence a search for an edge-list pair takes $O(t \log n) = O(\tau \log n)$ time.

Finally, we analyze the time complexity to compute $F(S, w)$ for a fixed sensible-pair (S, w) in step (2) of Algorithm modified-DP(). First, observe that $k = |\mathrm{Ch}(w)| = degree(w) \leq \tau$, and $|\mathrm{Ch}(S)| \leq \tau$ by Lemma 2. Hence the number of partitioning of $\mathrm{Ch}(S)$ our algorithm will check is bounded by $O(|\mathrm{Ch}(S)|^k) = O(\tau^\tau)$. For each partition $S = S_1 \cup \cdots S_k$, and for each $j \in [1, k]$, it takes $O(\tau \log n)$ time to search for (S', w') as required in (modified F-1). Hence overall, it takes $O(k\tau \log n) = O(\tau^2 \log n)$ to check conditions (modified F-1) or (F-2) for all $j \in [1, k]$. Putting everything together, we have that the time complexity of our algorithm is bounded from above by $O(n^2 2^\tau \tau^{\tau+2} \log n)$, as claimed. This finishes the proof of Part (ii) of Theorem 3.

F Proof of Lemma 4

Imagine we run the dynamic programming algorithm DPgoodmap(T_1^f, T_2^g, δ^*) as introduced in the previous section. It should return 'yes'. In particular, consider the augmented trees $\widehat{T}_1^f$ and $\widehat{T}_2^g$ we construct w.r.t parameter δ^* as in Section 4.1, where $\widehat{T}_1^f$ (resp. $\widehat{T}_2^g$) is augmented from T_1^f (resp. T_2^g) by including all points in the super-levels also as tree nodes. Algorithm DPgoodmap(T_1^f, T_2^g, δ^*) will compute the feasibility $F(S, w)$ for all valid pairs (S, w) in a bottom-up manner, where S and w are from those super-levels $\mathrm{L}_i^{(1)}$ and $\mathrm{L}_i^{(2)}$ respectively. In the end, it will return $F(root(\widehat{T}_1^f), root(\widehat{T}_2^g))$, which should be equal to '1' in this case.

Now imagine that we will decrease δ^* by an infinitesimally small quantity $\nu > 0$ to $\delta' = \delta^* - \nu$. Algorithm DPgoodmap(T_1^f, T_2^g, δ') should return 'no' since δ^* is the smallest possible value on which the dynamic programming algorithm returns 'yes'. Recall that the super-levels w.r.t. a generic δ value are defined as follows:

$$\mathcal{L}_1 := \{L(c) \mid c \in \mathrm{C}_1\} \ \cup \ \{L(c - \delta) \mid c \in \mathrm{C}_2\} \quad \text{and}$$
$$\mathcal{L}_2 := \{L(c + \delta) \mid c \in \mathrm{C}_1\} \ \cup \ \{L(c) \mid c \in \mathrm{C}_2\},$$

where C_1 (resp. C_2) contains the f-function values of tree nodes in T_1^f (resp. g-values of tree nodes in T_2^g). Half of these super-levels are independent of the value of δ; these are levels in $\{L(c) \mid c \in \mathrm{C}_1\} \subset \mathcal{L}_1$ for T_1^f and $\{L(c) \mid c \in \mathrm{C}_2\} \subset \mathcal{L}_2$ for T_2^t, which pass through tree nodes in

T_1^f and T_2^g, respectively. We call these super-levels as *fixed super-levels.* The other half are *induced super-levels* and they are at height either δ above or below some fixed super-levels.

Now consider the super-levels w.r.t. δ^*. If any induced super-level, say $\mathrm{L}_i^{(1)} \in \mathcal{L}_1$, also passes through a tree node, say v, in T_1^f. This means that $\delta^* \in \Pi_1$: Indeed, since $\mathrm{L}_i^{(1)}$ is induced, it is of the form $\mathrm{L}_i^{(1)} = L(c - \delta^*)$ with $c = g(w) \in \mathrm{C}_2$ for some $w \in V(T_2^g)$. Then $\delta^* = g(w) - f(v)$ with $v \in V(T_1^f)$ and $w \in V(T_2^g)$. Thus $\delta^* \in \Pi_1$ and the lemma holds.

Hence from now on we assume that no induced super-level w.r.t δ^* passes through any tree node in T_1^f nor in T_2^g. As we decrease the value of δ^* slightly to δ', the induced super-levels move (up or down accordingly) by the same amount $\nu = \delta^* - \delta'$. Since no induced super-level contains any tree node in T_1^f or T_2^g, for any induced super-level L, each point in it only moves up or down in the interior of some tree edge of T_1^f or T_2^g, and no new point can appear in a super-level. Hence there is a canonical bijection between the tree nodes $V(\widehat{T}_1^f(\delta^*))$ (resp. $V(\widehat{T}_2^g(\delta^*))$) in the augmented tree $\widehat{T}_1^f(\delta^*)$ w.r.t. δ^*, and $V(\widehat{T}_1^f(\delta'))$ (resp. $V(\widehat{T}_2^g(\delta'))$) w.r.t. δ'. (This bijection also induces a bijection of points (augmented tree nodes) within the i-th super-level w.r.t δ^* and points in corresponding i-th super-level w.r.t δ'.) Under this bijection, we use $S(\delta')$ to denote the set of tree nodes in $\widehat{T}_1^f(\delta')$ (resp. in $\widehat{T}_2^g(\delta')$) corresponding to $S \subset V(\widehat{T}_1^f(\delta^*))$ (resp. $S \subset V(\widehat{T}_2^g(\delta^*))$).

Since $F(root(\widehat{T}_1^f(\delta')), root(\widehat{T}_2^g(\delta'))) =$ 'no', it means that during the bottom-up process of algorithm $\mathsf{DPgoodmap}(T_1^f, T_2^g, \delta')$, at some point $F_{\delta^*}(S, w) = 1$ (w.r.t. parameter δ^*), yet $F_{\delta'}(S(\delta'), w(\delta')) = 0$ (w.r.t. parameter δ'). Let (S, w) be such a pair from the lowest super-level that this happens.

This could be caused by the following reasons:

(1) $(S(\delta'), w(\delta'))$ is no longer a valid pair. This means that $u = LCA(S)$ (which is necessarily a tree node in $V(T_1^f)$) must be at height $h + 2\delta^*$, where $h = f(S)$. If S is from a fixed super-level, then there exists a tree node $v \in V(T_1^f)$ such that $f(v) = h$. It then follows that $2\delta^* = f(u) - f(v)$, implying that $\delta^* \in \Pi_2$. Otherwise, S is from an induced super-level $\mathrm{L}_i^{(1)}$. This means that there exists a tree node $w' \in V(T_2^g)$ such that $h = g(w') - \delta^*$. It then follows that $f(u) - g(w') = h + 2\delta^* - (h + \delta^*) = \delta^*$, implying that $\delta^* \in \Pi_1$.

(2) $(S(\delta'), w(\delta'))$ is still valid, $F_{\delta^*}(S, w) = 1$ but $F_{\delta'}(S(\delta'), w(\delta')) = 0$. There are two cases how $F_{\delta'}(S(\delta'), w(\delta')) = 0$ in algorithm $\mathsf{DPgoodmap}(T_1^f, T_2^g, \delta')$.

  (2.a) $\mathrm{Ch}(S(\delta')) \neq \emptyset$ but $\mathrm{Ch}(w(\delta')) = \emptyset$. This cannot happen as by the bijective relation between augmented tree nodes $V(\widehat{T}_1^f(\delta'))$ and $V(\widehat{T}_2^g(\delta'))$, this would have implied that $\mathrm{Ch}(S(\delta^*)) \neq \emptyset$ but $\mathrm{Ch}(w(\delta^*)) = \emptyset$. That is, $F(S, w) = 0$ in algorithm $\mathsf{DPgoodmap}(T_1^f, T_2^g, \delta^*)$ as well, which contradicts the assumption on (S, w).

  (2.b) Otherwise, since (S, w) is from the lowest level s.t. $F_{\delta^*}(S, w) = 1$ (w.r.t. parameter δ^*), yet $F_{\delta'}(S(\delta'), w(\delta')) = 0$ (w.r.t. parameter δ'), condition (F-2) must have failed when we process $(S(\delta'), w(\delta')$ in algorithm $\mathsf{DPgoodmap}(T_1^f, T_2^g, \delta')$. In other words, there exists a child w_j of w such that $depth(w_j) = 2\delta^* - (g(w) - g(w_j))$, as condition (F-2) will be violated as we decrease δ^* to δ'. Thus there must be a leaf node $\hat{w}$ in the subtree rooted at w_j so that $g(w_j) - g(\hat{w}) = depth(w_j)$, and thus $g(w) - g(\hat{w}) = 2\delta^*$. Assume that S is from $\mathrm{L}_i^{(1)}$ (and thus w is from $\mathrm{L}_i^{(2)}$). If super-level $\mathrm{L}_i^{(2)}$ contains a tree node of T_2^g, then this implies that $\delta^* \in \Pi_3$. If not, then the super-level $\mathrm{L}_i^{(2)}$ must be an induced super-level, meaning that there is a tree node $u \in V(T_1^f)$ such that $g(w) = \hat{h}_i = f(u) + \delta^*$. It then follows that $f(u) - g(\hat{w}) = \delta^*$ and thus $\delta^* \in \Pi_1$.

In all cases, we note that $\delta^* \in \Pi$. Lemma 4 then follows.

G Proof of Theorem 5

First, we compute and sort all candidate values in the set Π as in Section 5.1. Set $m = |\Pi|$ and let $\Delta = \{\delta_1 < \delta_2 < \cdots \delta_m\}$ be the ordered sequence of candidate values in Π.

On the other hand, observe that the degree-bound parameter $\tau_\delta(T_1^f, T_2^g)$ can take at most $O(n)$ integer values: $1, 2, 3, \ldots, 2n$, where n is the total number of nodes in the input trees. Set $\tau_k = k$, with $k \in \{1, 2, \ldots, 2n\}$. It is easy to see that the set of $\delta \in \Delta$ such that $\tau_\delta = \tau_k$ forms a consecutive subsequence of Δ, which we denote by $[\delta_{\ell_k}, \ldots, \delta_{r_k}]$. That is, for any $j \in [\ell_k, r_k]$, $\tau_{\delta_j} = \tau_k$.

For a fixed τ, for simplicity, we set $Time(n, \tau) := n^2 2^\tau \tau^{\tau+2} \log n$; by Theorem 3, $O(Time(n, \tau))$ is the time to solve the decision problem "Is $d_I(T_1^f, T_2^g) \le \delta$ where δ is any value such that its δ-degree-bound τ_δ equal to τ; that is, $\tau_\delta = \tau$.

Below, we first show that for any τ_k, we can identify the smallest $\delta \in \Delta$ *valid for* τ_k defined as (i) $\tau_\delta = \tau_k$, and (ii) $d_I(T_1^f, T_2^g) \le \delta$; or returns "Null" if such a δ does not exist. We call this procedure SmallestValidDelta(τ_k).

Procedure SmallestValidDelta(τ_k). First, we run the decision problem for $d_I(T_1^f, T_2^g) \le \delta_{r_k}$. If the answer is 'no', then no δ can be valid for τ_k, and we return "Null". Otherwise, we perform a binary search within the range $[\ell_k, r_k]$ to search for the smallest index $i \in [\ell_k, r_k]$ such that δ_i is valid for τ_k, and return δ_i. Overall, we execute the decision problem $O(\log |r_k - \ell_k + 1|) = O(\log n)$ times. For each of the δ value tested during this course, we have $\tau_\delta = \tau_k$. Hence by Theorem 3 the overall time complexity for this procedure is $O(Time(n, \tau) \cdot \log n)$.

Identifying optimal δ^*. Now to prove Theorem 5, we need to identify the smallest $\delta^* \in \Delta$ where the decision problem returns 'yes'. Let $\tau_{k^*} = \tau_{\delta^*}$ be the δ^*-degree bound parameter. If we can identify k^*, then Procedure SmallestValidDelta(τ_k) will computes δ^*.

On the other hand, k^* is the smallest k value such that Procedure SmallestValidDelta(τ_k) returns 'yes'. We compute k^* by a standard binary+exponential search procedure on $k \in \{1, 2, \ldots, 2n\}$. Specifically, starting with $k = 1$, we check whether Procedure SmallestValidDelta(τ_k) returns "Null" or not. If yes, then we double the size of k (i.e, in the i-th iteration, we will be checking for $k = 2^i$), till we reach the first k' such that the procedure returns a valid δ value. This means that the correct k^* must be from $[k'/2, k']$, which we further identify by a binary search within this range.

The time needed for the first "exponential" search of $k' = 2^{i'}$ is

$$\begin{aligned} &Time(n,1)\log n + Time(n,2)\log n + Time(n,4)\log n + \cdots + Time(n,2^{i'})\log n \\ =&O(Time(n,k')\log n), \end{aligned}$$

as the last term $Time(n, 2^{i'}) \log n = Time(n, k') \log n$ dominates the sum of this geometric-like series.

For the second binary search within the range $[k'/2, k']$, it takes $O(Time(n, k') \log n \log k')$. As $k' \le 2k^*$, the total time is thus $O(Time(n, 2k^*) \log^2 n)$, as claimed in Theorem 5.

H Proof of Theorem 6

By Claim 1, if we can compute $\mu = \min_{u\in V(T_1), w\in V(T_2)} d_I(T_1^{f_u}, T_2^{g_w})$, where $f_u : |T_1| \to \mathbb{R}$ (resp. $g_w : |T_2| \to \mathbb{R}$) is the distance function to point u (resp. w) in $|T_1|$ (resp. in $|T_2|$), then μ is a 14-approximation of $\hat{\delta^*} = \delta_{\mathcal{GH}}(\mathcal{T}_1, \mathcal{T}_2)$; that is, $\mu/14 \le \hat{\delta^*} \le 14\mu$ (the upper bound is in fact 2μ by Claim 1).

There are two issues that we need to address. First, consider an arbitrary pair of nodes $u \in V(T_1)$ and $w \in V(T_2)$. Our DP-algorithm to compute $d_I(T_1^{f_u}, T_2^{g_w})$ depends on the degree-bound $\tau_\delta(T_1^{f_u}, T_2^{g_w})$ parameter for the two merge trees $T_1^{f_u}$ and $T_2^{g_w}$. How does this relate to the metric-degree-bound $\widehat{\tau}_\delta(T_1, T_2)$? We claim:

Claim 4. $\widehat{\tau}_\delta \le \tau_\delta \le \widehat{\tau}_{2\delta}$.

Proof. It is easy to see that for any $x \in |T_1|$, $\widehat{B}_\delta(x, T_1) \subseteq B_\delta(x, T_1^{f_u})$, where $\widehat{B}_\delta$ is the geodesic ball w.r.t. metric d_1, while B_δ is the δ-ball defined at the beginning of Section 4 for the merge tree T_1^f. A symmetric statement hold for the two type of balls in T_2 and $T_2^{g_w}$. This implies that $\widehat{\tau}_\delta \le \tau_\delta$. We now show that for each $B_\delta(x, T_1^{f_u})$, there exists some $y \in |T_1|$ such that $B_\delta(x, T_1^{f_u}) \subseteq \widehat{B}_{2\delta}(y, T_1)$.

Indeed, consider any $B_\delta(x, T_1^{f_u})$: First, observe that by definition of the δ-ball $B_\delta(x, T_1^{f_u})$ (see the beginning of Section 4), any point $z \in B_\delta(x, T_1^{f_u})$ is connected to x and $f_u(z) \in [f_u(x)-\delta, f_u(x)+\delta]$. Set y be the point this ball with highest f_u function value. As $B_\delta(x, T_1^{f_u})$ is a connected subset of a tree, there is a monotone path $\pi(y, z)$ from this highest point y in it to every point $z \in B_\delta(x, T_1^{f_u})$. Since f_u is the shortest path distance to point $u \in T_1$ (which is the root of the merge tree $T_1^{f_u}$), and $\pi(y, z)$ is monotone, we thus have that $f_u(z) = f_u(y) + d_1(z, y)$. It then follows that $d_1(z, y) = f_u(y) - f_u(z) \le 2\delta$ for any point $z \in B_\delta(x, T_1^{f_u})$. Hence $B_\delta(x, T_1^{f_u}) \subseteq \widehat{B}_{2\delta}(y, T_1)$ as claimed.

A symmetric statement holds for balls for T_2 and $T_2^{g_w}$. Hence $\tau_\delta \le \widehat{\tau}_{2\delta}$. $\square$

Second, observe that in general, $d_I(T_1^{f_u}, T_2^{g_w})$ could be much larger than μ. This means that the time complexity of our DP algorithm to compute $d_I(T_1^{f_u}, T_2^{g_w})$ may not be dependent on τ_μ (and thus $\widehat{\tau}_{2\mu}$) any more. Hence to compute $\mu = \min_{u\in V(T_1), w\in V(T_2)} d_I(T_1^{f_u}, T_2^{g_w})$, we cannot afford to compute every $d_I(T_1^{f_u}, T_2^{g_w})$ for each pair of $u \in V(T_1)$ and $w \in V(T_2)$. Instead, we now enumerate all candidate sets $\widehat{\Pi} = \cup_{u\in V(T_1), w\in V(T_2)} \Pi_{u,w}$, where $\Pi_{u,w}$ denotes the candidate set of choices for δ w.r.t $T_1^{f_u}$ and $T_2^{g_w}$, for all $O(n^2)$ pairs of $u \in V(T_1)$ and $w \in V(T_2)$. We next sort all parameters $\delta_{u,w}$ in $\widehat{\Pi}$: the subscript u, w in $\delta_{u,w}$ means that this value is a potential δ value for interleaving distance $d_I(T_1^{f_u}, T_2^{g_w})$. Note that The total number of critical values in $\widehat{\Pi}$ is $O(n^2 \times n^2) = O(n^4)$. This takes $O(n^4 \log n)$ time.

Next, we perform a similar double-binary search procedure (once for the optimal degree-bound parameter τ, and once for δ) as in the proof of Theorem 5 to identify the smallest $\delta_{u,w}$ such that $d_I(T_1^{f_u}, T_2^{g_w}) \le \delta_{u,w}$ for any $u \in V(T_1)$ and $w \in V(T_2)$. Let $\tau' = \tau_{\delta_{u,w}}(T_1^{f_u}, T_2^{g_w})$ be the $\delta_{u,w}$-degree bound for merge trees f_u and g_w. This means that during the procedure, all the parameter τ values we ever tested are at most $2\tau'$, meaning that for each decision problem we ever tested, its time complexity is bounded by $O(Time(n, 2\tau') \log^2 n)$, where $Time(n, \tau) := n^2 2^\tau \tau^{\tau+2} \log n$ as in the proof of Theorem 5. Hence the total time complexity of this double-binary search procedure is $O(Time(n, 2\tau') \log^2 n)$.

On the other hand, by Claim 1, we have $\delta_{u,w}(= \min_{u\in V(T_1),w\in V(T_2)} d_I(T_1^{f_u}, T_2^{g_w})) \leq 14\hat{\delta^*}$. It then follows from Claim 4 that

$$\tau' = \tau_{\delta_{u,w}} \leq \tau_{14\hat{\delta^*}} \leq \widehat{\tau}_{28\hat{\delta^*}}.$$

Setting $\widehat{\tau} = 2\widehat{\tau}_{28\hat{\delta^*}}$, the total time for performing the double-binary search procedure is

$$O(Time(n, 2\tau') \log^2 n) = O(n^2 2^{\widehat{\tau}} \widehat{\tau}^{\widehat{\tau}+2} \log^3 n),$$

which, together the $O(n^4 \log n)$ time to compute and sort all candidate values in $\widehat{\Pi}$, gives the time complexity as claimed in Theorem 6.