FPT-Algorithms for Computing Gromov-Hausdorff and Interleaving Distances Between Trees

Elena Farahbakhsh Touli[1]
Stockholm University, Sweden
https://www.su.se/english/profiles/elfa1534-1.428605
elena.touli@math.su.se

Yusu Wang[1]
The Ohio State University, USA
https://web.cse.ohio-state.edu/~wang.1016/
yusu@cse.ohio-state.edu

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.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Gromov-Hausdorff distance, Interleaving distance, Merge trees

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.83

Funding 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.

Acknowledgements 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 19), which further leads to a similar improvement for approximating the Gromov-Hausdorff distance (Theorem 20).

[1] Corresponding author

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 20. 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[2] 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.

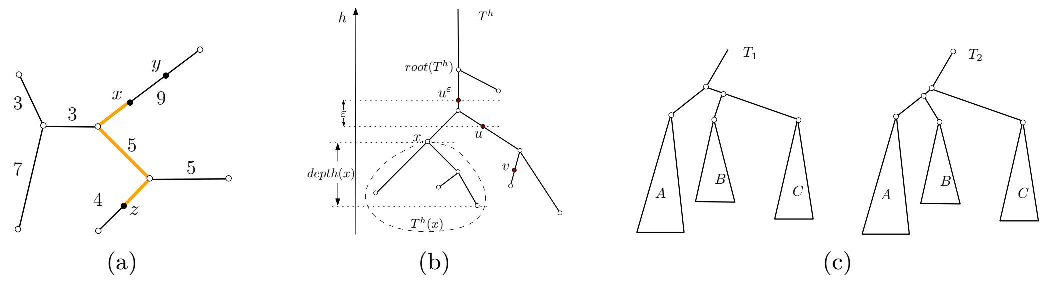

Figure 1 (a) A metric tree (T, d_T) with edge length marked. Tree nodes are white dots. $d_T(x, z) = 3 + 5 + 2 = 10$ is the length of the thickened path $\pi(x, z)$. (b) A merge tree T^h, with examples of $u \succ v$, u^ε, $T^h(x)$ and $depth(x)$ marked. (c) Tree alignment distance between T_1 and T_2 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 un-ordered trees [18, 26]. 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 $\mathbb{R}^3$.

[2] We note that the final time complexity for the optimization problem presented in Theorem 19 is based on an argument by Kyle Fox. His argument improves our previous n^4 factor (as in Theorem 18) by an almost n^2 factor, by performing a double-binary search, instead of a sequence search we originally used.

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 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[3] 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:

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

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

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 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:*

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

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:*

$$d_I(T_1^f, T_2^f) = \inf\{\ \varepsilon \ \mid \ there\ exist\ a\ pair\ of\ \varepsilon\text{-}compatible\ maps\ w.r.t\ T_1^f\ and\ T_2^g\}. \qquad (1)$$

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 3 ([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 4.** *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 5** (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 6.** *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 [25].

▶ **Theorem 7.** *Given any two merge trees T_1^f and T_2^g, then $d_I(T_1^f, T_2^g) \le \varepsilon$ if and only if there exists an ε-good map $\alpha : |T_1^f| \to |T_2^g|$.*

4 Decision Problem for Interleaving Distance

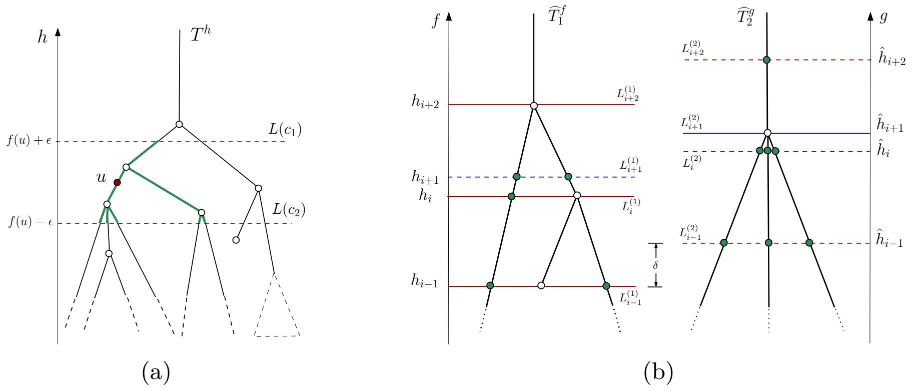

Figure 2 (a) Green component within the slab is $B_\varepsilon(u, T_1^f)$. The sum of degrees for nodes within this ε-ball is 13. The ε-degree bound $\tau_\varepsilon(T_1^f, T_2^g)$ is the largest value of this sum for any ε-ball in T_1^f or in T_2^g. (b) White dots are tree nodes of T_1^f and T_2^g. 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 $d_I(T_1^f, T_2^g) \le \delta$?". 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

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

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 $d_I(T_1^f, T_2^g) \le \delta$?" 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 8** (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,*

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

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:*

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

Now sort all levels in $\mathcal{L}_i$ in increasing order of their heights, denoted by $\mathcal{L}_1 = \{\mathrm{L}_1^{(1)}, \mathrm{L}_2^{(1)}, \ldots, \mathrm{L}_m^{(1)}\}$ and $\mathcal{L}_2 = \{\mathrm{L}_1^{(2)}, \ldots, \mathrm{L}_m^{(2)}\}$, 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 $h_1, \ldots, h_m$ be the sequence of height values for $\mathrm{L}_1^{(1)}, \mathrm{L}_2^{(1)}, \ldots, \mathrm{L}_m^{(1)}$; that is, $h_i = \textit{height}(\mathrm{L}_i^{(1)})$. Similarly, let $\widehat{h}_1, \widehat{h}_2, \ldots, \widehat{h}_m$ 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 *root*$(\widehat{T}_1^f)$ and *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 9** (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 1 DPgoodmap(T_1^f, T_2^g, δ).

Base case ($i = 1$): For each valid-pair (S, w) from level-1, set $F(S, w) = 1$ ("true") if and only if $depth(w) \leq 2\delta$; 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 $\mathrm{Ch}(S) \subseteq \mathrm{L}^{(1)}_{i-1}$ of nodes in S, and w's children $\mathrm{Ch}(w) = \{w_1, \ldots, w_k\}$ in $\mathrm{L}^{(2)}_{i-1}$.
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 $depth(w_j) \leq 2\delta - (\widehat{h}_i - \widehat{h}_{i-1})$; note that this implies that $\widehat{h}_i - \widehat{h}_{i-1} \leq 2\delta$ in this case.

Output: DPgoodmap(T_1^f, T_2^g, δ) 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 $V(\mathrm{L}^{(1)}_m)$ (resp. $V(\mathrm{L}^{(2)}_m)$).

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 10.**

(i) *Algorithm DPgoodmap(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 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 10.

Part (i) of Theorem 10: correctness

We first show the correctness of algorithm 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 5 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 $f(u_1^{2\varepsilon}) = f(u_1) + 2\varepsilon \geq \mathit{height}(S)$. First, we observe the following:

▷ Claim 11. At the top level where $\mathrm{L}_m^{(1)} = \{u = \mathit{root}(\widehat{T}_1^f)\}$ and $\mathrm{L}_m^{(2)} = \{w = \mathit{root}(\widehat{T}_2^g)\}$ 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 10) will follow from Claim 11 and Lemma 12 below. Lemma 12 is one of our key techincal results, and its proof can be found in [25].

▶ **Lemma 12.** *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 10: 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 [25].

▶ **Lemma 13.** *For any valid pair (S, w), we have $|S| \leq \tau$ and $|\mathrm{Ch}(S)| \leq \tau$, 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 14.** *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 $k = |\mathrm{Ch}(w)| = \mathit{degree}(w) \leq \tau$, and $|\mathrm{Ch}(S)| \leq \tau$ by Lemma 13. 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 14, 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 14. We can consider only what we call sensible-pairs, which we define now.

▶ **Definition 15.** *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 [25]. We only summarize the main theorem below.

▶ **Theorem 16.**

(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 [25].

▶ **Lemma 17.** *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 16, we can easily obtain the following trivial bound:

▶ **Theorem 18.** *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 [25] for the proof.

▶ **Theorem 19.** *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 [25].

▶ **Theorem 20.** *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).

References

1 Pankaj K. Agarwal, Kyle Fox, Abhinandan Nath, Anastasios Sidiropoulos, and Yusu Wang. Computing the Gromov-Hausdorff Distance for Metric Trees. *ACM Trans. Algorithms*, 14(2):24:1–24:20, 2018.

2 Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM Journal on Computing*, 28(3):1073–1085, 1998.

3 Nir Ailon and Moses 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 Noga Alon, Mihai Bădoiu, Erik D Demaine, Martin Farach-Colton, MohammadTaghi Hajiaghayi, and Anastasios Sidiropoulos. Ordinal embeddings of minimum relaxation: general properties, trees, and ultrametrics. *ACM Transactions on Algorithms (TALG)*, 4(4):46, 2008.

5 Ulrich Bauer, Xiaoyin Ge, and Yusu Wang. Measuring Distance between Reeb Graphs. In *30th Annual Sympos. on Comput. Geom.*, page 464, 2014.

6 Ulrich Bauer, Claudia Landi, and Facundo Mémoli. The Reeb Graph Edit Distance is Universal. *CoRR*, abs/1801.01866, 2018. `arXiv:1801.01866`.

7 Silvia Biasotti, Daniela Giorgi, Michela Spagnuolo, and Bianca Falcidieno. Reeb graphs for shape analysis and applications. *Theor. Comput. Sci.*, 392(1-3):5–22, 2008.

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

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

10 Mihai Bădoiu, Piotr Indyk, and Anastasios 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 Victor Chepoi, Feodor F Dragan, Ilan Newman, Yuri Rabinovich, and Yann Vaxes. Constant approximation algorithms for embedding graph metrics into trees and outerplanar graphs. *Discrete & Computational Geometry*, 47(1):187–214, 2012.

12 Vin de Silva, Elizabeth Munch, and Amit Patel. Categorified Reeb Graphs. *Discrete & Computational Geometry*, 55(4):854–906, June 2016.

13 Michael Fellows, Fedor Fomin, Daniel Lokshtanov, Elena Losievskaja, Frances A Rosamond, and Saket Saurabh. Parameterized Low-distortion Embeddings-Graph metrics into lines and trees. *arXiv preprint*, 2008. `arXiv:0804.3028`.

14 Xiaoyin Ge, Issam Safa, Mikhail Belkin, and Yusu Wang. Data Skeletonization via Reeb Graphs. In *Proc. 25th Annu. Conf. Neural Information Processing Systems (NIPS)*, pages 837–845, 2011.

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

16 Alexander Hall and Christos 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 Franck Hétroy and Dominique Attali. Topological quadrangulations of closed triangulated surfaces using the Reeb graph. *Graph. Models*, 65(1-3):131–148, 2003.

18 Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of Trees - An Alternative to Tree Edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995. `doi:10.1016/0304-3975(95)80029-9`.

19 Facundo Mémoli and Guillermo Sapiro. A Theoretical and Computational Framework for Isometry Invariant Recognition of Point Cloud Data. *Found. of Comput. Math.*, 5(3):313–347, 2005.

20 Dmitriy Morozov, Kenes Beketayev, and Gunther H. Weber. Interleaving Distance between Merge Trees. In *Workshop on Topological Methods in Data Analysis and Visualization: Theory, Algorithms and Applications*, 2013.

21 Facundo Mémoli. Some Properties of Gromov–Hausdorff Distances. *Discrete & Computational Geometry*, pages 1–25, 2012. 10.1007/s00454-012-9406-8. doi:10.1007/s00454-012-9406-8.

22 Felix Schmiedl. Computational Aspects of the Gromov–Hausdorff Distance and its Application in Non-rigid Shape Matching. *Discrete & Computational Geometry*, 57(4):854–880, June 2017. doi:10.1007/s00454-017-9889-4.

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

25 Elena Farahbakhsh Touli and Yusu Wang. FPT-algorithms for computing Gromov-Hausdorff and interleaving distances between trees. *CoRR*, abs/1811.02425, 2018. arXiv:1811.02425.

26 Kaizhong Zhang and Tao Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.