A Practical Algorithm for (Geometry-Aware) Interleavings Between Merge Trees

Thijs Beurskens
TU Eindhoven, The Netherlands

Emil Toftegaard Gæde
Technical University of Denmark, Kongens Lyngby, Denmark

Tim Ophelders
TU Eindhoven, The Netherlands

Willem Sonke
TU Eindhoven, The Netherlands

Bettina Speckmann
TU Eindhoven, The Netherlands

Kevin Verbeek
TU Eindhoven, The Netherlands

Abstract

Merge trees are a popular topological descriptor for scalar field data. A common measure to compare two merge trees is the *interleaving distance*, which relies on a mapping between the two merge trees, also referred to as an *interleaving*. Despite its desirable properties, the interleaving distance has not been used much in practice, largely due to the fact that computing the exact interleaving distance is NP-hard. In this paper, we show that the exact interleaving distance can be computed efficiently for merge trees encountered in practice: we present the first implementation of the exact fixed-parameter tractable (FPT) algorithm by Touli and Wang [17]. This algorithm uses a dynamic program to test if a specific interleaving distance δ is feasible. They bound the running time using a parameter τ that captures the number of mapping options between the two merge trees for the output distance δ. Our experiments show that, even though τ can become quite large for real-world merge trees, the running time of our implementation does not depend very heavily on τ. Furthermore, we modify the FPT algorithm into a sweepline algorithm that runs much faster in practice. Finally, we introduce a natural restriction for the interleaving distance capturing the geometric similarity between the underlying scalar fields. This *restricted interleaving distance* can be computed more efficiently and can, in some settings, also result in more meaningful interleavings. We extend our implementations to support these restrictions and demonstrate their effect on the running time of the algorithms.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Fixed parameter tractability

Keywords and phrases interleaving distance, geometry-aware, exact algorithm, implementation

Digital Object Identifier 10.4230/LIPIcs.SEA.2026.6

Funding *Thijs Beurskens*: Supported by the Dutch Research Council (NWO) under project no. OCENW.M20.089.
Tim Ophelders: Supported by the Dutch Research Council (NWO) under project no. VI.Veni.212.260.
Willem Sonke: Supported by the Dutch Research Council (NWO) under project no. OCENW.M20.089.

1 Introduction

Scalar fields are a common model to represent physical phenomena, such as temperature or air pressure, for computational purposes. They assign a numerical value to every point in a domain; when the data is geographic, then we also speak about *terrains*. Most scalar fields are too large to be used directly in computations. We hence use tools from topological data

analysis, such as *topological descriptors*, to summarize, abstract, or reduce scalar fields, which in turn enables analysis and visualization of their salient features. One popular topological descriptor are *merge trees* which encode how connected components of sub- or superlevel sets in the scalar field evolve (see Figure 1a). As such, they provide a compact abstraction of the hierarchy of critical points of a scalar field. Merge trees have been used to represent scalar fields in diverse research areas such as material science [15], meteorology [18], and uncertainty visualization [21].

If the data represented by a scalar field varies over time, then we want to quantify the rate of change. That is, we need comparative measures, both on the scalar fields themselves and on their topological descriptor [20]. One of the most studied distance measures for merge trees is the *interleaving distance* [11]. Intuitively, it measures how far two merge trees are from being "isometric" using structure-preserving maps between the trees. The interleaving distance has favorable theoretical guarantees, but computing it is NP-hard [1]. Touli and Wang [17] described an exact fixed-parameter tractable algorithm which uses a dynamic program. To the best of our knowledge, this algorithm is the only exact algorithm and until now it has not been implemented and tested in practice[1].

As a consequence, practical applications of the interleaving distance usually rely on heuristic algorithms. For example, Pegoraro [13] formulated an integer linear program to obtain upper and lower bounds for the interleaving distance. In practice, these bounds often coincide for trees with a small number of leaves (≤ 15), but the approach becomes computationally impracticable for larger instances. An alternative definition of the interleaving distance in terms of labelings [7, 12] has inspired several heuristics to estimate optimal labelings. In particular, Curry *et al.* [6] use the Gromov-Wasserstein framework to find good labelings, while Yan *et al.* [19, 20] use the underlying scalar fields to obtain geometry-aware labelings.

Contributions. Our main contribution is the first implementation for computing the interleaving distance between two merge trees exactly. More precisely, we implement (1) the fixed-parameter tractable algorithm of Touli and Wang [17], and (2) a modification of their approach that replaces the dynamic program with a sweepline algorithm through the trees. Additionally, inspired by recent geometry-aware approaches [3, 19], we propose a novel heuristic: the *restricted interleaving distance*. More precisely, we use the geometry of the underlying scalar fields to impose additional geometry-aware restrictions on the allowed maps. We integrate these restrictions in both the dynamic program and the sweepline algorithm. We evaluate the runtime of our implementation through an experimental study on real-world data, and we study the effect the restrictions have on the interleaving distance.

We first give the necessary definitions in Section 2. In the same section we also give a detailed description of the algorithm by Touli and Wang. In Section 3, we describe our sweepline algorithm and show that it correctly computes the interleaving distance. Next, in Section 4, we define the restricted interleaving distance and we propose concrete geometry-aware restrictions based on the underlying scalar fields. Lastly, in Section 5, we discuss our experimental evaluation. Omitted proofs can be found in the appendix.

2 Preliminaries

Let T be a *rooted tree*, that is, a tree with one vertex identified as the *root*. We identify T with a *topological realization*: the disjoint union of unit intervals $[0, 1]$ that each represent an edge of T, whose endpoints are connected according to the adjacencies of T. Intuitively we think of T as a topological space; we refer to elements of T as *points*.

[1] The authors of [17] confirmed via personal communication that they did not implement their algorithm.

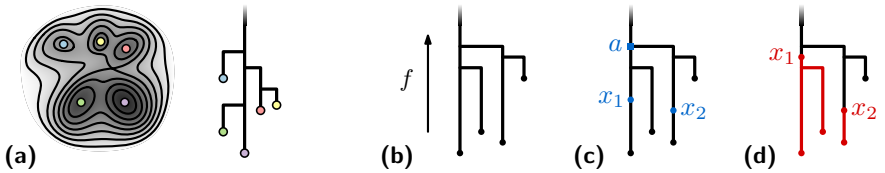

Figure 1 (a) A merge tree based on a scalar field. **(b)** Example of a merge tree T. The root of T at ∞ is omitted. **(c)** The lowest common ancestor a of $\{x_1, x_2\}$. **(d)** Subtrees $T[x_1]$ and $T[x_2]$.

▶ **Definition 1.** *A* merge tree *is a pair (T, f), where T is a rooted tree and $f\colon T \to \mathbb{R} \cup \{\infty\}$ is a continuous height function defined on the topological realisation of T such that (i) it is strictly increasing towards the root, and (ii) $f(x) = \infty$ if and only if x is the root.*

We refer to the highest non-root vertex in T as the *top vertex* in T. For two points x_1 and x_2 of T, we say x_1 is a *descendant* of x_2 if there is an f-monotonically increasing path from x_1 to x_2. If moreover $x_1 \neq x_2$, we say x_1 is a *strict descendant* of x_2. Correspondingly, x_2 is called a *(strict) ancestor* of x_1. For a point x of T at height h, and some $\delta \geq 0$, we say the 2δ-*ancestor* of x is the ancestor of x at height $h + 2\delta$. For a set of points X of T, we use $\mathrm{lca}(X)$ to denote the lowest common ancestor of all points in X. Note that if all $x \in X$ have the same height h, then X has a common 2δ-ancestor if and only if $\mathrm{lca}(X)$ has height at most $h + 2\delta$. For a point x of T, we use $T[x]$ to denote the *subtree* of T rooted at x (i.e., consisting of all descendants of x); for a finite set of points X of T, we write $T[X]$ for the union of all subtrees $T[x]$ for $x \in X$. We define the *depth* of x as the maximum height difference between x and any point in the subtree $T[x]$. See Figure 1 for examples.

Scalar field-based merge trees. Merge trees are typically used to analyze and study *scalar fields*, that is, a connected domain together with a function that assigns to every point a numeric value. Formally, let M be a compact manifold and let $f\colon M \to \mathbb{R}$ be a Morse function on M. Two points p_1, p_2 of M are *equivalent* with respect to f, written $p_1 \sim_f p_2$, if (1) they have the same height value $h = f(p_1) = f(p_2)$, and (2) they belong to the same connected component of the sublevel (superlevel) set at height h. The merge tree (T, f) based on (M, f) is the quotient space $M/\!\sim_f$, that is, the space that arises when we "glue" together all points that are equivalent under $\sim_f$. In literature [5, 19], a merge tree defined on sublevel sets is typically referred to as a *join tree*, and a merge tree defined on superlevel sets is typically referred to as a *split tree*. The vertices of a scalar field-based merge tree correspond to extrema in the scalar field; the root at ∞ represents the entire scalar field.

2.1 Interleaving distance

Consider two merge trees (T_1, f_1) and (T_2, f_2). The interleaving distance between T_1 and T_2 was originally defined in terms of two maps, one from T_1 to T_2, and one from T_2 to T_1 [11]. We use an equivalent definition [7] in terms of a single map from T_1 to T_2 (see Figure 2):

▶ **Definition 2** (adapted from [7, Definition 10])**.** *Given $\delta \geq 0$ and two merge trees (T_1, f_1) and (T_2, f_2), a continuous map α from T_1 to T_2 is a* δ-good map *if*

P1. *for all $x \in T$ we have $f_2(\alpha(x)) = f_1(x) + \delta$,*
P2. *for all $y \in \mathrm{im}\,\alpha$ with $x' := \mathrm{lca}(\alpha^{-1}(y))$, we have $f_1(x') - f_1(x) \leq 2\delta$ for all $x \in \alpha^{-1}(y)$,*
P3. *for all $y \notin \mathrm{im}\,\alpha$, the depth of y is at most 2δ.*

The interleaving distance *$d(T_1, T_2)$ between T_1 and T_2 is defined as the minimum*[2] *δ for which there exists a δ-good map from T_1 to T_2.*

[2] This was originally defined using the infimum. However, this infimum is always attained (see [2, 13]).

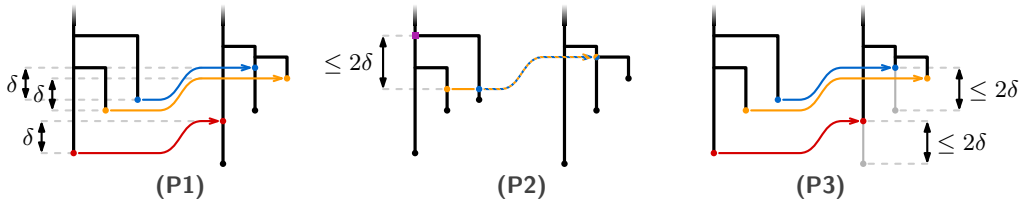

Figure 2 A δ-good map from T_1 to T_2: **P1** maps each point of T_1 to a point of T_2 whose height is exactly δ greater; **P2** ensures that points of T_1 with the same image in T_2 have a common 2δ-ancestor; **P3** does not leave unmapped subtrees of T_2 that are taller than 2δ.

Touli and Wang [17] describe a decision procedure to determine if a δ-good map[3] exists for a given value δ. We sketch this procedure below; see the original paper for a full description. To compute the interleaving distance, they search on a finite set of candidate values. In particular, it has been shown [1, 17] that $d(T_1, T_2)$ is always present in the set $\Delta(T_1, T_2)$ that contains (1) all height differences between a vertex in T_1 and a vertex in T_2, and (2) all height differences, divided by 2, between vertex pairs in the same tree.

Consider a pair (X, y), where X is a set of points of T_1, all at the same height h, and y is a single point of T_2 at height $h + \delta$. A continuous map α from $T_1[X]$ to $T_2[y]$ is a *partial δ-good map* if **P1**–**P3** hold for all points of $T_1[X]$ and $T_2[y]$. Note that for **P2**, we still determine the lowest common ancestor in the complete tree T_1. We say (X, y) is *feasible* if a partial δ-good map from $T_1[X]$ to $T_2[y]$ exists. Touli and Wang [17, Claim 2] show that a δ-good map between the complete trees T_1 and T_2 exists if and only if there is a feasible pair (X, y) for which X and y both lie above the top vertices in their respective trees.

Decision procedure. To decide whether a δ-good map exists, Touli and Wang describe a dynamic program (DP) that computes, for each level, which pairs are feasible. A *level* is a set of points of T_1 at a height h and a set of points of T_2 at height $h + \delta$ (see Figure 3a). They consider only levels that contain a vertex in either tree; this results in a discrete set of levels $\{L_1, \ldots, L_m\}$ in increasing height order. To compute the set of feasible pairs at some level L_i, the algorithm first determines all sets of points of T_1 in L_i that have a common 2δ-ancestor (see Figure 3b). Then, for all combinations ("valid pairs") (X, y) of such a subset X and a point y of T_2 in L_i, the algorithm determines whether it is feasible:

(1) If y is a leaf, then (X, y) is feasible if and only if X consists of only leaves.
(2) Else, let $y_1, \ldots, y_k$ be the k descendants of y in level L_{i-1}. Now (X, y) is feasible iff the descendants of X in level L_{i-1} can be partitioned into k sets $X_1, \ldots, X_k$, s.t. for all j:
  (a) if X_j is not empty, then (X_j, y_j) is feasible,
  (b) if X_j is empty, then all leaves in $T_2[y_j]$ are at most 2δ below the height of y.

Note that the first level contains only leaves, so the separate base case in [17] is unnecessary.

Touli and Wang show that this decision procedure is fixed-parameter tractable for a parameter τ. Specifically, for any $h \geq 0$, they consider connected parts of T_1 and T_2 of height $2h$, and define τ_h to be the maximum sum of degrees over these parts:

$$\tau_h = \max\{\text{degree-sum}(T) \mid y \in \mathbb{R},\ T \text{ is a component of } f_1^{-1}([y, y + 2h]) \text{ or } f_2^{-1}([y, y + 2h])\}.$$

[3] We note that Definition 2 slightly differs from the one given in [17]. A proof that these definitions are equivalent can be found in [7, Appendix A].

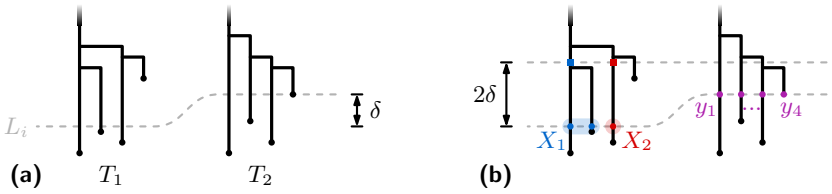

Figure 3 (a) Two merge trees T_1 and T_2, with one level L_i shown. **(b)** On level L_i, X_1 and X_2 are sets with a common 2δ-ancestor. So, the valid pairs are (X, y) where X is any subset of X_1 or X_2, and y is any of $y_1, \ldots, y_4$.

Now, $\tau \coloneqq \tau_\delta$, where δ is the interleaving distance between the trees. Touli and Wang bound the running time of the decision procedure by $O(n^3 2^\tau \tau^{\tau+1})$ [17, Theorem 2], where n is the sum of the number of vertices in both trees.

Search procedure. Given the δ-good map decision procedure and the candidate set $\Delta(T_1, T_2)$, we compute the interleaving distance by finding the smallest candidate for which the decision procedure returns YES. The simplest way to do this is via a linear search, resulting in a running time bound of $O(n^4 2^\tau \tau^{\tau+2} \log n)$ [17, Theorem 4]. Alternatively, we can also use an exponential search on τ. We can compute τ_δ for each candidate $\delta \in \Delta(T_1, T_2)$; note that τ_δ monotonically increases for increasing δ. We then perform an exponential search over the set of $2n$ possible values for τ_δ, to find the first τ_δ for which a δ-good map exists. Lastly, we perform a binary search to find the exact δ. This results in a running time bound of $O(n^2 2^{2\tau} (2\tau)^{2\tau+2} \log^3 n)$ [17, Theorem 5]. This improves the dependency on n by a factor $\sim n^2$ over linear search, but deteriorates the dependency on τ by a large exponential factor. As τ can be relatively large in practice, this is a poor tradeoff to make; for this reason we have not implemented this method and will not report on it in the remainder of this paper.

We propose another simple search procedure that avoids the need to evaluate every single candidate, but still tries to avoid overshooting too much. We first perform an exponential search to upper bound δ, and then binary search to find the exact interleaving distance. Because in our experiments the decision procedure tends to take significantly longer as δ increases (see Section 5), instead of simply doubling δ, we take as the new upper bound the highest candidate that is still within a factor of 1.5 of the previously established lower bound.

3 Sweepline decision procedure

We can interpret the decision procedure described in Section 2 as a sweepline algorithm: two coordinated sweeplines sweep up through T_1 and T_2, where the height of the sweepline in T_2 always stays exactly δ above the one in T_1 (see Figure 4a). Whenever either of the sweeplines hits a vertex, an event occurs, and the set of feasible pairs for the corresponding level is computed. We propose an alternative decision procedure that uses this sweepline interpretation: instead of computing all feasible pairs when an event happens, we update only the ones affected by the event. As opposed to the DP, we generally do not need to enumerate all valid pairs. In many cases, the set of feasible pairs is (much) smaller than the set of valid pairs, resulting in a speedup. To compute the interleaving distance, we use the same search procedures described above, but replace the DP by our sweepline procedure.

To simplify the exposition, we assume that the vertices are in general position: every internal vertex has exactly two children, and no two vertices are hit simultaneously by the sweepline. We justify this assumption in Section 3.3: for any merge tree and any $\varepsilon > 0$, we can construct a merge tree within interleaving distance ε that satisfies the assumption.

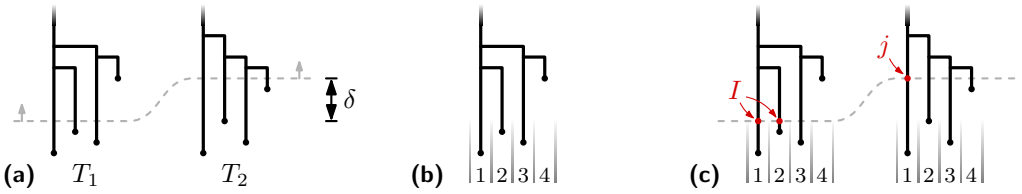

Figure 4 **(a)** The sweepline sweeps through T_1 and T_2 simultaneously. **(b)** We index points on the sweepline using leaf indices; this corresponds to numbering "columns" in a drawing of the merge tree. **(c)** $F[j]$ contains all sets I of indices such that (I, j) is feasible. Illustrated here: $\{1, 2\} \in F[1]$.

3.1 Event handling

We maintain for each point y of T_2 on the sweepline, all non-empty sets X of points of T_1 on the sweepline such that (X, y) is feasible. Let ℓ_1 and ℓ_2 be the number of leaves in T_1 and T_2, respectively. We assume the leaves in each tree T_k are indexed $1, 2, \ldots, \ell_k$ from left to right in such a way that the indices of the set of leaves in a subtree always form an integer interval; this can be achieved using an in-order tree walk on the input. We index each point on the sweepline by the lowest-index leaf in its subtree (see Figure 4b); when we discuss points on the sweepline, we write "vertex i in T_k" instead of "vertex in T_k with index i". If an internal vertex i is on the sweepline, its two descendants just below the sweepline have indices i and i', where $i < i'$. We refer to i and i' as the *children* of i.

We maintain the set of feasible pairs in a list F of ℓ_2 elements. Consider an index j from T_2. If no point y of T_2 on the sweepline has index j, then $F[j] = \text{NIL}$. Otherwise, if there is a point y, then $F[j]$ represents all non-empty sets X of points of T_1 on the sweepline such that (X, y) is feasible. We store every such set X as the corresponding set I of indices (see Figure 4c). When the sweepline hits a vertex, an event happens and we need to update F:

(a) Leaf j in T_2: Initialize $F[j] = \varnothing$.

(b) Internal vertex j in T_2: Let j, j' be the children of j. Set $G = \varnothing$. For each $I_1 \in F[j]$ and $I_2 \in F[j']$, add $I_1 \cup I_2$ to G if and only if I_1 and I_2 (i) have the same 2δ-ancestor, and (ii) are disjoint (see Figure 5). Additionally, if the depth of the left (right) subtree is at most 2δ, add all elements of $F[j']$ ($F[j]$) to G. Finally, set $F[j] = G$ and $F[j'] = \text{NIL}$.

(c) Leaf i in T_1: For each index j from T_2 on the sweepline: (1) for each set I in $F[j]$, add the set $I \cup \{i\}$ to $F[j]$ if and only if I and i have the same 2δ-ancestor, and (2) if the depth of the point of T_2 on the sweepline with index j is at most 2δ, add $\{i\}$ to $F[j]$.

(d) Internal vertex i in T_1: Let i, i' be the children of i. For each index j from T_2 on the sweepline, for each set I in $F[j]$, if I contains exactly one of i and i', delete I from $F[j]$; if I contains both, replace I by $I \setminus \{i'\}$.

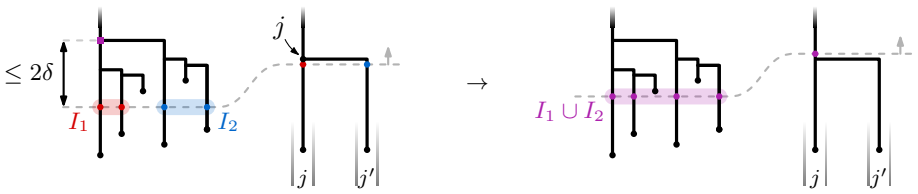

Figure 5 Handling a type-**(b)** event on internal vertex j in T_2.

The sweepline terminates after handling the last event, that is, when the top vertices of both T_1 and T_2 have been handled. Consider the set F after the sweepline terminates. Let y be the top vertex in T_2. By construction, y has index 1. We conclude that a δ-good map exists if and only if (a) the depth of y is at most 2δ, or (b) $F[1]$ contains at least one element.

To prove that our sweep is correct, we make two observations. Consider a partial δ-good map α from $T_1[X]$ to $T_2[y]$, for some set of points X of T_1 and a point y of T_2. Then we can extend α until we hit an internal vertex: let X' be the ancestors of points in X at a fixed height, and assume that there are no internal vertices between points in X and their ancestor in X'. We define the *extension* of α to $T_1[X']$: for every point x of $T_1[X']$ that is not in $T_1[X]$, let y' be the ancestor of y at height $f_1(x) + \delta$. We set $\alpha(x) \coloneqq y'$. Since there are no new internal vertices in $T_1[X']$, the extension is well-defined for all points of $T_1[X']$. Moreover, α is continuous and satisfies **P1**–**P3**.

▶ **Observation 3.** *Let α be a partial δ-good map. Any extension of α that does not extend above an internal vertex is a partial δ-good map.*

On the other hand, we can also restrict α such that its image remains a subtree of T_2. The resulting map is again a partial δ-good map: **P1**–**P3** follow directly.

▶ **Observation 4.** *Let α be a partial δ-good map from $T_1[X]$ to $T_2[y]$, and let X' be a set of points in $T_1[X]$ with the same image. Then α restricted to $T_1[X']$ is a partial δ-good map.*

We are now ready to prove that the sweepline maintains all non-empty feasible pairs.

▶ **Lemma 5.** *Assume T_1 and T_2 are in general position. After handling any event, the set F correctly represents all non-empty feasible pairs at the height of that event.*

3.2 Data structure

For efficient event handling, we store F in a suitable data structure. Specifically, we store each (non-NIL) element $F[j]$ in a balanced binary search tree, such as a red–black tree, which allows efficient searches, insertions, and deletions. When storing some $I \in F[j]$ in the tree, we use the lowest-indexed element of I as its key. This way, the sets in $F[j]$ are always ordered by their lowest-indexed element, which allows us to efficiently handle the 2δ-ancestor check in **(b)** events as follows.

Event **(b)** asks us to try all combinations of $I_1 \in F[j]$ and $I_2 \in F[j']$ and see if they have the same 2δ-ancestor. Instead, we check for each potential 2δ-ancestor which sets I_1 and I_2 descend from it, and only try these combinations. Let $A = [a_1, \ldots, a_k]$ be the (ordered) sequence of indices of T_1 at height 2δ above the sweepline, that is, A is the sequence of potential 2δ-ancestors of I_1 and I_2. The problem boils down to finding, for a given ancestor a_i, the subset of $F[j]$ that have a_i as its 2δ-ancestor. This is the subset of all $I_1 \in F[j]$ for which the lowest-indexed element of I_1 is between a_i (inclusive) and a_{i+1} (exclusive). As $F[j]$ is sorted on the lowest-indexed element, this subset of elements is a contiguous range, which we can delineate by searching in the binary search tree for $\{a_i\}$ and $\{a_{i+1}\}$.

We note that this data structure admits an easy and efficient implementation in practice if we represent each set $I \in F[j]$ as a bit string of which the i-th bit is set if and only if $i \in I$. Such a representation not only allows us to compute unions and intersections efficiently, but also sorting the bit strings lexicographically ensures that they are sorted by their lowest-indexed element.

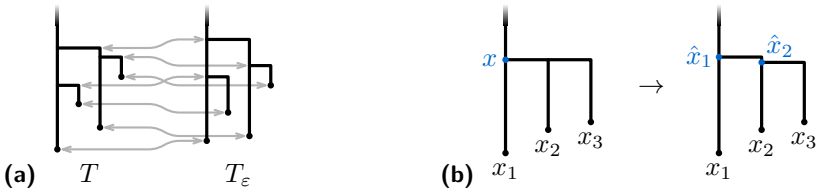

Figure 6 (a) An ε-perturbation T_ε of a merge tree T. **(b)** Expanding vertex x into $\hat{x}_1$ and $\hat{x}_2$.

3.3 General position

Let (T, f) be a merge tree, and let $\varepsilon > 0$ such that ε is smaller than the height difference between any two adjacent vertices in T. A merge tree $(T_\varepsilon, f_\varepsilon)$ is an *ε-perturbation* of (T, f) if there is a homeomorphism η from T to T_ε such that $|f(x) - f_\varepsilon(\eta(x))| \leq \varepsilon$ for all points x of T (see Figure 6a). It is not hard to show that the interleaving distance between (T, f) and an ε-perturbation $(T_\varepsilon, f_\varepsilon)$ is at most ε.

▶ **Lemma 6.** *Let (T, f) be a merge tree and let $(T_\varepsilon, f_\varepsilon)$ be any ε-perturbation for sufficiently small $\varepsilon > 0$. Then $d(T, T_\varepsilon) \leq \varepsilon$.*

Suppose T has a vertex x that has $k > 2$ children $x_1, \ldots, x_k$. We *expand* x into $k-1$ distinct vertices $\hat{x}_1, \ldots, \hat{x}_{k-1}$ such that each vertex $\hat{x}_i$ has two children: x_i and $\hat{x}_{i+1}$ (see Figure 6b). We define the *expansion* $\hat{T}$ of T by expanding all vertices in T, together with the height function $\hat{f}$ that assigns to each new vertex $\hat{x}_i$ the same height as x. An extension $(\hat{T}, \hat{f})$ is not a merge tree; however, for any value $\varepsilon > 0$ there does exists an ε-perturbation of $(\hat{T}, \hat{f})$ that is a merge tree. We define the *interleaving distance* between two expansions $\hat{T}_1$ and $\hat{T}_2$ as the limit of the interleaving distance between such ε-perturbed merge trees $\hat{T}_1^\varepsilon$ and $\hat{T}_2^\varepsilon$:

$$d(\hat{T}_1, \hat{T}_2) \coloneqq \lim_{\varepsilon \to 0} d(\hat{T}_1^\varepsilon, \hat{T}_2^\varepsilon).$$

Lemma 6 then directly implies Corollary 7.

▶ **Corollary 7.** *The interleaving distance between any two merge trees is equal to the interleaving distance between their expansions.*

In the algorithm we handle expansions by placing the expanded vertices on exactly the same height, and separating them symbolically. That is, the vertices all lie on the same level, and when we process that level, we handle them one by one.

4 Restrictions

We consider a generalization of the interleaving distance that imposes additional restrictions on the allowed δ-good maps. Intuitively, each restriction specifies for some leaf x in T_1 all points of T_2 that are "reachable" by x. Formally, let (T, f) be a merge tree. A subset of points of T is called a *pruning* of T if it is a merge tree again, that is, it is connected and contains the root of T (see Figure 7a). For two merge trees (T_1, f_1) and (T_2, f_2), a *restriction* $\mathcal{R}$ from T_1 to T_2 is a set of prunings $\mathcal{R}[x]$ for each leaf x in T_1. In other words, if a point y of T_2 is reachable by x, then all ancestors of y are also reachable by x. This condition ensures that a δ-good map that respects the restriction always exists. If $\mathcal{R}[x] = T_2$ for each leaf x in T_1, we recover the unrestricted interleaving distance.

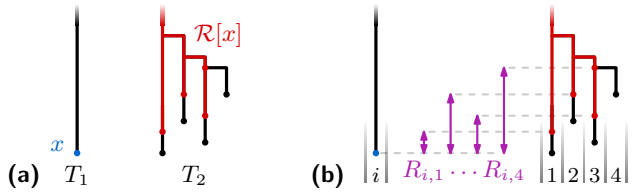

Figure 7 (a) A pruning $\mathcal{R}[x]$. **(b)** The row in the restriction matrix corresponding to $\mathcal{R}[x]$.

▶ **Definition 8.** *Given two merge trees (T_1, f_1) and (T_2, f_2) and a restriction $\mathcal{R}$ from T_1 to T_2, a δ-good map α from T_1 to T_2 is* $\mathcal{R}$-restricted *if*

P4. *for all leaves $x \in T$ we have $\alpha(x) \in \mathcal{R}[x]$.*

The $\mathcal{R}$-restricted interleaving distance $d(T_1, T_2, \mathcal{R})$ *between T_1 and T_2 is defined as the infimum δ for which there exists an $\mathcal{R}$-restricted δ-good map from T_1 to T_2.*

Analogous to the unrestricted setting, we define a *partial $\mathcal{R}$-restricted δ-good map* as a partial δ-good map that additionally satisfies **P4**. A pair (X, y) is *$\mathcal{R}$-feasible* if there exists a partial $\mathcal{R}$-restricted δ-good map from $T_1[X]$ to $T_2[y]$. We show that the value $d(T_1, T_2, \mathcal{R})$ is always in a finite set $\Delta(T_1, T_2, \mathcal{R})$ that consists of (1) all values in $\Delta(T_1, T_2)$, and (2) the height differences between a leaf x in T_1 and a leaf in $\mathcal{R}[x]$.

▶ **Lemma 9.** $d(T_1, T_2, \mathcal{R}) \in \Delta(T_1, T_2, \mathcal{R})$.

The proof of Lemma 9 directly implies that the infimum in Definition 8 can be replaced by a minimum: for any $\delta \geq 0$, if an $\mathcal{R}$-restricted δ-good map exists then there also exists an $\mathcal{R}$-restricted δ'-good map with $\delta' \in \Delta(T_1, T_2, \mathcal{R})$ and such that $\delta' < \delta$.

▶ **Corollary 10.** *For any two merge trees T_1 and T_2 and any restriction $\mathcal{R}$ from T_1 to T_2, there exists a $\mathcal{R}$-restricted δ-good map from T_1 to T_2 with $\delta = d(T_1, T_2, \mathcal{R})$.*

4.1 Computing the restricted interleaving distance

We compute the $\mathcal{R}$-restricted interleaving distance between T_1 and T_2 using a similar approach as for computing the unrestricted interleaving distance. More precisely, we can still search on the set of candidate values, but instead of searching over all values in $\Delta(T_1, T_2)$, we search over all values in $\Delta(T_1, T_2, \mathcal{R})$. Moreover, to decide for a given value δ whether an $\mathcal{R}$-restricted δ-good map exists, we use a modified version of either the DP as described in Section 2, or the sweepline algorithm as described in Section 3.

Recall that ℓ_1 and ℓ_2 denote the number of leaves in T_1 and T_2, respectively. We represent a restriction $\mathcal{R}$ using a matrix R of size $\ell_1 \times \ell_2$. Consider a leaf x in T_1 with index i, and a leaf y in T_2 with index j. The entry R_{ij} stores the height of the lowest ancestor of y that is in the pruning $\mathcal{R}[x]$. We refer to R as the *restriction matrix* of $\mathcal{R}$ (see Figure 7b).

Sweepline decision procedure. Intuitively, we only need to verify that when we add a pair (X, y) for which X contains a leaf x in T_1, that $y \in \mathcal{R}[x]$. It suffices to modify event **(c)**:

(c') **Leaf i in T_1:** Let h be the height of the sweepline within T_2. For each index j from T_2 on the sweepline with $R_{ij} \leq h$: (1) for each set I in $F[j]$, add the set $I \cup \{i\}$ to $F[j]$ if and only if I and i have the same 2δ-ancestor, and (2) if the depth of the point of T_2 on the sweepline with index j is at most 2δ, add $\{i\}$ to $F[j]$.

The sweepline terminates after handling the last event, that is, when the top vertices of both T_1 and T_2 have been handled. Consider the set F after the sweepline terminates and let y be the top vertex in T_2. Analogous to the unrestricted setting, we conclude that a $\mathcal{R}$-restricted δ-good map exists if and only if (a) the depth of y is at most 2δ, or (b) $F[1]$ contains at least one element.

▶ **Lemma 11.** *Assume T_1 and T_2 are in general position. After handling any event, the set F correctly represents all non-empty feasible pairs at the height of that event.*

DP decision procedure. We need a similar modification of the dynamic program: if we consider a pair (X, y) for which X contains a leaf x in T_1, we need to verify that additionally $y \in \mathcal{R}[x]$. We can perform this check when computing all valid pairs. The proof that this computes the restricted interleaving distance is analogous to the proof of Lemma 11.

4.2 Geometry-aware restrictions

Consider a (finite) sequence of scalar fields (M, f_t), that is, a 2D or 3D domain M whose height function changes over time. Let d_e denote the distance between points of M. Fix two instances (M, f_1) and (M, f_2), and let (T_1, f_1) and (T_2, f_2) be the corresponding scalar field-based merge trees. Without loss of generality, assume T_1 and T_2 are based on sublevel sets; their leaves correspond to minima in their respective scalar fields. Intuitively, assuming the two instances are not too far apart and the scalar field evolves smoothly over time, we do not expect big changes in the geometry of the extrema: each minimum in (M, f_1) has a corresponding minimum in (M, f_2) close by. We use this to define a restriction from T_1 to T_2: a leaf in T_1 is allowed to map to a point of T_2 if the corresponding minima are not "too far" apart. Formally, we define a restriction $\mathcal{R}_\rho$, where $\rho \geq 0$ is a threshold variable that captures how far is too far. Let x be a leaf in T_1. Then x corresponds to a minimum in (M, f_1); let p_x be the point of M where the minimum lies. Similarly, each point y of T_2 corresponds to a sublevel set component; let P_y denote the sublevel set component that contains y. We define $y \in \mathcal{R}_\rho[x]$ if and only if $d_e(p_x, P_y) \leq \rho$. This defines a pruning: if $y \in \mathcal{R}_\rho[x]$, then the sublevel set component $P_{y'}$ of any ancestor y' of y contains P_y, so $y' \in \mathcal{R}_\rho$.

▶ **Observation 12.** *For any leaf x in T_1 and any value $\rho \geq 0$, the set $\mathcal{R}_\rho[x]$ is a pruning.*

Our idea of a geometry-aware heuristic is inspired by a heuristic by Yan *et al.* [19]. Specifically, they also use the geometry of the underlying scalar fields as a heuristic to compute the interleaving distance. However, their method uses a matching between extrema of the scalar fields to compute a labeling on the merge trees; the resulting interleaving distance is then the labeled interleaving distance for that specific labeling. Our method is more general: each leaf is allowed to map to more than one point. In particular, if we set $\rho = \infty$, then $\mathcal{R}_\rho[x] = T_2$ for each leaf x in T_1 and we recover the unrestricted interleaving distance.

5 Experimental evaluation

We implemented the algorithms described above and ran them on two different datasets. In this paper, we study only the running time of our implementation; we do not study the quality of the resulting interleavings for the analysis of scalar fields.

Datasets. We used two datasets that Yan *et al.* [19] also used to evaluate their methods. For both datasets, we used the Topology Toolkit (TTK) [4, 10, 16] to generate split trees. The HeatedFlow[4] dataset comes from a simulation of a 2D flow generated by a heated cylinder [8, 14]. The RedSea[5] dataset was used in the 2020 SciVis Contest [9, 22, 23]. For both datasets, we used the same preprocessing steps as Yan *et al.* [19], except that we did not use persistence simplification to prune the merge trees, as we want to demonstrate the running times on larger merge trees.

To create pairs (T_1, T_2) and corresponding restrictions $\mathcal{R}$ to run the algorithm on, we selected sets of frames from both datasets (frames $1000, 1050, \ldots, 1500$ for HeatedFlow and frames $0, 5, \ldots, 55$ for RedSea) and considered all pairs of frames (where T_1 always corresponds to the earlier frame). This resulted in 66 instances for HeatedFlow, and 78 instances for RedSea. The number of leaves per tree instances ranges from 83 to 108 (average 96.4) for HeatedFlow and from 39 to 57 (average 46.9) for RedSea. For each instance, we generated geometry-aware restriction matrices R_ρ, where $\rho \in \{\infty, 10, 0\}$ for HeatedFlow and $\rho \in \{\infty, 25, 10, 0\}$ for RedSea (measured in pixels). Recall that $\rho = \infty$ represents the unrestricted interleaving distance.

Implementation details. We implemented the algorithm in C++ and ran all experiments on a workstation with an AMD Ryzen 9 9950X CPU and 64 GB RAM running Ubuntu 25.10. Although the CPU has 16 cores, we ran the experiments single-threaded to be able to measure the running times as accurately as possible. For the decision procedure, we implemented both the DP and the sweepline variants. For the search procedure, we implemented both linear search and the δ-exponential/binary search (referred to as "exponential search" in the remainder) as described at the end of Section 2.1.

While implementing, we noticed that a naive implementation of the algorithm can be sensitive to floating-point rounding errors. In particular, if δ is the interleaving distance between T_1 and T_2, there exists a δ-good map from T_1 to T_2; however, if we call the decision procedure with δ exactly, it may fail to find this δ-good map due to rounding errors. This may lead the search procedure to output an incorrect interleaving distance. We alleviate this problem by calling the decision procedure for values between the candidates, instead of for the candidates themselves. Specifically, let $\Delta(T_1, T_2) = [\delta_1, \delta_2, \ldots, \delta_{k-1}, \delta_k]$ be the sorted candidate list. We then perform the search procedure on the list $[\frac{\delta_1+\delta_2}{2}, \frac{\delta_2+\delta_3}{2}, \ldots, \frac{\delta_{k-1}+\delta_k}{2}]$. Given an outcome that a $\frac{\delta_i+\delta_{i+1}}{2}$-good map exists but a $\frac{\delta_{i-1}+\delta_i}{2}$-good map does not, we can conclude that the true interleaving distance must be δ_i.

Furthermore, in our implementation we optimized the size of the candidate set $\Delta(T_1, T_2)$. Recall that $\Delta(T_1, T_2)$ contains (1) all height differences between a vertex in T_1 and a vertex in T_2, and (2) all height differences, divided by 2, between vertex pairs in the same tree. However, it turns out that the interleaving distance is always realized by a subset $\Delta^*(T_1, T_2)$ of these values, that contains (1) all height differences between a leaf in T_1 and a leaf in T_2 (or, in the restricted setting, in its pruning), (2) all height differences between an internal vertex in T_1 and an internal vertex in T_2, and (3) all height differences, divided by 2, between a leaf and an ancestor vertex in the same tree. Note that it is possible for $\Delta^*(T_1, T_2)$ to contain (nearly) as many candidates as the full set $\Delta(T_1, T_2)$, so this optimization does not affect a worst-case running time analysis. However, it could save some time in practice by reducing the number of binary search steps needed.

[4] https://cgl.ethz.ch/research/visualization/data.php
[5] https://kaust-vislab.github.io/SciVis2020/

Table 1 Accumulated results of the experimental evaluation on two datasets: HeatedFlow (66 instances) and RedSea (78 instances). In each column, we report the average computed interleaving distances across all instances, and the total running time spent on all instances for the DP and the sweepline decision procedure. All instances used the exponential search procedure.

dataset	HeatedFlow			RedSea			
restriction radius ρ	∞	10	0	∞	25	10	0
avg. interleaving distance	0.1172	0.4710	0.5717	0.1657	0.1766	0.2200	0.3216
total time DP (s)	4453	3706	3054	7.752	6.866	6.550	5.937
total time sweepline (s)	652.2	154.4	144.9	2.425	2.308	2.426	2.391

5.1 Results

We ran our algorithm on all instances of HeatedFlow and RedSea. We used only the exponential search procedure, as running the linear search procedure was exceedingly slow and did not finish in a reasonable amount of time. We also set a timeout at 300 seconds for each run. The DP procedure reached the timeout 5 times for HeatedFlow, while the sweepline procedure never reached the timeout. We summarize the accumulated results in Table 1; in this table we count timeouts as contributing just 300 seconds to the total time.

Decision procedure. The results clearly show that, in practice, the sweepline decision procedure is much faster than the DP decision procedure. For both datasets and for each restriction radius ρ, the total running time using the DP procedure is ~ 2.5 – $25\times$ higher than the total running time using the sweepline procedure. Due to the algorithm hitting timeouts with the DP procedure, the "true" total time for DP would have been even higher.

To investigate the variation in running time between instances of the same dataset, we show the running time of every HeatedFlow instance with $\rho = \infty$ in Figure 8a. We observe that the sweepline procedure outperforms the DP procedure on every single instance.

Restrictions. For HeatedFlow, the running time of both procedures decreases if we choose a smaller restriction radius ρ; for the sweepline decision procedure this decrease (a factor ~ 4) is more significant than for the DP procedure. However, this comes at a cost: choosing a smaller restriction radius results in a distance that is significantly larger (a factor ~ 4) than the unrestricted interleaving distance. These results raise the question of how accurately the interleaving distance between two scalar field based merge trees reflects the similarity of the scalar fields. After all, if the restrictions increase δ this much, apparently the lower interleaving distances observed in the unrestricted case match critical points that lie far apart in the scalar fields. We leave this question for future research.

For RedSea, the distance results are slightly better. In fact, for $\rho = 25$ they are similar to the actual distance. However, in this dataset the running times do not decrease significantly; this is likely due to the fact that these instances are smaller than the HeatedFlow instances, and therefore a larger fraction of the running time is spent on preprocessing.

Search procedure. The theoretical running time analysis by Touli and Wang [17] (see Section 2.1) indicates that the running time is highly exponential in τ. Because τ_δ increases as a function of δ, this suggests that overshooting δ could be detrimental for performance. Linear search is a way to avoid overshooting and hence could be a sensible search strategy.

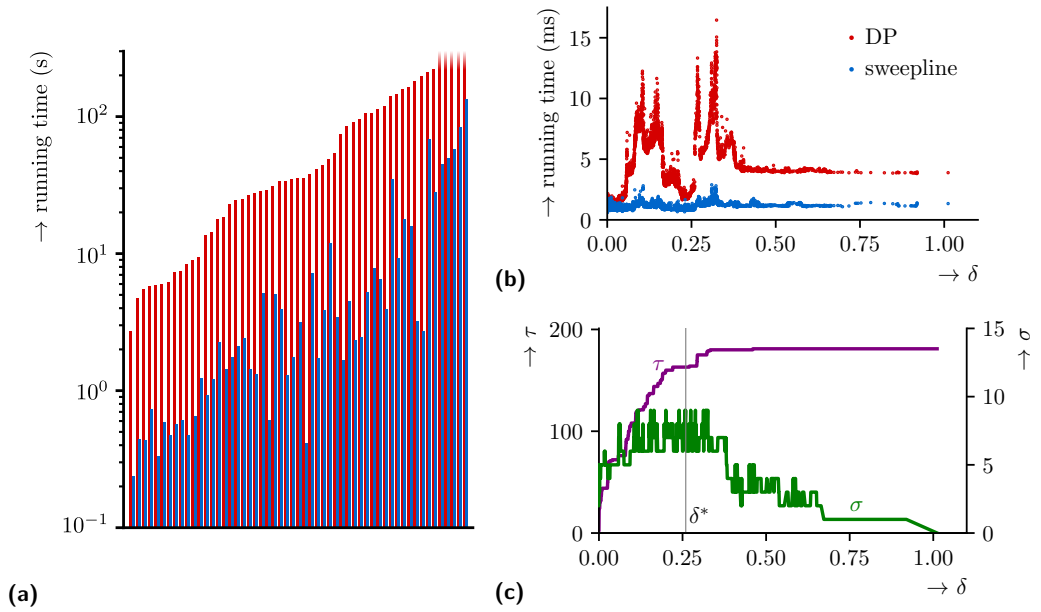

Figure 8 (a) Bar chart showing the DP (in red) and sweepline (in blue) running times for each $\rho = \infty$ instance from the HeatedFlow dataset (excluding instances with $T_1 = T_2$). The instances are sorted by their DP running time. The rightmost five bars for DP are truncated due to hitting the timeout. **(b)** Scatter plot showing the running times for a single call to the DP decision procedure (in red) and the sweepline decision procedure (in blue) for each candidate value δ for one RedSea instance (frames 15–30) with $\rho = \infty$. **(c)** We plot the parameters τ_δ and σ_δ for the dataset of **(b)**, and also indicate the value of the interleaving distance $\delta^* := d(T_1, T_2)$.

However, in practice it turns out linear search is overwhelmingly much slower than exponential search, so much so that it did not complete in a reasonable time for many instances, while exponential search is often very fast. To investigate this behavior, we plotted (see Figure 8b) for one of the instances in the RedSea dataset (frames 15–30, no restrictions) for which linear search was feasible, how long the DP and sweepline decision procedures took for all candidate values $\delta \in \Delta^*(T_1, T_2)$ (also the candidate values higher than $d(T_1, T_2)$). It is clear that the running time does in fact not at all increase monotonically as a function of δ or τ_δ (see also Figure 8c), so overshooting is not inherently a problem.

In addition to τ_δ, we consider a second parameter σ_δ. For a point p of T_1 or T_2, consider the number of descendants of p that lie exactly 2δ below p. We define σ_δ to be this number, maximized over all points p of T_1 and T_2. Although σ_δ does not increase monotonically in δ, we observe a correlation between the running times and the σ_δ, see Figure 8b–c.

6 Conclusion

We have presented the (to the best of our knowledge) first implementation for computing the exact interleaving distance between merge trees, and thereby demonstrated that computing the exact interleaving distance is feasible for many instances in practice, especially when using our sweepline procedure. We see this implementation as a starting point for further algorithm engineering on computing the (exact) interleaving distance. Our (geometry-aware) restrictions are a step in this direction and do indeed result in speedups, specifically for larger instances. However, the interleaving distance also significantly increases with restrictions; future research should determine whether these restrictions are too restrictive or if the restricted interleavings better reflect the similarity between the underlying scalar fields.

References

1 P.K. Agarwal, K. Fox, A. Nath, A. Sidiropoulos, and Y. Wang. Computing the Gromov-Hausdorff distance for metric trees. *ACM Transactions on Algorithms*, 14(2):1–20, 2018. doi:10.1145/3185466.

2 T. Beurskens, T. Ophelders, B. Speckmann, and K. Verbeek. Locally correct interleavings between merge trees. arXiv:2512.16474.

3 T. Beurskens, T. Ophelders, B. Speckmann, and K. Verbeek. Relating interleaving and Fréchet distances via ordered merge trees. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA25)*, pages 5027–5050. Society for Industrial and Applied Mathematics, 2025. doi:10.1137/1.9781611978322.170.

4 T. Bin Masood, J. Budin, M. Falk, G. Favelier, C. Garth, C. Gueunet, P. Guillou, L. Hofmann, P. Hristov, A. Kamakshidasan, C. Kappe, P. Klacansky, P. Laurin, J.A. Levine, J. Lukasczyk, D. Sakurai, M. Soler, T. Steneteg, J. Tierny, W. Usher, J Vidal, and M. Wozniak. An overview of the topology toolkit. *Topological Methods in Data Analysis and Visualization VI: Theory, Applications, and Software*, pages 327–342, 2021. doi:10.1007/978-3-030-83500-2_16.

5 H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003. doi:10.1016/S0925-7721(02)00093-7.

6 J. Curry, H. Hang, W. Mio, T. Needham, and O. B. Okutan. Decorated merge trees for persistent topology. *Journal of Applied and Computational Topology*, 6(3):371–428, February 2022. doi:10.1007/s41468-022-00089-3.

7 E. Gasparovic, E. Munch, S. Oudot, K. Turner, B. Wang, and Y. Wang. Intrinsic interleaving distance for merge trees. *La Matematica*, 4(1):40–65, 2025. doi:10.1007/s44007-024-00143-9.

8 T. Günther, M. Gross, and H. Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics*, 36(4):141:1–141:11, 2017. doi:10.1145/3072959.307368.

9 I. Hoteit, X. Luo, M. Bocquet, A. Kohl, and B. Ait-El-Fquih. Data assimilation in oceanography: Current status and new directions. *New frontiers in operational oceanography*, pages 465–512, 2018. doi:10.17125/gov2018.ch17.

10 E. Le Guillou, M. Will, P. Guillou, P. Lukasczyk, P. Fortin, C. Garth, and J. Tierny. TTK is Getting MPI-Ready. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

11 D. Morozov, K. Beketayev, and G. Weber. Interleaving distance between merge trees. Manuscript (accessed on 06-03-2025), 2013. URL: https://mrzv.org/publications/interleaving-distance-merge-trees/manuscript/.

12 E. Munch and A. Stefanou. The ℓ^∞-Cophenetic metric for phylogenetic trees as an interleaving distance. In *Research in Data Science*, volume 17 of *Association for Women in Mathematics Series*, pages 109–127. Springer International Publishing, 2019. doi:10.1007/978-3-030-11566-1_5.

13 M. Pegoraro. A graph-matching formulation of the interleaving distance between merge trees. *AIMS Mathematics*, 10(6):13025–13081, 2025. doi:10.3934/math.2025586.

14 S. Popinet. Free computational fluid dynamics. *ClusterWorld*, 2(6), 2004. URL: http://gfs.sf.net/.

15 S.S. Thygesen, A.I. Abrikosov, P. Steneteg, T.B. Masood, and I. Hotz. Level of detail visual analysis of structures in solid-state materials. In *EuroVis (Short Papers)*, pages 55–59, 2023. doi:10.2312/evs.20231043.

16 J. Tierny, G. Favelier, J.A. Levine, C. Gueunet, and Michaux M. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics*, 1, 2017. doi:10.1109/TVCG.2017.2743938.

17 E.F. Touli and Y. Wang. FPT-algorithms for computing the Gromov-Hausdorff and interleaving distances between trees. *Journal of Computational Geometry*, 13:89–124, 2022. doi:10.20382/jocg.v13i1a4.

18 A.A. Valsangkar, J.M. Monteiro, V. Narayanan, I. Hotz, and V. Natarajan. An exploratory framework for cyclone identification and tracking. *IEEE Transactions on Visualization and Computer Graphics*, 25(3):1460–1473, March 2019. doi:10.1109/TVCG.2018.2810068.

19 L. Yan, T. B. Masood, F. Rasheed, I. Hotz, and B. Wang. Geometry aware merge tree comparisons for time-varying data with interleaving distances. *IEEE Transactions on Visualization and Computer Graphics*, 29(8):3489–3506, 2022. doi:10.1109/TVCG.2022.3163349.

20 L. Yan, T. B. Masood, R. Sridharamurthy, F. Rasheed, V. Natarajan, I. Hotz, and B. Wang. Scalar field comparison with topological descriptors: Properties and applications for scientific visualization. *Computer Graphics Forum*, 40(3):599–633, 2021. doi:10.1111/cgf.14331.

21 L. Yan, Y. Wang, E. Munch, E. Gasparovich, and B. Wang. A structural average of labeled merge trees for uncertainty visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):832–842, 2020. doi:10.1109/TVCG.2019.2934242.

22 P. Zhan, G. Krokos, D. Guo, and I. Hoteit. Three-dimensional signature of the Red Sea eddies and eddy-induced transport. *Geophysical Research Letters*, 46(4):2167–2177, 2019. doi:10.1029/2018GL081387.

23 P. Zhan, A.C. Subramanian, F. Yao, and I. Hoteit. Eddies in the Red Sea: A statistical and dynamical study. *Journal of Geophysical Research: Oceans*, 119(6):3909–3925, 2014. doi:10.1002/2013JC009563.

A Omitted proofs

▶ **Lemma 5.** *Assume T_1 and T_2 are in general position. After handling any event, the set F correctly represents all non-empty feasible pairs at the height of that event.*

Proof. We prove by induction on the number of events. After the first event there are no non-empty feasible pairs, which the algorithm correctly determines. Assume that the sweepline is at the r-th event; the induction hypothesis (IH) states that the set F' after handling the $(r-1)$-st event correctly represents all non-empty feasible pairs. Consider a non-empty set of points X of T_1 on the sweepline and a single point y of T_2 on the sweepline. We refer to the descendants of X and y on the sweepline at the $(r-1)$-st event as the *children* of X and y, respectively. Let I be the indices of points in X, and let j be the index of y. We need to show that (X, y) is feasible if and only if $I \in F[j]$.

If X does not contain any vertex in T_1 and y is not a vertex in T_2, then then let X' be the children of X, and let y' be the child of y. By construction, the points in X' and the point y' are indexed the same as the points in X and the point y, respectively. We do not address (I, j) in between these events, so $I \in F[j]$ if and only if $I \in F'[j]$. Similarly, the pair (X, y) is feasible if and only if (X', y') is feasible: any partial δ-good map α can be extended or restricted. By the IH, we know that (X', y') is feasible if and only if $I \in F'[j]$. Combining: (X, y) is feasible if and only if $I \in F[j]$. It remains to show the case that either X contains a vertex, or y is a vertex. We distinguish four cases. In each case, if (X, y) is feasible, α denotes a partial δ-good map from $T_1[X]$ to $T_2[y]$.

y is a leaf – (a) The only valid map from $T_1[X]$ to $T_2[y]$ takes each point of $T_1[X]$ to y. By assumption, the set X is non-empty. As no two events happen simultaneously, the subtree $T_1[X]$ must contain points below the sweepline. In other words, there cannot be a partial δ-good map from $T_1[X]$ to $T_2[y]$, and hence (X, y) is not feasible. The procedure correctly determines that $F[j]$ is empty; in particular, $I \notin F[j]$.

y is an internal vertex – (b) Let y_1, y_2 be the children of y and let X' be the children of X. By construction, the indices of the children of y are j and $j' > j$; without loss of generality assume y_1 has index j and y_2 has index j'. As no two events happen simultaneously, all indices in I are on the sweepline at the previous event. Suppose (X, y) is feasible. By **P2**,

all points in X have the same 2δ-ancestor. Let $X_1 \coloneqq \alpha^{-1}(y_1)$ and $X_2 \coloneqq \alpha^{-1}(y_2)$. By Observation 4, the restrictions of α to $T_1[X_1]$ and $T_1[X_2]$ are both partial δ-good maps. Let I_1 and I_2 be the sets of indices of X_1 and X_2, respectively. If I_1 is non-empty, then by the IH, it follows that $I_1 \in F'[j]$. Similarly, if I_2 is non-empty, then $I_2 \in F'[j']$. By definition, I_1 and I_2 are disjoint and their union is equal to I. If X_1 is empty, then by **P3**, the depth of the left subtree at y is at most 2δ, and symmetrically, if X_2 is empty, the depth of the right subtree at y is at most 2δ. In all cases, we get $I \in F[j]$.

For the other direction, suppose $I \in F[j]$. One of three cases must be true: (i) there are disjoint non-empty sets $I_1 \in F'[j]$ and $I_2 \in F'[j']$ such that $I_1 \cup I_2 = I$, (ii) $I \in F'[j]$ and the depth of the right subtree at y is at most 2δ, or (iii) $I \in F'[j']$ and the depth of the left subtree at j is at most 2δ. In case (i), by the IH, there exist partial δ-good maps α_1 from $T_1[X_1]$ to $T_2[y_1]$ and α_2 from $T_1[X_2]$ to $T_2[y_2]$. Their extensions to the height of the sweepline do not extend above any vertices of T_1, so by Observation 3 they are again partial δ-good maps. Together, they map each point of X to y, so their combination is a partial δ-good map from $T_1[X]$ to $T_2[y]$. Hence, (X, y) is feasible. It remains to show case (ii); case (iii) is symmetric. Since $I \in F'[j]$, by the IH, there is a partial δ-good map α from $T_1[X']$ to $T_2[y_1]$. The extension of I is a partial δ-good map from $T_1[X]$ to $T_2[y]$: **P1** and **P2** are trivially satisfied, and since the depth of the right subtree is at most 2δ, **P3** is also satisfied. So, the pair (X, y) is feasible.

X contains a leaf x – (c) Let X' denote the children of X, and let I' denote the set of indices of X'. Suppose (X, y) is feasible. If X' is empty, then by **P3**, it follows that the depth of y is at most 2δ, and after handling the event we have $I \in F[j]$. If X' is not empty, then as no two events happen simultaneously, all points of X' have the same image. Hence, the restriction of α to $T_1[X']$ is a partial δ-good map (by Observation 4). By the IH, it follows that $I' \in F'[j]$. Moreover, by **P2**, all points of X have the same 2δ-ancestor, so $I \in F[j]$.

To show the other direction, suppose $I \in F[j]$. If X' is empty, then the depth of $T_2[y]$ is at most 2δ. In other words, the map α with $\alpha(x) = y$ is a partial δ-good map from $T_1[X]$ to $T_2[y]$. So the pair (X, y) is feasible. Otherwise, if X' is not empty, we know that $I' \in F'[j]$ and that the ancestors of X' on the sweepline have the same 2δ-ancestor as x. By the IH, there exists a partial δ-good map α from $T_1[X']$ to $T_2[y']$, where y' is the child of y. By Observation 3, the extension of α to the sweepline is again a partial δ-good map. If we moreover set $\alpha(x) \coloneqq y$ we do not violate **P1**–**P3**, so the new map α is a partial δ-good map from $T_1[X]$ to $T_2[y]$. So, the pair (X, y) is feasible.

X contains an internal vertex x – (d) Let x_1, x_2 be the children of x, with indices i_1 and i_2. Without loss of generality, assume $i_1 < i_2$; by construction, x has index i_1, so $i_1 \in I$. Let X' denote the children of X with indices I'. Suppose (X, y) is feasible. By Observation 4, the restriction of α to $T_1[X']$ is a partial δ-good map. So, it follows by the IH that $I' \in F'[j]$. The set I' contains both i_1 and i_2, so the algorithm correctly determines $I = I' \setminus \{i_2\} \in F[j]$.

For the other direction, suppose $I \in F[j]$. Then the set I' must be in $F'[j]$. By the IH there is a partial δ-good map α from $T_1[X']$ to $T_2[y']$, where y' is the child of y. The extension of α to $T_1[X]$ is a partial δ-good map, so (X, y) is feasible. ◀

▶ **Lemma 6.** *Let (T, f) be a merge tree and let $(T_\varepsilon, f_\varepsilon)$ be any ε-perturbation for sufficiently small $\varepsilon > 0$. Then $d(T, T_\varepsilon) \leq \varepsilon$.*

Proof. Let η be a homeomorphism from T to T_ε that satisfies $|f(x) - f_\varepsilon(\eta(x))| \leq \varepsilon$. We define a map α from T to T_ε as follows. For each point x of T, the height of $\eta(x)$ is at most $f(x) + \varepsilon$. Hence, the ancestor y of $\eta(x)$ at height $f(x) + \varepsilon$ is well-defined. We define $\alpha(x) \coloneqq y$. By construction, the map α satisfies **P1**. Moreover, since η is a homeomorphism and we only take ancestors at a fixed height, the map α is continuous.

It remains to show that α satisfies **P2** and **P3**. Consider a point y of T_ε. If $y \in \operatorname{im}\alpha$, let x_1 and x_2 be distinct points in $\alpha^{-1}(y)$, and let x be the lowest common ancestor of x_1 and x_2. We have $f(x_1) = f_\varepsilon(y) - \varepsilon = f(x_2)$, so x_1 and x_2 lie on distinct edges. Since η is a homeomorphism, the points $\eta(x_1)$ and $\eta(x_2)$ also lie on distinct edges, so the lowest common ancestor y' of $\eta(x_1)$ and $\eta(x_2)$ must be a descendant of y. Both x and y' are vertices, so by choice of η, we get $\eta(x) = y'$ and $f(x) - f_\varepsilon(y') \leq \varepsilon$. In particular, it follows that $f(x) - f(x_1) \leq (f_\varepsilon(y') + \varepsilon) - (f_\varepsilon(y) - \varepsilon) \leq f_\varepsilon(y) + \varepsilon - f_\varepsilon(y) + \varepsilon = 2\varepsilon$. So **P2** holds.

If $y \notin \operatorname{im}\alpha$, then let x be the point of T such that $\eta(x) = y$; then $f(x) - f_\varepsilon(y) \leq \varepsilon$. Let $y' \coloneqq \alpha(x)$, which must be an ancestor of y in the image of α. By construction, $f_\varepsilon(y') - f(x) = \varepsilon$. We obtain $f_\varepsilon(y') - f(y) = (f_\varepsilon(y') - f(x)) + (f(x) - f_\varepsilon(y)) \leq 2\varepsilon$; this shows **P3**. ◀

▶ **Lemma 9.** $d(T_1, T_2, \mathcal{R}) \in \Delta(T_1, T_2, \mathcal{R})$.

Proof. Let α be an $\mathcal{R}$-restricted δ-good map, for some $\delta \geq 0$, and let δ' be the largest value in $\Delta(T_1, T_2, \mathcal{R})$ such that $\delta' \leq \delta$. We show that there always exists an $\mathcal{R}$-restricted δ'-good map α'; this directly implies the lemma. If $\delta' = \delta$ we are immediately done, so assume $\delta' < \delta$. We define the *lower endpoint* of a point x of T_1 as the lower endpoint of the edge in T_1 that contains x; if x is a vertex, then the lower endpoint of x is x itself. We use a similar shorthand for points of T_2. For each point x of T_1, let x' be the lower endpoint of x. Similarly, let y' be the lower endpoint of $\alpha(x')$. Both x' and y' are vertices, so the value $|f_1(x') - f_2(y')| \in \Delta(T_1, T_2, \mathcal{R})$. By choice of δ', it then follows that $f_2(y') \leq f_1(x') + \delta'$. Therefore, there exists an ancestor y of y' at height $f_1(x) + \delta'$. We define $\alpha'(x) = y$.

To show that α' is continuous, it suffices to show that $\alpha'(x_1)$ is a descendant of $\alpha'(x_2)$. Let x_1, x_2 be points of T_1 such that x_1 is a descendant of x_2. Let x_1' and x_2' be the lower endpoints of x_1 and x_2, respectively, and let y_1' and y_2' be the lower endpoints of $\alpha(x_1')$ and $\alpha(x_2')$, respectively. We directly obtain that x_1' is a descendant of x_2'. Since α is continuous, the point $\alpha(x_1')$ is a descendant of $\alpha(x_2')$, and as a result y_1' is a descendant of y_2'. It follows that $\alpha'(x_1)$ is a descendant of $\alpha'(x_2)$. So, α' is continuous.

It remains to show that α' satisfies **P1**–**P3**. By construction, **P1** directly holds. To see that **P2** holds, we first observe that $\alpha'(x)$ is a descendant of $\alpha(x)$ for all points x of T_1: since α is continuous, we know that $\alpha(x')$ is a descendant of $\alpha(x)$, where x' is the lower endpoint of x. Since $\alpha'(x)$ is an ancestor of $\alpha(x')$, it follows that $\alpha'(x)$ lies on the path between $\alpha(x')$ and $\alpha(x)$; so indeed, $\alpha'(x)$ is a descendant of $\alpha(x)$. Consider two points x_1 and x_2 of T_1 with $\alpha'(x_1) = y = \alpha'(x_2)$, and let x_1', x_2', and y' denote the lower endpoints of x_1, x_2 and y, respectively. Without loss of generality, we assume that $f_1(x_1') < f_1(x_2')$. Let h be the maximum of $f_1(x_2')$ and $f_2(y') - \delta$, and let x_1'' and x_2'' denote the ancestors of x_1' and x_2' at height h; note that by construction the point x_1'' lies on the edge between x_1' and x_1 (and similarly x_2'' lies between x_2' and x_2), and that $\operatorname{lca}(x_1'', x_2'') = x$. The points $\alpha(x_1'')$ and $\alpha(x_2'')$ are both ancestors of y' and hence lie on the same edge as y. As α is continuous, it follows that $\alpha(x_1'') = \alpha(x_2'')$. By **P2**, we thus obtain $f_1(x) - f_1(x_2'') \leq \delta$. If $h = f_1(x_2')$, then both x_2'' and x are vertices in T_1, so by choice of δ' it follows that $f_1(x) - f_1(x_2'') \leq \delta'$. In particular, this means that $f_1(x) - f_1(x_2) \leq \delta$. Otherwise, if $h = f_2(y') - \delta$, then since both y' and x are vertices we know that $f_1(x) - f_2(y') \leq \delta$. By choice of δ', it then follows that $f_1(x) - f_2(y') \leq \delta'$. Moreover, by construction of α', we know that $f_2(y') - f_1(x_2) \leq f_2(\alpha'(x_2)) - f_1(x_2) \leq \delta'$. Together, we obtain $f_1(x) - f_1(x_2) \leq 2\delta'$. So, α' satisfies **P2**.

Recall that for each point x, the point $\alpha'(x)$ is a descendant of $\alpha(x)$. As a result, if y is in the image of α, then it is also in the image of α'. It directly follows that α' satisfies **P3**. Lastly, consider a leaf x in T_1. The map α is $\mathcal{R}$-restricted, so $\alpha(x) \in \mathcal{R}[x]$. In particular, let y_x be

any leaf in $\mathcal{R}[x]$ that is a descendant of $\alpha(x)$. Then the height difference δ'' between x and y_x is at most δ. Moreover, $\delta'' \in \Delta(T_1, T_2, \mathcal{R})$. Hence, by choice of δ', we know that $\delta'' \leq \delta'$. It follows that $\alpha'(x) \in \mathcal{R}[x]$, so **P4** holds. ◀

▶ **Lemma 11.** *Assume T_1 and T_2 are in general position. After handling any event, the set F correctly represents all non-empty feasible pairs at the height of that event.*

Proof. We use an analogous proof as for Lemma 5, but we need additional arguments for handling **(c)** events. More precisely, consider a set of points X of T_1 on the sweepline, and a point y of T_2 on the sweepline, and let I and j denote the set of indices of X and the index of y, respectively. If X does not contain a leaf, then (X, y) is $\mathcal{R}$-feasible if and only if (X, y) is feasible, and (X, y) is feasible if and only if $I \in F[j]$. It remains to show that if X contains a leaf, then (X, y) is feasible if and only if $I \in F[j]$ after handling the corresponding event. We note here that Observations 3 and 4 hold also for $\mathcal{R}$-restricted δ-good maps.

***X* contains a leaf *x* – (c')** Let X' denote the children of X, and let I' denote the set of indices of X'. By construction, we have $I = I' \cup \{i\}$, where i is the index of x. Moreover, let h be the height of the sweepline within T_2, that is, $h \coloneqq f_2(y)$. Suppose (X, y) is feasible, that is, there exists an $\mathcal{R}$-restricted partial δ-good map from $T_1[X]$ to $T_2[y]$. In particular, this means that $y \in \mathcal{R}[x]$. In other words, we have $R_{ij} \leq f_2(y) = h$. If X' is empty, then by **P3**, it follows that the depth of y is at most 2δ, and hence $I = \{i\} \in F[j]$. If X' is not empty, then as no two events happen simultaneously, all points of X' have the same image. Hence, the restriction of α to $T_1[X']$ is an $\mathcal{R}$-restricted partial δ-good map. By the IH, it follows that $I' \in F'[j]$. Moreover, by **P2**, all points of X have the same 2δ-ancestor: we obtain $I \in F[j]$.

To show the other direction, suppose $I \in F[j]$. Then $R_{ij} \leq h$, so $y \in \mathcal{R}[x]$. If X' is empty, the depth of $T_2[y]$ is at most 2δ. Then, the map α defined by $\alpha(x) = y$ is an $\mathcal{R}$-restricted δ-good map from $T_1[X]$ to $T_2[y]$. In other words, (X, y) is $\mathcal{R}$-feasible. Otherwise, if X' is not empty, we know that $I' \in F'[j]$ and that the ancestors of X' on the sweepline and x have the same 2δ-ancestor. By the IH, there exists an $\mathcal{R}$-restricted partial δ-good map α from $T_1[X']$ to $T_2[y']$, where y' is the child of y. By Observation 3, the extension of α to the sweepline is again an $\mathcal{R}$-restricted partial δ-good map, and extending it by setting $\alpha(x) = y$ does not affect **P1**–**P3**. So, (X, y) is feasible. ◀