DOI: 10.1111/cgf.14547
Eurographics Conference on Visualization (EuroVis) 2022
R. Borgo, G. E. Marai, and T. Schreck
(Guest Editors)

Volume 41 (2022), Number 3

Branch Decomposition-Independent Edit Distances for Merge Trees

Florian Wetzels, Heike Leitte, and Christoph Garth

Technische Universität Kaiserslautern

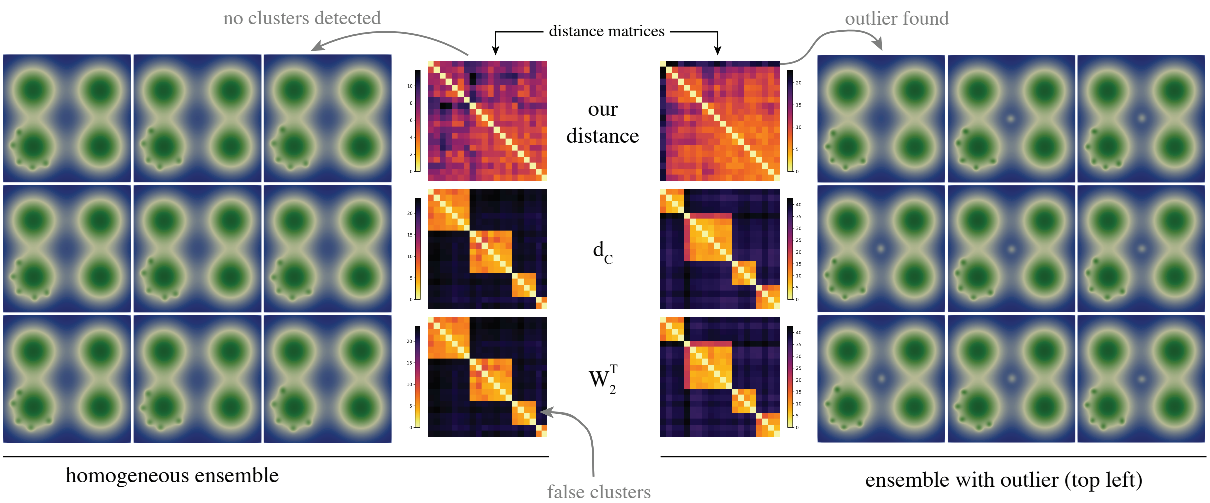

Figure 1: *Two scalar field ensembles (illustrated by nine representatives each) with clustermap distance matrices using branch mappings (top) and classic edit mappings (center and bottom). In both ensembles, false clusters are identified by previous approaches, which prevent detection of an outlier in the right ensemble (which is visible but not detectable without previous knowledge of the domain, i.e. its existence). Our new distance does not find the false clusters and is therefore able to identify the outlier clearly in the right example.*

Abstract

Edit distances between merge trees of scalar fields have many applications in scientific visualization, such as ensemble analysis, feature tracking or symmetry detection. In this paper, we propose branch mappings, a novel approach to the construction of edit mappings for merge trees. Classic edit mappings match nodes or edges of two trees onto each other, and therefore have to either rely on branch decompositions of both trees or have to use auxiliary node properties to determine a matching. In contrast, branch mappings employ branch properties instead of node similarity information, and are independent of predetermined branch decompositions. Especially for topological features, which are typically based on branch properties, this allows a more intuitive distance measure which is also less susceptible to instabilities from small-scale perturbations. For trees with $\mathcal{O}(n)$ nodes, we describe an $\mathcal{O}(n^4)$ algorithm for computing optimal branch mappings, which is faster than the only other branch decomposition-independent method in the literature by more than a linear factor. Furthermore, we compare the results of our method on synthetic and real-world examples to demonstrate its practicality and utility.

1. Introduction

The study or visualization of scalar fields, either acquired through simulation or real-world experiments, is a major avenue of research within scientific visualization. As computational power increases, so does the size and complexity of scalar fields. Furthermore, modeling of uncertainty is rapidly becoming increasingly popular as ensemble simulation is becoming more accessible. Due to this development, efficient measures for the similarity of scalar fields are of high interest, e.g in clustering tasks or outlier detection in scalar field ensembles, pattern or periodicity recognition in the analysis of time-dependent scalar fields, and symmetry or self-similarity detection within complex scalar fields of high dimension. Furthermore, the problem of finding mappings between substructures of two given objects is closely related to distance measures, as many algorithms such as edit distances compute both concurrently. Uses for such algorithms include feature tracking in time-varying scalar

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

fields and identifying common structures or features in ensemble data, which are of importance in many visualization scenarios.

Since working directly on a scalar field becomes computationally infeasible rather quickly, these tasks are often executed on abstract representations. One prominent example for those is the *contour tree* or a slightly simpler variant of it, the *merge tree* [EH10]. Both concepts originate in the fields of Topological Data Analysis, more specifically Persistent Homology [EH10]. They have a variety of applications (see [HLH*16]), either used directly as a (abstract) visualization of the represented scalar field, for generating visualizations, for improving interaction, or simply as a representation amenable to other methods, e.g. distance functions.

In this paper, we present a novel distance measure for scalar fields and merge trees that is based on the tree edit distance. While edit distances and related methods on topological structures have been used before to measure the distance between scalar fields (e.g. [SMKN20, PVDT21]), our method generalizes previous approaches to overcome restrictions that prevent these methods from capturing certain semantic properties of the data in the distance.

More precisely, tree edit distances work on mappings between nodes or edges of trees, which are local features, whereas the topological features of interest are global ones such as branches. Recent methods usually overcome this problem by working on branch decomposition trees or labeling the nodes with properties of their corresponding branch. Both approaches make the distance depend on a fixed branch decomposition, typically derived by the elder rule [EH10]. These are, however, not stable under small-scale perturbations, leading to semantically imprecise labels of the mapped features and therefore poor mappings. In this paper, we present a novel method that works on mappings between branches of arbitrary decompositions, making it therefore robust against such instabilities. We call them *branch mappings*. Figure 1 illustrates two example applications where branch mappings show better results than previous approaches. Furthermore, our method generalizes the notion of tree edit distances as a whole, and we aim to provide an entry point for further research into this field and its application in scientific visualization and computational topology, for which it was developed. In detail, our contributions are the following:

- We collect prior work on edit distances in topology-based visualization, and provide a novel categorization based on the desired characteristics discussed above. An overview is given in Table 1.
- We describe a *branch decomposition-independent* edit distance that focuses on paths and branches instead of nodes and edges; this stands in contrast to previous methods that are either branch decomposition-dependent or focus on edges.
- We illustrate the benefits of this distance in visualization applications by showcasing several synthetic and real-world examples.

After discussing related work in Section 2, we provide basic definitions, an introduction into tree edit distances and the categorization in Section 3.1. In Section 3, we define the concept of *branch mappings*, which are the core of our new distance measure and study it formally in detail. Application examples and practical comparisons with previous methods can be found in Section 5, before we conclude in Section 6 and give an outlook on future work.

2. Related Work

Similarity or dissimilarity measures based on topological abstractions of scalar fields, or more generally topological descriptors, have been studied in various ways and a variety of techniques have been proposed in increasing frequency. A recent survey by Yan et al. [YMS*21] gives a good overview over these methods. An introduction into topology based visualization methods in general can be found in the survey by Heine et al. [HLH*16].

Out of these methods, the works of Sridharamurthy et al. [SMKN20, SN21], Pont el al. [PVDT21], Saikia et al. [SSW14], Lohfink et al. [LWL*20] and Beketayev et al. [BYM*14] are most related to ours, as they are all based on edit mappings or similar mappings between contour or merge trees and their branch decomposition trees. In Section 3.2, we discuss these methods in detail in context together with the classification.

Furthermore, our work can be seen as a generalization of tree edit distances without the context of merge trees. An overview over various versions of tree edit distances and related problems was given by Bille in 2005 [Bil05]. Closely related techniques are the *constrained edit distance* [Zha96] and the *tree alignment distance* [JWZ94] on unordered rooted trees, or the *one-degree edit distance* on ordered trees [Sel77]. We provide a more detailed discussion of these methods in Section 3.3.

Topology-based Similarity Measures. Apart from edit distances on merge trees or contour trees, other distance measures based in topological descriptors have been used; among these, the persistence diagram is frequently used. Mapping persistence pairs of two scalar fields based on some metric between them is, in principle, largely equivalent to computing edit distances on branch decomposition trees; the only difference is that persistence diagrams do not take into account the nesting of the persistence pairs. Prominent examples of such distances / mapping methods are the *Wasserstein distance* [CSEHM10] and the *bottleneck distance* [CEH07]. Pont et al. [PVDT21] explicitly compare their edit distance method to these distance measures. An example for a more advanced method based on persistence diagrams is the work by Rieck et al. [RSL20].

Other graph-based distance measures work, for example, on Reeb Graphs [BDFL16, BGW13, CO17, SMP15], extremum graphs [NTN15], or also on merge and contour trees without specifically using an edit distance [MBW13, TN11, YWM*20].

Topology-based Feature Tracking. Edit mappings between topological descriptors can be used to track and visualize features in time-varying scalar fields. Here, they are typically applied to determine correspondence between topologically-characterized features e.g. across time. The distance measures discussed above can therefore all be used for this task (e.g. [LGW*21] or [PVDT21]). Furthermore, a further set of tracking techniques is based on topological descriptors directly without relying on edit distance (e.g. [OHW*17, EHMP04, BWP*10]). Finally, a last class of methods relies on topological descriptors only for feature *identification*, while the actual tracking is then done by other means, e.g. a measure of spatial overlap [LWM*17, SW17, SGL*16, SHD*20].

Topology-based Ensemble Visualization. Another area where often methods similar to edit distances are used for comparative analysis or similarity assessment is uncertainty visualization via ensembles, specifically the task of finding a representative for the topology of an ensemble of scalar fields. Lohfink et al. [LWL*20] and Pont et al. [PVDT21] derive a representative tree structure for the ensemble from the induced edit mappings. Other examples for contour tree based visualizations of ensemble or uncertain data can be found in the works of Wu and Zhang [WZ13], Kraus [Kra10] and Günther et al. [GST14]. An example that does not work on contour trees but persistence diagrams can be found in [TMMH14].

3. Merge Trees and Tree Edit Distances

In this section, we start with basic definitions of merge trees and notation for paths and branches in the Subsection 3.1. Then, we will give an overview over edit distances on general trees that are relevant for edit distances on merge trees in 3.2. After that, we will review previous approaches for mapping based merge tree distances and categorize them in 3.3. An overview is given in Table 1.

3.1. Merge Trees

A detailed introduction into persistent homology and a definition of merge trees can be found in [ELZ00] or [MW13]. For this paper, we restrict to an abstract model that captures merge trees of scalar fields of dimension > 1.

Abstract Merge Trees. We now define the abstract model for merge trees, to which we will refer as *abstract* merge trees. These should just be the class of trees, which can be interpreted as merge trees for some domain of dimension at least 2.

Definition 1 An unordered, rooted tree T of (in general) arbitrary degree (i.e. number of children) with node labels $f : V(T) \to \mathbb{R}$ is an *Abstract Merge Tree* if the following properties hold:

- The root node has degree one, $\deg(\mathrm{root}(T)) = 1$
- All inner nodes have a degree of at least two, $\deg(v) \neq 1$ for all $v \in V(T)$ with $v \neq \mathrm{root}(T)$
- All nodes have a larger scalar value than their parent node, $f(c) > f(p)$ for all $(c,p) \in E(T)$

(Abstract) Merge trees can have arbitrary further edge- or vertex-labels, e.g. persistence, volume or the actual segment of an arc.

Since the root of an abstract merge tree always has degree one and inner nodes do not, subtrees rooted in an inner node are not abstract merge trees themselves. Therefore, we identify subtrees by root edges, rather than root nodes: Formally, for a node p with children $c_1,\dots,c_k$, the subtree rooted in (c_1,p) is the "classic" subtree rooted in p where the subtrees rooted in $c_2,\dots,c_k$ are removed. Given an abstract merge tree T with subtree T' rooted in the edge (c,p), we define $T-T'$ to be the tree T'', which we obtain by removing all edges and all vertices of T' from T except the root p. If p has degree two in T, then we also remove it from T'', as otherwise p would be an inner node of degree one in T''. With this definition, it holds that T' and T'' are abstract merge trees as well.

From now on, we will often just use the term *merge tree* and it should be clear from the context if this refers to an abstract merge tree or an actual merge tree corresponding to a given scalar field.

Paths and Branches. As for general graphs, a *path* of length k in a merge tree T is a sequence of vertices $p = v_1 \dots v_k \in V(T)^k$ with $(v_i, v_{i-1}) \in E(T)$ for all $2 \leq i \leq k$ (note the strict root-to-leaf direction).We denote the edges within a path p by $\mathrm{edges}(p) := \{(v_i, v_{i-1}) \mid 2 \leq i \leq k\}$. A *branch* of T is a path that ends in a leaf. A branch $b = b_1 \dots b_k$ is a parent branch of another branch $a = a_1 \dots a_\ell$ if $a_1 = b_i$ for some $1 < i < k$. We also say a is a child branch of b.

A set of branches $B = \{B_1,\dots,B_k\}$ of a merge tree T is called a *Branch Decomposition* of T if $\{\mathrm{edges}(B_1),\dots,\mathrm{edges}(B_k)\}$ is a partition of $E(T)$. By $B(T)$ we denote the set of all branch decompositions of T. Every branch decomposition $B \in B(T)$ of an abstract merge tree contains exactly one branch $b \in B$ with $\mathrm{root}(T) \in b$. We call this branch the *main branch* of B.

The parent-child relations of the branches in a branch decomposition $B \in B(T)$ form a tree structure by themselves. The tree build from the vertex set $V = B$ and edge set

$$E = \{(a,b) \mid a,b \in B,\ b \text{ is a parent branch of } a\}$$

is called the *Branch Decomposition Tree* (BDT) of B.

Let T be a merge tree with $B \in B(T)$, T' a subtree of T rooted in (c,p) and $T'' = T - T'$. If there is a branch $b = b_1 \dots b_k$ in B that starts in (c,p), i.e. $p = b_1, c = b_2$, then B induces branch decompositions $B' \in B(T')$ and $B'' \in B(T'')$. B' and B'' can be obtained in the obvious way: let $a = a_1 \dots a_\ell$ with $b_1 = a_i$ be the parent branch of b. We put b into B' and we put the branch $a_1 \dots a_{i-1} a_{i+1} \dots a_\ell$ into B'' if p has degree two and a otherwise. All other branches from B are put into B' if they are descendants of b and into B'' otherwise. We denote the branch decompositions B' and B'' by $B[T']$ and $B[T'']$.

3.2. Distance Measures for Trees

Various distance measures for rooted unordered trees have been proposed and investigated in the last decades. In contrast to the ordered case, they differ significantly in computational complexity, ranging from quadratic algorithms of running time $\mathcal{O}(n \cdot m)$ to NP-hard or even MAX SNP-hard problems. We now provide a short overview over often-used methods to provide insight into the framework of tree edit distances in which we aim to place our new method. Furthermore, this elucidates the relation between the various other edit distances that have been applied to merge trees.

The classic edit distance for rooted, ordered, node-labeled trees was introduced by Tai [Tai79] and is defined to be the cost-optimal sequence of the following edit operations:

- **node-deletion**, where all children of the deleted node are appended to its parent,
- **node-insertion**, where a new child is inserted for some node and a subset of its children is made the children of the new node, and
- **node-relabel**, where the label of a node is changed.

The cost of the single operations is defined by some cost function $c : ((L \cup \{\perp\}) \times (L \cup \{\perp\})) \to \mathbb{R}_{\geq 0}$ where L is the set of node labels and $\perp$ stands for the null node needed to represent deletions and insertions. If the basic cost function defines a metric on the set $L \cup \{\perp\}$, then the edit distance between two trees defines a metric on the set of all rooted, unordered, L-labeled trees [Tai79].

Edit sequences induce a mapping between the vertices of the

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

trees, called *edit mappings* or *Tai mappings* [Tai79]. These are ancestor-preserving, one-to-one mappings between the two vertex sets. Their cost is defined as the sum of the relabel costs for all pairs in the relation representing the mapping together with the sum of all insertion/deletion costs for all vertices from both trees that are not present in the relation. These mappings are induced as all nodes in the first tree that are not changed within an edit sequence as well as those relabeled can be matched to one unique node in the second tree. Due to this correspondence between edit sequences and mappings, it can be shown that the cost of the optimal sequence and the optimal mapping is equal. Typical algorithms for edit distances therefore compute the induced mappings and their cost, not the actual edit sequence (which can of course be derived from it).

In contrast to the ordered case, where the general edit distance can be computed in cubic time [DMRW07, DG18], the problem of computing the edit distance for unordered trees (which we will call $\mathbf{d_e}$) is NP-hard [ZSS92] and even MAX SNP-hard [ZJ94], and therefore not arbitrarily close to approximate by a tractable algorithm unless P=NP. This has led to the development of many simplified versions of the edit distance. Two very popular approaches are tree alignments [JWZ94] and the constrained edit distance [Zha96].

An *alignment* of two trees is a supertree of both that is obtained by inserting in both trees until they are isomorphic. An alignment induces a mapping and corresponding costs in a similar fashion as before. The cost of the optimal alignment is equivalent to an edit distance where all insertions have to occur before all deletions. We denote this distance in the case for general trees as $\mathbf{d_a}$.

An edit mapping between two trees is called *constrained*, if it also fulfills the property that disjoint subtrees are strictly mapped to disjoint subtrees [Zha96]. The *constrained edit distance* is defined to be the cost of an optimal constrained edit mapping between two trees. We denote this distance in the case for general trees as $\mathbf{d_c}$.

Pont et al. [PVDT21] further restricted the constrained edit mappings specifically for the usecase of BDTs of merge trees. They introduced the constraint that if a node is deleted, i.e. mapped to null, the entire subtree rooted in this node has to be deleted, too. This distance is an unordered version of the 1-degree edit distance which has been proposed by Stanley Selkow [Sel77] for ordered trees. However, we do not provide a formal proof for this equivalence. We denote the distance on unordered trees by $\mathbf{d_1}$.

All three versions discussed can be computed more efficiently than the general edit distance. The alignment distance is still NP-hard [JWZ94] for trees of arbitrary degree, but can be computed in quadratic time if the trees have bounded degree [JWZ94], which is a reasonable assumption for merge trees (however, not true in general). Constrained and 1-degree edit distances are computable in polynomial time for both bounded degree trees and arbitrary degree trees, specifically quadratic time for bounded degree [Zha96, PVDT21], and in time $\mathcal{O}(n_1 \cdot n_2 \cdot (\deg_1 + \deg_2) \cdot \log(\deg_1 + \deg_2))$ for arbitrary degree. Furthermore, all three distances form a hierarchy in terms of how restrictive they are for the allowed mappings and therefore also their search spaces form a hierarchy of inclusions. For the four distances and arbitrary trees T_1, T_2 the following holds if they are applied using the same base metric (derived from a clear hierarchy of the recursions in [JWZ94, Zha96, Sel77]):

$$d_e(T_1,T_2) \le d_a(T_1,T_2) \le d_c(T_1,T_2) \le d_1(T_1,T_2).$$

	Edit Dist.	BDI	Global Prop.	Time
d_C [SMKN20]	✓ (d_c)	–	✓	$\Theta(n^2)$
W_2^T [PVDT21]	✓ (d_1)	–	✓	$\Theta(n^2)$
d_S [SSW14]	✓ (d_1)	–	✓	$\Theta(n^2)$
d_A [LWL*20]	✓ (d_a)	✓	–	$\Theta(n^2)$
d_M [BYM*14]	–	✓	✓	$\omega(n^5)$
d_B (This paper)	✓ (d_B)	✓	✓	$\mathcal{O}(n^4)$

Table 1: *The categorization of mapping based merge tree distances. A distance should be a true edit distance (Edit Dist.), should be branch decomposition-independent (BDI) and should use global properties (Global Prop.). The runtimes refer to bounded degree.*

3.3. Edit Distances for Contour Trees and Merge Trees

Distance metrics between contour trees or merge trees have been used as similarity measures for scalar fields in many varieties, especially through the use of tree edit distances and their corresponding edit mappings. We will now go through those previous approaches that are most similar to ours, which are those that either explicitly compute structure-preserving mappings between certain parts of the trees or at least internally use them to compute the distance based on a base metric for the mapped objects.

Categorization. To give a better intuition for the differences between these methods and our method, we categorize them in the following by two further properties (apart from the distinction whether they are explicitly edit distances or not). The first one is the type of the input objects. They can be actual contour or merge trees, or branch decompositions of these. Then, we distinguish the type of labels or properties of these features that are used by the base metric to compute distances of mapped pairs. Here we distinguish local and global properties as labels of the mapped objects. Examples could be edge persistence for a local property and branch persistence for a global property.

Sridharamurthy et al. [SMKN20] used the constrained edit distance $\mathbf{d_c}$ to measure the distance between merge trees. As a base metric for the merge tree nodes, they used L_∞ and overhang costs between the corresponding branches. In [SN21] they adapted this distance to also work on subtrees (similar to eBDGs in [SSW14]). Pont et al. [PVDT21] used a very similar distance to compute geodesics between merge trees and barycenter merge trees as a representative of an ensemble of scalar fields. They applied the 1-degree edit distance $\mathbf{d_1}$ to *unordered* (see Section 4.3 and App. A in the supp. material) branch decomposition trees using the Wasserstein metric to compare branches. Another closely related approach is the one by Saikia et al. [SSW14]. They also compute the one-degree edit distance $\mathbf{d_1}$ between branch decomposition graphs of two merge trees to find self-similarities within the scalar fields. In contrast to Pont et al. they considered *ordered* BDTs (see App. A in the supp. material). As for the categorization, these three methods fall into the same category: they use fixed branch decompositions as an input (Pont et al. and Saikia et al. even do so explicitly by working on BDTs) while using global properties (i.e. persistence, birth, death, etc. of branches) as labels. We will refer to these distances as d_C, W_2^T and d_S and by this mean the mapping method independent on the used base metric (which means we do use the notation W_2^T also for distances that do not use the Wasserstein metric for sin-

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

gle branches). We should note that Pont et al. use a normalization step as preprocessing to adapt the edit distance for geodesics and barycenters that modifies the branch labels according to the branch decomposition. To obtain a meaningful comparison, we only consider their metric without the normalization step, as it then fits better into the hierarchy of different edit distances.

Lohfink et al. [LWL*20] used tree alignments to represent the topology of a scalar field ensemble and to obtain a joint layout of all the contour trees in the ensemble. The underlying distance is the tree alignment distance $\mathbf{d_a}$. In contrast to labeling the nodes with their corresponding branches and using a base metric on branches, they labeled the nodes with their unique parent edge and used euclidean distances for different arc properties as the base metric. This approach gets actual contour trees as an input and uses local properties (edge properties) as labels. We will refer to this distance as d_A, again independent of the used base metric.

A branch decomposition-independent approach that does not fit into the concept of edit distances was introduced by Beketayev et al. [BYM*14] (d_M). They also compute a cost-optimal mapping between two branch decompositions of two given merge trees. However, in contrast to the edit distances, the total cost of such a mapping is not the sum of the mapped branches, but the value of the highest cost pair in the mapping. Furthermore, they also compute the optimal pair of branch decompositions that minimizes the costs. This falls into the category of methods that get contour trees as inputs (although their core method compares branch decompositions, they actually find the optimal one) and also use global properties (branch persistence etc.) for the base metric.

Advantages of using a true edit distance. Almost all approaches from the last section use an actual edit distance. A very useful property of edit distances is that they always come with a mapping of different features of the compared objects. In our case they induce a mapping between the nodes, edges or branches of the two input trees. In fact, given a cost function for matching two such features or deleting them, the edit distance value is equal to sum of all mapped or deleted features in the optimal mapping. We usually compute the distance and the mapping simultaneously.

An outlier in the list above is the method of Beketayev et al. [BYM*14]. They also consider underlying mappings, but since the value of the mapping is only the value of the worst match or largest deletion, the correspondence between mappings and distances is weaker. Moreover, their distance does not correspond to the cost of an edit sequence between the two input trees.

Using local properties only. The alignment distance used in [LWL*20] uses arc properties for the base metric. While independent of a specific branch decomposition, this approach has a substantial downside, which we will discuss in the following example. If one scalar field has two nested maxima of similar scalar value with the saddle connecting them close to the extreme points, and the other one only has one of the maxima, then the distance between the two trees should be small, as there is only a small split of the maximum from the second tree, which can also be seen as an insertion of one very small maximum branch. However, the persistence of the two arcs corresponding to the leaf nodes are both very small whereas the persistence of the arc leading to the single maximum in the other tree is large. Therefore the edit costs are high, contradicting the (intuitively) desired behaviour. Figure 2a shows two example trees where this problem appears. A practical example could be two trees where we apply a simplification by persistence to both of them and in one tree the second maximum stays whereas it is removed in the other tree.

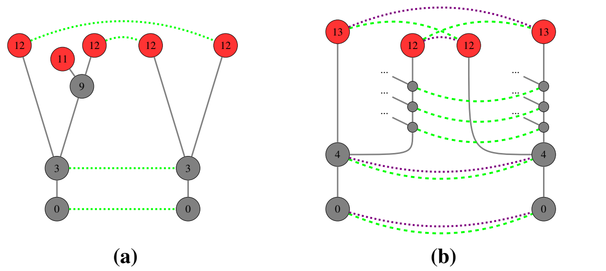

Figure 2: *Examples for matching problems. For local properties, in (a) we see two merge trees with the optimal alignment matching in green. Intuitively, the cost should be* 2*, as only one arc of persistence* 2 *is removed. However, the actual cost is* 8 *since mapping the right maximum in the left tree (persistence* 3*) to the left maximum in the right tree (persistence* 9*) has cost of* 6*. For a fixed branch decompostion, in (b) two merge trees with two different branch decompositions are shown. The two best matching options are shown in violet and green. Both yield non-optimal results in comparison to the desired distance.*

Using fixed branch decompositions. The problem stated in the last section can be fixed by using branch persistences instead of arc persistences. Most methods that we listed above make use of this approach. However, such a metric would depend on a fixed branch decomposition, but these are not unique and two badly chosen branch decompositions of two very similar trees can lead to huge distances. Usually, a very specific branch decomposition is used, namely the one derived by the elder rule, preferring the most persistent branches over small ones. But also this specific decomposition can have crucial instabilities which we will again see in an example. Consider a merge tree with two very persistent branches, one of which is feature-rich and one so simple that it does not have any child branches. Now consider another tree with the same properties. Then, for a low distance, we need to match the feature-rich branches in both trees onto each other as well as the simple branches. If, however, the two branches have similar persistence, the order of the branches can flip between the two trees. Then a good matching between the two branch decompositions has only the following options: either map the child branches of the two feature-rich branches onto each other and take the cost of the persistence difference of the main branches or match the main branches according to their persistence and take the cost of not matching the smaller branches. Figure 2b illustrates this example.

This problem can be illustrated on the following practical example. Consider a scalar field ensemble consisting of 20 scalar fields as illustrated in Figure 1 (all members shown in Figure 2, supp. material). All fields have four large peaks of similar size with five

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

smaller peaks arising from one of them. The height, size and exact position of all peaks are chosen randomly within small ranges. This leads to the following scenario: Which of the four main peaks (the one with smaller peaks or one of those without) represents the global maximum differs across the ensemble, and hence also the main branch of the branch decomposition derived by the elder rule differs. All fields are otherwise highly similar, as are their merge trees. There are no significant differences between the fields apart from small-scale, noise-like instabilities. However, the merge trees resemble those in Figure 2b and therefore cannot be mapped properly by branch decomposition-dependent distances.

On this dataset, we observe that d_C [SMKN20] and W_2^T [PVDT21], using the Wasserstein base metric proposed in [PVDT21], distinguish four clusters within the members, which does not convey an intuitive understanding of the data. A clustermap visualization of the edit distances can be seen in Figure 1. In contrast to that, our new distance function, which we define in the next section, does not find these false clusters and therefore resembles an intuitive understanding of the dataset much closer.

4. Branch Mappings

All of the methods listed above have one of the mentioned shortcomings. They either use branch properties to overcome the problem of local properties at the expense of the need of a fixed branch decomposition or they are independent of branch decompositions but only use local properties or they combine the positive aspects of both but do not offer the benefits of a true edit distance.

Here, we present a new approach that can be seen as a combination of the ones from [PVDT21], [SMKN20] and [BYM*14]. We modify the constrained tree edit distance so that it maps branches of two given merge trees onto each other and determine the total cost of such a mapping by taking the sum of distances of all mapped structures while computing the optimal branch decompositions for these costs at the same time. By staying within the notion of edit distances, we keep their global characteristics (in contrast to the more local concept of taking the cost of the worst matched feature pair) and the induced mappings, while still doing the optimization over all possible branch decompositions and mapping the actual features of interest (namely branches, not nodes or edges).

The distance we will introduce is based on the new concept of *branch mappings* that are similar to edit mappings and can be seen as an adaptation of those to branches of trees instead of nodes. Branch mappings can be applied to merge trees with arbitrary branch decompositions as well as merge trees with a fixed, given branch decomposition. They can be computed in $\mathcal{O}(n^2 \cdot m^2)$, which is faster than the only other branch decomposition-independent method (from [BYM*14]) by more than a linear factor. We achieve this by intertwining the dynamic programming that iterates all branch decompositions with the dynamic programming for the classic edit distance. However, this method of course also has its downsides. We will show that the branch mapping distance, while being a metric on the domain of branch decompositions of merge trees, it is not a metric on the set of merge trees. Furthermore, the computation cost is higher than for the typical tractable edit distances, which usually can be computed in around $\mathcal{O}(n \cdot m)$.

We start with the core concept underlying our distance, *branch mappings*. Similar to edit mappings, constrained edit mappings, or alignments, which are mappings between the vertex sets of two given trees with different restrictions to keep certain structures, branch mappings map branches of one tree to branches of another tree while keeping the ancestor relations of the mapped branches.

Definition 2 Given two abstract merge trees T_1, T_2, a branch mapping between T_1 and T_2 is a mapping $M \subseteq B_1 \times B_2$ with branch decompositions $B_1 \in B(T_1), B_2 \in B(T_2)$ such that

1. If $(a,b),(a',b') \in M$ then $a = a' \Leftrightarrow b = b'$,
2. $(m,m') \in M$, where m and m' are the main branches of B_1 and B_2,
3. If a and b are parent branches of a', b' and $(a',b') \in M$, then $(a,b) \in M$,
4. If $(a,b),(a',b') \in M$ and a_1 is a descendant of a'_1 then b_1 is a descendant of b'_1,

where $a = a_1...a_k, a' = a'_1...a'_{k'} \in B_1$ and $b = b_1...b_\ell, b' = b'_1...b'_{\ell'} \in B_2$ are arbitrary branches of the two trees.

Similar to branch decompositions, we define induced mappings as follows: let T'_1, T'_2 be subtrees of T_1, T_2, $B_1 \in B(T_1), B_2 \in B(T_2)$, $T''_1 = T_1 - T'_1, T''_2 = T_2 - T'_2$ and M be a branch mapping between B_1 and B_2. Then we define $M[B'_1]$ to be the map

$$\{(a,b) \in M \mid a \text{ is a descendant of } m'\}$$

where $B'_1 = B_1[T'_1]$ and m' is the main branch of B'_1. For $B''_1 = B_1[T''_1]$ we define $M[B''_1]$ to be the map

$$\{(a,b) \in M \mid a \notin M[B'_1] \text{ and } a \neq p\} \cup \{(p'', M(p))\}$$

where B'_1 and m' are defined as before, p is the parent branch of m' in B_1 and p'' the corresponding branch in B''_1. We define $M[B'_2]$ and $M[B''_2]$ analogously.

4.1. Branch Mapping Distance

In analogy to basing tree edit distances on their corresponding mappings, we now define a distance function on merge trees and their branch decompositions. For ease of notation, we define for a branch mapping M the set of all edit operations as

$$\overline{M} = M \cup \{(a,\perp) \mid \nexists (a,b) \in M\} \cup \{(\perp,b) \mid \nexists (a,b) \in M\}.$$

Given a cost function on branches of two merge trees T_1, T_2,

$$\mathrm{c} : \big(\big(\bigcup_{B_1 \in B(T_1)} B_1 \cup \{\perp\}\big) \times \big(\bigcup_{B_2 \in B(T_2)} B_2 \cup \{\perp\}\big)\big) \to \mathbb{R},$$

we define the cost of a branch mapping M to be

$$\mathrm{c}(M) = \sum_{(a,b) \in \overline{M}} \mathrm{c}(a,b).$$

For a branch decomposition $B \in B(T)$ of some merge tree, we define an empty mapping $M_\perp(B) = \{(b,\perp) \mid b \in B\}$ to be the mapping that corresponds to deleting the whole tree T. We can now define the distance functions based on the branch mappings.

Definition 3 Given two abstract merge trees T_1, T_2 with branch decompositions $B_1 \in B(T_1), B_2 \in B(T_2)$, define the distance function

$$d_B(B_1, B_2) = \min\{\mathrm{c}(M) \mid M \text{ is a branch mapping between } B_1, B_2\}.$$

Analogously, for two abstract merge trees T_1, T_2, we define

$$d_B(T_1,T_2) = \min\{d_B(B_1,B_2) \mid B_1 \in B(T_1), B_2 \in B(T_2)\}.$$

Furthermore, we also include empty trees into the definition of the branch mapping distance:

$$\begin{aligned} d_B(B_1,\perp) &= \mathrm{c}(M_\perp(B_1)), d_B(\perp,B_2) = \mathrm{c}(M_\perp(B_2)),\\ d_B(T_1,\perp) &= \min\{d_B(B_1,\perp) \mid B_1 \in B(T_1)\} \end{aligned}$$

and $d_B(\perp,T_2)$ symmetrically.

For practical purposes, we want to restrict the base metric for single branches or the branch labels to semantically meaningful ones. In our case, this means that those properties captured by the labels or costs are actually *branch* properties. Formally, we want a cost function $\mathrm{c}(a_1...a_k, b_1...b_\ell)$ between two branches to only depend on the start and end points of the branches, i.e. a_1, a_k, b_1 and b_ℓ. The number and position of child branches should be irrelevant. We then call it a *pure* branch distance.

4.2. Important Properties

Metric properties of the branch mapping distance We now show that d_B is a metric on the set of all branch decompositions of abstract merge trees but not on the set of all abstract merge trees.

Theorem 1 d_B is a metric on the set

$$\{(T,B) \mid T \text{ is an abstract merge tree}, B \in B(T)\},$$

as long as the cost function c on the branch labels is a metric.

Proof See supplementary material, App. B. □

Theorem 2 d_B is not a metric on the set of all abstract merge trees.

Proof If we consider the distance function d_B applied to merge trees with arbitrary branch decompositions, the triangle inequality does (in contrast to the previous result) no longer hold. The previous argument is not possible in this case since the optimal mapping between T_1, T_2 and T_2, T_3 can use a different branch decompositions which leads to mapping costs that are not possible with a single branch mapping. A counterexample can be found in Figure 3. □

Recursive Structure A core property of the branch mapping distance is the following: as long as the base metric is a pure branch metric, we can decompose a branch mapping into the different subtrees of the given branch decompositions and if we do so, we can do it in the same way for the cost of the mapping.

Lemma 1 Given two merge trees T_1, T_2 with branch decompositions B_1, B_2 and a branch mapping $M \subseteq B_1 \times B_2$, for any subtrees T_1' and T_2' of T_1 and T_2 for which $B_1[T_1']$ and $B_2[T_2']$ exist, and any pure branch distance c, it holds that:

- $\mathrm{c}(M) = \mathrm{c}(M[B[T_1']]) + \mathrm{c}(M[B[T_1 - T_1']])$,
- $\mathrm{c}(M) = \mathrm{c}(M[B[T_2']]) + \mathrm{c}(M[B[T_2 - T_2']])$.

Proof Follows directly from the pureness of c. □

Lemma 2 Given two merge trees T_1, T_2 with roots v_1, u_1, let v_2, u_2 be the unique children of the two roots and let those have children v_3, v_4 and u_3, u_4. Let T_1' be the subtree rooted in (v_2,v_3), T_1'' rooted in (v_2,v_4), T_2' rooted in (u_2,u_3) and T_2'' in (u_2,u_4). Let M be an optimal branch mapping for T_1 and T_2. Then, for the optimal cost of M it holds that:

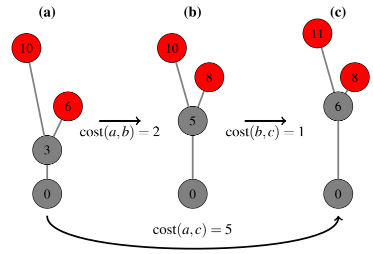

Figure 3: *Branch mapping distances for three merge trees using $|b_1 - b_2| + |p_1 - p_2|$ as base metric for branches with birth values b_1, b_2 and persistences p_1, p_2. The optimal map between (a) and (b) uses branches $(0,10)$, $(3,6)$ and $(5,8)$, whereas the one for (b) and (c) uses $(0,8)$, $(5,10)$ and $(6,11)$. These maps yield branch mapping distances of 2 and 1. The optimal map between (a) and (c) uses branches $(0,10)$, $(3,6)$, $(0,11)$ and $(6,8)$ which yields a total distance of 5. This violates the metric property as the direct distance between (a) and (c) is greater than the way over (b).*

- $d_B(T_1,T_2) = d_B(T_1',\perp) + d_B(T_1 - T_1', T_2)$ or
- $d_B(T_1,T_2) = d_B(\perp,T_2') + d_B(T_1, T_2 - T_2')$ or
- $d_B(T_1,T_2) = d_B(T_1'',\perp) + d_B(T_1 - T_1'', T_2)$ or
- $d_B(T_1,T_2) = d_B(\perp,T_2'') + d_B(T_1, T_2 - T_2'')$ or
- $d_B(T_1,T_2) = d_B(T_1',T_2') + d_B(T_1 - T_1', T_2 - T_2')$ or
- $d_B(T_1,T_2) = d_B(T_1'',T_2'') + d_B(T_1 - T_1'', T_2 - T_2'')$ or
- $d_B(T_1,T_2) = d_B(T_1',T_2'') + d_B(T_1 - T_1', T_2 - T_2'')$ or
- $d_B(T_1,T_2) = d_B(T_1'',T_2') + d_B(T_1 - T_1'', T_2 - T_2')$

Proof See supplementary material, App. C. □

If one of the trees only consists of two nodes, i.e. it only has one possible branch decomposition with one branch, we can just omit the corresponding recursions. Next, we consider the recursion for the case where one tree is empty.

Lemma 3 Given a merge tree T_1 with root v_1, let v_2 be the unique children of the root and let it have children v_3, v_4. Let T_1' be the subtree rooted in (v_2,v_3) and T_1'' rooted in (v_2,v_4). Then, for the optimal cost of $M_\perp(T_1)$ it holds that:

- $d_B(T_1,\perp) = d_B(T_1',\perp) + d_B(T_1 - T_1',\perp)$ or
- $d_B(T_1,\perp) = d_B(T_1'',\perp) + d_B(T_1 - T_1'',\perp)$,

and $d_B(\perp,T_2)$ decomposes symmetrically.

Proof See supplementary material, App. D. □

We should note that the previous lemmas only consider binary merge trees. However, it is easy to adapt for trees of arbitrary degree and we therefore omit this proof for better readability. We continue with the base cases of the recursion.

Lemma 4 Given two merge trees $T_1 = (\{v_1,v_2\},\{(v_2,v_1)\})$, $T_2 = (\{u_1,u_2\},\{(u_2,u_1)\})$ that only have one branch, the following holds for d_B:

- $d_B(T_1,\perp) = \mathrm{c}(v_1v_2,\perp)$,
- $d_B(\perp,T_2) = \mathrm{c}(\perp,u_1u_2)$ and
- $d_B(T_1,T_2) = \mathrm{c}(v_1v_2,u_1u_2)$.

Proof See supplementary material, App. E. □

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

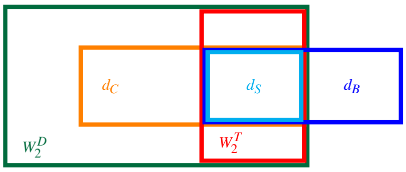

Figure 4: *A schematic illustration of the search spaces of branch based distance measures for merge trees or scalar fields. W_2^D (Wasserstein Distance between Persistence Diagrams, compare [PVDT21]) maps the branches of a fixed branch decomposition without any structural constraints. d_C and W_2^T restrict the search space by introducing structural constraints based on the merge trees. Since d_C allows for gaps in the mapping and W_2^T is not order preserving, the two models can be seen as orthogonal to each other. d_S also works on fixed branch decompositions and is strictly more restrictive than the others. For details, see App. A (supp. material). d_B then takes the structural constraints from d_S but lifts them to the space of arbitrary branch decompositions, thereby defining a model that is orthogonal to both d_C and W_2^T.*

We can use these lemmas to derive a recursive formula for $d_B(T_1,T_2)$ by taking the minimum of the different terms in each case. This gives a recursion very similar to the ones for the classic edit distances that can be computed with a dynamic programming approach. In the Section 4.4 we will discuss this algorithm.

4.3. Search Space Comparison to other Methods

We now want to illustrate more precisely how the branch mapping distance relates to other known distance measures. Since the contour tree alignments from [LWL*20] form some kind of outlier by only considering local properties, we will only look at those methods that use actual branch properties. We therefore try to fit the new method into a hierarchy of branch based methods that we extracted from the papers describing the two closest methods on merge trees [SSW14, SMKN20, PVDT21]. We do this by illustrating the relationship in a schematic drawing of the considered search spaces for mappings between branches/features shown in Figure 4.

4.4. Algorithm

In this section, we will use the recursive structure of optimal branch mappings shown in Lemmas 2-4 to derive a recursive algorithm that computes the branch mapping distance.

First, consider all subtrees T_1', T_1'', T_2', T_2'', $T_1 - T_1'$, $T_1 - T_1''$, $T_2 - T_2'$ and $T_2 - T_2''$ from Lemma 2. All of them (in all recursive steps) are abstract merge trees with a root of degree one. We can identify the trees appearing in this recursion by two nodes: the root node of the subtree and its unique child (which does not have to be a child in the original tree T_1/T_2). Therefore, we generally switch to identifying trees in this manner in contrast to identifying them just by one node like in the recursions for classic tree edit distances and define an algorithm computing $d_B(n_1,p_1,n_2,p_2)$ where n_1,n_2 stand for the current nodes and p_1,p_2 stand for the parent nodes of the considered subtrees. For example, $d_B(T_1,T_2)$ is represented by $d_B(v_1,v_2,u_1,u_2)$, whereas $d_B(T_1 - T_1',T_2 - T_2'')$ is represented by $d_B(v_4,v_1,u_3,u_1)$. Intuitively, p_1,p_2 can be considered as the last matched nodes in the currently tracked branch, which is illustrated in Figure 5. In App. F (supp. material), we present pseudo code showing the algorithm for binary trees and discuss how to adapt the algorithm for non-binary trees.

Running time. An obvious bound on the running time of Algorithm 1 in App. F (supp. material) using memoization is $\mathcal{O}(|T_1|^2 \cdot |T_2|^2)$, since the number of subproblems is constant, they always smaller than the current one and there are only $|T_1|^2 \cdot |T_2|^2$ many 4-tuples of nodes. However, this can be tightened to $\mathcal{O}(|T_1| \cdot \mathrm{depth}(T_1) \cdot |T_2| \cdot \mathrm{depth}(T_2))$, since we know that p_1,p_2 are always ancestors of n_1,n_2. A short discussion on the runtime for non-bounded degree can be found in App. F (supp. material).

These bounds of course only hold if the base metric is computable in constant time. Typical metrics like difference in persistence or bottleneck, Wasserstein and L_∞-distances are all constant time operations. For branch mappings, it is also important that these are pure branch metrics, since then they only depend on the start and end point of a branch, which are always available when evaluating the base metric (through n_1,p_1 or n_2,p_2).

Variants. There are a few adaptations that can be applied and should be noted here. First, saddle swap instabilities are a key problem of all tractable edit distances on merge trees and are of course still present in the branch mapping distance. To handle such instabilities, [SMKN20] and [PVDT21] use a preprocessing step to collapse small edges (through an ε-parameter) such that branches in a parent-child relation that are probable to be influenced by such instabilities become siblings instead. Of course, this preprocessing is also applicable to our method and should yield the same improvements, however, we did not implement or test it. Furthermore, it is possible to adapt our algorithm to not try all branch decompositions and use a fixed one instead. The computed distance and mapping then becomes equivalent to the one from [SSW14].

It is also possible to not just sum up the costs of all pairs in the optimal matchings but to sum up the squared pair costs and then take the square root of the sum. This approach was used in [PVDT21] and makes the resulting distance directly comparable to the classic Wasserstein distance for persistence diagrams. This is of course also possible with our method or d_C and we used this variant for the motivational examples in Section 1.

Another property to note is that branch mappings induce vertex mappings in the sense of constrained (or even 1-degree) edit mappings. Therefore, it is possible to construct an alignment of two trees by computing the optimal branch mapping. This alignment is a supertree of the two input trees and can therefore again be aligned. This gives us the possibility to use an iterated heuristic for an alignment of a set of merge trees (based on the branch mapping distance) as it has been done for contour trees in [LWL*20]. We plan to implement such a method in future work.

5. Experiments

In the following, we demonstrate the utility of our technique as a basis for typical tasks in visualization, and compare it against other,

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

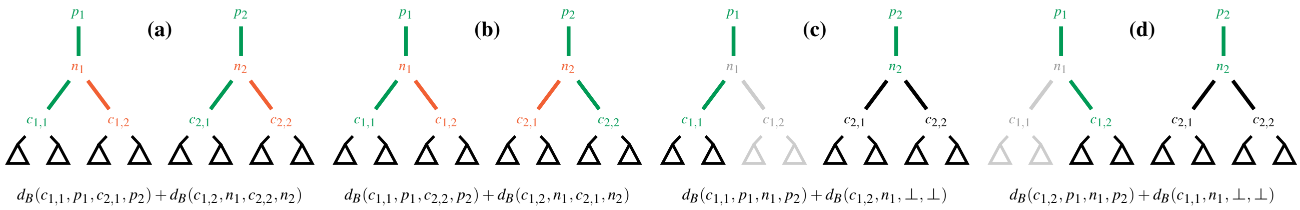

Figure 5: *Exemplary illustration of recursive structure of $d_B(n_1, p_1, n_2, p_2)$ in Algorithm 1 in App. F (supp. material). In (a) we continue main branch tracking with the left subtree in both trees and match the right subtrees onto each other, whereas in (b) we continue main branch tracking with the left subtree in T_1 and the right subtree in T_2 and match the other two subtrees onto each other. In (c), we continue main branch tracking with the left subtree in T_1 and delete the right subtree. In T_2, we do nothing. In (d), we continue main branch tracking with the right subtree in T_1, delete the left subtree and do nothing in T_2.*

similar techniques. The basis for these experiments is a straightforward Python implementation of the algorithm given in Section 4.4 and the constrained edit distance. Merge trees are computed using TTK [TFL*18], and ParaView [SAL*07] is used for result visualization. The three experiments described in the following correspond to three different uses of distance functions between scalar fields as found in many visualization applications: outlier identification, periodicity detection, and feature tracking. App. H (supp. material) discusses the results on noisy versions of the same data.

For benchmarks, we use a C++ implementation which is publicly available on Github as well as the easier to read Python version [WLG21]. Computation of individual distances for the here used trees with up to 80 vertices took only a few milliseconds. On noisier trees with sizes between 100 and 400, times went up to the range of seconds. For more details, see App. G (supp. material).

5.1. Outlier Detection

Outlier detection is a major concern in ensemble analysis, and the distance between merge trees of two scalar ensemble members can be used towards this. We consider a synthetic example dataset shown in Figure 1. This dataset is based on the example ensemble from Section 3.3, with one modification: we add a fifth peak in the middle of the four large peaks, but create an outlier by omitting this peak in a single ensemble member. Outlier detection then proceeds by computing the pairwise merge tree distances for all pairs of trees, using the squared sum of the mapping costs.

Using clustermaps to visualize the distance matrices, it is easy to see that the branch mapping distance (d_B) distinguishes the outlier clearly, whereas the constrained edit distance d_C and the Wasserstein distance W_2^T do not. For the latter two, the cost of the missing branch in the outlier ensemble member is concealed by the cluster effects when working with fixed branch decompositions. A visualization of the entire ensemble and further results obtained using additional parameter choice can be found in the supplementary material (App. H). Note that this dataset was constructed specifically to illustrate the problem of fixed branch decompositions, as discussed in Section 3.3. We make both datasets available publicly with our implementation [WLG21].

5.2. Periodicity Detection

Periodicity detection is an often encountered problem in time-varying data; we here consider a time-varying dataset consisting of 1001 time steps of a 400×50 grid representing the scalar velocity magnitude of the flow around a cylinder that forms a periodic Kármán vortex street. It was simulated by Weinkauf [WT10] using the *GerrisFlowSolver* [Pop04].

Reproducing the experiment of Sridharamurthy et al. [SMKN20] with our distance measure, we compute the merge tree distance matrix for all pairs of time steps and visualize this as a heatmap (cf. Figure 7). Here, the branch mapping distance exhibits exactly the same periodicity pattern as the constrained edit distance (Fig. 13 in [SMKN20]), giving a period/half-period of 75/37 time steps, respectively (see App. H (supp. material) for a direct comparison).

5.3. Feature Tracking over Time

As a last experiment, we utilize our distance metric for feature tracking within a time series of scalar fields. Here, we consider two different time series. The **SciVis contest 2008 dataset** [WN08] describes development of ion density during universe formation, and was also used as in a tracking case study in prior work [PVDT21]. Second, the **heated cylinder dataset** describes an ensemble of flows around a heated pole in a fluid. This dataset was also used in prior work on ensemble analysis [LWL*20], and we make it available publicly [WLG21]. We again consider velocity magnitude as the variable of interest for tracking.

For both datasets, we provide a proof-of-concept for the proposed branch mapping distance by replicating previous results. We are able to extract semantically meaningful matchings of features, comparable to those found earlier [LWL*20, PVDT21] from the branch mappings for both datasets (see App. H (supp. material) for a direct comparison). Visualizations of the matched features are shown in Figure 6, indicating that our distance measure is useful for tracking applications in topology-based visualization.

6. Conclusion and Outlook

In this paper, we presented a novel variant of edit distances tailored specifically to matching branches of merge trees. We provided a formal definition and analysis of metric properties as well as an algorithm and implementation. We showed that it is as expressive as

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

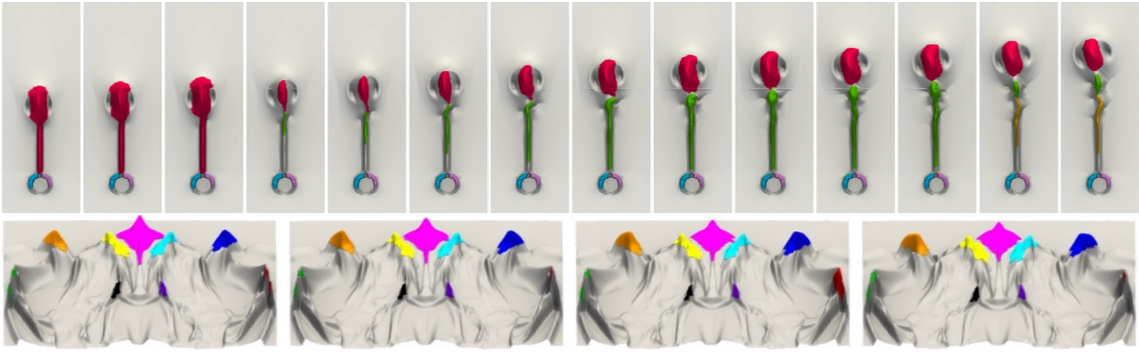

Figure 6: *Feature Tracking for the Heated Cylinder (top) and ion density (bottom) datasets. We rendered all maxima matched by the optimal branch mapping in consistent colors over all time points. The matching for the heated cylinder shows how the rising plume splits up over time into two and then three distinct maxima while the two peaks around the pole stay constant. The rendering for the ion density dataset shows that the branch mapping represents the intuitively correct mapping of the corresponding peaks, which is for example not true for a mapping obtained by Wasserstein distances on persistence diagrams according to [PVDT21].*

previous approaches on practical datasets and that it can improve the quality of the matching and distance measure significantly on datasets containing specific structures. We also discussed limitations of the new method: its higher complexity and the fact that it is not a metric on merge trees, which does, however, not influence its utility for the provided tasks. In the context of our classification, we have described an edit distance-based method that works on merge trees and uses global properties. However, this method does so in a very specific and restricted way, which we now discuss together with options for less restrictive methods and future work.

Other Branch Decomposition-Independent Methods. A restriction we used was to only map *branches*. We could generalize this to arbitrary *paths* of a merge tree. As branches are paths, this would just extend the concept of branch mappings by again increasing the search space. Such *path mappings* would actually be computable in the same time and space bounds. In future work, we want to investigate the formal foundations of path mappings.

Another restriction is that the new distance corresponds to the 1-degree edit distance in the classic setting. We also want to consider adaptions of constrained edit mappings, alignments, the general edit distance or even unrooted versions of those. For these options, we have to find suitable definitions and study their complexity to see which of these options are applicable in practice.

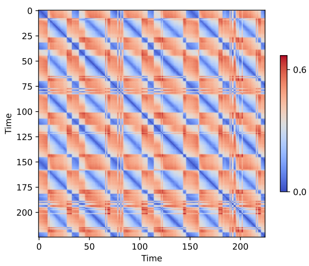

Figure 7: *A heatmap of the distance matrix for all pairs of the first 225 time steps of the vortex street dataset (full matrix cf. supp. material, App. H), clearly indicating a period/half-period 75/37.*

Other Applications. Furthermore, as hinted in Section 4.4, we plan to integrate branch mappings (and maybe path mappings) into the alignment framework introduced in [LWL*20] (which is definitely possible) and investigate the possibility of combining it with the geodesic/barycenter methods introduced in [PVDT21]. The latter seems more likely to be possible for certain variants of path mappings which we believe to be a metric, since the normalization step from [PVDT21] seems hard to apply to non-fixed branch decompositions and barycenter computation could be hindered by the missing triangle inequality. A very typical application of distance measures for scalar fields that we have not studied in this paper is symmetry or self-similarity detection. As the memoization table computed by the algorithm includes distances for all subtrees of both input trees, it is possible to use it to find substructures of high similarity by applying methods similar to those from [SSW14]. We aim to implement them in future work, too. Furthermore, we plan to investigate if similar distances can be defined for contour trees instead of merge trees. Hurdles to overcome here would be, for example, that paths in contour trees are not necessarily monotone, and by restricting to monotone branches, the impact of saddle-swap instabilities increase significantly.

Parallel Implementation. For a more efficient implementation, we want to look deeper into optimization options and also develop a parallel implementation of our algorithm. This could lead to a significantly improved performance, as the branching factor in the $\mathcal{O}(n^4)$-sized search space is high, which (intuitively) should lead to high parallelization potential.

Acknowledgements.

We would like to thank Julien Tierny, Mathieu Pont, Markus Anders and Jonas Lukasczyk for valuable discussions. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 442077441.

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

References

[BDFL16] BAUER U., DI FABIO B., LANDI C.: An edit distance for Reeb graphs. In *Proceedings of the Eurographics 2016 Workshop on 3D Object Retrieval* (Goslar Germany, Germany, 2016), 3DOR '16, Eurographics Association, pp. 27–34. doi:10.2312/3dor.20161084. 2

[BGW13] BAUER U., GE X., WANG Y.: Measuring distance between Reeb graphs. *Proceedings of the Annual Symposium on Computational Geometry* (07 2013). doi:10.1145/2582112.2582169. 2

[Bil05] BILLE P.: A survey on tree edit distance and related problems. *Theoretical Computer Science 337*, 1-3 (2005), 217–239. doi:10.1016/j.tcs.2004.12.030. 2

[BWP*10] BREMER P.-T., WEBER G., PASCUCCI V., DAY M., BELL J.: Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics 16*, 2 (Mar. 2010), 248–260. doi:10.1109/tvcg.2009.69. 2

[BYM*14] BEKETAYEV K., YELIUSSIZOV D., MOROZOV D., WEBER G. H., HAMANN B.: Measuring the distance between merge trees. In *Topological Methods in Data Analysis and Visualization III, Theory, Algorithms, and Applications*, Bremer P., Hotz I., Pascucci V., Peikert R., (Eds.). Springer, 2014, pp. 151–165. doi:10.1007/978-3-319-04099-8_10. 2, 4, 5, 6

[CEH07] COHEN-STEINER D., EDELSBRUNNER H., HARER J.: Stability of persistence diagrams. *Discret. Comput. Geom. 37*, 1 (2007), 103–120. doi:10.1007/s00454-006-1276-5. 2

[CO17] CARRIÈRE M., OUDOT S.: Local equivalence and intrinsic metrics between reeb graphs. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia* (2017), Aronov B., Katz M. J., (Eds.), vol. 77 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 25:1–25:15. doi:10.4230/LIPIcs.SoCG.2017.25. 2

[CSEHM10] COHEN-STEINER D., EDELSBRUNNER H., HARER J., MILEYKO Y.: Lipschitz functions have lp-stable persistence. *Foundations of Computational Mathematics 10*, 2 (2010), 127–139. doi:10.1007/s10208-010-9060-6. 2

[DG18] DUDEK B., GAWRYCHOWSKI P.: Edit distance between unrooted trees in cubic time. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic* (2018), Chatzigiannakis I., Kaklamanis C., Marx D., Sannella D., (Eds.), vol. 107 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 45:1–45:14. doi:10.4230/LIPIcs.ICALP.2018.45. 4

[DMRW07] DEMAINE E. D., MOZES S., ROSSMAN B., WEIMANN O.: An optimal decomposition algorithm for tree edit distance. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings* (2007), Arge L., Cachin C., Jurdzinski T., Tarlecki A., (Eds.), vol. 4596 of *Lecture Notes in Computer Science*, Springer, pp. 146–157. doi:10.1007/978-3-540-73420-8_15. 4

[EH10] EDELSBRUNNER H., HARER J.: *Computational Topology - an Introduction*. American Mathematical Society, 2010. URL: http://www.ams.org/bookstore-getitem/item=MBK-69. 2

[EHMP04] EDELSBRUNNER H., HARER J., MASCARENHAS A., PASCUCCI V.: Time-varying reeb graphs for continuous space-time data. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (New York, NY, USA, 2004), SCG '04, Association for Computing Machinery, p. 366–372. doi:10.1145/997817.997872. 2

[ELZ00] EDELSBRUNNER H., LETSCHER D., ZOMORODIAN A.: Topological persistence and simplification. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA* (2000), IEEE Computer Society, pp. 454–463. doi:10.1109/SFCS.2000.892133. 3

[GST14] GÜNTHER D., SALMON J., TIERNY J.: Mandatory critical points of 2d uncertain scalar fields. *Comput. Graph. Forum 33*, 3 (2014), 31–40. doi:10.1111/cgf.12359. 3

[HLH*16] HEINE C., LEITTE H., HLAWITSCHKA M., IURICICH F., DE FLORIANI L., SCHEUERMANN G., GARTH C.: A survey of topology-based methods in visualization. *Computer Graphics Forum 35* (06 2016), 643–667. doi:10.1111/cgf.12933. 2

[JWZ94] JIANG T., WANG L., ZHANG K.: Alignment of trees - an alternative to tree edit. In *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, California, USA, June 5-8, 1994, Proceedings* (1994), Crochemore M., Gusfield D., (Eds.), vol. 807 of *Lecture Notes in Computer Science*, Springer, pp. 75–86. doi:10.1007/3-540-58094-8_7. 2, 4

[Kra10] KRAUS M.: Visualization of uncertain contour trees. In *IMAGAPP 2010 - Proceedings of the International Conference on Imaging Theory and Applications and IVAPP 2010 - Proceedings of the International Conference on Information Visualization Theory and Applications, Angers, France, May 17 - 21, 2010* (2010), Richard P., Braz J., (Eds.), INSTICC Press, pp. 132–139. doi:https://doi.org/10.5220/0002817201320139. 3

[LGW*21] LOHFINK A. P., GARTZKY F., WETZELS F., VOLLMER L., GARTH C.: Time-varying fuzzy contour trees. In *2021 IEEE Visualization Conference, IEEE VIS 2021 - Short Papers, New Orleans, LA, USA, October 24-29, 2021* (2021), IEEE, pp. 86–90. doi:10.1109/VIS49827.2021.9623286. 2

[LWL*20] LOHFINK A. P., WETZELS F., LUKASCZYK J., WEBER G. H., GARTH C.: Fuzzy contour trees: Alignment and joint layout of multiple contour trees. *Comput. Graph. Forum 39*, 3 (2020), 343–355. doi:10.1111/cgf.13985. 2, 3, 4, 5, 8, 9, 10

[LWM*17] LUKASCZYK J., WEBER G., MACIEJEWSKI R., GARTH C., LEITTE H.: Dynamic nested tracking graphs. *Computer Graphics Forum 36*, 3 (2017), 12–22. doi:10.1111/cgf.13164. 2

[MBW13] MOROZOV D., BEKETAYEV K., WEBER G.: Interleaving distance between merge trees. *Discrete and Computational Geometry 49*, 22-45 (2013), 52. 2

[MW13] MOROZOV D., WEBER G. H.: Distributed merge trees. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013* (2013), Nicolau A., Shen X., Amarasinghe S. P., Vuduc R. W., (Eds.), ACM, pp. 93–102. doi:10.1145/2442516.2442526. 3

[NTN15] NARAYANAN V., THOMAS D. M., NATARAJAN V.: Distance between extremum graphs. In *2015 IEEE Pacific Visualization Symposium, PacificVis 2015, Hangzhou, China, April 14-17, 2015* (2015), Liu S., Scheuermann G., Takahashi S., (Eds.), IEEE Computer Society, pp. 263–270. doi:10.1109/PACIFICVIS.2015.7156386. 2

[OHW*17] OESTERLING P., HEINE C., WEBER G. H., MOROZOV D., SCHEUERMANN G.: Computing and visualizing time-varying merge trees for high-dimensional data. In *Mathematics and Visualization*. Springer International Publishing, 2017, pp. 87–101. doi:10.1007/978-3-319-44684-4_5. 2

[Pop04] POPINET S.: Free computational fluid dynamics. *ClusterWorld 2*, 6 (2004). URL: http://gfs.sf.net/. 9

[PVDT21] PONT M., VIDAL J., DELON J., TIERNY J.: Wasserstein distances, geodesics and barycenters of merge trees. *IEEE Transactions on Visualization and Computer Graphics* (2021), 1–1. doi:10.1109/TVCG.2021.3114839. 2, 3, 4, 6, 8, 9, 10

[RSL20] RIECK B., SADLO F., LEITTE H.: Hierarchies and ranks for persistence pairs. In *Topological Methods in Data Analysis and Visualization V: Theory, Algorithms, and Applications* (02 2020). doi:https://doi.org/10.1007/978-3-030-43036-8_1. 2

[SAL*07] SQUILLACOTE A. H., AHRENS J., LAW C., GEVECI B., MORELAND K., KING B.: *The paraview guide*, vol. 366. Kitware Clifton Park, NY, 2007. 9

[Sel77] SELKOW S. M.: The tree-to-tree editing problem. *Inf. Process. Lett. 6*, 6 (1977), 184–186. doi:10.1016/0020-0190(77)90064-3. 2, 4

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.

[SGL*16] SHU Q., GUO H., LIANG J., CHE L., LIU J., YUAN X.: Ensemblegraph: Interactive visual analysis of spatiotemporal behaviors in ensemble simulation data. In *2016 IEEE Pacific Visualization Symposium, PacificVis 2016, Taipei, Taiwan, April 19-22, 2016* (2016), Hansen C., Viola I., Yuan X., (Eds.), IEEE Computer Society, pp. 56–63. doi:10.1109/PACIFICVIS.2016.7465251. 2

[SHD*20] SCHNORR A., HELMRICH D. N., DENKER D., KUHLEN T. W., HENTSCHEL B.: Feature tracking by two-step optimization. *IEEE Trans. Vis. Comput. Graph. 26*, 6 (2020), 2219–2233. doi:10.1109/TVCG.2018.2883630. 2

[SMKN20] SRIDHARAMURTHY R., MASOOD T. B., KAMAKSHIDASAN A., NATARAJAN V.: Edit distance between merge trees. *IEEE Trans. Vis. Comput. Graph. 26*, 3 (2020), 1518–1531. doi:10.1109/TVCG.2018.2873612. 2, 4, 6, 8, 9

[SMP15] SILVA V., MUNCH E., PATEL A.: Categorified Reeb graphs. *Discrete and Computational Geometry 55* (01 2015). doi:10.1007/s00454-016-9763-9. 2

[SN21] SRIDHARAMURTHY R., NATARAJAN V.: Comparative analysis of merge trees using local tree edit distance. *IEEE Transactions on Visualization and Computer Graphics* (2021), 1–1. doi:10.1109/TVCG.2021.3122176. 2, 4

[SSW14] SAIKIA H., SEIDEL H., WEINKAUF T.: Extended branch decomposition graphs: Structural comparison of scalar data. *Comput. Graph. Forum 33*, 3 (2014), 41–50. doi:10.1111/cgf.12360. 2, 4, 8, 10

[SW17] SAIKIA H., WEINKAUF T.: Global feature tracking and similarity estimation in time-dependent scalar fields. *Comput. Graph. Forum 36*, 3 (2017), 1–11. doi:10.1111/cgf.13163. 2

[Tai79] TAI K.: The tree-to-tree correction problem. *J. ACM 26*, 3 (1979), 422–433. doi:10.1145/322139.322143. 3, 4

[TFL*18] TIERNY J., FAVELIER G., LEVINE J. A., GUEUNET C., MICHAUX M.: The topology toolkit. *IEEE Trans. Vis. Comput. Graph. 24*, 1 (2018), 832–842. doi:10.1109/TVCG.2017.2743938. 9

[TMMH14] TURNER K., MILEYKO Y., MUKHERJEE S., HARER J.: Fréchet means for distributions of persistence diagrams. *Discret. Comput. Geom. 52*, 1 (2014), 44–70. doi:10.1007/s00454-014-9604-7. 3

[TN11] THOMAS D. M., NATARAJAN V.: Symmetry in scalar field topology. *IEEE Trans. Vis. Comput. Graph. 17*, 12 (2011), 2035–2044. doi:10.1109/TVCG.2011.236. 2

[WLG21] WETZELS F., LEITTE H., GARTH C.: Branch decomposition-independent edit distances (supplementary source code). https://github.com/scivislab/bdi-ed, 2021. 9

[WN08] WHALEN D., NORMAN M. L.: Competition data set and description, in 2008 ieee visualization design contest, 2008. URL: http://vis.computer.org/VisWeek2008/vis/contests.html. 9

[WT10] WEINKAUF T., THEISEL H.: Streak lines as tangent curves of a derived vector field. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2010) 16*, 6 (November - December 2010), 1225–1234. URL: http://tinoweinkauf.net/. 9

[WZ13] WU K., ZHANG S.: A contour tree based visualization for exploring data with uncertainty. *International Journal for Uncertainty Quantification 3*, 3 (2013). doi:https://doi.org/10.1615/INT.J.UNCERTAINTYQUANTIFICATION.2012003956. 3

[YMS*21] YAN L., MASOOD T. B., SRIDHARAMURTHY R., RASHEED F., NATARAJAN V., HOTZ I., WANG B.: Scalar field comparison with topological descriptors: Properties and applications for scientific visualization. *Comput. Graph. Forum 40*, 3 (2021), 599–633. doi:10.1111/cgf.14331. 2

[YWM*20] YAN L., WANG Y., MUNCH E., GASPAROVIC E., WANG B.: A structural average of labeled merge trees for uncertainty visualization. *IEEE Trans. Vis. Comput. Graph. 26*, 1 (2020), 832–842. doi:10.1109/TVCG.2019.2934242. 2

[Zha96] ZHANG K.: A constrained edit distance between unordered labeled trees. *Algorithmica 15*, 3 (1996), 205–222. doi:10.1007/BF01975866. 2, 4

[ZJ94] ZHANG K., JIANG T.: Some MAX snp-hard results concerning unordered labeled trees. *Inf. Process. Lett. 49*, 5 (1994), 249–254. doi:10.1016/0020-0190(94)90062-0. 4

[ZSS92] ZHANG K., STATMAN R., SHASHA D. E.: On the editing distance between unordered labeled trees. *Inf. Process. Lett. 42*, 3 (1992), 133–139. doi:10.1016/0020-0190(92)90136-J. 4

© 2022 The Author(s)
Computer Graphics Forum © 2022 The Eurographics Association and John Wiley & Sons Ltd.