





Fuzzy Contour Trees: Alignment and Joint Layout of Multiple Contour Trees

Anna-Pia Lohfink ¹, Florian Wetzels ¹, Jonas Lukasczyk ², Gunther H. Weber ³, Christoph Garth ¹

¹Technische Universität Kaiserslautern, Germany

²Arizona State University

³Lawrence Berkeley National Laboratory and University of California, Davis

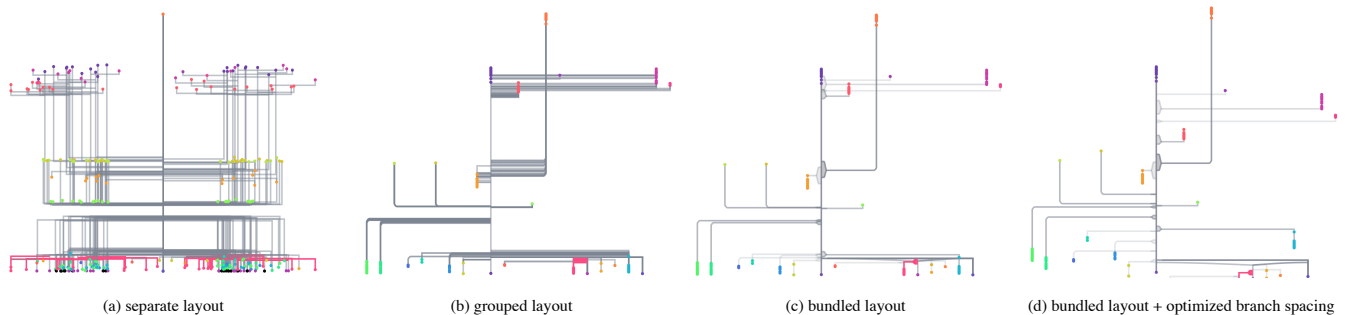


Figure 1: An illustration of the fuzzy contour tree layout. (a) Separate layout of multiple contour trees yields a cluttered representation, while grouping (b) and bundling (c) lay out aligned branches jointly. To better leverage vertical space, saddles can be shifted (d). In each layout, the same branch on the bottom is highlighted in pink.

Abstract

We describe a novel technique for the simultaneous visualization of multiple scalar fields, e.g. representing the members of an ensemble, based on their contour trees. Using tree alignments, a graph-theoretic concept similar to edit distance mappings, we identify commonalities across multiple contour trees and leverage these to obtain a layout that can represent all trees simultaneously in an easy-to-interpret, minimally-cluttered manner. We describe a heuristic algorithm to compute tree alignments for a given similarity metric, and give an algorithm to compute a joint layout of the resulting aligned contour trees. We apply our approach to the visualization of scalar field ensembles, discuss basic visualization and interaction possibilities, and demonstrate results on several analytic and real-world examples.

1. Introduction

Topology-based methods have a long tradition in the visualization of scalar fields. Founded on mathematical principles, they provide an abstract representation of scalar field structure. Among a variety of methods, the contour tree serves as the well-understood basis for a plethora of techniques, ranging from the straightforward generation of visualization images (e.g. [PCMS09]) to clever analysis user interfaces (e.g. [WDC*07a]).

As modeling and simulation of uncertainty are becoming increasingly prominent aspects of computational science, however, it has proven challenging to adapt topology-based visualization to the resulting novel data modalities. Here, we consider the example of contour tree visualization of ensemble data sets. Such ensembles

result from sampling of parameter spaces and stochastic properties of models, and consist of multiple realizations of a model, called ensemble members. Identification of similarities and differences between members and detection of outliers are among the elementary analysis tasks that should be supported by visualization. The randomized nature of prevalent contour tree layout techniques and their large parameter spaces often result in strongly different representations for very similar scalar fields. Thus, in naïve form, these techniques are not suited to the needs of ensemble visualization.

In this paper, we propose a novel strategy for the joint visualization of many contour trees: the *fuzzy contour tree*. We utilize tree alignments—a graph-theoretical concept similar to edit distance mappings—to identify common branches across multiple

contour trees, where commonality is identified through a semantically meaningful similarity metric. Using such alignments, we devise an algorithm to lay out common branches for all trees identically, and improve this basic resulting layout through the use of edge bundling and further abstraction. In essence, we replace the independent layout of each tree by the layout of a common super-tree, and use this to draw individual trees in a similar manner. This approach yields a coherent, easy-to-interpret representation of multiple contour trees. We leverage our approach to provide improved contour tree visualization of ensemble data sets. After discussing related work in Section 2, we make the following contributions:

- We describe the application of tree alignment to contour trees, and provide a heuristic algorithm to quickly compute the alignment of multiple contour trees with a problem-specific similarity metric in Section 3.
- We devise a layout algorithm that uses both alignment and individual trees to achieve a simultaneous, easy-to-interpret visualization of multiple contour trees, with basic interaction possibilities (Section 4).
- We identify elementary visualization tasks for visualization of scalar ensembles, and show how our technique supports these tasks. Several examples are used to demonstrate this (Section 5).

The work we present here is intended to provide proof-of-concept towards the use of tree alignments for layout of multiple contour trees, and we provide a detailed discussion of our approach and a comparison to other topology-based ensemble visualization approaches in Section 6. We conclude with a summary of open questions, limitations and further opportunities inherent in our approach (Section 7).

2. Related Work

Contour Trees in Visualization. Our work is rooted in the topology-based visualization of scalar fields using contour trees [CSA03]. Such visualizations provide a mathematically well-founded and effective abstraction, and have been used to define features of interest [BWP*10] or provide user interfaces [KRS03, WDC*07b, CSv04]. A general overview of topology-based techniques is given by Heine et al. [HLH*16].

Applying topology-based visualization to multiple scalar fields at once has several major use cases. In ensemble analysis, an understanding of commonalities and differences between ensemble members is sought [WHLs19], while the study of time-dependent scalar fields aims to identify feature evolution over time [BWP*10]. In both cases, an important problem is to establish feature correspondence by topological means. A common approach is to use branches or sub-trees of contour trees to characterize regions that are then examined for correspondence using overlap measures; however, this does not take the contour tree structure into account. An example is the comparison of two scalar fields based on contours obtained from the contour tree by Schneider et al. [SWC*08]. Similarly, Lukaszczuk et al. uses merge tree segmentations to compute the correlation between features [LWM*17]. Space-filling structures in turbulent flows are tracked by Schnorr et al. using the volume overlap of 3D Morse-Smale cells, which serve as input to a maximum-weight, maximal matching [SGKH15].

Instead of considering the spatial overlap of topologically-characterized regions of scalar fields, a further class of methods focuses primarily on correspondence directly from a graph-centric perspective. For example, Saikia et al. [SSW14] compare all sub-trees of two merge trees against each other to find repeating structures, and Thomas and Natarajan [TN11] adopt a similar approach to identify symmetries in scalar fields.

Heine et al. present a review of different layout strategies for contour trees as well as a new layout method [HSCS11]. We adapt their orthogonal layout method for our fuzzy contour trees. This orthogonal layout method, as well as most commonly used layout methods for sufficiently large contour trees, is at heart a randomized algorithm, since optimal layout is an NP-hard problem. Apart from direct computation [CSA03], contour trees can be extracted in parallel [CWS*19], and generalizations of contour trees to multidimensional data are presented by Carr and Duke [CD13].

Ensemble Visualization using Contour Trees. A use case we consider in this paper is the visualization of scalar field ensembles through the contour trees of the ensemble members. Visualizing the information in, and differences between, multiple trees was achieved e.g. by Schulz et al. through an edge-bundled visualization of multiple samples from a probabilistic graph model [SNG*17]. Location and sub-tree structure uncertainty of two different graphs were visualized by Lee et al. [LRCP07]. Shu et al. discuss EnsembleGraphs to visualize hierarchical clustering across an ensemble [SGL*16]. A recent survey on graph visualization was given by Hu et al. [HN18].

Contour trees of uncertain scalar fields were considered by Kraus [Kra10]. Here, two contour trees of morphologically filtered versions of an uncertain volume data set represent the range of uncertainty, visualized by combining both trees in one image. Günther et al. [GST14] also use two realizations of an uncertain scalar field that represent estimations of the support of the PDF of the input data. They characterize mandatory critical points in the given range of realizations and provide mandatory merge and split trees.

Contour tree-based uncertainty visualization as proposed by Wu et al. [wZ13] includes a layout algorithm for contour trees. The idea of assigning slots to branches is identical to the one by Heine et al. that we use in this paper. However, the proposed space-saving strategy is not applicable in our case, since a separation in upward/downward and mixed branch zones is not possible considering multiple saddles and leaves for each branch. The same authors visualize the mean contour tree obtained from the pointwise ensemble mean, with uncertainty added from contour differences between individual members.

In the fuzzy contour tree, we are able to visualize collections of scalar fields quantifying uncertainty indirectly. It is thus possible and desirable to incorporate information from individual contour trees into the overall visualization. In contrast to the work by Wu et al. [wZ13], the fuzzy contour tree is not calculated using a mean of all data points, but is based on a topology-driven matching of individual contour trees. Thus, outlier behavior remains clearly visible.

Distance and Merging of Graph-Based Topological Descriptors. A central aspect of the work we present here is aimed at identifying an optimal matching of contour trees to create a common

layout. We therefore provide a short overview of the major classes of techniques in this active research area.

The general problem of finding a distance between (rooted) trees arises in different fields of computer science, such as computational biology [SZ90], AI [KTSK00] and code compilation [HO82]. Various types of edit distances, based on defining a cost function for edit operations in trees, have been applied to solve this problem, with the tree edit distance [Tai79] being the most general and complex approach. An overview is given by Bille [Bil05]. Tree alignments, a computationally cheaper alternative, were introduced by Jiang et al. [JWZ94]. Apart from being easier to compute, alignments exhibit some properties which make them a good fit for our purpose. Those properties will be explained in Section 3 as well as our method to apply the alignment to an ensemble of more than two unrooted, unordered contour trees.

Recently, different types of edit distances have been applied to merge trees and other graph based descriptors representing the topology of a scalar field. Saikia et al. [SSW14] applied the 1-degree edit distance to branch decomposition trees of merge trees to find self similarities in scalar fields. Sridharamurthy et al. used the constrained edit distance [Zha96] on merge trees for feature tracking in time-dependent data [SBKN18]. Beketayev et al. [BYM*14] propose a method to compare merge trees based on the minimum edit distance between all possible branch decompositions of the two compared trees. Rieck et al. use the edit distance for ordered trees on persistence hierarchies [RSLng]. Moreover, many metrics other than edit distances have been proposed for merge trees, often obtained by restricting a metric on the more general Reeb graph [BDFL16, BGW13, CO17, SMP15, MBW13]. Yan et al. introduced a metric between labeled merge trees, allowing the definition of an average of several merge trees [YWM*19]. In contrast to this, in our work, the distance between single contour trees is not important, but a matching of their nodes is required to achieve a common layout. In addition, the resulting matching needs to incorporate all paths and all features of the single contour trees. This makes the edit distance in terms of the tree alignment the preferred approach for our purpose, resulting in a super-tree with the required properties.

Contour trees yield more difficulties than merge trees when searching for a distance metric or matching algorithm. They are in general more complex data structures with potentially high variance for small changes in the considered scalar field. As recently shown by Hristov et al., also branch decomposition poses additional challenges for contour trees [HCng]. Applying their method to contour trees, Saikia et al. [SSW14] describe similar problems. Therefore, we conclude that the application of edit distances and general merging of contour trees is not yet understood well. In this paper, we adapt and apply tree alignments to contour trees to obtain a matching and a super-tree of multiple contour trees.

3. Tree Alignment of Contour Trees

We aim to devise a combined representation or layout of multiple contour trees that respects and leverages similarities among the trees and the scalar fields they represent to facilitate common, topology-based analytical tasks. A central problem is therefore to

identify such similarities. This can be accomplished – on a tree level – by constructing a matching between the nodes and arcs of all individual contour trees, such that matched nodes and arcs correspond to similar structures in the scalar fields. A good way to find such matchings is using tree edit distances, which induce a mapping of nodes in the compared trees [Bil05] such that the trees become minimally different w.r.t. edit distance. An interesting approach for merge trees is described by Sridharamurthy et al. [SBKN18].

In brief, edit distance between two *labeled* trees measures the minimum number of operations (i.e. insert, delete, relabel) required to transform one tree into the other. More generally, operations can carry arbitrary cost, and a cost-minimal sequence of edit operations is sought. An edit sequence S for two trees T_1 and T_2 induces a mapping of the vertices $M_S \subset V(T_1) \times V(T_2)$ where for all $(v_1, w_1), (v_2, w_2) \in M_S$

- $v_1 = v_2$ if and only if $w_1 = w_2$, and
- v_1 is an ancestor of v_2 if and only if w_1 is an ancestor of w_2 .

Given two ordered trees T_1 and T_2 , the edit distance $\delta(T_1, T_2)$ can be computed in time $\mathcal{O}(|T_1| \cdot |T_2| \cdot |L_1| \cdot |L_2|)$ using dynamic programming, where L_1 and L_2 are the depths of the trees [Tai79]. Given two unordered rooted trees T_1, T_2 , the problem of computing the value of $\delta(T_1, T_2)$ is known to be NP-hard [Bil05].

Contour trees are unordered, unrooted trees, thus general edit distance is not suitable for our purpose. However, many restricted variants of the edit distance, which are easier to compute and applicable to unrooted trees, have been introduced [Bil05]. From these, we utilize *tree alignments* and the corresponding *tree alignment distance*. A tree alignment \mathcal{A} of trees T_1, \dots, T_n is a super-tree of the aligned trees, i.e. it contains each aligned tree as a sub-tree. In general, \mathcal{A} is not unique and can be computed from each individual tree through sequences of insert operations and node relabelings. A *minimal tree alignment* minimizes a cost function over the edit sequences that yield \mathcal{A} from each T_i , thus intuitively providing a "small" alignment that captures the similarity between the individual trees.

In comparison to general edit mappings, whose computation is NP-hard, minimal alignments can be found in quadratic time in the number of nodes for (arbitrarily rooted) contour trees. Furthermore, an important property towards a joint layout of contour trees is the *path property*: all paths in the individual trees map to paths in the super-tree. A detailed description and comparison of these concepts can be found in the survey by Bille [Bil05].

We next consider alignment of two general trees, and on this basis proceed to describe an algorithm for computing an alignment of n contour trees.

3.1. Minimal Contour Tree Alignment

An alignment of two trees T_1 and T_2 is obtained by first inserting nodes labeled with a blank symbol λ into T_1 and T_2 , making them isomorphic. Let T'_1, T'_2 be the resulting trees and T_{align} be the unlabeled tree isomorphic to both. Labeling a node $v \in V(\mathcal{A})$ with $l(v) := (l(v_1), l(v_2))$, where v_1 and v_2 are the nodes in T'_1 and T'_2 corresponding to v , and l is the labeling, gives the alignment \mathcal{A} .

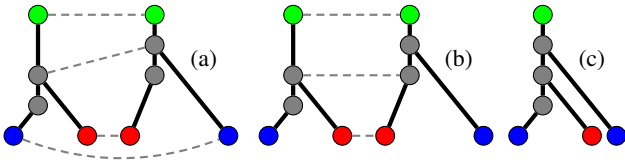


Figure 2: Differences between alignment and edit distance: (a) An intuitive mapping between the two trees; it can only be achieved with an edit distance mapping, as the lower gray node must be deleted as a parent of the blue node and a new gray node must be inserted as parent of the red node. In an alignment though, insertions must occur before deletions. The minimal alignment and corresponding mapping are shown in (b) and (c). Matching the blue nodes is impossible since it would result in a cycle.

The aligned label $l(v)$ represents an edit operation, and is associated with a cost $\gamma(l(v_1), l(v_2))$, which is an arbitrary metric. The overall cost of \mathcal{A} is then

$$\gamma(\mathcal{A}) = \sum_{v \in V(\mathcal{A})} \gamma(l(v)),$$

allowing to define the alignment distance as the minimal cost. $\delta_{\text{align}}(T_1, T_2) = \min\{\gamma(\mathcal{A}) \mid \mathcal{A} \text{ is alignment for } T_1, T_2\}$. Thus, each \mathcal{A} minimizing $\gamma(\mathcal{A})$ is a minimal tree alignment for the chosen metric. It corresponds to a restricted edit distance, where all insertions are performed before all deletions. This yields the super-tree property, and nodes labeled without λ represent the induced matching.

Differences between Alignment and Edit Distance. It is illustrative to highlight a number of differences between tree alignments and edit mappings. From an edit distance mapping between T_1 and T_2 , one can construct a tree of the mapped nodes in a natural way (following from the mapping properties). This tree will always be a sub-tree of T_1 and T_2 . Given the cost function

$$\gamma(\lambda, l) = \gamma(l, \lambda) = 1, \gamma(l_1, l_2) = \begin{cases} 0 & \text{if } l_1 = l_2 \\ 2 & \text{otherwise} \end{cases}$$

the minimal alignment from T_1 to T_2 will be the smallest common super-tree, and the sub-tree induced by the minimal edit sequence will be the largest common sub-tree [Bil05]. Therefore, alignment mappings are not able to match certain corresponding structures; consider e.g. the alignment and edit mapping in Figure 2: the more intuitive matching cannot be achieved by an alignment, a restriction inherent in tree alignments.

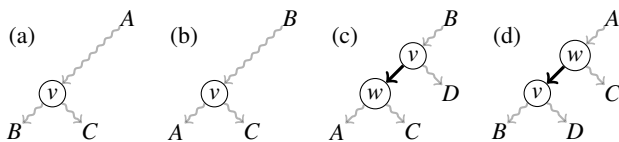


Figure 3: Illustration of consistent root and path properties. If the root A of the alignment (a) is not kept consistent and changed to B in (b), the following alignment could be (c), which violates the ancestor property for v and C if again rooted in A (see (d)).

However, the super-tree property provides substantial advantages. First, it allows the construction of a heuristic for aligning more than two trees (cf. Section 3.2). Furthermore, a super-tree of all contour trees contains all features (critical points) of the original fields. In contrast, an edit mapping only induces a sub-tree. The most important advantage of alignments in our context however is reduced computational complexity: for two unordered trees with bounded degree, the alignment can be computed in time $\mathcal{O}(|T_1| \cdot |T_2|)$; this assumption is fulfilled for contour trees in most practical settings (e.g. in the strongly prevalent piecewise linear case). In contrast, the corresponding edit distance problem is NP-hard [Bil05].

3.2. Alignment Heuristics

We extend the minimal alignments introduced above for two trees to n trees as follows. Given n scalar fields, the alignment of the corresponding contour trees can be used as a representation of the topology of the ensemble. In general, the problem of aligning n trees is again known to be NP-hard, even for bounded degree or ordered trees, since it is a generalization of the multiple sequence alignment [WJ94]. Thus, direct computation is not feasible. Furthermore, the alignment procedure requires rooted trees, whereas contour trees are unrooted. To address both problems, we adopt the following interlocking heuristics:

Sequential Alignment of Multiple Trees. Let \mathcal{A}_2 be the minimal alignment of T_1 and T_2 , and define \mathcal{A}_{k+1} as the minimal alignment of T_{k+1} and \mathcal{A}_k . The final matching is the one induced by \mathcal{A}_n .

In this manner, we construct an alignment of n trees sequentially. This alignment will in general not be a minimal alignment. However, \mathcal{A}_2 contains all features of T_1 and T_2 . Aligning a third tree T_3 which has features similar to T_1 but not to T_2 , with \mathcal{A}_2 , the resulting alignment \mathcal{A}_3 will still match them, since they are present in \mathcal{A}_2 and T_3 . For example, consider the two trees in Figure 2. The blue and red nodes are swapped. If further trees with this swap are aligned, there will likely be two blue and two red nodes in the alignment. Our experiments (cf. Section 5) indicate that this heuristic works well in practice and is cheap to compute.

Rooting Contour Trees. To align two unrooted contour trees, it appears possible to minimize alignment over all possible choices of roots. For the sequential alignment, this can however lead to problems; in Figure 3, the edit mapping property is violated after aligning with respect to different roots. This problem does not arise if the root of the alignment is kept consistent. Thus, in each step of the sequential alignment, the alignment node corresponding to the previously chosen roots has to be chosen as the root of the new alignment as well. In contrast, the root of T_{k+1} can be chosen freely to obtain an optimal result.

3.3. Cost Metrics

The cost of edit operations that induce the minimal alignment can be chosen as an arbitrary metric, providing flexibility in steering minimal alignments towards matching nodes that are semantically related. For example, nodes can be labeled by scalar value, and the

difference between the scalar values of two nodes can be chosen as cost. A similar construction, independent of the absolute scalar value, can be obtained by labeling nodes in a rooted tree with the difference in scalar value to their origin, i.e. with the persistence of the unique edge pointed to this node. Again, the metric is the difference of the two values. One could also use the area corresponding to this edge in the field, or the sum or product of several of these quantities, depending on application needs. Following Sakia et al. [SSW14], we call the size of the edge segmentation *volume*, independent of the dimension, and the product of *volume* and *persistence metric* the *combined metric*. In their use case, the combined metric performed best. A purely combinatorial matching is possible by defining fixed costs per edit operation type, but this appears less useful in the application scenarios we envision here.

Furthermore, a meaningful way to combine labels of the form $(l(v_1), l(v_2))$ into a single label after each alignment step needs to be chosen. For example, for scalar value labels, the average of $l(v_1)$ and $l(v_2)$ can be chosen as the new label. Similar constructions can be used for the other examples discussed above.

Importantly, to preserve the semantics of the individual contour trees in an alignment, we penalize the matching of nodes of different critical point type (minimum, maximum, saddle) by choosing prohibitively large cost for such relabelings. Hence, we ensure that it is always cheaper to insert a non-matching new node than to match critical points of different types.

3.4. Algorithm

The overall algorithm to approximate the minimal alignment for n contour trees T_1, \dots, T_n with cost metric c is shown in Algorithm 1.

The runtime of the above algorithm is in $\mathcal{O}(n^2 \cdot |V_{max}|^4)$ for n trees, where $|V_{max}|$ is the number of nodes of the largest tree. Because this is still expensive for large trees, and it is not sensible to lay out contour trees with hundreds of nodes, we apply contour tree simplification (e.g. [CSv04]) before alignment. This results in very good computation times for trees with several hundreds of nodes, as given in Table 1 for the examples discussed in Section 5.

The given algorithm is at heart, a randomized algorithm; finding an ordering of trees to ensure optimal alignment is an NP-hard problem, and thus we turn to randomization and randomly permute the input ordering of trees, as is done in many other algorithms that would otherwise have to employ exhaustive combinatorial search. In practice, to increase repeatability, the random ordering is computed using a fixed chosen seed. In our experiments, we have found that while alignments differ, the quality of the resulting layouts is largely independent of the chosen seed. Figure 10 shows layouts resulting from two different seeds for the same set of contour trees.

Our algorithm applies to arbitrary choices of the cost metric c , which can be chosen to suit the needs of a particular application domain. We provide corresponding examples in Section 5.

3.5. Properties of the Contour Tree Alignment

The output of our algorithm is an alignment of the n contour trees, where each T_i is rooted in a chosen leaf, and all roots are matched

Algorithm 1: Heuristic for minimal alignment of n contour trees

```

Let  $\mathcal{A}_{min}$  be some alignment tree with infinite cost
foreach leaf  $r_1$  of  $T_1$  do
  Let  $T_1^{r_1}$  be  $T_1$  rooted in  $r_1$ 
   $\mathcal{A} = T_1^{r_1}$ 
  for  $i = 2 \dots n$  do
    Let  $\mathcal{A}'_{min}$  be some alignment tree with infinite cost
    foreach leaf  $r_i$  of  $T_i$  do
      Let  $T_i^{r_i}$  be  $T_i$  rooted in  $r_i$ 
       $\mathcal{A}' = \text{align}(\mathcal{A}, T_i^{r_i})$ 
      if  $c(\mathcal{A}') < c(\mathcal{A}'_{min})$  then
        |  $\mathcal{A}'_{min} = \mathcal{A}'$ 
      if  $c(\mathcal{A}'_{min}) < c(\mathcal{A})$  then
        |  $\mathcal{A} = \mathcal{A}'_{min}$ 
    if  $c(\mathcal{A}) < c(\mathcal{A}_{min})$  then
      |  $\mathcal{A}_{min} = \mathcal{A}$ 

```

to the root of the alignment. \mathcal{A} fulfills a set of properties that are important for the layout algorithm:

- \mathcal{A} is a super-tree, therefore all inner nodes of the individual trees are matched to inner nodes of the alignment and all leaves of the alignment represent leaves of the individual trees.
- The alignment preserves the node type, i.e. the alignment nodes also have a specific type (minimum, maximum or saddle).
- All paths in individual contour trees which start at the chosen root are matched to sub-paths in the alignment (path property).

Some properties complicate laying out the fuzzy contour tree:

- In contrast to contour trees, an inner node of the alignment can be a minimum or maximum. For visualization purposes, these inner extrema nodes can be turned into leaf nodes by attaching their children to their parent node. (cf. Figure 4 and Section 4.1).
- Matched leaves from different contour trees are not necessarily connected to a single matched saddle. However, the path property ensures that different saddles will be either on the parent branch of the leaf in the alignment or in its sub-tree.
- For a saddle node in the alignment, that matches saddle nodes from the member trees, there is not necessarily a single leaf node matching leaves from exactly the same member trees.

4. Fuzzy Contour Trees

In the following, we describe a layout algorithm that allows an intuitive joint depiction of multiple contour trees in a sensible manner – the *fuzzy contour tree*. The layout procedure centrally relies on the alignment described in the previous section: It is used to achieve

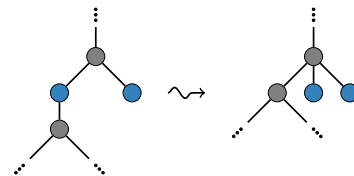


Figure 4: Turning an inner minimum/maximum node into a leaf.

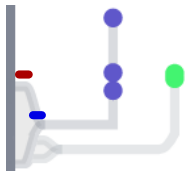


Figure 5: Bundled edges. Red: plateau-width, blue: bundle-width. The child branch is expanded to compensate the bundling.

layouts of the individual contour trees that are then combined into the fuzzy contour tree.

In order to achieve a high recognition factor for the fuzzy contour tree, we use the well-established and often-used orthogonal layout [HSCS11] as a basis for our algorithm. In this layout, branches are drawn as vertical lines, and are connected by saddles, which are drawn as horizontal lines rather than points. Finding an orthogonal layout for the alignment (and thus for all aligned contour trees) is done in analogy to finding a layout for (single) contour trees. First, a branch decomposition is recursively established, then the resulting branches are assigned horizontal positions, with the vertical positions of the nodes given by their isovalues. For the individual contour (sub-)trees of the alignment, matched nodes are assigned equal positions. They are then combined, and further layout improvements for the resulting fuzzy contour tree are performed to significantly increase visual clarity. In the following, we describe differences to the layout approach for single contour trees and these layout improvements in detail.

4.1. Branch Decomposition of the Alignment

A key ingredient in contour tree layout is the branch decomposition. To identify a branch decomposition of a contour tree, first, a root and a main branch are selected. From saddles in this main branch, further branches can be identified recursively until the entire contour tree is decomposed. In case of a single contour tree, the leaf with minimum isovalue is chosen as root, and the main branch is chosen as the monotone increasing path with maximal persistence starting in this root. These properties may vary across multiple contour trees, and thus the choice of root and main branch fundamentally affects the tree layout.

The alignment we compute above provides a dedicated root node. This root is guaranteed to exist in all individual contour trees and ensures the path property of the alignment (see Section 3.2 and Figure 3). In the process of branch decomposition, this root might turn out as the maximum of the main branch instead of the minimum in the individual contour trees. In this case, the minimum of the main branch is considered as root.

Starting in the chosen root node, a main branch is chosen by considering both, alignment and individual contour trees, as follows: All possible paths in the alignment from the root node to each leaf are initially considered as candidates for the main branch. Note that paths that are monotone in one or more individual trees are not necessarily monotone in the alignment, due to insertion of nodes and averaging of labels (isovalues). Separately for increasing and decreasing directions, each candidate path in the alignment is

then considered in each individual tree, and counted if it is monotone, giving its *path frequency* F . This frequency is then used as a rating R for candidate paths. $R := F\%$ is the percentage of aligned trees that contain the considered path. Here, other ratings could be employed; see Section 4.4 for further details.

Choosing the path with the highest rating R as the main branch and proceeding recursively for each sub-branch (i.e. saddle) of the main branch yields a branch decomposition for the alignment. A corner case occurs if no contour tree contains a path from the currently considered saddle to any leaf. The frequency of the branch is then considered zero, and the rating is based on the path persistence in the alignment.

4.2. Layout Algorithm

After a branch decomposition for the alignment is established, many known layout algorithms for contour trees could be employed. To obtain a suitable layout for the fuzzy contour tree representing the combination of all individual contour trees, additional information from the individual trees need to be taken into account when optimizing layout clarity, e.g. by minimizing crossings. Currently, we incorporate the isovalues of nodes from individual contour trees into the layout, resulting in value ranges (as opposed to individual isovalues) for leaves and saddles. Further consideration of branches in the individual contour trees is part of future work.

As a basic layout strategy, we adapt the (partly randomized) method proposed by Heine et al. in their permutation phase [HSCS11]: we attempt to find an ordering of branch groups that minimizes a weighted number of edge crossings. Instead of branch persistence, we weight crossing by the rating R obtained during branch decomposition. Thus, branches that have been chosen as main branches for the entire alignment or sub-trees are less likely to be crossed in the resulting ordering. The optimum ordering is sought as proposed by Heine et al. using a combination of random walk and simulated annealing. While this approach does not ensure an optimal layout, it gives very good results in practice (cf. Section 5). Furthermore, the non-deterministic nature of the algorithm may yield different layouts given identical input; we inherit this property from Heine's algorithm. Further discussion is given in Section 4.4.

In our setting, all branches are considered as individual branch groups. This is a natural choice, since the decomposition of the alignment into branch groups, taking multiple isovalues per node into account, tends to result in small branch groups, often containing only a single branch. The resulting order of branches is translated directly into horizontal coordinates for the layout, such that each branch occupies one vertical slice of the overall layout.

Grouped Layout. The horizontal coordinates obtained in the alignment layout can be propagated to the individual contour trees via the node matching from the alignment. Thus, across all contour trees, matched nodes are assigned identical horizontal positions. "Superimposing" all individual contour trees with the assigned mutual layout results the *grouped layout* (Figure 1(b)).

While this layout presents a significant improvement over separate layout of individual trees with superimposition (shown in Figure 1(a)), visual clutter is still an issue and can be disruptive.

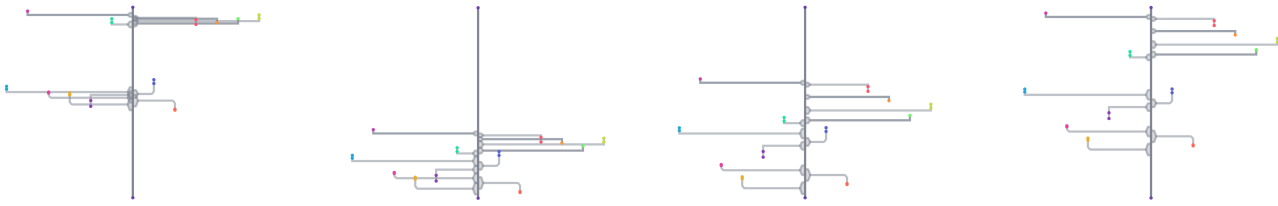


Figure 6: Branch spacing optimization (left to right): bundled fuzzy contour tree; branches stacked at the bottom; bounding space inserted; fitting to previous distribution.

Bundled Layout. To reduce visual clutter, we further abstract the grouped layout through edge bundling. On the basis of the grouped layout, the *bundled layout* (Figure 1(c)) bundles all edges of a branch group and assigns an opacity to edges and nodes based on the rating R of the branch. Branch edges are bundled close to their origin to the mean vertical position of the group's saddles. The distance to the origin consists of *plateau width* and a *bundle width*. Both are illustrated in Figure 5 and can be customized.

To further simplify the representation, we only draw the edges of a branch group originating at the respective maximum and minimum saddle values. The inbetween area is filled with appropriate opacity to accentuate its affiliation. If a child branch must be connected at an isovalue that is not spanned by the bundled edge, the child branch is extended to the plateau. If this is not possible, i.e. if its parent and its plateau are not on the same side of the child branch, the child branch's horizontal position is shifted accordingly. In the future this can be avoided by incorporating further knowledge on individual contour trees in the alignment layout algorithm. A plateau size that is large enough to make a clear distinction between a connection to the parent branch and a connection to the plateau ensures that the parent-child relation remains clear even in this special case. Again, Figure 5 provides an example.

Optimized Vertical Branch Spacing. In many cases, ensemble members will have a similar topological structure, resulting in a strong resemblance of their contour trees after alignment. This may result in clustered branch origins in the fuzzy contour tree. To disambiguate in these cases, we propose to shift branches vertically to better leverage available vertical space. Although vertical node position no longer indicates scalar value, we preserve the vertical ordering of branches. Furthermore, given sufficient vertical space, the vertical distribution of branches on each parent branch adheres to as much as possible, and the saddle isovalue ranges of two branches left and right of the parent branch overlap only if they do so in the original tree; they are never forced to overlap by our algorithm.

The shifting procedure is performed across all sub-trees in a bottom-up manner, beginning with the branches farthest from the chosen main branch. Available space on a sub-tree's main branch is filled in three steps, with different types of spaces considered in each step (cf. Figure 6 for an illustration):

- Step 1:** All saddles are stacked in correct order without space in between. Overlaps of saddles left and right of the main branch are reflected. The occupied vertical space is marked as "saddle".
- Step 2:** Based on the bounding box of the sub-tree's main branch,

"bounding" spaces are added above and below every "saddle" space, if the current space is smaller than the bounding box.

- Step 3:** The original space above and below every child branch on the sub-tree's main branch is compared to the current spacing. Space is added to obtain a distribution of the child branches similar to the original layout.

After each step, the amount of occupied vertical space relative to the available height is checked. If it exceeds available height, the spaces added in the previous step are "compressed" by scaling all vertical heights down such that the maximum available height is not exceeded; all further steps are omitted. If this occurs after the first step, this means that an overlap of the isovalue ranges cannot be avoided. After step 2, it implies the possibility of overlaps between main branches of sub-trees. This shifting can be applied to the grouped layout and the bundled layout alike and significantly disambiguates overlapping structures and reduces clutter. Figure 1(d) provides an example.

4.3. Basic Interaction

In the following, we consider basic interaction modalities enabled by the fuzzy contour tree. The resulting interactive visualizations (including the layout algorithm discussed above) are implemented in a lightweight JavaScript prototype based on the d3.js library [BOH11]. An overview of the user interface is given in Figure 7. For the datasets we consider in Section 5, all algorithms run quickly enough to enable fully fluid interaction.

We implement **branch highlighting** (Figure 8a): hovering a branch in the fuzzy contour tree highlights this branch and all its bundled edges and ancestors while all other branches are grayed out. The selection can be fixed and released by clicking. For a fixed selection, no ancestors are highlighted (cf. Figure 1).

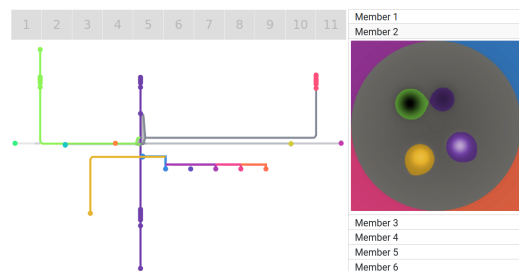


Figure 7: An overview of the fuzzy contour tree user interface.

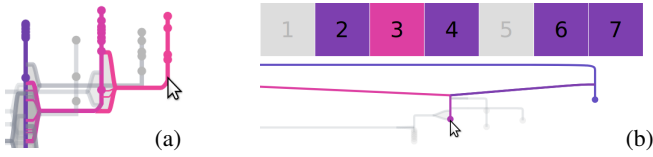


Figure 8: (a) Branch highlighting. The selected branch and all ancestors are highlighted with all connected saddles. (b) Member highlighting. Each member containing an edge in the selected bundle is highlighted in the associated color.

At the top of the UI, a grid providing information on individual contour trees is linked to the fuzzy contour tree. Figure 9 shows **tree highlighting**: selecting the index of an individual contour tree in the grid highlights this particular contour tree. To clarify membership of each branch, highlighting a branch in the fuzzy contour tree also triggers **member highlighting** in the grid (See Figure 8b). All members that contain one of the contained edges are highlighted in the corresponding edge color.

It is furthermore sensible to link the fuzzy contour tree to a spatial representation of each of the individual analyzed scalar fields. We implement this **component highlighting** for the 2D case, as shown in Figure 12. While we do not implement similar functionality for 3D datasets, volume rendering or isosurface visualization can be applied in these cases.

4.4. Layout Parameters

Several parameters influence the layout process. The temperature function of the simulated annealing and its parameters are adopted from the layout algorithm by Heine et al. [HSCS11] and are discussed there. Additional parameters in our approach are the branch rating R , used to obtain the alignment branch decomposition, and the cost function used to weight crossings during the simulated annealing. The weights for this cost function are here chosen as the rating R of the crossing branches. Other ratings may be chosen to customize the layout.

For example, it appears natural to consider persistence of branches in the alignment as rating. However, since node values in the alignment differ from those in the individual contour trees, this persistence cannot be considered an intuitive stability measure, making its impact difficult to interpret. Thus, we do not consider

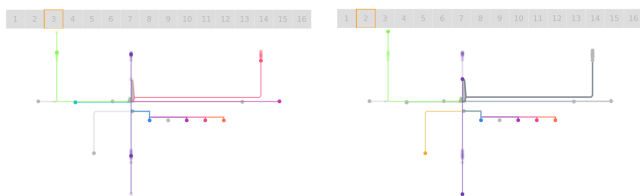


Figure 9: Tree highlighting applied to two contour trees of the analytical ensemble. (Left) A typical member is highlighted. Its branches follow branches with maximal frequency. (Right) The dissimilarity of the structure of the outlier is clearly visible.

persistence further here, and we reserve a more detailed consideration of ratings for future work.

4.5. Challenges

Several specific challenges arise when visualizing a fuzzy contour tree that are not present in the visualization of individual contour trees. Among special cases for the layout treating leaf and saddle positioning, the problem of multiple saddles appearing as the origin of a single branch is the most prominent one. In this case, the opacity of branches is determined by the path occurring in the largest number of individual contour trees, and for each origin minimum and maximum branch are visible. Furthermore, every origin is assigned an individual color for connecting edges. Hence, the existence of multiple origins is emphasized when the affected edge is highlighted, and also in member highlighting, cf. Figure 8b for an example. When highlighting individual trees, only the edge to the origin occurring in the tree is highlighted. In addition, several layout questions on the bundle positions need to be considered since having multiple origins provides multiple choices for the start position of bundled edges.

5. Results

In this section, we apply our algorithm to the specific problem of visualizing the contour trees of scalar field ensembles. Using our approach, this is easily accomplished: from each ensemble member, the contour tree is extracted. Optional preprocessing of the data such as noise removal or contour tree simplification can be applied as desired to individual contour trees. The alignment is computed and the layout strategies and interactions discussed in the previous section are applied.

We illustrate our technique and its properties on several analytical and real-world examples in both 2D and 3D. Running times for a sequential implementation of the alignment algorithm, formulated as a C++ TTK [TFL*18] filter, and general dataset properties, are given in Table 1. All times were obtained on a standard workstation with an Intel Core i7-7700 and 16GB of RAM. Contour trees were computed and, if sensible, persistence-simplified using TTK.

Tasks. The fuzzy contour tree as combination of multiple individual contour trees in one visualization is aimed at analyzing and understanding scalar ensembles. In this context, we aim to support the following domain-agnostic elementary analysis tasks:

Dataset	Size	n	$ V_{\max} $	$ \mathcal{A} $	$t_{\text{align}} [s]$
analytical	128×128	16	20	28	0.06
cylinder 2D	128×256	23	32	62	0.15
cylinder 3D	$64^2 \times 128$	10	60	144	1.10
viscous fingers	$64^2 \times 45$	15	48	128	0.83

Table 1: Properties and runtimes of the example data sets. n is number of ensemble members in each case, while $|V_{\max}|$ and $|\mathcal{A}|$ denote maximal contour tree size (after simplification) and alignment size, respectively. t_{align} denotes the alignment computation time.

T1 - Compare : Identify similarities and differences in topological structure across the ensemble. Example: *Which scalar values induce topological changes in contours in some or all of the ensemble members?*

T2 - Combine : Identify ensemble members with common topological segmentation or threshold values. Example: *Which contours represent the ensemble well? Which critical points occur in all members (common topological denominator [GST14])?*

T3 - Separate : Separate groups of ensemble members with similar behavior and identify outliers. Example: *Which ensemble members contain a specific branch?*

We illustrate the concretization of these tasks for fluid flow applications in the examples described below, where topological analysis is used to identify features of interest.

5.1. Analytical Example

To demonstrate the usefulness of fuzzy contour trees and give a straightforward example that illustrates both alignment and layout, we devise an analytical two-dimensional data set with simple structure: Each of 16 ensemble members contains a small local maximum in the center and 4 local extrema of varying height around the center peak. In 15 members these extrema are three maxima and one minimum, in one further (outlier) member there are two minima and two maxima. Figure 10 shows the fuzzy contour tree for this data set, computed using the persistence metric from Section 3.

Comparing the different ensemble members using the fuzzy contour tree (**T1**) is straightforward: Using the bundled layout, the branches with high frequency are easily determined as those with highest opacity. Also, the existence of three maxima and one minimum in most ensemble members is clearly apparent, as are the isovalues inducing topological changes.

Identifying the members that share the structure with three maxima and one minimum (**T2**) can be achieved using member highlighting. Topological structures contained in every member of the ensemble are given by the branches that are highlighted in both, Figure 9 (a) and (b). As expected, two maxima and one minimum are part of the common structure of the whole ensemble, as well as four small, linked branches at the vertical center of the tree, representing the four corners of the domain, as can be seen using component highlighting (Figure 11). Note that the small maximum in the center of each ensemble member is not visible as a common structure in this case, but as multiple (nearly) horizontal branches. This results from the super-tree property and a chosen metric that favors high persistence (cf. Section 3.1 and Figure 2).

Outlier identification (**T3**) can be accomplished using member highlighting on the single minimum with low frequency on the left of the tree. Tree highlighting then provides all information on the topological structure of the outlier (cf. Figure 9).

5.2. Heated Cylinder

The heated cylinder ensemble describes the flow around a heated pole in both 2D and 3D domains. The ensemble was obtained by simulating the corresponding model with stochastically perturbed

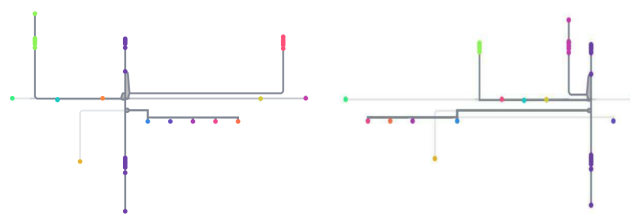


Figure 10: Fuzzy contour tree of the analytical example, aligned and laid out using different randomizations (left and right).

initial and boundary conditions for velocity. Fluid initially at rest is heated around the pole, begins to rise, and forms a plume. Scalar values describe flow vorticity, and topological segmentation identifies vortices as the attracting basins of maxima. The contour trees for both data sets were simplified using persistence, with the same threshold for all members.

2D ensemble. Using the combined cost metric (cf. Section 3) for the alignment of the ensemble contour trees results in a highly intuitive matching, as can be verified in the component view. Several highlighted components across different members are shown in Figure 12. For example, the global maximum at the center (represented by the long orange branch in the fuzzy contour tree) is matched in all members, as indicated by the small variance of the matched nodes. Fuzzy contour tree visualization using different layout options is shown in Figure 1. Clearly, bundled layouts are visually most intuitive and exhibit the least amount of clutter.

3D ensemble. In the 3D case, matching according to the same combined cost metric provides good results for the alignment as well. Common topological structures present in all ensemble members are clearly identifiable in the fuzzy contour tree visualization given in Figure 13. An intuition on what contours have been matched can be built by exploring the 3D data set and matched components in an appropriate viewer.

Approaching topological tasks using the fuzzy contour tree. An example on how to compare ensemble members (**T1**) in the 3D case is given in Figure 12. Here, all branches with a high frequency are marked. They can be determined by their opacity and using member highlighting. Without considering frequency, common topological structures can be determined: at the bottom, only minima exist, then a layer of maxima occurs, followed by another area with

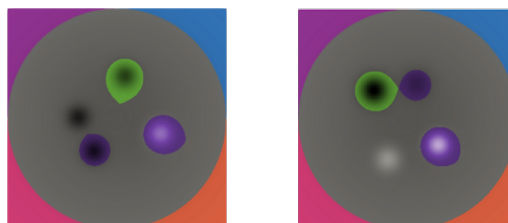


Figure 11: Component highlighting: Selected branches shared by the members in Figure 9 are highlighted in the ensemble members.

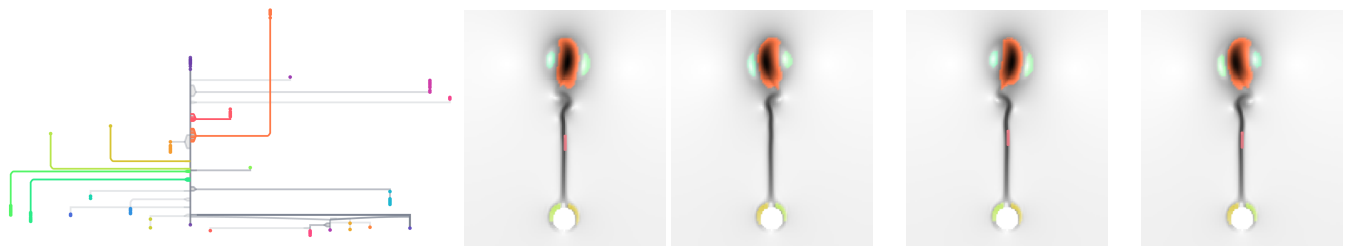


Figure 12: Component highlighting shows alignment quality: Exemplarily marked components (vortices) in the fuzzy contour tree (bundled layout, optimized branch spacing) of the 2D heated cylinder ensemble and representative examples of possible matching behavior.

mainly maxima; these extrema indicate vortices of different rotational direction. Whether this structure is present in all members can be checked using tree highlighting, providing a common segmentation for the relevant members (**T2**). The single blue minimum (vortex) between the two layers of maxima distinguishes one ensemble member from all others. It can be determined easily using member highlighting for this branch (**T3**).

5.3. Viscous Fingering

To illustrate the behavior of our method in a setting where searching for topological similarities in the member's level sets is not meaningful, we consider the fuzzy contour tree for 15 members of the viscous fingering ensemble [Sci]. To derive three-dimensional piecewise linear scalar fields from the given point clouds we follow the approach by Lukaszczuk et al. [LAS* 17]. The contour trees are simplified by persistence.

As expected, the alignment algorithm computes the alignment of the contour trees without problems, but as Figure 14 illustrates, the resulting fuzzy contour tree is complex. The large variance in the scalar values of critical points of the branch groups indicates that the matching is not semantically meaningful. Hence, non-meaningful alignments are easily identified.

6. Discussion

As shown in the previous section, fuzzy contour trees are useful to visualize topological structures across ensembles. Fundamentally, tree alignment, i.e. the matching of individual contour tree nodes



Figure 13: Common topological structures of the 3D heated cylinder ensemble are clearly shown in the fuzzy contour tree. All branches occurring in at least 8 out of 10 members are highlighted.

and arcs into a super-tree, enables the joint layout of all contour trees as a fuzzy contour tree, but also imposes some limits w.r.t. possible applications: often, overlap measures are used to map features defined by the contour tree segmentation onto each other. In contrast, our method is independent of position and area in the field of matched arcs and nodes. If the same major features are shared among multiple fields in a similar topological structure (regarding relative positioning and connectivity in the contour tree) but are scattered differently over the domain, our approach is still able to find and match them. Naturally, this is only possible as long as the overall topological structure provides a sufficient amount of similarity for a meaningful matching.

If the structure of the different contour trees shows only small or no topological similarity, as discussed in the viscous fingers example above, a minimal alignment will exist (and is computed by our algorithm), but the resulting fuzzy contour tree will not yield a meaningful visualization. As explained in Section 3 and Figure 2, alignment matching can be limited by the positioning of semantically similar features in the individual trees. If a semantically meaningful match is not possible, there are two alignment options: either the features are not aligned, resulting in two separate branches (desired behaviour), or a semantically not meaningful matching to a different part of the tree is achieved (see Section 5.3).

Finally, our method cannot automatically identify semantically meaningless alignments automatically. While it would appear intuitive to consider the alignment cost as a criterion and declare alignment failed if the cost is too high, this cost is a heuristic that does not allow an absolute comparison that is easy to generalize across a large range of datasets. However, identifying alignments containing matchings of unrelated topological components can be achieved by a user when comparing matched segments via component highlighting. Indications that an alignment is not meaningful are apparent in the fuzzy contour tree: large differences in the vertical coordinates of saddles and leaves belonging to a branch mark potential matchings of originally unrelated branches in the individual contour trees.

Comparison to Similar Techniques. Here, we discuss similarities and differences of the fuzzy contour tree with similar approaches.

As merge trees can be seen as a special case of contour trees with a fixed root, fuzzy merge trees could be generated with a significant speedup compared to fuzzy contour trees since testing different roots to generate the alignment is unnecessary.

Compared to the combined visualization of the fuzzy contour tree, displaying multiple contour or merge trees side-by-side provides much less support for the tasks defined in Section 5. Independent visualization of individual contour trees results in different layout and scaling; thus, a sensible comparison of the contour trees, especially of value ranges, is not feasible. Even if identical layout and scaling could be obtained, there are strong limits on the visual scalability of a side-by-side approach, and manual or “visual” matching of subtrees has to be performed by a viewer, making the approach non-practical overall.

Favelier et al. [FFST19] cluster ensemble members based on an embedding of their persistence maps in Euclidean space. Using the notion of mandatory critical points, confidence regions for each cluster are calculated and visualized. Athawale et al. process a given 2D Morse complex ensemble to obtain a probabilistic map and a survival map, called *summary maps* for 2D Morse complex ensembles [AMJ*19]. The probabilistic map shows the probabilistic classification of all points in the plane based on the mandatory maximum their integral curve ascends to over all ensemble members, while the survival map traces the behavior of gradient flows under persistence simplification, where unchanged gradient flow direction after a simplification step is counted as a survived step.

Both techniques are suited for ensembles of arbitrary size, but do not consider or present single or combined contour trees. Furthermore, a fuzzy contour tree incorporates information from individual contour trees into a single overall visualization, and links this combined visualization back to the individual contour trees; this possibility fundamentally enables **T3** and is not available in either summary maps or the persistence atlas. While the persistence atlas provides clustering of the ensemble members (**T2**), users are not provided sufficient information on individual members to identify those with common segmentations, unless they are part of the ensemble’s *common topological denominator*.

A further limitation of the two approaches in comparison to our approach is the fixed comparison metric. While the persistence atlas relies on trend and location of critical points, and the approach by Athawale et al. is based on the gradient field, our approach *can* incorporate these parameters when matching nodes in the alignment, but it also can be based only on the topological structure or any other parameters. This flexibility makes fuzzy contour trees highly adaptable to domain-specific needs.

7. Conclusions and Future Work

In this paper, we combine tree alignments with a novel layout algorithm to achieve simultaneous depiction of multiple contour trees as a fuzzy contour tree in a semantically meaningful manner with minimal clutter. We illustrate the usefulness of these visualizations for ensemble analysis, and demonstrate interaction enabled by the fuzzy contour tree on several examples.

We identify several opportunities for future work:

- We will investigate an algorithm to compute minimal tree alignments deterministically. While the current algorithm works well in practice, a deterministic algorithm would be preferential.
- An automated manner to identify non-meaningful alignments

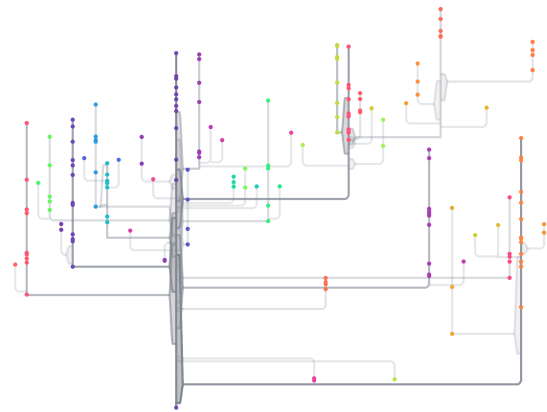


Figure 14: Large differences between saddles and leaves in a single branch indicate the matching of semantically unrelated branches for the viscous fingers ensemble.

appears as an important aspect towards real-world use of our technique, and we will consider several options in future work.

- Identifying suitable cost metrics for a more general set of tasks and application domains would benefit the applicability of our approach. For example, an appropriately chosen, location-dependent metric could shift the focus on variation among ensemble members rather than commonalities.
- Further optimizations of the layout, as well as a systematic investigation of visual analysis on top of fuzzy contour trees, appear fruitful. Furthermore, we will investigate how other scenarios involving several contour trees can benefit from our approach.
- Finally, we will investigate the combination of our approach with image databases [AJO*14] for in situ visualization.

While we obtain promising results, our work aims to provide an initial demonstration that the fuzzy contour tree is useful for topology-based visualization of scalar ensembles.

Acknowledgements

We would like to thank Frederike Gartzky and Noël Ströhmer-Lohfink for implementation support, and Heike Leitte for valuable discussion. This research was funded by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation) under contract 252408385 as part of IRTG 2057 Physical Modeling for Virtual Manufacturing. Open access funding enabled and organized by Projekt DEAL. [Correction added on 17 November 2020, after first online publication: Projekt Deal funding statement has been added.]

References

- [AJO*14] AHRENS J. P., JOURDAIN S., O’LEARY P., PATCHETT J., ROGERS D. H., PETERSEN M.: An image-based approach to extreme scale in situ visualization and analysis. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014* (2014), Damkroger T., Dongarra J. J., (Eds.), IEEE Computer Society, pp. 424–434. doi: 10.1109/SC.2014.40.11

- [AMJ*19] ATHAWALE T., MALJOVEC D., JOHNSON C. R., PASCUCCI V., WANG B.: Uncertainty visualization of 2d morse complex ensembles using statistical summary maps, 2019. [arXiv:1912.06341](https://arxiv.org/abs/1912.06341). 11
- [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](https://doi.org/10.2312/3dor.20161084). 3
- [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](https://doi.org/10.1145/2582112.2582169). 3
- [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](https://doi.org/10.1016/j.tcs.2004.12.030). 3, 4
- [BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D³ data-driven documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec 2011), 2301–2309. [doi:10.1109/TVCG.2011.185](https://doi.org/10.1109/TVCG.2011.185). 7
- [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 (05 2010), 248–60. [doi:10.1109/TVCG.2009.69](https://doi.org/10.1109/TVCG.2009.69). 2
- [BYM*14] BEKETAYEV K., YELIUSSIZOV D., MOROZOV D., WEBER G., HAMANN B.: *Measuring the Distance Between Merge Trees*. Springer, 01 2014, pp. 151–165. [doi:10.1007/978-3-319-04099-8_10](https://doi.org/10.1007/978-3-319-04099-8_10). 3
- [CD13] CARR H., DUKE D.: Joint contour nets. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (12 2013), 1100–1113. [doi:10.1109/TVCG.2013.269](https://doi.org/10.1109/TVCG.2013.269). 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](https://doi.org/10.4230/LIPICs.SoCG.2017.25). 3
- [CSA03] CARR H., SNOEYINK J., AXEN U.: Computing contour trees in all dimensions. *Computational Geometry* 24, 2 (2003), 75 – 94. Special Issue on the Fourth CGC Workshop on Computational Geometry. [doi:10.1016/S0925-7721\(02\)00093-7](https://doi.org/10.1016/S0925-7721(02)00093-7). 2
- [CSv04] CARR H., SNOEYINK J., VAN DE PANNE M.: Simplifying flexible isosurfaces using local geometric measures. In *IEEE Visualization 2004* (Oct 2004), pp. 497–504. [doi:10.1109/VISUAL.2004.96](https://doi.org/10.1109/VISUAL.2004.96). 2, 5
- [CWS*19] CARR H., WEBER G. H., SEWELL C., RÜBEL O., FASEL P., AHRENS J.: Scalable contour tree computation by data parallel peak pruning. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. [doi:10.1109/TVCG.2019.2948616](https://doi.org/10.1109/TVCG.2019.2948616). 2
- [FFST19] FAVELIER G., FARAJ N., SUMMA B., TIERNY J.: Persistence atlas for critical point variability in ensembles. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan 2019), 1152–1162. [doi:10.1109/TVCG.2018.2864432](https://doi.org/10.1109/TVCG.2018.2864432). 11
- [GST14] GÜNTHER D., SALMON J., TIERNY J.: Mandatory critical points of 2d uncertain scalar fields. *Computer Graphics Forum (Proc. EuroVis)* 33 (06 2014), 31–40. [doi:10.1111/cgf.12359](https://doi.org/10.1111/cgf.12359). 2, 9
- [HCng] HRISTOV P., CARR H.: W-structures in contour trees. In *Topological Methods in Visualization: Theory, Software and Applications* (forthcoming). 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](https://doi.org/10.1111/cgf.12933). 2
- [HN18] HU Y., NÖLLENBURG M.: *Graph Visualization*. Springer International Publishing, Cham, 2018, pp. 1–9. [doi:10.1007/978-3-319-63962-8_324-1](https://doi.org/10.1007/978-3-319-63962-8_324-1). 2
- [HO82] HOFFMANN C. M., O'DONNELL M. J.: Pattern matching in trees. *J. ACM* 29, 1 (1982), 68–95. [doi:10.1145/322290.322295](https://doi.org/10.1145/322290.322295). 3
- [HSCS11] HEINE C., SCHNEIDER D., CARR H., SCHEUERMANN G.: Drawing contour trees in the plane. *IEEE Transactions on Visualization and Computer Graphics* 17, 11 (Nov 2011), 1599–1611. [doi:10.1109/TVCG.2010.270](https://doi.org/10.1109/TVCG.2010.270). 2, 6, 8
- [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), pp. 75–86. [doi:10.1007/3-540-58094-8_7](https://doi.org/10.1007/3-540-58094-8_7). 3
- [Kra10] KRAUS M.: Visualization of uncertain contour trees. In *IMA-GAPP/IVAPP* (01 2010), pp. 132–139. 2
- [KRS03] KETTNER L., ROSSIGNAC J., SNOEYINK J.: The safari interface for visualizing time-dependent volume data using iso-surfaces and contour spectra. *Computational Geometry - Theory and Applications*, v.25, 97-116 (2003) 25 (05 2003). [doi:10.1016/S0925-7721\(02\)00132-3](https://doi.org/10.1016/S0925-7721(02)00132-3). 2
- [KTSK00] KLEIN P. N., TIRTHAPURA S., SHARVIT D., KIMIA B. B.: A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA* (2000), pp. 696–704. 3
- [LAS*17] LUKASCZYK J., ALDRICH G., STEPTOE M., FAVELIER G., GUEUNET C., TIERNY J., MACIEJEWSKI R., HAMANN B., LEITTE H.: Viscous Fingering: A Topological Visual Analytic Approach. In *Applied Mechanics and Materials* (2017), vol. 869, Trans Tech Publ, pp. 9–19. [doi:10.4028/www.scientific.net/AMM.869.9](https://doi.org/10.4028/www.scientific.net/AMM.869.9). 10
- [LRCP07] LEE B., ROBERTSON G., CZERWINSKI M., PARR C.: Canditree: Visualizing structural uncertainty in similar hierarchies. *Information Visualization* 6 (10 2007). [doi:10.1057/palgrave.ivs.9500157](https://doi.org/10.1057/palgrave.ivs.9500157). 2
- [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](https://doi.org/10.1111/cgf.13164). 2
- [MBW13] MOROZOV D., BEKETAYEV K., WEBER G.: Interleaving distance between merge trees. *Discrete and Computational Geometry* 49 (01 2013), 22–45. 3
- [PCMS09] PASCUCCI V., COLE-MCLAUGHLIN K., SCORZELLI G.: *The TOPORRERY: computation and presentation of multi-resolution topology*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 19–40. [doi:10.1007/b106657_2](https://doi.org/10.1007/b106657_2). 1
- [RSLng] 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 forthcoming). 3
- [SBKN18] SRIDHARAMURTHY R., BIN MASOOD T., KAMAKSHIDASAN A., NATARAJAN V.: Edit distance between merge trees. *IEEE Transactions on Visualization and Computer Graphics* (2018), 1–1. [doi:10.1109/TVCG.2018.2873612](https://doi.org/10.1109/TVCG.2018.2873612). 3
- [Sci] IEEE VIS scientific visualization contest 2016. <http://www.uni-kl.de/sciviscontest/>. Accessed: 2019-11-28. 10
- [SGKH15] SCHNORR A., GÖBBERT J. H., KUHNEN T. W., HENTSCHEL B.: Tracking space-filling structures in turbulent flows. In *5th IEEE Symposium on Large Data Analysis and Visualization, LDAV 2015, Chicago, IL, USA, October 25-26, 2015* (2015), pp. 143–144. [doi:10.1109/LDAV.2015.7348089](https://doi.org/10.1109/LDAV.2015.7348089). 2
- [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)* (April 2016), pp. 56–63. [doi:10.1109/PACIFICVIS.2016.7465251](https://doi.org/10.1109/PACIFICVIS.2016.7465251). 2
- [SMP15] SILVA V., MUNCH E., PATEL A.: Categorized Reeb graphs. *Discrete and Computational Geometry* 55 (01 2015). [doi:10.1007/s00454-016-9763-9](https://doi.org/10.1007/s00454-016-9763-9). 3

- [SNG*17] SCHULZ C., NOCAJ A., GOERTLER J., DEUSSEN O., BRANDES U., WEISKOPF D.: Probabilistic graph layout for uncertain network visualization. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (Jan 2017), 531–540. doi:10.1109/TVCG.2016.2598919. 2
- [SSW14] SAIKIA H., SEIDEL H., WEINKAUF T.: Extended branch decomposition graphs: Structural comparison of scalar data. *Computer Graphics Forum* 33, 3 (2014), 41–50. doi:10.1111/cgf.12360. 2, 3, 5
- [SWC*08] SCHNEIDER D., WIEBEL A., CARR H. A., HLAWITSCHKA M., SCHEUERMANN G.: Interactive comparison of scalar fields based on largest contours with applications to flow visualization. *IEEE Transactions on Visualization and Computer Graphics* 14, 6 (2008), 1475–1482. doi:10.1109/TVCG.2008.143. 2
- [SZ90] SHAPIRO B. A., ZHANG K.: Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in the Biosciences* 6, 4 (1990), 309–318. doi:10.1093/bioinformatics/6.4.309. 3
- [Tai79] TAI K.: The tree-to-tree correction problem. *J. ACM* 26, 3 (1979), 422–433. doi:10.1145/322139.322143. 3
- [TFL*18] TIERNY J., FAVELIER G., LEVINE J. A., GUEUNET C., MICHAUX M.: The topology toolkit. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (Jan 2018), 832–842. doi:10.1109/TVCG.2017.2743938. 8
- [TN11] THOMAS D. M., NATARAJAN V.: Symmetry in scalar field topology. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec 2011), 2035–2044. doi:10.1109/TVCG.2011.236. 2
- [WDC*07a] WEBER G. H., DILLARD S. E., CARR H., PASCUCCI V., HAMANN B.: Topology-controlled volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 13, 2 (March 2007), 330–341. doi:10.1109/TVCG.2007.47. 1
- [WDC*07b] WEBER G. H., DILLARD S. E., CARR H., PASCUCCI V., HAMANN B.: Topology-controlled volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 13, 2 (March 2007), 330–341. doi:10.1109/TVCG.2007.47. 2
- [WHL19] WANG J., HAZARIKA S., LI C., SHEN H.: Visualization and visual analysis of ensemble data: A survey. *IEEE Transactions on Visualization and Computer Graphics* 25, 9 (Sep. 2019), 2853–2872. doi:10.1109/TVCG.2018.2853721. 2
- [WJ94] WANG L., JIANG T.: On the complexity of multiple sequence alignment. *Journal of Computational Biology* 1, 4 (1994), 337–348. doi:10.1089/cmb.1994.1.337. 4
- [wZ13] WU K., ZHANG S.: A contour tree based visualization for exploring data with uncertainty. *International Journal for Uncertainty Quantification* 3 (01 2013), 203–223. doi:10.1615/Int.J.UncertaintyQuantification.2012003956. 2
- [YWM*19] YAN L., WANG Y., MUNCH E., GASPAROVIC E., WANG B.: A structural average of labeled merge trees for uncertainty visualization. *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. doi:10.1109/TVCG.2019.2934242. 3
- [Zha96] ZHANG K.: A constrained edit distance between unordered labeled trees. *Algorithmica* 15, 3 (1996), 205–222. doi:10.1007/BF01975866. 3