



Mergemaps: Treemaps for Scientific Data

Adhitya Kamakshidasan, Vijay Natarajan

► To cite this version:

Adhitya Kamakshidasan, Vijay Natarajan. Mergemaps: Treemaps for Scientific Data. 2020. hal-02458816v2

HAL Id: hal-02458816

<https://inria.hal.science/hal-02458816v2>

Preprint submitted on 7 Feb 2020 (v2), last revised 9 Aug 2020 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Mergemaps: Treemaps for Scientific Data

Adhitya Kamakshidasan and Vijay Natarajan

Abstract Topology driven methods for analysis of scalar fields often begin with an exploration of an abstract topological structure such as the merge tree. Such abstractions are hard to interpret and time-consuming, particularly for feature-rich data. Current visualization schemes often place less emphasis on enriching user experience, human perception, or interaction. In this work, we aim to bridge that gap by utilizing treemaps towards effective topological analysis. We present *mergemaps*, a treemap based interactive design, to better understand merge trees. To aid the perceptual understanding of large merge trees, we provide fusing and diffusing operations to reduce its hierarchical size while preserving topological features. We show multiple examples where our design lead to easy interpretations.

1 Introduction

Topological methods for data analysis have proven to be useful in multiple contexts ranging from exploring cosmic filaments [25] to extracting voids in proteins [27]. Contour trees, Reeb graphs, Morse-Smale complexes, persistence diagrams [5, 9, 24, 11, 13], to name a few, provide abstract representations that aid in topological analysis of scalar fields. Despite these abstractions being extremely powerful, they are still yet to gain widespread popularity because their interpretation requires background in algebraic topology and Morse theory [10]. To this extent, multiple attempts have been made to provide user interfaces that convey topological information in an intuitive manner. In this work, we aim at improving the user experience of exploring a particular topological structure called the merge tree.

Adhitya Kamakshidasan
Inria, Saclay e-mail: adhitya.kamakshidasan@inria.fr

Vijay Natarajan
Indian Institute of Science, Bangalore e-mail: vijayn@iisc.ac.in

1.1 Related Work

A *merge tree* traces the connectivity evolution of sub/super-level sets in a scalar field, a *contour tree* contains the combined information of both sub- and super-level sets. From a data analysis perspective, merge trees have been used in various interesting applications. Bock et al. [3] used it for extracting fishes from micro-CT scans, Valsangkar et al. [32] track cyclones by understanding how their contours join and split, Doraiswamy et al. [8] identify congestion in New York roads.

Weber et al. [34] introduced the notion of topological landscapes, which uses a terrain metaphor for presenting the contour tree. Topological landscapes capture the nesting behavior of contours, with the intuition that humans better perceive topographic information. Their two-step algorithm involves placing branches by adaptively subdividing a mesh, and rebalancing the mesh to improve space utilization. While the resulting landscape allows users to grasp high-level features, the number of triangles increases sharply for deep hierarchies and the terrain can contain large empty spaces.

Building on the same paradigm, Harvey and Wang [12], observed the connection between topological landscapes and treemaps. Their proposed method computes a landscape corresponding to each edge interpreted as the root of the tree, and chooses the best landscape by defining a metric distance between each of them. While this method is not computationally expensive, it can result in skinny boundaries. Bekatayev et al. [1] proposed a solution that preserves the geometry proximity while constructing the topological landscapes by routing edges across a Voronoi diagram, but works only for small-sized trees. Demir et al. [7] layout branches as square boxes and render landscapes with a first-fit box packing scheme, followed by hierarchically triangulating the corresponding grid. This algorithm is an improvement over the original algorithm by Weber et al., but the resulting landscape looks artificial and lacks a spatial context. Topological landscapes have also been used to study higher dimensional point clouds [18]. Other visual representations based on tree drawing or 1D profiles [14, 19] have also been proposed for contour trees and merge trees. In contrast, the nesting behavior is better perceived in a treemap based representation. In addition, the treemap based approach also supports various visual analysis tasks.

Existing techniques that present the contour tree using a landscape metaphor are affected by the limits of human perception and interaction. While a contour tree is well suited to be represented as a terrain, the interactive study of terrains is perceptually difficult. In particular, we believe that the simultaneous exploration of sub-level and super-level sets is difficult using topological landscapes. Therefore, instead of directly representing the contour tree, we study the merge tree, which captures either sub- or super-level set connectivity. Focusing on the merge tree enables the exploration of simple representations that utilize and highlight its hierarchical properties.

1.2 Summary of results

We present *mergemaps*, a treemap based visual presentation of a merge tree¹. We construct a mergemap, by processing the branch tree representation of the merge tree to compute an intermediate proxy called the *aggregate tree*, that stores the hierarchy of persistence pairs. We believe that a mergemap simplifies the process of comprehending and interacting with the merge tree. We build upon existing work on treemaps and landscape metaphors and extend it to visualize merge trees. We also provide fusing and diffusing operations to reduce hierarchical clutter and hence improve user experience. We demonstrate the utility of our designs for understanding scientific datasets. Our designs are simple to implement and we hope that they would be adopted by the community.

2 Background

2.1 Merge Tree

Consider a scalar function $f: D \rightarrow \mathbb{R}$ defined on a simply connected manifold domain D . A value c in the range of f is called an *isovalue*. Given an isovalue, an *isocontour* or *level set* is defined as the collection of all points $x \in D$ such that $f(x) = c$. A merge tree captures the connectivity of sub-level sets $f^{-1}(-\infty, c]$ (*join tree*) or super-level sets $f^{-1}[c, \infty)$ (*split tree*) of f . Both join and split trees are referred to as merge trees. For the sake of convenience, we use only the split tree for explanation. Figure 1 shows the merge trees and contour tree of the height function defined on a simply connected domain.

A split tree is constructed by sweeping the domain in decreasing order of function value. It records the points at which the number of connected components of the super-level set changes. Nodes of a split tree consist of maxima, split saddles, and the global minimum. In practice, a split tree can be conceptualized as a rooted binary tree, in which every interior node has at least two children. The root is defined by the global minimum, the leaves are maxima, and the interior nodes are saddles. All maxima can be paired with saddles based on the notion of topological persistence, except for the global maximum which is paired with the global minimum. Each such pair represents a topological feature and its *persistence* can be defined as the absolute difference between the two scalar function values.

Similarly, a join tree can be defined using minima, join saddles and the global maximum. The *contour tree* contains the combined information present in a split and a join tree, it captures the evolution of level set connectivity. Collectively, minima, maxima, join saddles and split saddles are called critical points.

A merge tree can be decomposed into a set of branches, such that each branch contains a persistence pair. This generates a nested hierarchy of branches, wherein

¹ Video illustrating mergemaps at <https://youtu.be/xuj9jG4E3IM>

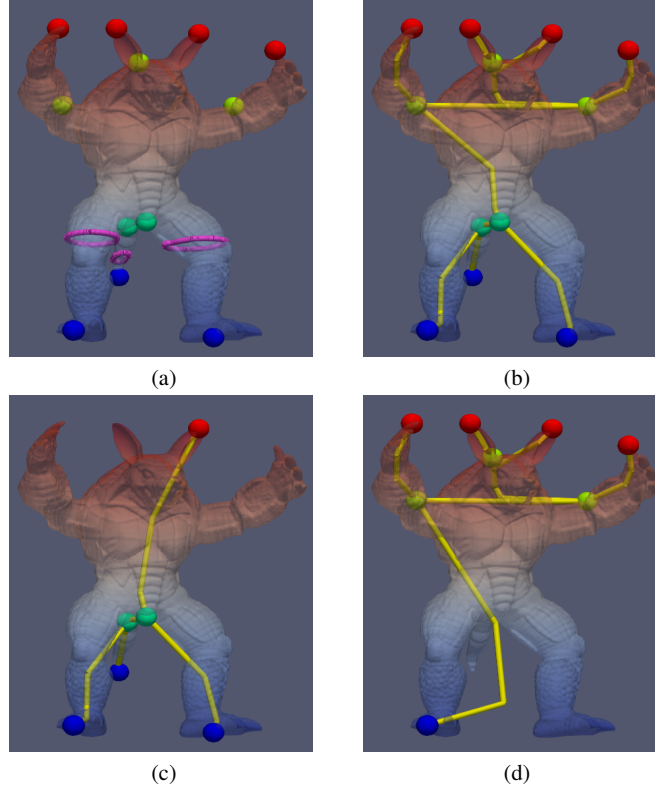



Fig. 1: Height field of Armadillo rendered using a blue-red colormap (). (a) critical points of the field and a level set, shown in pink. (b, c, d) show the contour tree, join tree, and split tree respectively.

each parent branch has a persistence greater than that of its children. This hierarchy of branches is called the *branch decomposition* representations of the merge tree.

2.2 Treemap

Trees are generally visualized as node-link diagrams. Such a visualization is inadequate while exploring large datasets, since navigating the structure is difficult and content information is often hidden within the individual nodes [33]. To counter this, Shneiderman [26] proposed treemaps as a technique to display tree structures using a two-dimensional nested space-filling approach similar to that of Venn diagrams. The original algorithm requires dividing the display into nested rectangles, each with an area corresponding to the weight of the associated node.

The utility of a treemap can be understood with an example, see Figure 2. The file system hierarchy of a computer can be comprehended using a treemap. In this case, the tree to be visualized consists of files and folders. The leaves of the tree represent the files. All interior nodes, including the root, represent the folders. Each file is represented as a box with an area corresponding to its file size. All folders are represented as containers with an area equivalent to the sum of the file sizes of its children. Every node of the input tree serves as a container for its children.

By definition, treemaps take an input of n weights, a hierarchy upon these weights (the tree), and a shape (generally a rectangle). Since these weights correspond to the leaves of the tree, the size of a parent container (present as an interior node of the tree) should be equal to the sum of the sizes of its children.

Several treemap layouts have been presented in the literature – for improved presentation of the tree hierarchy, better display of values associated with nodes, enhanced aspect ratio of rectangles, and multiple other criteria. We refer the reader to the survey by Schulz et al. [23] for a more elaborate discussion on these variants. In this paper, we choose an appropriate existing variant based on the requirement. For our case studies in Section 4, we use squarified treemaps [4], zoomable treemaps [2], spatial treemaps [37], and cascaded treemaps [16].

3 Mergemap

A *mergemap* is a treemap based visual representation of a merge tree. In this section, we will describe an algorithm for constructing a mergemap, show how to interact with it, and use its hierarchical properties to improve perception.

3.1 Motivation

Our main goal is to be able to represent a merge tree using a treemap and allow for better perceptual and interactive analysis. However, treemaps cannot directly be used out-of-the-box for representing merge trees. We describe the reason using an example. Figure 2 shows the directory structure as an input tree, visually depicted as a treemap. Each box in the treemap corresponds to a file, and each container corresponds to a folder. The area of a treemap box is proportional to the corresponding file’s size. Note that the folder does not require additional space, therefore the area of the corresponding containers is equal to the sum of the areas of its children. In contrast, the internal nodes of a merge tree (the saddles), always have an associated weight (scalar function value) and cannot be shown in a treemap. Therefore, a merge tree should first be converted into an intermediate structure that preserves the topological abstraction, and whose visual representation is amenable to interaction, perception, and analysis.

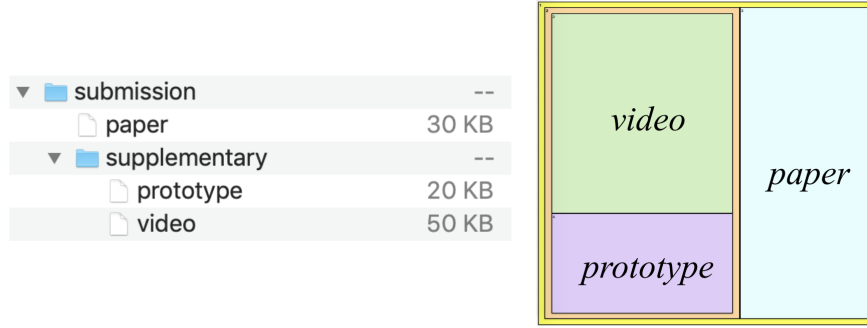
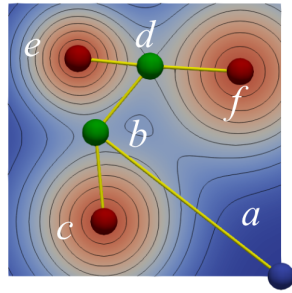
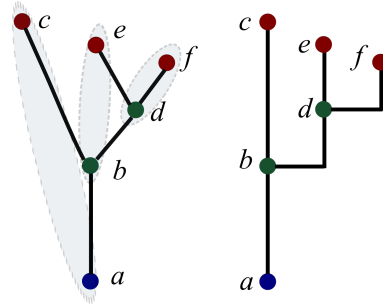


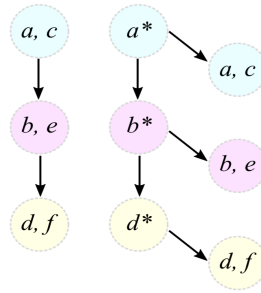
Fig. 2: An example file directory and its treemap visualization



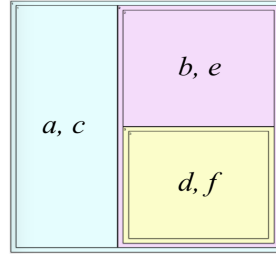
(a) 2D scalar field with Gaussians rendered using a blue-red colormap ()



(b) Merge tree and its branch decomposition



(c) Persistence hierarchy and aggregate Tree



(d) Mergemap

Fig. 3: Constructing a mergemap for a scalar field

3.2 Algorithm

There are three basic steps for generating a mergemap. First, we compute the branch decomposition. Second, using the branch decomposition, we construct an aggregate tree to introduce imposter nodes. Third, we visualize this aggregate tree using a treemap. Optionally, the hierarchy of the treemap is reduced using a sequence of fusing/diffusing operations and is spatially ordered. Figure 3 shows a mergemap and the output of the different steps.

We use the algorithm by Pascucci et al. [20], to convert a merge tree into a persistence based branch decomposition. The root branch is referred to as the trunk. The trunk has the global maximum and global minimum at each of its ends. Depending on which merge tree is used, all other branches have a minimum or maximum, and a saddle. Each branch can have an importance value like persistence, hypervolume or volume associated with it. For the purpose of our discussion, we assume that this branch decomposition has only one value associated with each branch, say persistence.

While the branch decomposition is often displayed as a collection of L-shaped branches, it can also be represented as a rooted tree, whose nodes represents individual branches. Such a representation is called a persistence hierarchy [21]. Understanding the branch decomposition in terms of persistence hierarchy, makes visualization easier.

The persistence hierarchy cannot be directly presented as a treemap. Each node of the persistence hierarchy has a value associated with it, whereas a treemap requires values to be associated only with leaf nodes. We construct a new tree, called the aggregate tree, whose leaves store values corresponding to all nodes of the persistence hierarchy tree. The aggregate tree can be presented as a treemap.

Every node in the persistence hierarchy is duplicated (without edges) and inserted as a child of the same node. The nodes are duplicated during a preorder traversal of the persistence hierarchy. After duplication, every non-leaf node is assigned a value equal to the sum of its children. This tree contains twice the number of nodes as the persistence hierarchy and is suitable for visualization using a treemap. Algorithm 1 shows this procedure.

Let us consider the example shown in Figure 3 to understand the algorithm. The input scalar field has six critical points, (a, b, c, d, e, f) , associated with the split tree. The branch decomposition consists of 3 branches: (a, c) , (b, e) , (d, f) . These pairs have a persistence value and an implicit hierarchy defined upon them. Since the pairs cannot be directly visualized using a treemap, we create an aggregate tree. For each node in the persistence hierarchy, an *imposter* is created, by duplicating all properties of the original node, except for parent-child relationships. This is then inserted back as a child of the original node. The imposter can be considered as a symbolically perturbed saddle with function value lower than its parent.

This aggregate tree satisfies some desirable properties:

- (i) the imposter will appear as a leaf node in the aggregate tree,

Algorithm 1 Create Aggregate Tree

```

1: function MAKEIMPOSTER(node)
2:   node  $\leftarrow$  DUPLICATE(node)
3:   for child  $\in$  node.children do
4:     if child not processed then
5:       imposter  $\leftarrow$  MAKEIMPOSTER(child)
6:       node.value  $\leftarrow$  node.value + imposter.value
7:   MARK(node, processed)
8:   return node

9: function DUPLICATE(node)
10:  imposter  $\leftarrow$  COPY(node)
11:  imposter.value  $\leftarrow$  node.value
12:  imposter.parent  $\leftarrow$  node
13:  DELETE(imposter.children)
14:  INSERT(node.children, imposter)
15:  MARK(imposter, processed)
16:  return node

17: AggregateTree  $\leftarrow$  MAKEIMPOSTER(node)

```

- (ii) the branch will be represented both as an internal node as well as a leaf node in the aggregate tree, and
- (iii) the internal node that represents the branch contains a copy that retains its original value.

When a node has imposters for itself and its initial children, it takes up a value equivalent to the sum of all its children. We call such a node as an *aggregate node* and denote it as $saddle^*$. In our example, d^* will be equal to the value of (d, f) , b^* will be equal to the sum of (b, e) and d^* , and a^* will be equal to the sum of (a, c) and b^* . Topologically, an aggregate node has a value equal to the total persistence of all branches beneath it.

Using these imposters and aggregate nodes, we can now show the branch decomposition using a treemap. The leaf nodes, (a, c) , (b, e) , (d, f) are shown as boxes, while their respective parents, a^* , b^* and d^* are shown as containers. Each branch has been represented twice, using a box and a container. The boxes show the exact value represented by the branch, while the containers express the hierarchical relationship amongst the branches.

3.3 Interaction

To explore a large dataset, a mergemap allows a user to navigate through multiple interesting features. Interaction with mergemaps is straightforward, and can be done in two ways: focus + context exploration, or using external widgets. In this section, we look at some best practices for interacting with mergemaps.

3.3.1 Focus + Context Exploration

A direct way to use a mergemap is by linking it with subvolumes of the domain [35]. A user can select individual boxes to interactively view the corresponding volumes. Meta-data associated with a branch is visible by hovering over its box in the mergemap and more specific information like branch identifiers and persistence values may be displayed as labels on top of a box. Even though containers and boxes represent the same branch, providing the same interactive capabilities is redundant. Containers can be used for size reduction operations (Section 3.4) and boxes for volume selection. A rectangle's area can also be selected for annotation using text or colour.

In order to focus on a single feature and its properties without getting distracted by the rest of the treemap, we use zoomable treemaps [2]. Zoomable treemaps provide capabilities that allow users to navigate up and down the hierarchy of the tree using animated rolling up and drilling down views. Zooming into a container causes a new treemap associated with its branches to be rendered into the original area. Zooming out causes the original context to be restored. A user may use this interaction capability to select the smallest features of the branch decomposition without the need for excessive simplification.

3.3.2 Linked Widgets

While interacting with a large branch decomposition, it can be cumbersome to repeatedly select features, one after another, in a hit-and-trial fashion. Hence, the designs presented in this paper may be used in conjunction with interaction widgets and tools. For instance, 'top/bottom k-persistent features' is a useful query to support. The traditional persistence diagram, where persistence pairs are plotted as vertical bars on the diagonal, is not best suited for selection. An alternative representation, called barcodes, shows the pairs as a sorted bar chart. This representation may be used in conjunction with mergemaps and standard brushing and linking by simply changing colors or shades of the corresponding boxes. Topological spines [6] is also an interesting candidate for a linked widget, since it preserves the underlying geometry of the scalar field.

3.4 Operations

User perception may be affected due to two different reasons. First, the mergemap shows a branch both as a container and a box, even when it has few or no children. This may result in clutter. Second, a branch decomposition often contains large nested hierarchies. To address this difficulty, we propose operations that reduce the size of the branch decomposition. We first describe these operations and then describe how they may be used together with mergemaps.

3.4.1 Reduction Operations

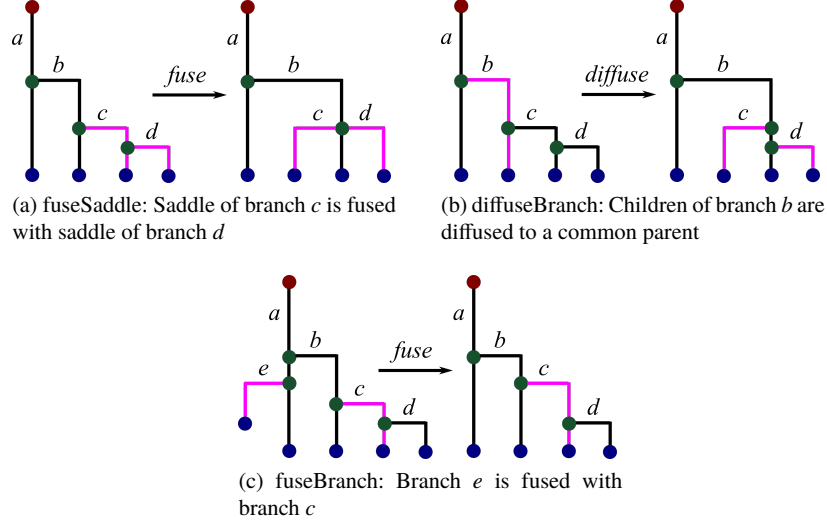


Fig. 4: Examples of reduction operations

We present three operations: *fuseSaddle*, *diffuseBranch*, and *fuseBranch*. These operations adapted from [34, 29], either fuse different branches into one or spreads branches or flatten a deep hierarchy. Figure 4 shows an illustration for each operation.

- (i) ***fuseSaddle***: If the critical value at the saddle of a branch is close to that of its parent, then both branches may be fused together so that they share the same saddle. This is a saddle stabilization operation.
- (ii) ***diffuseBranch***: If a branch has multiple nested children, the hierarchy below the branch may be flattened by spreading all nested descendant branches as immediate children. This is a hierarchy compression operation.
- (iii) ***fuseBranch***: If the minima of two branches in a join tree are in close proximity in the spatial domain, then both branches may be fused into a single branch. Children of both branches are hierarchically placed into the merged branch. The new branch is pushed up the hierarchy, its saddle value is set to the larger of the two saddles, and the value of the minimum is set to the smaller of the two minima. In a split tree, the saddle value is set to the smaller of the two saddles and the value of the maximum is set to the larger of the two maxima. This is a proximity-based simplification operation.

3.4.2 Reducing Mergemaps

The above operations can be performed in two ways, directed by uniform thresholds or by a human-in-the-loop. We first describe the use of a uniform threshold to direct each operation.

The *fuseSaddle* operation traverses up the hierarchy from the leaves up to the trunk, merging branches whose saddle values are closer than a given saddle value proximity threshold. The *diffuseBranch* operation selects all branches at a chosen depth threshold and diffuses the descendants for each branch. The *fuseBranch* operation identifies groups of branches that are pairwise closer than a given spatial proximity threshold. For each group, a single branch is inserted into the hierarchy replacing all branches in the group.

Rectangles in a mergemap are easy to select. So, it makes sense to ask a user to manually identify the branches to be fused or diffused. To perform a fuse operation, the user selects all containers that are to be merged. Children of the selected branch are placed within a single container. In case of a diffuse operation, the user selects a single container and all descendants are hierarchically compressed.

From our experience, it is best to reduce a mergemap in the following order: perform persistence based simplification of the scalar field [31] and construct a branch decomposition, use uniform thresholds for the fuse/diffuse operations before the mergemap is rendered, finally apply the operations manually based on expert input.

3.5 Area Distortion

The size of containers in the mergemap is not exactly equal to the sum total size of its boxes. Each container has a constant padding that acts as a small cushion. This padding eats into the area of the children. So the area of the boxes in the illustrations is not exactly proportional to the value associated with it, but slightly smaller. The padding helps users perceive the hierarchy of the tree and hence select containers to perform the reduction operations. On a similar note, treemaps cannot directly show non-positive scalar values.

4 Case Studies

We describe four applications to demonstrate the utility of the mergemap. The datasets used in this section are available in the public domain [15, 36]. We use TTK [30] for computing the merge trees and persistence based simplification. Previous designs [12, 34] have also used the same datasets in Sections 4.2 and 4.3, and therefore we compare mergemaps with their representation for those case studies.

4.1 Ethane-1,2-diol

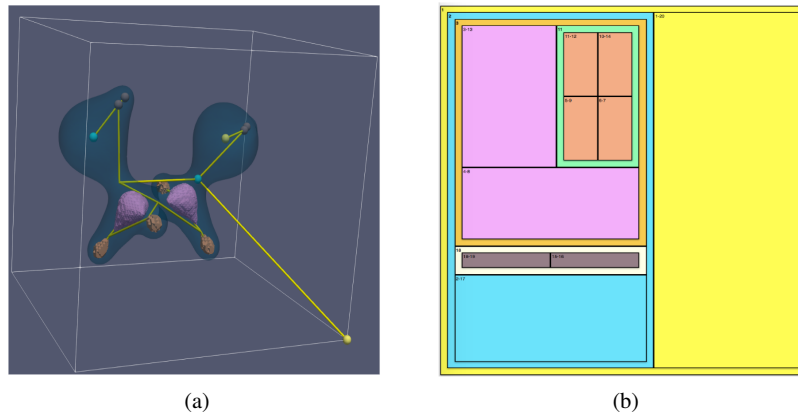
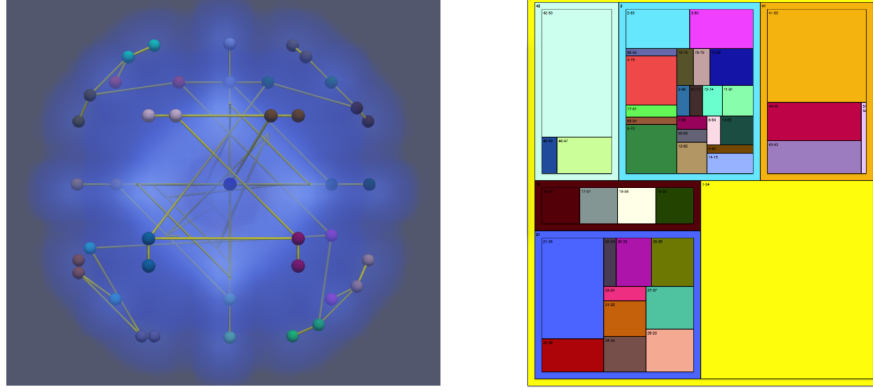


Fig. 5: Bonds in Ethane-1,2-diol: (a) Isosurfaces and the merge tree. (b) Mergemap. Similar sized boxes are assigned a common color. We observe that this corresponds to similar bonding regions.

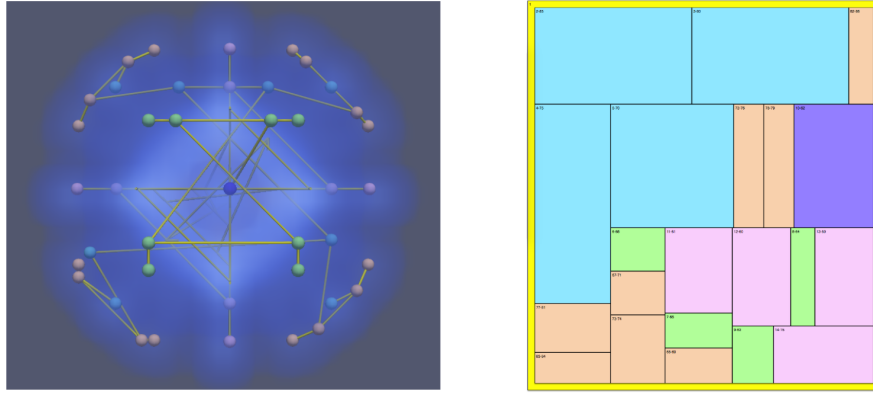
The Ethane-1,2-diol dataset is a 3D electron density distribution over a small molecule. Higher density regions correspond to atom centers. This is a relatively small dataset, the merge tree contains only 20 critical points. We use a squarified treemap [4] for the layout. This layout creates approximate squares as opposed to elongated rectangles, for easy selection and comparison. After performing a few reduction operations, we get the mergemap shown in Figure 5. Using brushable persistence barcodes, we assign a common color to similarly sized boxes. The similarly sized boxes directly correspond to similar bonding regions in this dataset. The orange boxes correspond to Carbon-Hydrogen bonds, brown boxes correspond to Oxygen-Hydrogen bonds, and the pink boxes correspond to the Carbon-Carbon bond.

4.2 Fuel


The dataset represents fuel density in a combustion chamber after fuel is injected. Understanding its structure is important for finding better combustion schemes. Past work [34, 29] have shown that the dataset exhibits radial symmetry. We attempt to replicate their results using mergemaps. Figure 6 shows our results. Initially, even after uniformly diffusing the mergemap, there were too many colors and nodes that affected the ability to locate the interesting features. However, since each container represents a topological feature, we were able to quickly locate the turbulent region.



(a) Fuel and its uniformly diffused mergemap. The turbulent feature container is shown in light blue.



(b) Turbulent feature of fuel and its mergemap after zooming and annotation.

Fig. 6: Volume rendering of the density field in the Fuel dataset using a blue-red color map () and the corresponding mergemaps.

In order to remove clutter due to the presence of the other features, we zoomed into this turbulent container. Next, we used brushable persistence bar codes and found several boxes of the same size, which directly corresponded to symmetrical features.

We now compare the interface of mergemaps with that of Denali and topological landscapes for this dataset, see Figure 6 (g)-(i) from [12]. One, mergemaps supports an interactive query-driven approach that helps locate the symmetrical structures. Denali and topological landscapes do not allow for such interaction. Two, Denali presents the (unrooted) contour tree. It considers the landscape corresponding to every possible root edge, and then chooses the best landscape by defining a metric distance between them. As a result, each such landscape can lead to a different

interpretation and analysis. For instance, it is perceptually difficult to understand why the turbulent part of the fuel dataset (highlighted in yellow) appears to be below the stable part (highlighted in green) in Denali’s landscape. Three, topological landscapes often have large spaces in the terrain, that diminishes simple comparative perceptual tasks.

4.3 Silicium

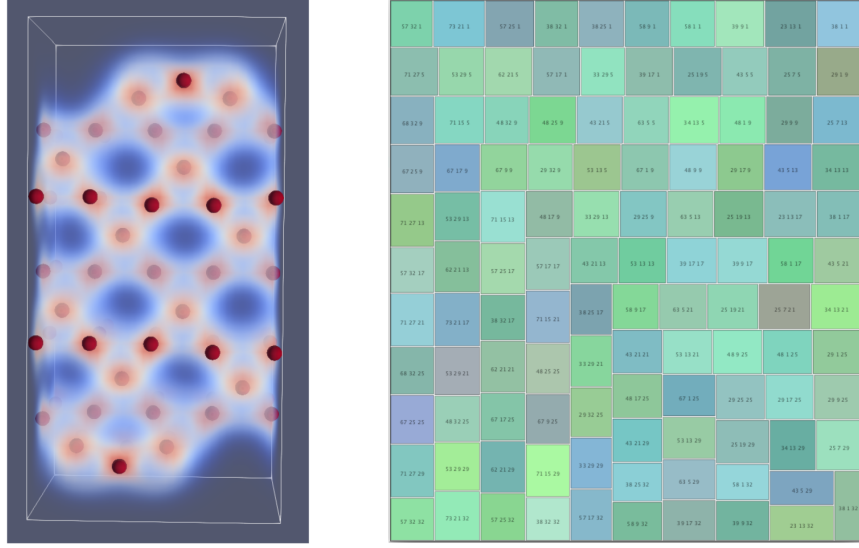


Fig. 7: Volume rendering of the Silicium dataset using a blue-red colormap (left) and maxima of the density field (left). Spatially ordered treemap where the box size indicates persistence (right).

In the mergemaps that we have studied so far, if two containers are adjacent to each other, it implies that the corresponding nodes have the same depth in the persistence hierarchy tree. This notion of adjacency between containers can also be extended to show spatial relationships amongst them. We use a spatially ordered treemap [37] for this purpose.

We illustrate this using a silicium grid, where the atoms appear as maxima. Such a grid is tightly packed, and the atoms are very close to each other at regular intervals. Scientists are generally interested in understanding impurities in such datasets and the atoms that are affected by them. Understanding spatial relationships is significant for this study. For this dataset, we suppress the hierarchy of the merge tree by fusing

all saddles together, resulting in a bush where all maxima are connected to a single saddle.

There is one hurdle in depicting the spatial proximity. The coordinates of the maxima are in 3D and the treemap has a 2D layout. We perform dimensionality reduction on the coordinates of the maxima using Principal Component Analysis (PCA) to project them onto the plane. Using the reduced principal coordinates and their persistence values as input to a spatially ordered treemap, we can visualize spatial relationships amongst the maxima, see Figure 7.

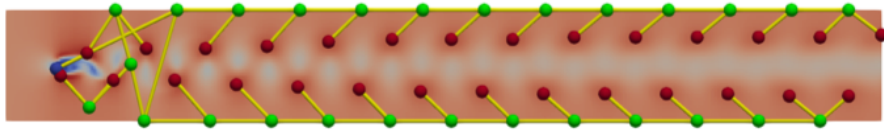
All the boxes are of the same size. So, we infer that there are no impurities in the dataset. We expect impurities to have a very small or large persistence and hence appear disproportionately in the mergemap. If we identify such a box in a spatially ordered mergemap, we can define a radius and select all atoms that are affected by this impurity. This technique could potentially be used to understand higher dimensional datasets that are projected onto the plane using a topology preserving dimensionality reduction method [38].

We now compare the interface of mergemaps with that of topological landscapes for this dataset, see Figure 12 from [34]. In both representations, it is difficult to distinguish individual maxima/minima because their sizes are similar. Mergemaps provides an additional guarantee that spatially proximal critical points appear close to each other in the visual representation. Further, we believe that a terrain representation is cumbersome to perceive contour trees, since it is difficult to analyze both maxima and minima at the same time without constantly rotating the 3-D view of the terrain. This is probably why a flipped version of the terrain is used to show minima. Mergemaps avoids this issue by focusing on merge trees.

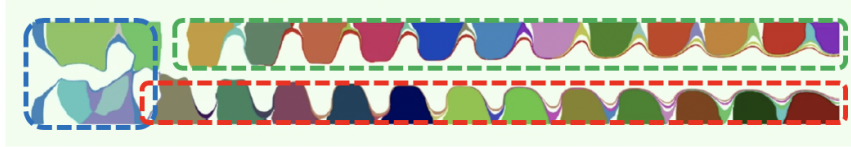
4.4 von Kármán street

The von Kármán street dataset is obtained from a simulation of 2D viscous flow behind a cylinder. This dataset is widely used as a demonstration for understanding time-varying data using topological analysis. Maxima of the velocity magnitude field directly correspond to vortices. Previous topological analysis of this dataset [22, 17, 28] have reported periodicity in its vortex shedding. Here, we study a single time step of the data set. In particular, we are interested in answering two questions. (a) What is the spatial structure of the flow and the vortices? (b) How are individual vortices different from one another? We propose a minor variant of mergemaps that helps answer the two questions.

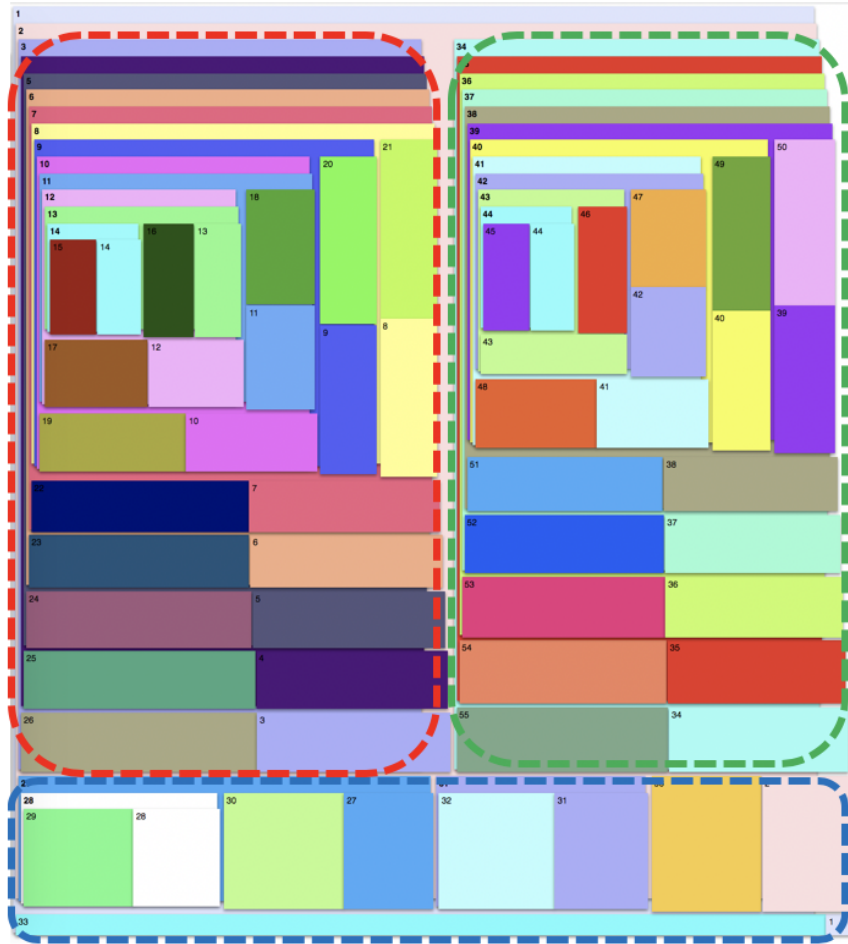
First, we introduce an additional constraint while constructing the branch decomposition from its merge tree that helps capture the spatial structure. A branch is attached as a child only if the index of its saddle end point in the preorder traversal of the merge tree lies in between the preorder indices of the end points of its parent branch. Further, we use a cascaded treemap [16] to display the aggregate tree. By design, cascaded treemaps use layering and offsetting rectangles to convey depth and hence showcase the structure of a tree.



(a) First time step of the von Kármán street rendered using a blue-red colormap () and merge tree of the magnitude field.



(b) Segmentation using a merge tree



(c) Cascaded mergemap for von Kármán street

Fig. 8: Cascaded mergemap captures the structure of the von Kármán street. Circled boxes in the mergemap correspond to the segments shown in (b).

Differences between vortices can be observed by studying maximum-saddle pairs in the merge tree. This requires the function values associated with the maximum and the saddle to be presented in the mergemap. So, instead of inserting a single imposter node to represent a branch, we insert two imposter nodes each one representing the two critical points. The size of a container now represents the sum total of function values of all critical points beneath it *i.e.*, the approximate hypervolume corresponding to the descendant branches. The size of a box represents the scalar function value associated with the critical point. The box of a saddle and its container are assigned the same color. One undesirable outcome of this variant is that a low persistent feature with high function value will be shown as a large container. For example, given a height function defined over a hand, low persistent features near the tip of the fingers, will be shown as large rectangles.

We now answer both questions about the dataset. Figure 8 shows how the mergemap captures the structure of the merge tree. It indicates that the domain can be split into three parts. The clockwise vortices on the top and counter clockwise vortices on the bottom of the vortex street correspond to the left and right parts of the mergemap, respectively. The bottom part of the mergemap corresponds to instabilities behind the cylindrical obstacle. We are able to answer the second question even though the padding results in area distortion. The lower region of the left and right part contains similar large boxes indicating that the vortices near the cylinder are extremely stable. However, as we move up, going deeper into the nested hierarchy, the size of the boxes reduce. This indicates that the vortices away from the cylinder obstacle have considerably lost speed. We also observe that both vortex streams "mirror" each other. To check whether our findings from a single time step are true for other time intervals, we computed and rendered the mergemap for each time step. The mergemap for all the time steps contains three parts but several small noisy features appear and disappear behind the cylindrical obstacle.

5 Conclusions

We have presented a treemap based design, that enables improved perception and interaction while exploring merge trees. We also discuss the best practices to interact with and analyze data using such a presentation. We demonstrate their utility on multiple datasets. They are simple to implement and lead to easy interpretations for better topological analysis.

Acknowledgments. This work is partially supported by the Department of Science and Technology, India (DST/SJF/ETA-02/2015-16), a Mindtree Chair research grant, and the Robert Bosch Centre for Cyber Physical Systems, Indian Institute of Science, Bangalore.

References

1. K. Beketayev, G. H. Weber, D. Morozov, A. Abzhanov, and B. Hamann. Geometry-preserving topological landscapes. In *Proceedings of the Workshop at SIGGRAPH Asia, WASA '12*, pages 155–160, New York, NY, USA, 2012. ACM.
2. R. Blanch and E. Lecolinet. Browsing zoomable treemaps: Structure-aware multi-scale navigation techniques. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1248–1253, Nov. 2007.
3. A. Bock, H. Doraiswamy, A. Summers, and C. T. Silva. Topoangler: Interactive topology-based extraction of fishes. *IEEE Trans. Vis. Comput. Graph.*, 24(1):812–821, 2018.
4. M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *In Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. Press, 1999.
5. H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *IEEE Visualization 2004*, pages 497–504, Oct 2004.
6. C. Correa, P. Lindstrom, and P.-T. Bremer. Topological spines: A structure-preserving visual representation of scalar fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1842–1851, Dec. 2011.
7. D. Demir, K. Beketayev, G. H. Weber, P.-T. Bremer, V. Pascucci, and B. Hamann. Topology exploration with hierarchical landscapes. In *Proceedings of the Workshop at SIGGRAPH Asia, WASA '12*, pages 147–154, New York, NY, USA, 2012. ACM.
8. H. Doraiswamy, N. Ferreira, T. Damoulas, J. Freire, and C. T. Silva. Using topological analysis to support event-guided exploration in urban data. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2634–2643, 2014.
9. H. Doraiswamy and V. Natarajan. Computing reeb graphs as a union of contour trees. *IEEE Trans. Vis. Comput. Graph.*, 19(2):249–262, 2013.
10. H. Edelsbrunner and J. L. Harer. *Computational Topology, An Introduction*. American Mathematical Society, 2010.
11. A. Gyulassy, N. Kotava, M. Kim, C. D. Hansen, H. Hagen, and V. Pascucci. Direct feature visualization using morse-smale complexes. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1549–1562, Sep. 2012.
12. W. Harvey and Y. Wang. Topological landscape ensembles for visualization of scalar-valued functions. *Comput. Graph. Forum*, 29(3):993–1002, 2010.
13. C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth. A survey of topology-based methods in visualization. *Comput. Graph. Forum*, 35(3):643–667, June 2016.
14. C. Heine, D. Schneider, H. Carr, and G. Scheuermann. Drawing contour trees in the plane. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1599–1611, Nov 2011.
15. P. Klacansky. Open scivis datasets, April 2019. <https://klacansky.com/open-scivis-datasets/>.
16. H. Lü and J. Fogarty. Cascaded treemaps: Examining the visibility and stability of structure in treemaps. In *Proceedings of Graphics Interface*, 2008.
17. V. Narayanan, D. M. Thomas, and V. Natarajan. Distance between extremum graphs. In *2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 263–270, April 2015.
18. P. Oesterling, C. Heine, H. Janicke, G. Scheuermann, and G. Heyer. Visualization of high-dimensional point clouds using their density distribution’s topology. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1547–1559, Nov 2011.
19. P. Oesterling, C. Heine, G. H. Weber, and G. Scheuermann. Visualizing nd point clouds as topological landscape profiles to guide local data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):514–526, Mar. 2013.
20. V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli. Multi-resolution computation and presentation of contour trees. In *Proc. IASTED Conference on Visualization, Imaging, and Image Processing*, 2005.
21. B. Rieck, H. Leitte, and F. Sadlo. Hierarchies and ranks for persistence pairs. Workshop on Topology-Based Methods in Visualization (TopoInVis), Feb. 2017.

22. H. Saikia, H.-P. Seidel, and T. Weinkauff. Extended branch decomposition graphs: Structural comparison of scalar data. *Computer Graphics Forum*, 33(3):41–50, 2014.
23. H.-J. Schulz, S. Hadlak, and H. Schumann. The design space of implicit hierarchy visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):393–411, Apr. 2011.
24. N. Shivashankar and V. Natarajan. Parallel computation of 3d morse-smale complexes. *Comput. Graph. Forum*, 31(3):965–974, 2012.
25. N. Shivashankar, P. Pranav, V. Natarajan, R. van de Weygaert, E. G. P. Bos, and S. Rieder. Felix: A topology based framework for visual exploration of cosmic filaments. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1745–1759, 2016.
26. B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, Jan. 1992.
27. R. Sridharamurthy, T. B. Masood, H. Doraiswamy, S. Patel, R. Varadarajan, and V. Natarajan. Extraction of robust voids and pockets in proteins. In L. Linsen, B. Hamann, and H. Hege, editors, *Visualization in Medicine and Life Sciences III, Towards Making an Impact.*, Mathematics and Visualization, pages 329–349. Springer, 2016.
28. R. Sridharamurthy, T. B. Masood, A. Kamakshidasan, and V. Natarajan. Edit distance between merge trees. *IEEE Transactions on Visualization and Computer Graphics*, 26(3):1518–1531, March 2020.
29. D. M. Thomas and V. Natarajan. Symmetry in scalar field topology. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2035–2044, Dec. 2011.
30. J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics*, 2017. <https://topology-tool-kit.github.io/>.
31. J. Tierny and V. Pascucci. Generalized topological simplification of scalar fields on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2005–2013, Dec 2012.
32. A. A. Valsangkar, J. M. Monteiro, V. Narayanan, I. Hotz, and V. Natarajan. An exploratory framework for cyclone identification and tracking. *IEEE Trans. Vis. Comput. Graph.*, 25(3):1460–1473, 2019.
33. K. J. Vicente, B. C. Hayes, and R. C. Williges. Assaying and isolating individual differences in searching a hierarchical file system. *Human Factors*, 29(3):349–359, 1987. PMID: 3623569.
34. G. Weber, P. Bremer, and V. Pascucci. Topological landscapes: A terrain metaphor for scientific data. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1416–1423, 2007.
35. G. H. Weber, S. E. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):330–341, 2007.
36. T. Weinkauff and H. Theisel. Streak lines as tangent curves of a derived vector field. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1225–1234, November - December 2010.
37. J. Wood and J. Dykes. Spatially ordered treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1348–1355, Nov 2008.
38. L. Yan, Y. Zhao, P. Rosen, C. Scheidegger, and B. Wang. Homology-preserving dimensionality reduction via manifold landmarking and tearing. *CoRR*, abs/1806.08460, 2018.