

# Computing and Visualizing Time-Varying Merge Trees for High-Dimensional Data

Patrick Oesterling, Christian Heine, Gunther H. Weber, Dmitriy Morozov, and Gerik Scheuermann

**Abstract** We introduce a new method that identifies and tracks features in arbitrary dimensions using the merge tree—a structure for identifying topological features based on thresholding in scalar fields. This method analyzes the evolution of features of the function by tracking changes in the merge tree and relates features by matching subtrees between consecutive time steps. Using the time-varying merge tree, we present a structural visualization of the changing function that illustrates both features and their temporal evolution. We demonstrate the utility of our approach by applying it to temporal cluster analysis of high-dimensional point clouds.

## 1 Introduction

With increasing size and dimensionality, time-varying data become difficult to visualize and analyze. One solution to this challenge is to detect features, i.e., salient data subsets, at each point in time and track their evolution to obtain more compact and less cluttered visualizations. For example, when meaningful features of scalar fields are defined by thresholding, a topological structure called the *merge tree* compactly encodes features for all possible thresholds.

To track features over time, existing methods commonly fix a threshold; changing this value requires expensive recomputation of the tracking information [30]. Sup-

---

Patrick Oesterling and Gerik Scheuermann  
Computer Science Dept, University of Leipzig, D-04009, Germany, e-mail: {oesterling, scheuermann}@informatik.uni-leipzig.de

Christian Heine  
Department of Computer Science at ETH Zürich, 8092 Zürich, Switzerland e-mail: cheine@inf.ethz.ch

Gunther H. Weber and Dmitriy Morozov  
Computational Research Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA-94720 e-mail: ghweber@lbl.gov, dmitriy@mrzv.org

porting threshold changes over time or on a per-feature basis depends on specifying a large number of parameters *a priori*. Moreover, many existing methods correlate features via spatial overlap in consecutive time steps—an approach that can lead to ambiguities [25] and becomes computationally intractable in higher dimensions. Time-varying Reeb graphs [11, 19] require complicated distinction of cases for topological events; their number increases with the dimensionality and no case tables have been presented beyond the 3-D case.

Here, we introduce time-varying merge trees—a topological summary of time-varying scalar fields that addresses these issues as follows. Instead of using a single threshold, we track features for all thresholds, enabling threshold selection guided by the time-varying merge tree’s visualization. To avoid tracking ambiguities, our algorithm records all necessary changes to the merge tree within the time interval between adjacent linearly-interpolated time steps, establishing a clean topological foundation for feature tracking. By focusing on merge trees for threshold-based feature tracking, we are able to provide a complete set of cases valid for all dimensions.

We represent time-varying merge trees as sequences of landscape profiles [22] and link hills visually to indicate structural changes. To reduce the clutter caused by showing many hills and links, we use topological simplification and represent preservation of topological feature groups by one visual link rather than many.

We demonstrate the utility of our approach by applying it to the analysis of time-varying, high-dimensional point clouds, where the merge tree of the density function captures clusters and their nesting [21].

## 2 Related Work

Defining and tracking features [23] is a common solution to visualizing large time-varying data sets. We focus on scalar fields, where many feature definitions are based on isosurfaces [20].

Szymczak [26] annotates contour trees for consecutive time steps to support queries of contours that evolve in particular ways or hit the boundary. Sohn and Bajaj [25] track contours using a similarity measure that considers spatial overlap of the inside and outside of contours. Ji and Shen [16] use the earth mover’s distance to determine correspondence among contours.

For a family of real-valued functions on a common  $d$ -manifold without boundary, Edelsbrunner et al. [10] define Jacobi sets, which can be used to track critical points. On this basis they compute the time-varying contour tree of a function on the 3-sphere [11]. However, changes to the contour tree require detailed case analysis and the algorithm is difficult to extend to higher dimensions [11, 18]. By restricting considerations to the merge tree, our algorithm considers fewer and simpler cases, and—more importantly—makes it independent of the domain’s dimension. Instead of tracking critical points explicitly, Cohen-Steiner et al. [9] use evolving persistence diagrams to trace critical value pairs visually.

Keller and Bertram [17] present a method to compute a time-varying isosurface from a so-called hyper-Reeb graph, i.e., a Reeb graph augmented with Betti numbers indicating, e.g., genus changes. While isosurface extraction is applicable for arbitrary dimensions [2],  $d$ -dimensional regular grids of hypercubes and isosurfaces extracted as sets of  $(d - 1)$ -dimensional simplices are quickly becoming impractical in higher dimensions, and Bhaniramka et al. [2] provide only 4-D and 5-D examples.

Bremer et al. [4] use the Morse-Smale complex to compute burning regions restricted to an isotherm for a range of fuel consumption thresholds. Once an appropriate fuel consumption threshold is identified, they use the Reeb graph of a 4-D space-time isosurface to track these regions over time [4, 28]. In later work, Bremer et al. [5] use the merge tree to compute statistics about burning regions in combustion simulations within a single time step. Once an appropriate threshold is identified, burning regions are tracked over time via overlap.

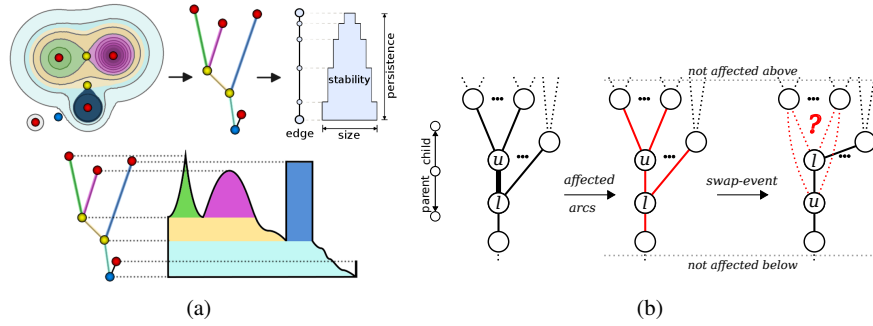
Visualizing tracked features can be challenging. Bremer et al. [3] survey methods and applications of topological feature tracking in molecular analysis, combustion simulations, and porous materials analysis. The results are shown as feature tracks embedded in the original domain. Similarly, Chen et al. [8] showed an embedding of the Reeb graph in the original data to track level sets of particle data. Other approaches store the tracking in a graph that is then shown using graph layout techniques. Widanagamaachchi et al. [30] present tracking graphs that update quickly when the user changes the isovalue of the tracked features. Because we compute tracking information for all features up front, we can layout the tracking information directly and show more feature properties, e.g. their size and robustness.

For point cloud data—the high-dimensional example application in this paper—Turkay et al. [27] present a system to explore time-varying clusters along with their structural properties, but this splits information into multiple views that need to be integrated in the analyst’s mind. In contrast, our visualization technique shows all relevant metrics in the same view as the cluster evolution.

### 3 Background

A *superlevel set* of a function  $f : \Omega \rightarrow \mathbb{R}, \Omega \subset \mathbb{R}^d$  for a value  $v$  is the set of all points  $x \in \Omega$  where  $f(x) \geq v$ . It may consist of multiple connected components, whose evolution with varying  $v$  is of particular interest to us. Imagine the function  $f$  as a landscape, initially fully submerged by water, and the value  $v$  as the current water level. When the water is slowly drained, i.e.,  $v$  is decreased, hills will start to emerge from the water, corresponding to new superlevel sets created at  $f$ ’s local *maxima*. When draining the water further, hills/superlevel sets will merge at points called *saddles*. The draining process stops when  $v$  reaches  $f$ ’s *global minimum*. Although the metaphor uses a 2-D landscape, the concepts are applicable in any dimension.

The *merge tree* encodes changes of superlevel set connectivity. Leaves represent local maxima, inner nodes represent saddles, the root represents the global minimum, and *arcs*, i.e., directed edges, connect nodes according to the process outlined. Each



**Fig. 1** (a) A scalar function's superlevel set boundaries and critical points indicate feature candidates. The merge tree encodes superlevel set evolution; its arcs and properties can be represented as a landscape profile. Color indicates correspondences among regions, merge tree arcs, and hills. A hill's height and width signify the corresponding superlevel set's persistence and size, respectively. The distribution of implicitly stored function values is reflected by a hill's shape; and thus its area reflects stability. The green hill is not very stable and the blue hill is maximally stable. (b) Left: If nodes  $u$  and  $l$  transpose, the merge tree needs to be reconstructed locally. Center: Only the arcs incident to  $u$  and  $l$  (red) are affected, whereas the rest of the tree remains unchanged. Right: Some arcs (black) are implicitly correct, while others (dotted red) need to be validated.

node is labeled with the value  $v$  of its corresponding event. Each node and each point on an arc correspond to exactly one connected component of a superlevel set for one value  $v$  (cf. Fig. 1a).

For piecewise-linear functions on simplicial grids, Carr et al. [6] give an algorithm to compute the *augmented merge tree*: Initially, each grid vertex is represented by one node in an otherwise empty tree and one set in a union-find data structure. The algorithm processes all grid vertices  $u$  in order of decreasing function value and (1) determines the sets of  $u$ 's *upper link*, i.e., grid neighbors with higher function value, (2) adds an arc from each set's lowest node to  $u$ 's node, and (3) unites these sets with  $u$ 's set and declares  $u$ 's node as the new set's lowest node. In addition to leaves, saddles, and the root node, the augmented merge tree also consists of *regular nodes*, which have exactly one outgoing and one incoming arc. Removing all regular nodes yields the merge tree, consisting of so-called *superarcs* and *supernodes*.

Noise in the data complicates the merge tree, but it can be detected and removed by *topological simplification*. In this process, superarcs are annotated with a measure of robustness. Repeatedly, the leaf superarc of lowest robustness is removed from the tree, potentially turning a saddle into a regular node, which, together with its two incident superarcs, is then replaced by a superarc. For each region mapped onto a superarc, we use three measures: *persistence* [12], the difference between the maximum and minimum function value of a region, *size*, the number of grid vertices of a region, and *stability* [22], a summed function value distribution of a region.

The merge tree can be represented as a 2-D landscape profile [22], which is a variation of 3-D topological landscapes [29, 15]—a terrain metaphor for the more complex contour tree. A topological landscape has the same topology like its input

tree and can represent structures of any dimension without structural occlusion. In a 2-D landscape profile, nesting hills represent superarcs of the merge tree; free of any occlusion and perspective distortion. Feature size is signified by width, persistence by height, and the distribution of function values in a region by the shape of the hills.

## 4 Overview

We visualize time-dependent, real-valued functions defined on domains of any dimension by studying how superlevel sets evolve with time. Our algorithm (1) computes the merge tree for each scalar field in the input sequence; (2) finds the sequence of structural changes that transforms each merge tree into its successor, storing these changes as *tracking records*; (3) uses topological simplification to remove noise from the merge trees, adjusting the tracking information accordingly, and (4) represents each tree as a landscape profile and translates tracking records into visual links connecting related hills. Steps (1)–(3) can be run concurrently for pairs of consecutive snapshots.

## 5 Merge Tree Transformation

To compute the merge tree evolution between two time steps, suppose that we computed the merge tree at each point in time for linear interpolation of the function between the two time steps. The construction algorithm outlined in Sec. 3 implies that the augmented merge tree’s structure does not change as long as the ordering of grid vertices based on their function value stays the same. Instead of an infinite set of merge trees, we only need to consider the finite number of changes to the augmented merge tree that arise when vertices *transpose*, i.e., their function values’ ordering changes. Moreover, because the augmented merge tree represents domain subset relations via arcs and paths, transpositions of nodes not connected by an arc do not affect the tree structure. It suffices to observe changes in the tree whenever two nodes joined by an arc transpose and their common arc *collapses*. We need to work on the augmented version of the merge tree because otherwise we would miss when a regular node pair becomes critical through a transposition.

Because reconstructing the whole merge tree from scratch after every single arc collapse would be computationally expensive, we consider how the merge tree can change after a single transposition. Fig. 1b illustrates a transposition for arc  $(u, l)$ , with  $u$  and  $l$  being the upper and lower nodes, respectively. There are two types of arcs: arcs not affected by the transposition that are implicitly preserved in the tree and arcs that may change, thus requiring additional validation.

**Lemma 1 (Arc Lemma).** *For a fixed domain and a fixed vertex order  $F$ , there is an arc  $(u, l)$ , with  $F(u) > F(l)$ , in the merge tree of  $F$  if and only if the component of*

$l$  in the domain restricted to the vertices  $w$  with  $F(w) > F(l)$  contains the vertex  $u$  and does not contain any vertex  $w$  with  $F(u) > F(w) > F(l)$ .

The proof of the lemma follows immediately from the algorithm used to construct merge trees, described in Sec. 3. It follows from the Arc Lemma that the transposition of  $u$  and  $l$  affects the merge tree only locally.

*Property 1.* Any arc that does not contain  $u$  or  $l$  remains in the merge tree.

*Proof.* Since the only change in the order  $F$  is the transposition of vertices  $u$  and  $l$ , if the two properties of the Arc Lemma hold for an arc  $(x, y)$ , with  $x, y \notin \{u, l\}$ , before the transposition, they continue to hold after the transposition. (And if they do not hold before, they do not hold after.)  $\square$

This property implies that the merge tree remains unchanged above all of  $u$ 's and  $l$ 's children, as well as below  $l$ 's parent and thus only arcs incident to  $u$  and  $l$  need further validation. While it is immediate that  $u$  and  $l$  are still connected after the transposition, the Arc Lemma also implies an arc between  $u$  and  $l$ 's parent node.

*Property 2.*  $u$  inherits  $l$ 's parent node.

*Proof.* Let  $p$  be  $l$ 's parent node. The component of  $l$  before the transposition is the same as the component of  $u$  after the transposition. Since the order of nodes between  $l$  and  $p$  before, and  $u$  and  $p$  after the transposition are the same, the Arc Lemma implies that we have an arc  $(u, p)$  in the tree after the transposition.  $\square$

The validation of  $u$ 's and  $l$ 's child arcs depends on their connection in the underlying grid. The nodes' links do not change, but their upper links do, and thus require analysis. For node  $l$ , the change of its upper link is limited.

*Property 3.*  $l$  retains the arcs to all the components that remain in its upper link.

*Proof.* For any arc  $(w, l)$ , the two properties of the Arc Lemma hold before the transposition. After the transposition, the first property holds because if the component of  $w$  remains in  $l$ 's upper link, then  $u$  belonged to a different component of  $l$  than  $w$  (so its removal could not have disconnected  $w$  from  $l$ ). The second property holds because there is one less node between  $w$  and  $l$ . In other words,  $(w, l)$  remains an arc after the transposition.  $\square$

To understand the changes to  $u$ 's children, we need to determine how its upper link is affected by the transposition.  $l$  can become a new upper link component, it can become part of an existing upper link component (a regular node), or  $l$  can combine an arbitrary number of  $u$ 's previous upper link components. We need to check which of  $u$ 's upper link components are in  $l$ 's upper link, once  $l$  is higher than  $u$ . In other words, we determine whether  $l$  is connected to some of  $u$ 's upper link components and thus becomes a regular node or a saddle. If  $l$  is not connected to any of  $u$ 's upper link components, it becomes a new maximum within the upper link of  $u$ .

To achieve our goal, we start a traversal towards the merge tree's root from each grid node  $x$  in  $l$ 's upper link. The traversal follows the unique path from  $x$  to the root

of the tree. Node  $l$  lies on this path since  $x$  belongs to the superlevel set component of  $l$ . Node  $u$  lies on this path if and only if  $x$  falls in the superlevel set component of some node  $y$  in  $u$ 's upper link (possibly, with  $x = y$ ). The component of  $y$  in the upper link of  $u$  is represented, without loss of generality, by an arc  $(y, u)$ . In this case,  $l$  inherits the arc  $(y, l)$  after the swap. Indeed, both conditions of the Arc Lemma are satisfied. If  $u$  does not lie on the path from  $x$  to the root, then, after the transposition, there is no connection in the superlevel set component of  $l$  between  $l$  and  $u$ 's former upper link. In this case no arc is redirected to  $l$ . Most transpositions are between regular nodes, but only few of these produce a new maximum-saddle pair. We noticed that this can only happen when the involved regular nodes are grid neighbors. If not, we can safely swap them without performing a child traversal.

Our implementation starts with the augmented merge trees of the first time step and two lists of the grid vertices sorted descending by their values at the first and second time step. The first list will be reordered over time for correct determination of a node's (changing) upper link. The second list is to determine potential transpositions of new arcs that occur after a transposition. The time of an arc collapse is inferred from the linear interpolation between the vertices' values in the first and the second time step. We keep arc collapses in a priority queue, breaking ties based on the lexicographical order of their incident node IDs; a straightforward extension of simulation of simplicity [13].

The number of arc collapses depends on how many node pairs change their relative order in both time steps; i.e. on the structural variation of the function. In the worst case, i.e., if the node ordering is reversed, their number is bounded by  $O(n^2)$  for  $n$  tree nodes. Potential push-updates to the priority queue take  $O(\log e)$ , for  $e$  arcs in the queue. The traversal to validate node  $l$ 's upper link depends on the number of tree nodes on the paths between  $l$ 's upper link nodes and  $l$  itself. Trivially, this number is bounded by  $n$ ; better bounds depend on the function itself and on grid granularity. In our experiments, however, we observed that the number of arc collapses is usually less than 5% of  $n^2$  and that the 90th percentile of the traversal lengths is around 10% of  $n$ ; while no traversal was longer than 30% of  $n$ .

## 6 Feature Tracking

By using a continuous transformation, we can identify structural changes of the augmented merge tree and we know the exact time and the order of all events. The challenge is now to infer the changes to the unaugmented merge tree's superarcs.

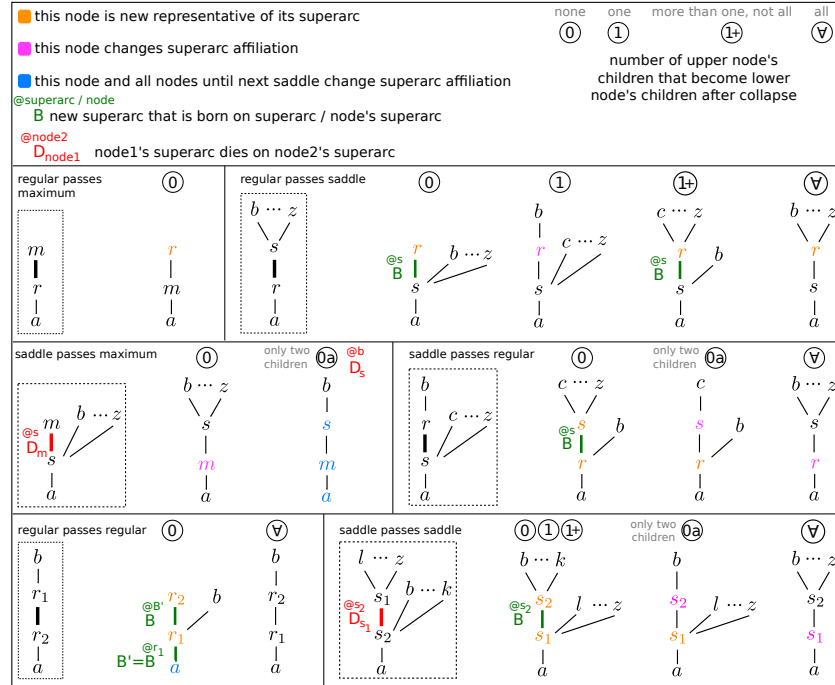
Between two time steps, superarcs may be born, die, or match with a superarc of the following merge tree. Fig. 2 illustrates how we distinguish these cases. For birth and death events we keep track on which parent superarcs they take place because the tree may undergo many changes. For example, a newborn superarc may move within the tree, or it may give birth to other superarcs. Similarly, a superarc on which another superarc died may move or die. In rare circumstances, an arc gives birth to the arc it dies on. Therefore, we have to store tracking information recursively.

To display superarc relations later on in the visualization, each superarc needs a representative. We use its upper supernode for this purpose. Leaf superarcs are thus represented by their maxima, and inner ones by their upper saddle node. For each transposition, we check if the superarcs of the tree are affected. If necessary, we create or destroy tracked arcs, and we record on which arcs these changes occur. If superarcs do not change, we record when regular nodes change their association to a superarc and, if necessary, update a superarc’s representative to handle moving features that are represented by a different set of grid vertices in the next time step.

For every superarc we maintain a *tracking record* that stores: the initial representative, the current representative, the superarc born from, and the superarc died on. Initially, the entries for superarc “born from” and “died on” are empty. We also store for each node to which superarc it belongs. One possibility to track superarcs—including precise place of birth/death and correct times—is to consider the node types before and after an arc collapse and to handle this event according to the case table in Fig. 2. This table summarizes all possible configurations of an arc collapse, using symmetry of events (e.g. “a saddle node rises above a maximum” versus “a maximum falls below a saddle”) and impossible events (e.g. “two maxima swap”) to reduce the number of considered cases. Possible results of a configuration reflect the changes of  $l$ ’s upper link. Furthermore, the table indicates in which configurations superarcs are born or die and in which cases a node’s superarc affiliation changes. Tracking then consists of simply creating or updating affected tracking records after every arc collapse. For example, if before a transposition the lower node is a saddle  $s$  and the upper node is a regular node  $r$  (Fig. 2, right column, second row) and both nodes are saddles after the transposition (Case 0), we first create a new superarc with  $s$  as its current representative and  $s$ ’s previous superarc as the “born from” entry. Then we set  $r$  to be  $s$ ’s previous superarc’s new representative and update the superarc connecting both nodes. For simplicity, events relating to the minimum of the tree are considered to be either regular or saddle events.

A function’s main features naturally appear as maximum-saddle pairs of significant persistence, size, or stability. Therefore, we restrict further processing and visualization to leaf superarcs. To create pairwise relations between original and final superarcs we post-process the tracking records after the transformation as follows: If the “died on” entry for an original leaf record is empty, we associate that record’s initial representative with its current representative, and store this as a *match record*. If the “died on” entry of an original leaf record is not empty, we recursively follow it until we reach the record where “died on” is empty. We associate the record’s initial representative with the record’s current representative found and store this as a *death record*. Finally, for each new leaf record, we recursively follow the “born from” entry, associate the found record’s initial representative and the new record’s current representative, and store this as a *birth record*. This gives us a set of records, classified into either match, birth, or death events, that tell us how features of the complete function relate to each other between the original and the final tree.





**Fig. 2** Case table indicating how tracking information is affected by the nodes' types before and after the transposition. For every possible configuration (dotted boxes), the bold arc is about to collapse and all possible results are shown to the right. In principle, superarcs die when supernodes pass each other, and a new superarc is born when the passing node becomes a supernode.

## 7 Simplification

Merge trees of noisy functions contain many small superarcs that represent features below meaningful thresholds. Topology-based simplification [7], i.e., the removal of those superarcs whose properties, e.g. their persistence, are below a user-specified threshold, has been applied successfully to reduce noise in the data.

If a superarc is removed from the tree, we also have to adjust those tracking records that have this superarc as their origin or target, i.e., as their initial or current representative. To this end, we redirect the tracking record by replacing the removed superarc by its parent superarc. Tracking records may become redundant by this process, e.g. if the target of a birth-record, or the origin of a death record is removed.

Note that for the time-varying analysis, changing the simplification threshold for the merge tree at time step  $i$  requires repeating simplification of tracking information between time steps  $i - 1$  and  $i$ , and between time steps  $i$  and  $i + 1$ .

## 8 Prototype Visualization and Examples

The time-varying merge tree consists of: the merge trees at the given time steps that describe the hierarchy as well as quantitative properties for all features, the tracking information of all features over time, and exact times for structural events of the complete function.

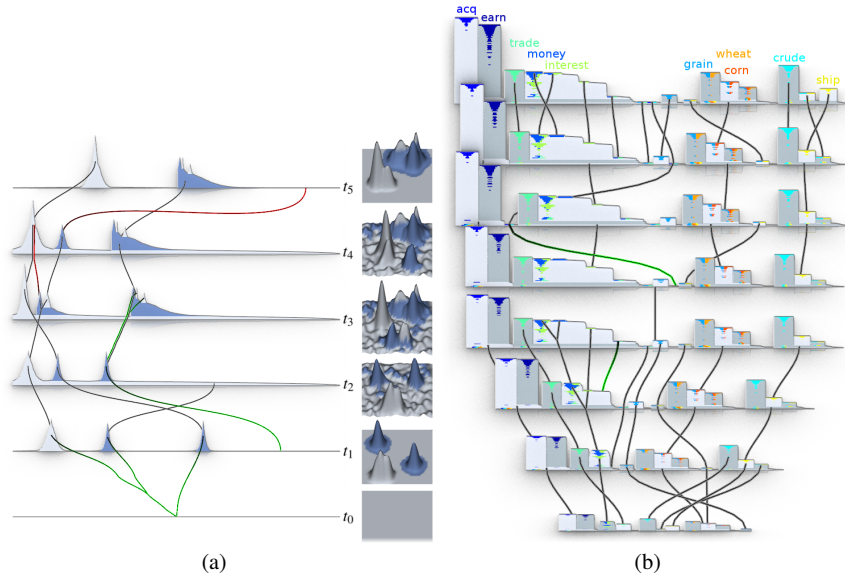
To reduce visual complexity, we use a discrete approach that separates the depiction of structure from that of temporal evolution. We display the merge trees as occlusion-free 2-D topological landscape profiles [22], stacked in the third dimension according to their time stamp, and using orthographic projection to facilitate feature comparison. We relate features over time using visual links between the profiles, similar to standard isotracking graphs [25, 5], but showing more feature properties and, most importantly, features for all thresholds in the landscape profiles.

To visualize the tracking records that associate two superarcs of two subsequent merge trees, we identify both superarcs using the record's initial and current representative, identify their areas in the profiles, and use these areas' centroids as the visual link's origin and target coordinates. To distinguish record types, we use black, green, and red links for match, birth, and death records, respectively. The user can filter tracking information by selecting arbitrary parts of the profiles. Tracking information is then filtered either forwards, backwards, or in both directions in time. More sophisticated analysis is achieved by combining simplification and interactive selection. For example, small features could be excluded from simplification if they are related to the evolution of user-selected features—e.g. if noise becomes a prominent user-selected feature later on.

Although topological simplification is the primary tool for removing irrelevant hills, we further reduce the remaining links to increase visual clarity by: *link aggregation* to group multiple incoming or outgoing links per hill by type/color and let them fork if necessary; *link unification* for links of different type/color that have exactly the same origin and target hills, which could happen for fine-grained topological events in one and the same feature, e.g., if a new arc is born inside a region on which the former maximum dies; and *match-link combination* to connect hierarchical subfeatures that do not change over time with a single black link at their lowest shared saddle. A single match link between two profiles indicates that structure is preserved entirely. Finally, *reordering* saddle node children can change hill positions in the profile without changing its topology. This strategy could be used to optimize crossing links; but does not allow to switch arbitrary hills.

### 2-D Example

Fig. 3a uses an artificial function in 2-D to illustrate the visualization of a time-varying merge tree. The grid consists of 2,500 vertices and the computation of the topologically most complex time step takes less than a second. We used a machine with two 2.4GHz Quad-Core AMD Opteron(tm) processors for all experiments. Because time steps are processed concurrently, the total time to obtain the image,



**Fig. 3** (a) Artificial, topologically simplified 2-D function (right):  $t_0$ : flat function.  $t_1$ : three main features are born.  $t_2$ : all features move and noise is added.  $t_3$ : all features move, one splits, the other two grow in size and persistence, respectively.  $t_4$ : features move and join.  $t_5$ : one feature dies, another is rotated, noise is removed. (b) Reuters data: Hills on the profiles for each time slice show the number, nesting, importance, and homogeneity of clusters with all relevant metrics (persistence, size, and stability) in the same view. Histograms, colored by class, verify that documents primarily accumulate by class and form subclusters for related categories. Over time, tracked features of the varying merge tree show how more subclusters break apart and grow individually. The profiles grow in their width and height to reflect that new documents are added with each additional time slice.

including topological simplification to remove small features, is approximately one second.

As can be seen, the tracking is robust with respect to noise and feature shape, and moving features are recognized if their spatial distance is small enough. Typically, a moving feature is identified as a superarc whose implicit regular nodes (grid vertices) only change or as a superarc that gives birth to a new one on which it dies afterwards. Time steps  $t_1$  and  $t_2$  show a counterexample. Because the spatial distance of the right feature in the function is too big, the tracking detects that a new feature gets born and the old one vanishes. A higher time resolution would be needed to correctly detect this as a fast moving feature. In general, death events are detected, but in time step  $t_5$  a feature matches to one of the maxima produced by added noise in that area. Because noise is removed by topological simplification, the link target correctly points to the removed hill's parent hill. If a feature splits, the new one typically matches to one of the former maxima ( $t_3$ , second hill) or a new feature is born ( $t_3$ , right hill). Likewise, if features join, they first share a higher saddle and then one feature would die on the other ( $t_4$ ). In  $t_3$ , the first two hills change their order because the topological

landscape profiles [22] sort subtrees of each merge tree saddle by persistence to put the highest hills to the left.

### High-Dimensional Example

To demonstrate the effectiveness and applicability of time-varying merge trees for high-dimensional data, we extend previous work by Oesterling et al. [21] on topology-based cluster analysis of static, high-dimensional points clouds. The authors assume the point cloud to be outcomes of a set of random variables with identical high-dimensional probability distributions. Using geometric graphs and a Gaussian kernel, they obtain an approximation of the high-dimensional probability density function whose merge tree topology they then visualize as a topological landscape. We extend this work by approximating a point cloud in space-time by its varying density function and analyze its superlevel set topology over time.

To make computation of the time-varying merge tree tractable, we construct a grid that remains unchanged over time, but still supports sampling the density functions of all time steps with sufficient accuracy. We merge all input point clouds into a single set of points, construct the neighborhood graph, and determine point densities using an appropriate Gaussian filter radius (see [21] for details) so separate all clusters.

We use categorized documents from the Reuters-21578 collection [1] that appeared on the Reuters newswire in 1987. To demonstrate accurate feature tracking in high dimensions, we extract documents for ten economy-related categories and use the *tf-idf* [24] document-term weighting to define word importances for each document’s dimensions in the vector space model. Using Linear Discriminant Analysis [14], a supervised projection that uses given classification information to minimize information-loss, we project the data to a ( $\#classes - 1 = 9$ )-dimensional space and end up with 5,309 documents, manually divided into eight time slices, 20 days each, from 02/21/1987 to 10/19/1987. Fig. 3b shows the visualization of the time-varying merge tree. The total time to obtain the image is around one second. The most complex transformation required processing approximately 33,000 arc collapses.

Based on classification information attached to the documents, we can place colored histograms on the hills to indicate how documents of different classes are distributed across the clusters. As can be seen, dense regions primarily match to documents of a single class, while some documents of related classes are in a subcluster relationship. The valley between hills of unrelated topics is typically low as it reflects the density between the corresponding clusters. Likewise, for related topics, like “grain”, “wheat” and “corn”, the subspace spanned by the vocabulary used in these documents also contains less specific documents between the cluster centers and thus saddle densities are higher. Another insight, suggested by the rectangular hill shape, is that clusters are very compact in the sense that document densities are close to the cluster’s maximum density; hence their positioning close to the hilltops. Over time, dense regions are primarily stable, but grow in size and persistence, and occasionally split into more subclusters that become increasingly prominent.

## 9 Conclusion

We introduced time-varying merge trees as a compact description of time-varying scalar fields and combined landscape profiles with a tracking information overlay to support exploration of time-varying scalar fields. We further demonstrated the utility of our method using a 9-D document collection data set. Our method can inform parameter selection for in-depth feature analysis.

The visualization is currently limited to showing a few time steps in a single image, and the optimal depiction of time-varying merge trees remains an open question. Future work will also focus on reducing the runtime of the transformation, which depends on the topological variance between two time steps. While the considered events are both necessary and sufficient for the augmented merge tree, it may be possible to process fewer events for unaugmented merge tree. We will also extend topological simplification to the time-varying merge tree, instead of simplifying time steps individually. While it is trivial to adapt our algorithm to the time-varying split tree and compute the contour tree for a given time, tracking the contour tree edges remains a topic of future work.

## Acknowledgements

The authors thank anonymous reviewers for valuable comments and assistance in revising the paper. This work was supported by a grant from the German Research Foundation (DFG) within the strategic research initiative on Scalable Visual Analytics (SPP 1335). This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. DOE under Contract No. DE-AC02-05CH11231 (Lawrence Berkeley National Laboratory) through the grant “Topology-based Visualization and Analysis of High-dimensional Data and Time-varying Data at the Extreme Scale”, program manager Lucy Nowell.

## Disclaimer

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the

Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

## References

1. K. Bache and M. Lichman. UCI machine learning repository, 2013.
2. P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurface construction in any dimension using convex hulls. *IEEE Transactions on Visualization and Comp. Graphics*, 10(2):130–141, 2004.
3. P. Bremer, E. Bringa, M. Duchaineau, A. Gyulassy, D. Laney, A. Mascarenhas, and V. Pascucci. Topological feature extraction and tracking. *Journal of Physics: Conference Series*, 78(1):012007, 2007.
4. P.-T. Bremer, G. H. Weber, V. Pascucci, M. Day, and J. B. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010.
5. P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2011.
6. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.
7. H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *Proceedings of the Conference on Visualization '04*, pages 497–504. IEEE CS, 2004.
8. F. Chen, H. Obermaier, H. Hagen, B. Hamann, J. Tierny, and V. Pascucci. Topology analysis of time-dependent multi-fluid data using the reeb graph. *Comp. Aided Geometric Design*, 2012.
9. D. Cohen-Steiner, H. Edelsbrunner, and D. Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, pages 119–126. ACM, 2006.
10. H. Edelsbrunner and J. Harer. Jacobi sets of multiple Morse functions. *Foundations of Computational Mathematics, Minneapolis*, pages 37–57, 2004.
11. H. Edelsbrunner, J. Harer, A. Mascarenhas, V. Pascucci, and J. Snoeyink. Time-varying Reeb graphs for continuous space-time data. *Computational Geometry*, 41(3):149–166, 2008.
12. H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Computational Geometry*, 28:511–533, 2002.
13. H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.
14. K. Fukunaga. *Introduction to statistical pattern recognition (2nd ed.)*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.
15. W. Harvey and Y. Wang. Topological landscape ensembles for visualization of scalar-valued functions. *Computer Graphics Forum*, 29(3):993–1002, 2010.
16. G. Ji and H.-W. Shen. Feature tracking using earth mover’s distance and global optimization. In *PG’06: Proceedings of the Pacific Graphics 2006*, 2006.
17. P. Keller and M. Bertram. Modeling and visualization of time-varying topology transitions guided by hyper Reeb graph structures. In *Proc. IASTED International Conference on Computer Graphics and Imaging*, pages 15–25, 2007.
18. A. Mascarenhas. private communication, 2013.
19. A. Mascarenhas and J. Snoeyink. Implementing time-varying contour trees. In *Proceedings of the 21. Annual Symposium on Computational Geometry*, pages 370–371. ACM, 2005.
20. A. Mascarenhas and J. Snoeyink. Isocontour based visualization of time-varying scalar fields. *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pages 41–68, 2009.

21. P. Oesterling, C. Heine, H. Jänicke, 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, 2011.
22. 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, 2013.
23. F. Reinders, F. H. Post, and H. J. Spoelder. Visualization of time-dependent data with feature tracking and event detection. *The Visual Computer*, 17(1):55–71, 2001.
24. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, Nov. 1975.
25. B.-S. Sohn and C. Bajaj. Time-varying contour topology. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):14–25, 2006.
26. A. Szymczak. Subdomain aware contour trees and contour evolution in time-dependent scalar fields. In *Shape Modeling and Applications, 2005 Int'l Conf.*, pages 136–144. IEEE, 2005.
27. C. Turkay, J. Parulek, N. Reuter, and H. Hauser. Interactive visual analysis of temporal cluster structures. *Computer Graphics Forum*, 30(3):711–720, 2011.
28. G. H. Weber, P.-T. Bremer, M. Day, J. Bell, and V. Pascucci. Feature tracking using Reeb graphs. *Topological Methods in Data Analysis and Visualization*, pages 241–253, 2011.
29. G. H. Weber, P.-T. Bremer, and V. Pascucci. Topological landscapes: A terrain metaphor for scientific data. *IEEE Transactions on Vis. and Comp. Graphics*, 13(6):1416–1423, 2007.
30. W. Widanagamaachchi, C. Christensen, V. Pascucci, and P.-T. Bremer. Interactive exploration of large-scale time-varying data using dynamic tracking graphs. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 9–17. IEEE, 2012.