
Book Section:
Carr, H orcid.org/0000-0001-6739-0283, Tierny, J and Weber, GH (2020) Pathological and Test Cases For Reeb Analysis. In: Topological Methods in Data Analysis and Visualization V. Mathematics and Visualization book series . Springer , pp. 103-120. ISBN 978-3-030-43035-1

https://doi.org/10.1007/978-3-030-43036-8_7

Pathological and Test Cases For Reeb Analysis

Hamish Carr and Julien Tierny and Gunther H. Weber

Abstract After two decades of computational topology, it is clearly a computationally challenging area. Not only do we have the usual algorithmic and programming difficulties with establishing correctness, we also have a class of problems that are mathematically complex and notationally fragile. Effective development and deployment therefore requires an additional step - construction or selection of suitable test cases. Since we cannot test all possible inputs, our selection of test cases expresses our understanding of the task and of the problems involved. Moreover, the scale of the data sets we work with is such that, no matter how unlikely the behaviour mathematically, it is nearly guaranteed to occur at scale in every run. The test cases we choose are therefore tightly coupled with mathematically pathological cases, and need to be developed using the skills expressed most obviously in constructing mathematical counter-examples. This paper is therefore a first attempt at reporting, classifying and analyzing test cases previously used for algorithmic work in Reeb analysis (contour trees and Reeb graphs), and the expression of a philosophy of how to test topological code.

Hamish Carr
School of Computing, University of Leeds, Woodhouse Lane, Leeds LS2 9JT, UK e-mail: h.carr@leeds.ac.uk

Julien Tierny
Sorbonne Universities, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, France e-mail: julien.tierny@lib6.fr

Gunther H. Weber
Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA e-mail: ghweber@lbnl.gov & University of California at Davis, One Shields Avenue, Davis, CA 95616, USA e-mail: ghweber@ucdavis.edu

1 Introduction

Computational topology began in the 1980s for scalar fields [14] in geographic information systems, and for vector fields [15] for scientific visualization, with scalar field analysis then developing for the analysis of 3D (volumetric) data.

Over time, topological analysis in scientific visualization has included techniques based on Reeb Analysis, Morse-Smale Analysis, Persistent Homology, Vector Field Analysis, and Tensor Field Analysis. While articles commonly describe new algorithms, they rarely describe testing strategies, although tools exist that could be applied, particularly for vector fields [21, 26]. We will, however, restrict our attention to Reeb Analysis, where we have prior experience. Similarly, while the strategies for debugging visualization described by Laramee [18] can be applied, we focus primarily on the test cases we use for topological algorithms.

While these techniques are powerful for understanding data, they are conceptually complex, but also particularly difficult to implement, as they are susceptible to a wide range of errors during program construction. Thus, in addition to the normal struggle to frame an algorithm accurately, robustly and efficiently, we have to contend with problems due to the difficulty of the underlying mathematics.

Between us, we have accumulated over forty years of experience in working with topological code. As a result, we have developed and employed a variety of strategies for constructing, testing and debugging programs. These strategies, however, rarely form part of publications, since there is usually barely room for all the technical details. Since these strategies are of value to other researchers or programmers attempting to grapple with complex algorithms, we therefore aim to start the discussion of test cases and testing strategies.

We do not have space for the details of all the algorithmic work, so we start with a quick overview instead in Section 2. We then sketch a number of conceptual approaches to test cases in Section 3 and introduce two types of pathological cases, flat regions in Section 4 and the *W-structure* in Section 5. Section 6 then gives some concrete examples of test sets that we have used, and Section 7 discusses some of the techniques that we use for visualizing intermediate results during debugging. Then Section 8 summarizes our experience and presents some conclusions.

2 Reeb Analysis

Reeb Analysis studies the relationships between isocontours to extract knowledge from a mathematical function or data set. Consider a *scalar field*, i.e. a function of the form $f : \mathscr{R}^d \rightarrow \mathscr{R}$. Since the data we wish to analyze is normally spatial in nature, we shall assume $d \in \{2,3,4\}$. A level set or inverse image of f is defined by choosing an *isovalue* $h \in \mathscr{R}$, then extracting all points in the domain of the function with function value h, i.e. $f^{-1}(h) = \{x \in \mathscr{R}^d : f(x) = h \in \mathscr{R}\}$. These level sets are often referred to as *isocontours* (*isolines* where $d = 2$, *isosurfaces* where $d = 3$).

Any given isocontour may have multiple connected components, which are ambiguously referred to as isolines and isosurfaces. We therefore use *isocontour components* to refer to the individual surfaces, in line with the literature.

If we contract each isocontour component to a single point, we construct the *contour tree* [3]. For more general functions, where the domain is a general manifold $\mathscr{M}$, the same construction gives the *Reeb graph* [19]. Although a special case of the Reeb graph, the contour tree is easier to compute [6], and the fastest Reeb graph algorithm reduces the input to a simple domain, computes the contour tree over that domain, then reconnects the domain (and the tree) to build the Reeb graph [24].

More recently, Reeb Analysis has been extended to functions of the form $f : \mathscr{M} \to \mathscr{R}^r$, where $r > 1$. These cases are covered by the mathematics of fiber topology, and we replace isocontours with *fibers* representing inverse images of the form $f^{-1}(h) = \{x \in \mathscr{R}^d : f(x) = h \in \mathscr{R}^r\}$. Continuous contraction of these fibers then results in the Reeb space [11]. This can be constructed approximately [4] for the general case, or precisely [22] for the case $f : \mathscr{R}^3 \to \mathscr{R}^2$.

Rather than recapitulate all of these algorithms, we refer the reader to the original papers, and assume some degree of familiarity with the details, as we are presently interested in describing debugging practice and test cases for them.

3 Approaches

Generally speaking, debugging complex code depends on testing representative types of data, since exhaustive testing of all possible inputs is combinatorially impossible. Within this, test sets may be analytic, stochastic, empirical, or synthetic, but the choice normally depends on the specific problem domain.

Analytic: Frequently, computation replicates an existing mathematical method, and as a result, test cases can be constructed from known mathematical examples, which were generally developed during mathematical debugging of an idea. Since these are likely to display interesting or challenging behaviour, they are commonly used as test functions for which the ground truth result is already well understood.

For our work in computational topology, this ideal approach has been less useful than it might seem. This occurs because mathematical development generally considers smooth infinitely differentiable functions. Since most code assumes simplicial or cubic meshes with linear, trilinear or ad hoc interpolation, the sampled data rarely captures the original mathematical function exactly unless sampled at high resolution. This is of particular concern when debugging, as manual validation of intermediate stages for anything over 10^3 is time-consuming and wearisome.

Moreover, mathematical reasoning is reductive, and attempts to deal with a small number of simple cases, in order to stay within the reasoning abilities of a human being. As a result, analytic examples tend to have simple topology – i.e. relatively small numbers of topological events. However, the sampling necessary to capture this causes them to be medium scale in terms of data, which makes them unattractive for early stage testing. Later on, simple topology has typically already been tested,

and medium scale examples are used to test combinations of simple topology. At this point, analytic functions are rarely complex enough to provide good medium scale tests. We therefore tend to avoid analytic functions except at the conceptual stage.

Stochastic: A second approach is to generate data sets stochastically - i.e. to choose randomly from all possible data sets. While this has the merit that it does not prefer any particular data sets, it fails to guarantee that challenging topology will be tested early, or indeed ever. As a result, we tend to avoid stochastic approaches.

Curiously, however, as the data scales up, stochastic effects mean that *every* pathological mathematical case will occur multiple times, leading to the problem that we refer to as **too much topology**. In practice, 1 gigabyte of data means that there may be tens of millions of topological events. If the isovalues at which they occur are independent, then even for double precision floating point, it is highly likely that multiple topological events will happen at the same isovalue, which means that robust handling of complex topology is always required. This has also driven much of the work on topological simplification, so paradoxically, while avoiding stochastic approaches in the abstract, we rely heavily on them in practice.

Empirical: Since the goal of computation is to process data, the third approach therefore looks to existing data, either from prior experience or from a current data problem. While it is generally simple to obtain data from a variety of sources, there are at least four problems with empirical data:

1. Scale: as with mathematical test data, empirical data is often at too large a scale to be useful in the early stages of development, although we commonly use empirical data for testing at the medium to large scale.
2. Noise: many acquired data sets, particularly medical data, are noisy due to the original acquisition process, and this tends to result in large numbers of topological events, which are undesireable for small scale testing. Noisy data types therefore tend to be of more use at the medium to large scale.
3. Blandness: clean empirical data can suffer the opposite problem: that the number of topological events is much smaller than the data set, again hampering manual validation. Clean simulation data is particularly prone to this.
4. Clumping: some types of empirical data, such as medical, tend to have heavily clumped values, as for example where isovalues correspond to different tissue types. This tends to result in many topological events over a narrow range of values, again hampering manual validation.

Having said that, empirical data becomes particularly useful at medium to large scales, since many data types naturally result in large numbers of topological events, providing a useful test of the scalability of the underlying approach. Moreover, we have found that terrain data, which is self-similar at different scales, is often useful for debugging, as discussed below.

Synthetic: Since neither analytic nor empirical approaches give good test sets for early stages, we find that early stage testing relies heavily on synthetic examples for algorithm development and debugging. We aim to keep input sizes small, and to exhibit a rich topological behaviour. We depend in particular on pathological cases

and counter-example construction. Scalable examples are then built algorithmically for data construction, by copying a known pathology, by working backwards from the desired output structure, or by replicating copies of smaller-scale features.

Although mathematical functions would seem to be the best strategy, these are most commonly C^∞, and are a poor fit to the demands of algorithmic development. We therefore tend to start with small sytnhetic examples, then scale by judicious selection of empirical data, usually starting either with small terrain examples or clean simulation data, and progressing to large complex data sets.

As we have noted, we tend to rely on experience with previous problems to choose data sets that have previously caused algorithmic, theoretical or interpretational difficulties. While not exclusive, there are two major types of data which we know routinely present these difficulties even at small to medium scales: flat regions and W-structures.

4 Flat Regions

Morse Theory assumes that critical points occur at unique values, and that there are no flat regions—i.e. regions with gradient of 0 but dimensionality > 0. While this considerably simplifies the mathematics, it tends to have the reverse effect in practical data. And, although perturbation through simulation of simplicity [12] allows us to reduce the problem to the mathematically tractable, it imposes both algorithmic and interpretational costs.

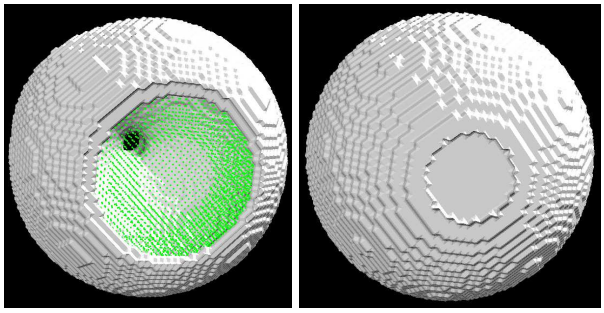


Fig. 1 Flat region in the hydrogen data set that is likely a quantization artifact and should be removed by simulation of simplicity. In the non-quantization version the hole would likely be filled in continuously until it disappears around a critical point.

To make matters worse, flat regions are frequently observed in quantized data sets. Many types of data have a narrow range of interesting values, with multiple topological events clustering tightly together. Even a small amount of quantization tends to result in flat regions. Moreover, for algorithmic purposes, flat regions are often broken up by symbolic perturbation [12], which adds a different mathematical

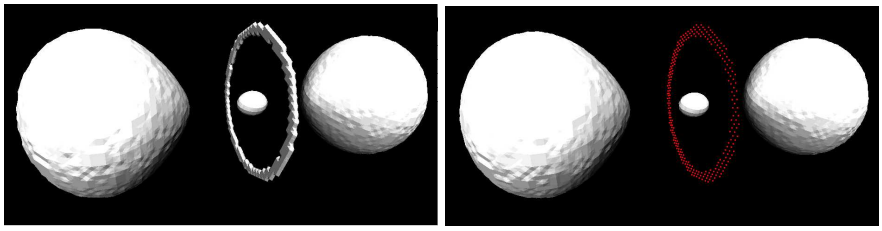


Fig. 2 Flat region in the hydrogen data set that is likely corresponding to a feature of interest and should be characterized in its entirety. The ring-structure likely has a correspondence in the smooth, real-valued function (but sampling on a grid without quantization would break it apart).

ε to each value to guarantee unique values throughout the data. This induces additional ε-persistent edges, which must then be suppressed in user interfaces and/or accounted for through simplification.

The converse of this is that these data sets are often valuable test cases of whether symbolic perturbation is implemented correctly and consistently. A good example of this occurs in the "nucleon" data set from VolVis. It contains small to moderately sized regions of constant value that can be resolved using symbolic perturbation.

Equally, the hydrogen data set [20] has extremely large constant regions that stress test symbolic perturbation implementations. For instance, most of the region around the hydrogen atom has a constant value, as shown in Figure 1. Here, the flat region is likely spurious and should be removed by symbolic perturbation. In the quantized data, a whole "cap" like structure forms around a flat region and subsequently closes an isosurface component. In this case, symbolic perturbation leads to the "correct" behavior: the component closes off smoothly.

However, not all flat regions are unimportant or spurious: they can be the regions of most interest in the data. Again considering the hydrogen data set, Figure 2 shows a flat region of particular interest where two protons interact. Here, even a continuous function would likely contain a region of constant value, i.e. the circle around which the ring forms. Since this behaviour is intrinsic to the underlying phenomenon, suppressing the region to ensure mathematically clean behaviour may actually mislead interpretation of results.

Ideally, we would be able to distinguish between "spurious" and real features, and to define stability for them in a mathematically sound framework. This may involve consideration of the difference between persistent simplification and geometric simplification [8], but is broadly speaking beyond the scope of the current discussion.

Phenomena like these have also led to approaches that aim at avoiding symbolic perturbation and identify critical regions directly [10, 25, 1, 17]. For example, in the hydrogen atom, the "ring structure" shown in Figure 2 appears around a region of constant function value. Symbolic perturbation breaks the ring up into at least a maximum (around which the ring starts to form in the perturbed version) and a saddle (where the ring closes in the perturbed version). One may argue that in this

instance it is desirable to detect the entire ring structure as a critical entity, i.e. a circle around which the ring forms.

5 W Structures

While flat regions test out ability to reconcile quantized data with mathematical formalisms, our other pathological case, the *W structure* can be constructed either mathematically or by observation in data.

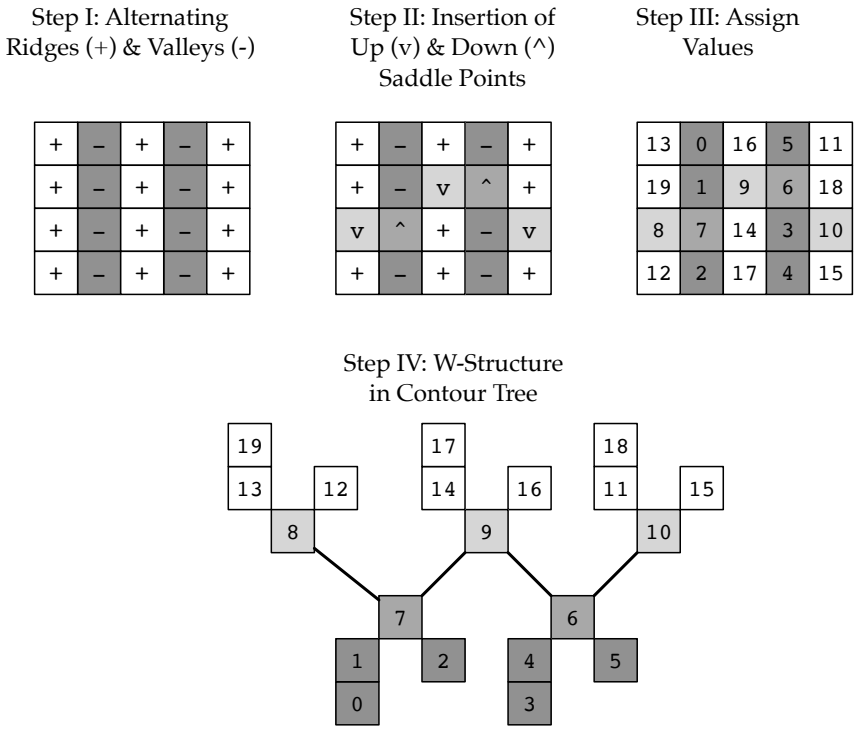


Fig. 3 An example of a "W structure" in a contour tree, with construction.

This structure was first shown as an illustration of a potential case by Carr et al. [7] (as Figure 2). However, the implications of this structure were not fleshed out, and the illustration was omitted from the later journal paper [8] for reasons of space. Since then, it has caused difficulties both in proofs [2], and in algorithmic analysis of recent parallel approaches [9].

We refer to this as a *W structure* since it consists of a horizontal zigzag of edges (or paths) in the contour tree. In 2D, these can be constructed as a sequence of nested

volcanic caldera. Similarly, in 3D, nested shells alternating between minima and maxima will also display this behaviour. However, once boundary conditions are taken into account, an alternating sequence of ridges and valleys stretching across the data set will also result in a W structure.

Once we realize the impact of boundary conditions, it is easy to construct W structures with any desired complexity, as illustrated in Figure 3. Here, we start with alternating ridges and valleys. Next, we insert saddle points in each ridge and valley to divide them into multiple extrema each. Finally, we assign values to each location, making sure we stay consistent with the assignment of saddle points. As a result, we see a horizontal zigzag emerge in the contour tree, and is clear that we can use this construction to make arbitrarily complex W structures.

We note that there are many variations possible. For example, we have chosen to make all downwards saddles lower than all upwards saddles. While this is not necessary, it is easy to enforce. Similarly, the exact ordering along the ridges and valleys can be altered: in larger examples, we can have multiple extrema for each. We have also chosen to place the saddles in the middle - there are boundary effects when they are at the edge of the data. Moreover, in practice we tend to use diagonal ridges and valleys, in order to pack more features into a small space. But the basic strategy is clear: alternate ridges and valleys generate W-structures, and this gives us useful test cases at any desired scale.

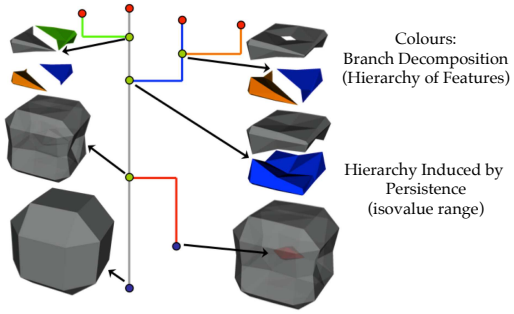
6 Concrete Examples

In addition to the specific examples of flat regions and W structures, we have historically used a range of data sets for testing. While the following list is not exhaustive, it covers a range of data sets we have found useful in practice for test purposes.

The 5b Dataset: This dataset was built around 2000 for testing contour tree construction. The intent was to pack the maximum number of topological features into the smallest possible space. As a result, it has shown up in a number of papers. It was constructed by electing to have two minima, one interior and one exterior, and four maxima arranged in pairs with toroidal isosurfaces nested around them. Initially, this data set was constructed as a $3 \times 3 \times 3$ grid, but was embedded in a layer of 0s - as a result, it is a $5 \times 5 \times 5$ grid, as shown in Figure 4.

Volvis Data: We also use the volvis data repository for testing, in particular the fuel dataset, which has around 100 critical points in a $64 \times 64 \times 64$ data set, although with more maxima than minima. The hydrogen dataset has also proved useful, as several of the features of interest form flat regions in the data, which is a useful test of the simulation of simplicity used to guarantee topological properties.

Protein Data Base: Another source of test data is the PDB protein data base (www.wwpdb.org). One of the utilities from this project generates sampled electrostatic potential fields at any desired resolution, and these tend to have many topological events occurring at bonds between atoms. One dataset from this source was



Plane z=0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Plane z=1

0	0	0	0	0
0	99	90	85	0
0	95	80	95	0
0	85	95	99	0
0	0	0	0	0

Plane z=2

0	0	0	0	0
0	75	50	65	0
0	55	15	45	0
0	60	40	70	0
0	0	0	0	0

Plane z=3

0	0	0	0	0
0	97	87	82	0
0	92	77	92	0
0	82	87	97	0
0	0	0	0	0

Plane z=4

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Fig. 4 5b: a synthetic $5 \times 5 \times 5$ dataset, constructed to have 4 maxima, 2 minima, 3 connectivity-critical points and 5 additional Morse critical points in the central $3 \times 3 \times 3$ block.

therefore instrumental in trapping a typographical error in a lookup table that caused the same surface to be extracted over a hundred times instead of once.

GTOPO30 Data: We have found that terrain data to be valuable for testing for several reasons. First, terrain data is defined in two dimensions, not three, and therefore tends to be more useful for small scale testing. Secondly, it is naturally self-similar, so interesting topology occurs throughout much of the world. Thirdly, thanks to US government policy, it is freely available from the US Geographical Survey, in particular the 30 arc-second GTOPO30 dataset (lta.cr.usgs.gov/GTOPO30).

We have found that a small section of terrain around Vancouver captures 40 odd topological events in about 400 data values, which is still feasible to verify manually. For scaling studies, sections of the Canadian Rockies proved exceptionally useful: they naturally give rise to W structures due to the parallel nature of mountain ranges. Finally, a section of low-relief terrain from the Canadian Shield tests both the handling of W structures and of clumped data values. This is not to say that these are ideal choices: merely that they have been good tests in the past.

Nasty W During the development of the most recent contour tree algorithm, W structures caused particular problems, and we therefore a number of small examples, which led us to the construction described above. These were crucial in constructing valid parallel algorithms, and had the side effect of developing our understanding of this pathological case. Eventually, we constructed an extreme case to track down a particularly nasty bug, simplifying the construction to a single triangle strip for ease of manual debugging. We show a developed version of this, called the "nasty W" in Figure 5, in which every vertex of the mesh is a supernode in the contour tree.

One of the particular values of this example is that the W structure ensures that only two branches are candidates at any given time, with the lower priority one chosen for removal. Here, our priority measure is the "height" of the branch, which is not in fact the persistence of the extremum [16]. The right-hand most superarc $(200-60)$ in the illustration has a priority of 140, higher than the priority (110) left-hand superarc $(230-120)$. The left-hand superarc is therefore removed first, revealing a new superarc $(0-130)$ with priority 130: note that at all stages, the right-hand superarc has the higher priority, so is never removed, and becomes part of the master branch of the decomposition.

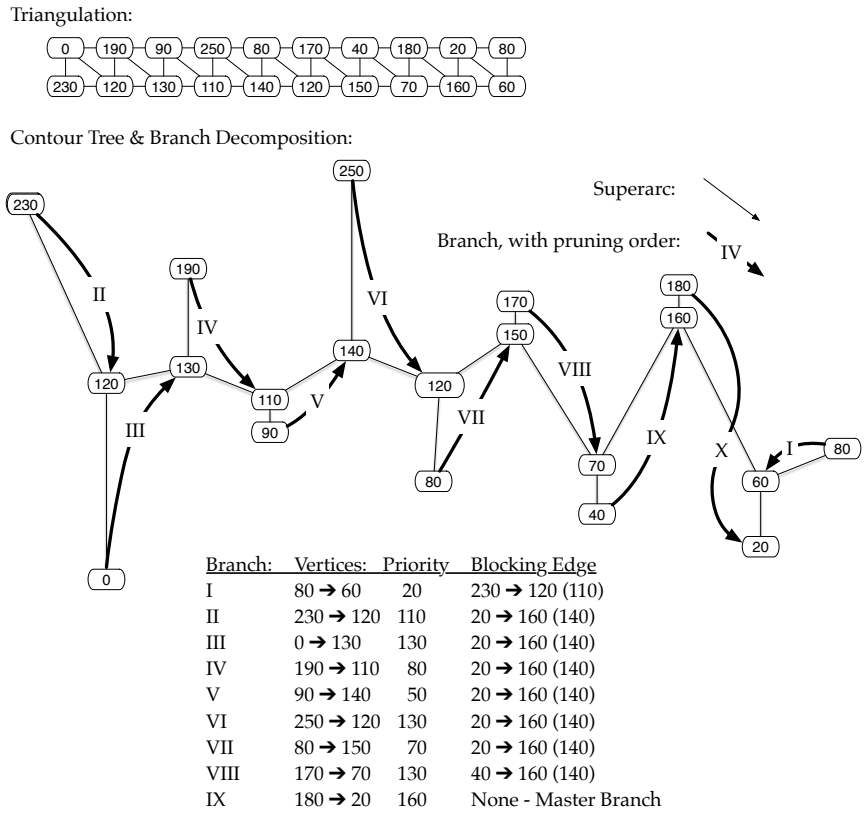
The Nasty W has several interesting properties. First, the master branch includes neither the global minimum nor the global maximum. Second, the global minimum and global maximum do not pair with each other, indirectly demonstrating that branch decomposition and contour tree simplification are not equivalent to persistence. And third, the master branch isn't even the longest monotone path in the tree! As a result, this (and several other W structures) are now part of our standard test suite for working on contour tree algorithms.

One Cube and One Cube Forking While less work has been done on Reeb spaces than on contour trees and Reeb graphs, the same pattern of test case construction was visible. Real data sets were too complex, while the existing mathematical examples were difficult to construct in practical data. As a result, we constructed three small examples of volumetric bivariate meshes, all employing 6 tetrahedra packed into a single cube with a shared major diagonal (i.e. a Freudenthal subdivision).

Here, the goal was not to maximize the complexity (at least for our first example), but rather to develop a small example with non-trivial Reeb space in order to assist in our own understanding of fiber topology. After several years of limited progress based on combinations of polynomial functions, this example was developed with a small test harness and immediately led to fiber surfaces, then played a role in algorithm development for Reeb spaces CGT15. It is therefore recommended both as a first tiny data set for testing correctness, but also as an example for the process of learning and reasoning about Reeb spaces and fiber topology.

This example followed a standard approach in fiber topology, by using a linear ramp for one of the functions, and choosing the first so that the contour tree on one face is an upwards fork, while the contour tree on the opposite face is a downwards fork. In its earliest incarnation in 2002, it was used by the first author to reason about time-varying contour trees. It then became an example used to explain Reeb Spaces to students: when the time came to construct a small data set for Reeb Space computation, it was natural to embody it as a small tetrahedral mesh.

Due to the tetrahedralization chosen, the result is slightly different. However, the downward fork is recognizable as two flaps (the white regions on the left), while the upward fork became another two (the white regions on the right). In the middle gray region, only one fiber exists, represented by gluing the two partial Reeb spaces together. We constructed paper models of this (and our other examples), and it was these models that led us to construct fiber surfaces [5].



Branch:	Vertices:	Priority	Blocking Edge
I	80 ➔ 60	20	230 ➔ 120 (110)
II	230 ➔ 120	110	20 ➔ 160 (140)
III	0 ➔ 130	130	20 ➔ 160 (140)
IV	190 ➔ 110	80	20 ➔ 160 (140)
V	90 ➔ 140	50	20 ➔ 160 (140)
VI	250 ➔ 120	130	20 ➔ 160 (140)
VII	80 ➔ 150	70	20 ➔ 160 (140)
VIII	170 ➔ 70	130	40 ➔ 160 (140)
IX	180 ➔ 20	160	None - Master Branch

Fig. 5 The Nasty W test example. The contour tree was constructed first, then the saddles arranged along one row of the mesh, the extrema along the other row.

We then constructed the second example, in Figure 7, by replacing the linear ramp with a rotated copy of the first function. Here there are up to three fiber components for any given value. Moreover, two "vertices" of the central square are not vertices of the mesh, but intersections of the projections of the mesh edges. This property showed us that certain algorithmic lines of attack would be unfruitful.

Finally, since all of the fibers in these examples were open at the boundary, we constructed a third to have closed loop fibers around the main diagonal of the cube. This was less useful than we had hoped, but still helped us to think about vertices and tetrahedra that overlapped in projection. We therefore omit this example.

These examples are small enough to print out and assemble manually, and we have found them very useful for comprehension and for debugging.

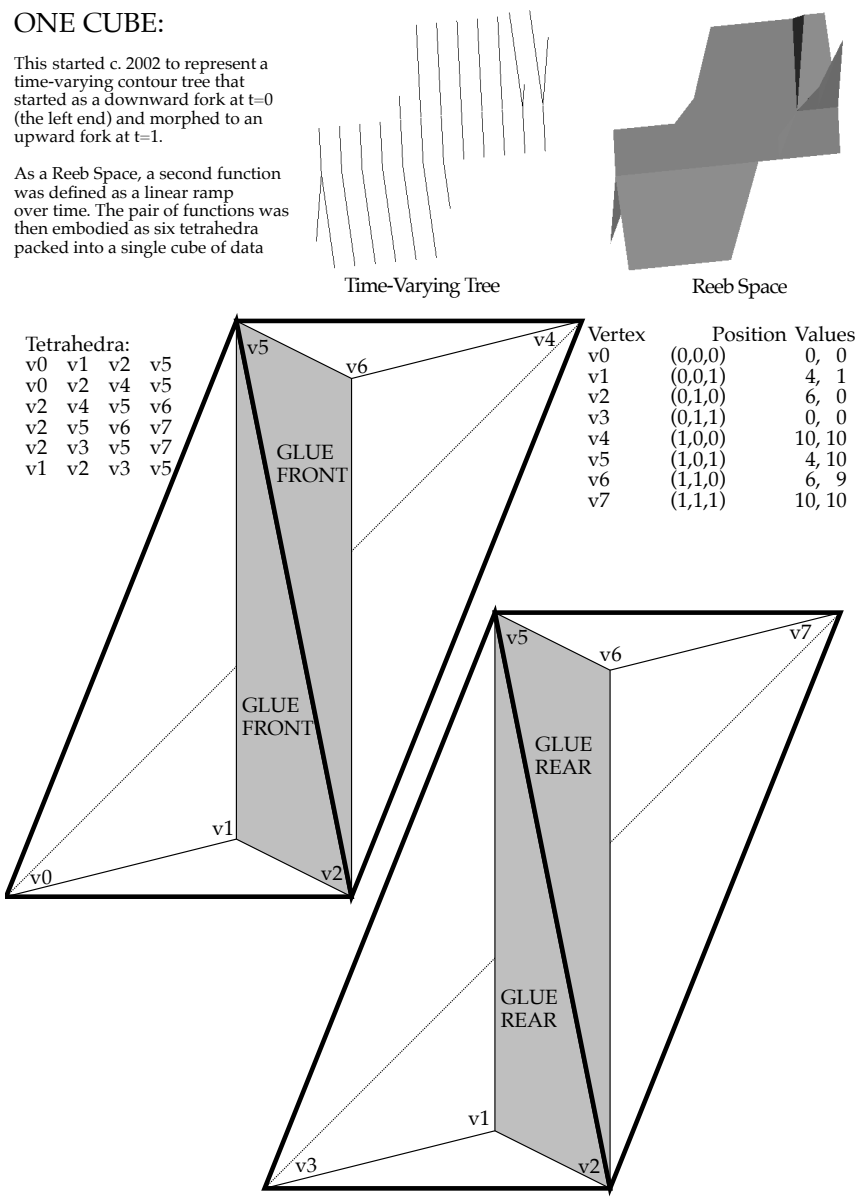


Fig. 6 A small Reeb space constructed from six tetrahedra packed into a single cube in the domain with shared edge v2v5.

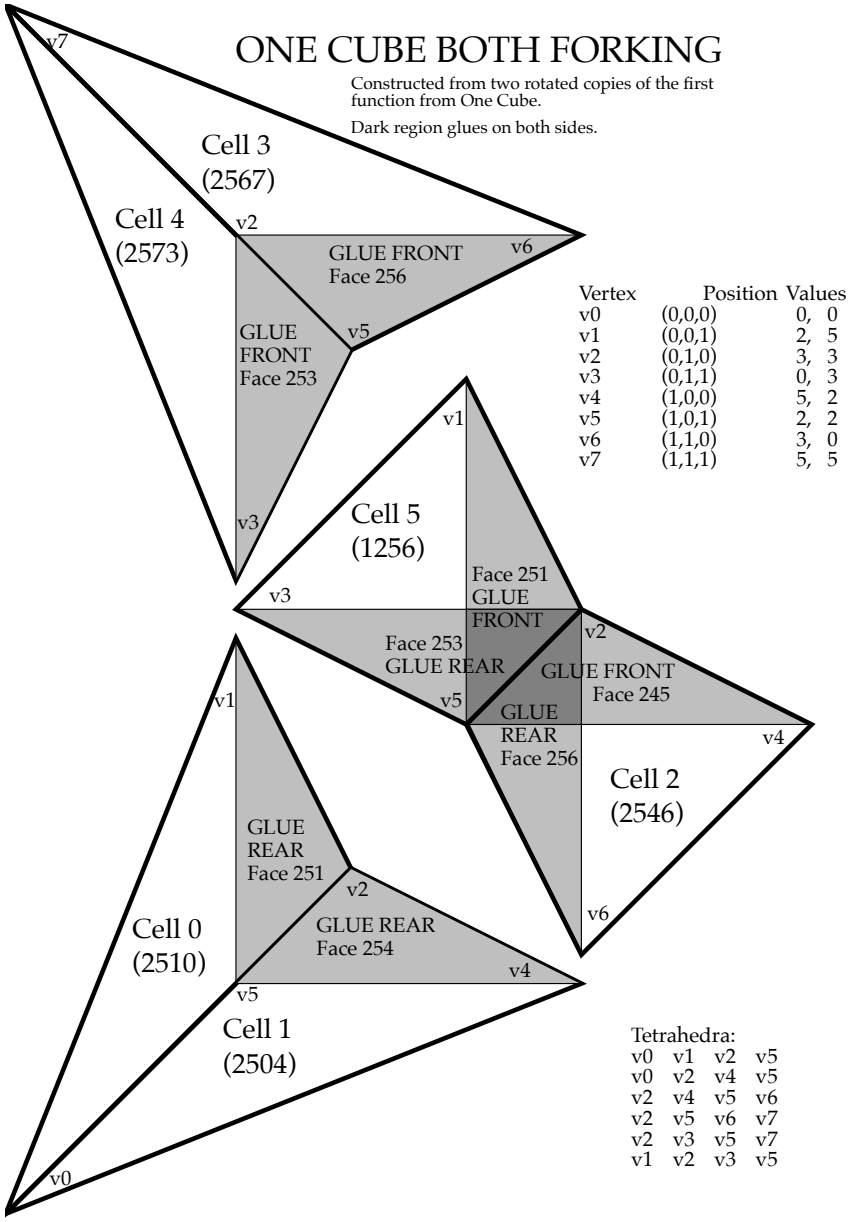


Fig. 7 A second Reeb space constructed from six tetrahedra packed into a single cube. Here, the first field is the same as in Figure 6, while the second as a copy of the first field, rotated by 90 degrees.

7 Debug Tricks

Once suitable test data is available, development and debugging can proceed. And here, too, experience indicates that a general debug procedure needs some modification to accommodate the demands of topological computation. We have noticed three basic tricks that we tend to repeat in different contexts:

Text Output: As with all code, text output is particularly useful. Part of this is because the internal structures are rarely intuitive. As a result, having a consistent well-formatted text output is priceless for tracking down bugs. Moreover, when improving an existing algorithm, we have found that sharing a common output format between versions makes it easier to identify where bugs occur, simply by using the command line tool 'diff' to the outputs of the versions. This technique is particularly valuable in practice as data scales: for example, this allowed us to validate new parallel algorithms [9] against old serial code [8].

Visual Output: Since our target is to visualize data, we normally operate in an environment where visual output is feasible. One approach is to export meshes at various stages of an algorithm, then use an external program such as paraview or TTK to inspect them. Another approach is to build a custom application, either to support the debug process, to play with particular ideas, or to illustrate the process for others. One example of this with two-dimensional data is to render a terrain with the contour tree superimposed: since the (x,y) and h coordinates are known, this means that visual inspection of relationships is straightforward at smaller scales, although difficult when many topological events occur.

Over time, the desire to have intermediate visual output was part of the motivation for the development of the Topology Tool Kit (TTK) by the second author [23], and readers may find its features valuable.

Graph Output: Since scalar Reeb analysis results in graph-like structures, one of the most useful debug tricks is to export the internal data structures to a graph format such as graphviz [13], then to invoke external programs such as dot to generate PDFs and display them. This is a variant of the stepping method described by Laramee [18] which we have found useful, especially for small test sets.

At one stage of developing a new parallel algorithm [9], debug involved manual cross-checking of a contour tree with over 1000 nodes, shown in Figure 8, which involved using a GUI-based graph editor on the dot format output. Most recently, improvements to internal data structures have been simplified considerably by graph outputs that show all of the internal cross-linked pointers (Figure 9), using colour-coding to show which vertices are processed in which iteration.

None of these techniques is unprecedented in general algorithmic procedure. However, simple inspection of data structures in memory is particularly difficult with topological code, so secondary routines such as those described are strongly recommended to accelerate the debug process, and we now ask at an early stage what debug visualizations we will need.

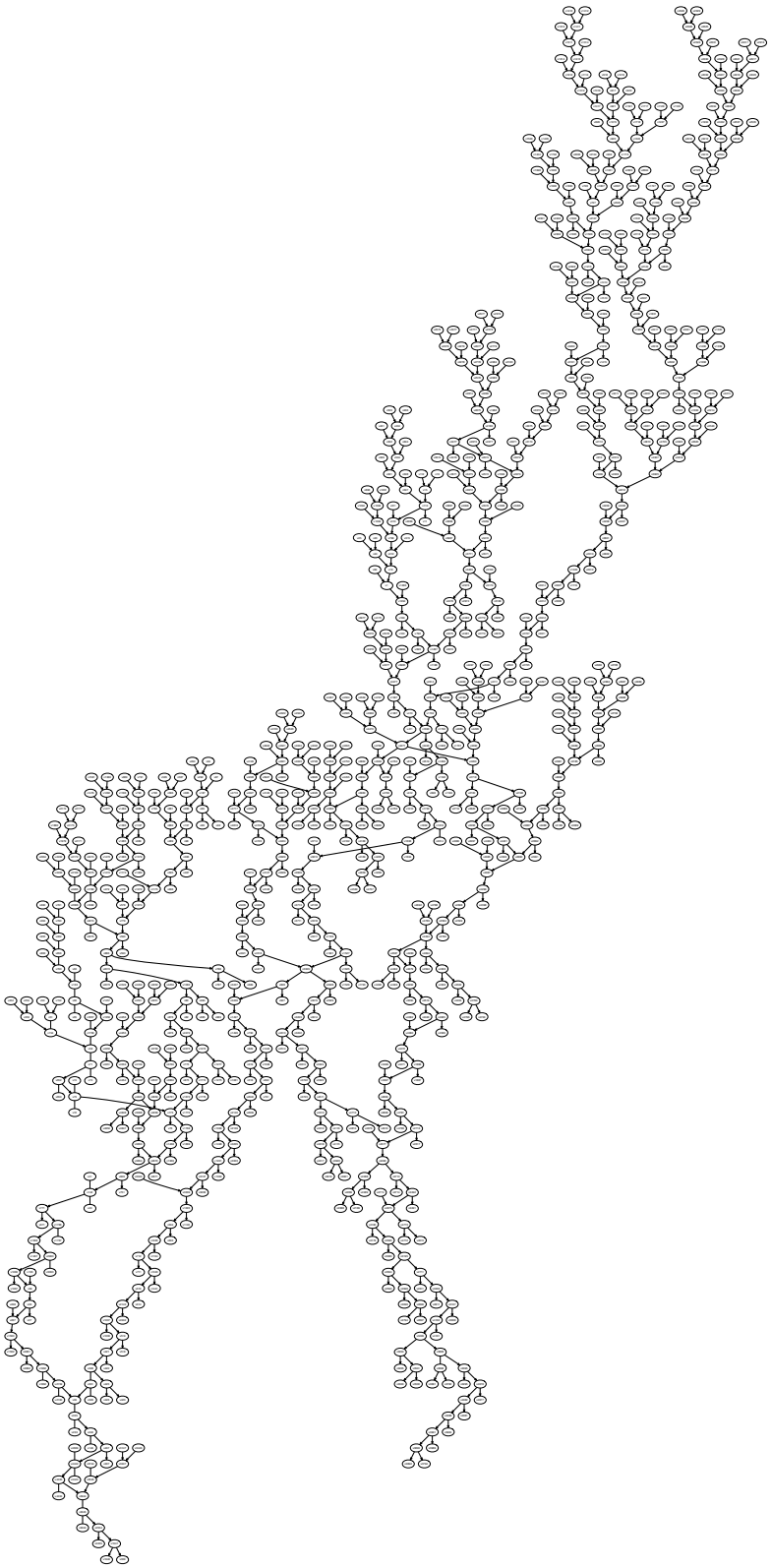


Fig. 8 A large contour tree visualized using graphviz for debug purposes. One edge had been mis-computed, and had to be identified manually.

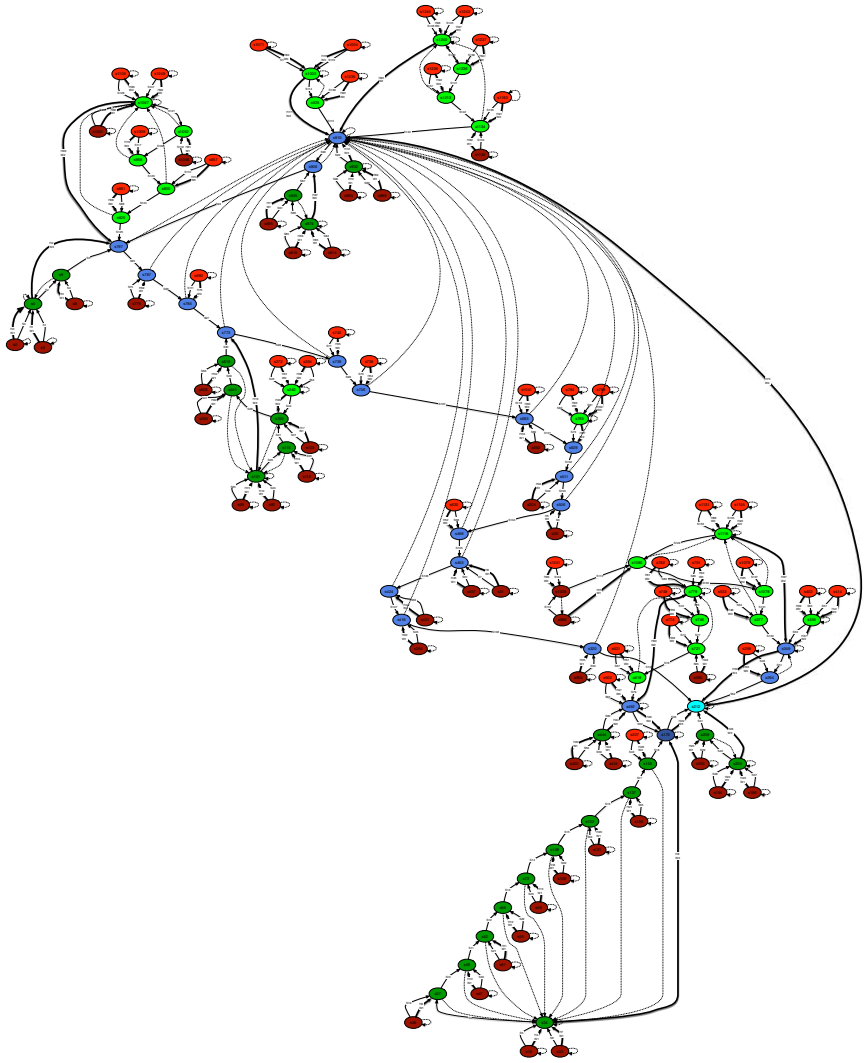


Fig. 9 Contour tree of a subset from Figure 8, with additional pointers, and colour-coding for the iterations of an algorithm.

8 Conclusions

In this paper, we have attempted to report on a crucial phase in algorithmic development for computational topology: selection of suitable test cases and debug procedure. As is apparent above, we have found that the skills of counter-example construction, and the consideration of pathological cases, have given the greatest insight into the mathematics, into our algorithms, and into the debug process.

We have already started work on the theoretical implications of the W-structure, and intend to report on it due course [16]. In the ideal case, we would also use this understanding to resolve the algorithmic implications, but these are non-trivial, and must wait until the current parallel work is fully reported.

Equally, we would encourage our colleagues to report on their test data and strategies, as these are crucial to developing modern topological algorithms, but are currently communicated by word of mouth, if at all. We note that, with the possible exception of the 5b data set, no standard benchmarks yet exist for topological algorithms, and suggest that this may be a fruitful direction for the community.

References

1. ALLILI, M., CORRIVEAU, D., DERIVIÈRE, S., KACZYNSKI, T., AND TRAHAN, A. Discrete dynamical system framework for construction of connections between critical regions in lattice height data. *Journal of Mathematical Imaging and Vision 28*, 2 (Jun 2007), 99–111.
2. ARGE, L. Personal communication to H. Carr.
3. BOYELL, R. L., AND RUSTON, H. Hybrid Techniques for Real-time Radar Simulation. In *IEEE 1963 Fall Joint Computer Conference* (1963), pp. 445–458.
4. CARR, H., AND DUKE, D. Joint Contour Nets. *IEEE Transactions on Visualization and Computer Graphics 20*, 8 (2014), 1100–1113.
5. CARR, H., GENG, Z., TIERNY, J., CHATTOPADHYAY, A., AND KNOLL, A. Fiber Surfaces: Generalizing Isosurfaces to Bivariate Data. *Computer Graphics Forum 34*, 3 (2015), 241–250.
6. CARR, H., SNOEYINK, J., AND AXEN, U. Computing Contour Trees in All Dimensions. *Computational Geometry: Theory and Applications 24*, 2 (2003), 75–94.
7. CARR, H., SNOEYINK, J., AND VAN DE PANNE, M. Simplifying Flexible Isosurfaces with Local Geometric Measures. In *IEEE Visualization* (2004), pp. 497–504.
8. CARR, H., SNOEYINK, J., AND VAN DE PANNE, M. Flexible Isosurfaces: Simplifying and Displaying Scalar Topology Using the Contour Tree. *Computational Geometry: Theory and Applications 43*, 1 (2010), 42–58.
9. CARR, H., WEBER, G., SEWELL, C., AND AHRENS, J. Parallel Peak Pruning for Scalable SMP Contour Tree Computation. In *IEEE Large Data Analysis and Visualization (LDAV)* (2016).
10. COX, J., KARRON, D., AND FERDOUS, N. Topological zone organization of scalar volume data. *Journal of Mathematical Imaging and Vision 18*, 2 (2003), 95–117.
11. EDELSBRUNNER, H., HARER, J., AND PATEL, A. K. Reeb Spaces of Piecewise Linear Mappings. In *ACM Symposium on Computational Geometry* (2008), pp. 242–250.
12. EDELSBRUNNER, H., AND MÜCKE, E. P. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics 9*, 1 (1990), 66–104.
13. ELLSON, J., GANSNER, E., KOUTSOFIOS, L., NORTH, S. C., AND WOODHULL, G. Graphviz—Open Source Graph Drawing Tools. In *International Symposium on Graph Drawing* (2001), Springer, pp. 483–484.
14. GOLD, C., AND CORMACK, S. Spatially Ordered Networks and Topographic Reconstruction. In *ACM Symposium on Spatial Data Handling* (1986), pp. 74–85.
15. HELMAN, J., AND HESSELINK, L. Representation and Display of Vector Field Topology in Fluid Flow Data Sets. *Computer* (1989), 27–36.
16. HRISTOV, P., AND CARR, H. W-Structures in Contour Trees. In preparation.
17. KACZYNSKI, T. Multivalued maps as a tool in modeling and rigorous numerics. *Journal of Fixed Point Theory and Applications 4*, 2 (Dec 2008), 151–176.

18. LARAMEE, R. Using Visualization to Debug Visualization Software. *IEEE Computer Graphics and Applications*, 6 (2009), 67–73.
19. REEB, G. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus de l'Acadèmie des Sciences de Paris 222* (1946), 847–849.
20. SFB 382 OF THE GERMAN RESEARCH COUNCIL (DFG). Hydrogen Atom, available at http://schorsch.efi.fh-nuernberg.de/data/volume/.
21. THEISEL, H. Designing 2D Vector Fields of Arbitrary Topology. *Computer Graphics Forum 21*, 3 (2002), 595–604.
22. TIERNY, J., AND CARR, H. Jacobi Fiber Surfaces for Bivariate Reeb Space Computation. *IEEE Transactions on Visualization & Computer Graphics 1* (2017), 960–969.
23. TIERNY, J., FAVELIER, G., LEVINE, J. A., GUEUNET, C., AND MICHAUX, M. The Topology ToolKit. *IEEE Transactions on Visualization & Computer Graphics 24*, 1 (2018), 832–842.
24. TIERNY, J., GYULASSY, A., SIMON, E., AND PASCUCCI, V. Loop Surgery for Volumetric Meshes: Reeb Graphs Reduced to Contour Trees. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2010), 1177–1184.
25. WEBER, G. H., SCHEUERMANN, G., AND HAMANN, B. Detecting critical regions in scalar fields. In *Visualization Symposium (VisSym)* (2003), EUROGRAPHICS and IEEE TCVG.
26. ZHANG, E., MISCHAIKOW, K., AND TURK, G. Vector Field Design on Surfaces. *ACM Transactions on Graphics 25*, 4 (2006), 1294–1326.