Propagate and Pair: A Single-Pass Approach to Critical Point Pairing in Reeb Graphs

Preprint · July 2018
DOI: 10.13140/RG.2.2.31490.79049

3 authors:

Junyi Tu
University of South Florida
11 PUBLICATIONS **109** CITATIONS
SEE PROFILE

Mustafa Hajij
The Ohio State University
29 PUBLICATIONS **39** CITATIONS
SEE PROFILE

Paul Rosen
University of South Florida
70 PUBLICATIONS **393** CITATIONS
SEE PROFILE

Propagate and Pair: A Single-Pass Approach to Critical Point Pairing in Reeb Graphs

Junyi Tu, Mustafa Hajij, Paul Rosen

Abstract—With the popularization of Topological Data Analysis, the Reeb graph, has found new applications as a summarization technique in the analysis and visualization of large and complex data. The Reeb graph provides a topological summary for a space, whose usefulness extends beyond just the graph itself. Pairing critical points enables forming topological fingerprints, often represented as persistence diagrams, that provides important insights into the structure and noise in the data. Although the body of work addressing the efficient calculation of Reeb graphs is large, the literature on pairing is limited. In this paper, we discuss two algorithmic approaches for the pairing of critical points in Reeb graphs. We first discuss a multipass approach that separates pairing for essential and non-essential critical points. Next, we introduce a new algorithm, called Propagate and Pair, that efficiently pairs all critical points in a single-pass. The algorithm is also general enough to support pairing critical points in Reeb graph approximation (e.g. Mapper), non-looping variants of Reeb graphs, such as contour trees, as well as split and join trees.

Index Terms—Reeb graph, Topological Data Analysis, Critical point pairing, Persistence diagram

1 Introduction

The last two decades have witnessed great advances in methods that rely on topological techniques to analyze and study data in a process known as Topological Data Analysis (TDA). The popularity of topology-based techniques is due in large part to their robustness and their applicability to a wide variety of datasets and scientific domains [24]. The *Reeb graph* [29] was originally proposed as a data structure to encode the geometric skeleton of 3D objects, but recently it has been re-purposed as an important tool in TDA.

The Reeb graph of a scalar function defined on a domain gives a topological summary of that domain. The Reeb graph encodes the evolution of the connectivity of the level sets induced by a scalar function defined on the domain by sweeping from negative infinity to positive infinity and tracking the birth and death of the connected components of the level sets. Beside their usefulness in handling large data [17], Reeb graphs and their non-looping variant, contour trees [7], have been successfully used in feature detection [35], data reduction and simplification [9,31], image processing [23], shape understanding [2], visualization of isosurfaces [3] and many other applications. One challenge with using Reeb graphs to directly visualize large data is that the graph may still be too large or complex to directly analyze, therefore requiring further abstraction.

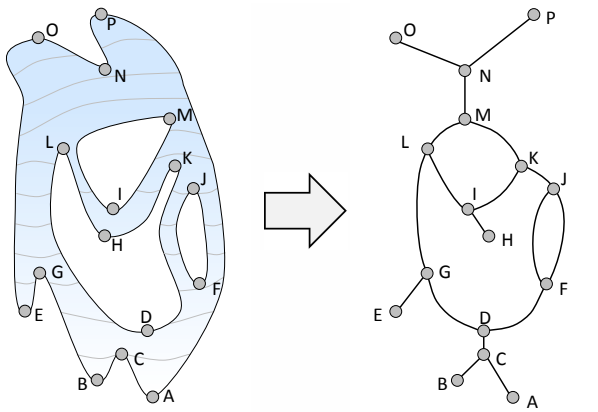
A fundamental tool in TDA is persistent homology, introduced by Edelsbrunner et al. [18]. Typically, persistent homology operates by transforming a point cloud data into a filtration (a nested sequence of spaces), performing persistent homology computation on the filtration, and parameterizing the obtained topological structures by their life-time in the filtration. As a result, persistent homology gives a topological description. This topological description is usually called the *persistence diagram*. The notion of persistence can be applied to any act of birth that is paired with an act of death. Since the Reeb graph encodes the birth and the death of the connected components of the iso-contours of a scalar function, the notion of persistence can be applied to pair the critical points in the Reeb graph [1].

While many algorithms have focused on the efficient calculation of Reeb graph structures themselves, few have described algorithms for pairing critical points. Although the critical point pairing problem itself is not a large data problem, it is a critical component in the analysis of large data. As seen in Figure 1, the dataset, which may be very high resolution, produces a much smaller Reeb graph. After that, the critical points are paired, and a persistence diagram displays the data, as seen in Figure 2. This final step can still be challenging, particularly when considering *essential critical points*—those critical points associated with loops in the Reeb graph. These require a complicated search that needs to be performed on each critical point.

In this paper, our contribution is the description and implementation of two efficient algorithms to directly compute persistence diagrams from Reeb graphs. Our first algorithm uses an 2-pass approach for non-essential critical point pairing that relies upon branch decomposition on a join and split tree. We then introduce an algorithm for pairing essential critical points, also based upon join trees. Finally, this leads to our second approach, a new single-pass algorithm for pairing both non-essential and essential critical points efficiently in Reeb graphs, Mapper graphs, contour trees, split trees, and join trees.

2 Reeb Graph

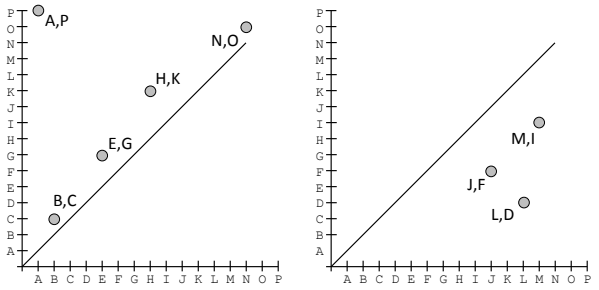
Let X be a triangulable topological space, and let $f : X \to \mathbb{R}$ be a continuous function defined on it. We define an equivalence relation $\sim$ on X, such that $x \sim y$, if and only if x and y belongs to the same connected component of $f^{-1}(r)$ for some $r \in \mathbb{R}$. The *Reeb graph* of the space X



(a) Data with a scalar function (b) Reeb graph of the data

Fig. 1. Topological data analysis and visualization pipeline using Reeb graphs shows (a) data being processed into (b) a Reeb graph. Using the Reeb graph, critical points are then paired, and a persistence diagram, as shown in Figure 2, is used to visualize the structure of the data.

- *Junyi Tu is with the University of South Florida. E-mail: junyi@mail.usf.edu.*
- *Mustafa Hajij is with the University of South Florida. E-mail: mhajij@usf.edu.*
- *Paul Rosen is with the University of South Florida. E-mail: prosen@usf.edu.*



(a) Persistence diagram (b) Extended persistence diagram

Fig. 2. (a) The persistence diagram $\mathrm{Dg}_0(f)$ and (b) extended persistence diagram $\mathrm{ExDg}_1(f)$ for the Reeb graph in Figure 1(b) provide a visual abstraction of the structures in the original data.

and the function $f : X \to \mathbb{R}$, denoted by $R_f(X)$, is the quotient space $X/\sim$ equipped with the quotient topology induced by the quotient map $\pi : X \to R_f(X)$. When X is clear from the context, we will denote the Reeb graph simply by R_f. Under some regularity conditions (which we shall define later) the Reeb graph has the structure of a CW complex.

The input function $f : X \to \mathbb{R}$ also induces a continuous function $\tilde{f} : R_f \to \mathbb{R}$ defined as $\tilde{f}(\tilde{x}) = f(x) = r$ for any preimage $x \in f^{-1}(r)$. Usually we plot the Reeb graph with the vertical coordinate of a point z as the function value $f(z)$. The Reeb graph can be thought of as a topological summary of the space X using the information encoded by the scalar function f. More precisely, the Reeb graph encodes the changes that occur to connected components of the level sets of $f^{-1}(r)$ as r goes from negative infinity to positive infinity. Figure 1 (a) and (b) shows an example of a Reeb graph defined on a surface.

The function $\tilde{f}$ can be used to classify points on the Reeb graph as follows. Let x be a point in R_f. The *up-degree* of x is number of branches (1-cells) incident to x that have higher values of $\tilde{f}$ than x. The down-degree is of x is defined similarity. A point x on R_f is said to be *regular* if its up-degree and down-degree are equal to one. Otherwise it is a *critical point*. A critical point on the Reeb graph is also a *node* of the Reeb graph. A critical point is called a minimum if its down-degree is equal to 0. Symmetrically, a critical point is said a maximum if its up-degree is equal to 0. Finally, a critical point is said to be a down-fork (up-fork) if its down-degree (up-degree) is larger than 1.

Without loss of generality we assume that Reeb graph is a single connected component, and all nodes on the Reeb graph have different function values. Moreover, we will assume also that every node in the Reeb graph is either a minimum, a maximum, a down-fork with down-degree 2, or an up-fork with up-degree 2. This is not a restriction to the general case since a Reeb graph that does not satisfy these conditions can be conditioned to fit them, as we will show in Section 4.

Regularity Condition on Reeb Graphs. Let $a \in \mathbb{R}$. We call $X_{\leq a} = \{x \in X \mid f(x) \leq a\}$ a *sublevel set* of f. Similarly, we call $X_{\geq a} = \{x \in X \mid f(x) \geq a\}$ a *superlevel set* of f. Let $H_p(X)$ denote the p-th homology group of the triangulable topological space X. In this paper we consider homology with coefficients in a finite field, so $H_p(X)$ is a vector space.

To ensure that R_f has the structure of a CW complex we need to assume that the scalar function f is *tame* in the following sense [4]. Let $\{a_1, ..., a_n\}$ be a finite sequence of real numbers such that the following conditions are satisfied:

1. $-\infty < min(f) = a_1 < \cdots < a_n = max(f) < \infty$ such that for all $i < n$ and $s,t \in [a_i, a_{i+1})$ with $s < t$, the homomorphism $H_p(X_{\leq s}) \longrightarrow H_p(X_{\leq t})$ induced by the natural inclusion $X_{\leq s} \hookrightarrow X_{\leq t}$ is an isomorphism.

2. In the same way, for all $s,t \in (a_i, a_{i+1}]$ with $s < t$, the homomorphism $H_p(X_{\geq t}) \longrightarrow H_p(X_{\geq s})$ induced by the inclusion $X_{\geq t} \hookrightarrow X_{\geq s}$ is an isomorphism.

3. $H_p(X_{\leq a_i}) < \infty$ for all i.

The first conditions insures that the topological changes that occur to a sublevel set $X_{\leq s}$ only occur as s goes through a_i for some i. Similarly, the second condition insures that topological changes occur to a superlevel set $X_{\geq s}$ when s passes through a_i for some i. The last condition insures that the rank of the homology group is finite at a_i for all i. All scalar functions in this paper will be assumed to be tame.

2.1 Persistent Homology and Persistence Diagram

The notion of persistent homology was originally introduced by Edelsbrunner et al. [18]. Here we present the theoretical setting for the computation of the persistence diagram associated with a scalar function defined on a triangulated topological space. We then show how this is related to the persistent pairing on Reeb graphs. We start by presenting a concise description of persistent homology. For more details the reader is referred to [18].

Consider the following sequence of vector spaces,

$$0 = H_p(X_0) \to H_p(X_1) \to \cdots \to H_p(X_n) = H_p(X), \qquad (1)$$

where $X_i = X_{\leq a_i}$ and each homomorphism $g_i^{i+1} : H(X_i) \to H(X_{i+1})$ on the homology groups is induced by the inclusion $X_i \hookrightarrow X_{i+1}$. We can define $g_i^j : H(X_i) \to H(X_j)$ for any $i \leq j$ by composition. We say that a class $\alpha \in H(X_\ell)$ is *born at (index) i* if

$$\alpha \in \operatorname{im} g_i^\ell \qquad \textbf{but} \qquad \alpha \notin \operatorname{im} g_{i-1}^\ell.$$

A class α born at index *i dies entering (index) j* if

$$g_i^j(\alpha) \in \operatorname{im} g_{i-1}^j \qquad \textbf{but} \qquad g_i^{j-1}(\alpha) \notin \operatorname{im} g_{i-1}^{j-1}.$$

In this case, the index pair (i, j) is called a *persistence pair*, and the difference $j - i$ is the *(index) persistence* of the pair.

Persistent homology records such birth and death events. In particular, the p-th *ordinary persistence diagram* of f, denoted by $\mathrm{Dg}_p(f)$, is a multiset of pairs (b,d) corresponding to the birth value b and death value d of some p-dimensional homology class. Since the homology $H_p(X)$ may not be trivial in general, any nontrivial homology class of $H_p(X)$, referred to as an *essential homology class*, will never die during the sequence in (1). These events are associated with the cyclic portions of the Reeb graph. By appending a sequence of relative homology groups to (1), we obtain the following sequence :

$$0 = H_p(X_0) \to \cdots \to H_p(X_n) = H_p(X) =$$
$$H_p(X, X_{\geq a_n}) \to H_p(X, X_{\geq a_{n-1}}) \to \cdots \to H_p(X, X_{\geq a_0}) = 0.$$

Since the last vector space $H_p(X, X_{\geq a_0}) = 0$, each essential homology class eventually dies in the *relative part* of the above sequence at some relative homology group $H_p(X, X_{\geq a_j})$. In other words, each essential homology class gets paired with some relative homology class in $H_p(X, X_{\geq a_j})$.

We refer to the multiset of points encoding the birth and death time of pth homology classes created in the ordinary part and destroyed in the relative part of the sequence as the *pth extended persistence diagram* of f, denoted by $\mathrm{ExDg}_p(f)$. In particular, for each point (b,d) in $\mathrm{ExDg}_p(f)$ there is an essential homology class in $H_p(X)$ that is born in $H_p(X_{\leq b})$ and dies at $H_p(X, X_{\geq d})$. Observe that for the extended persistence diagram the birth time b for an essential homology class in $H_p(X_{\leq b})$ is larger than or equal to death time d for the relative homology class in $H_p(X, X_{\geq d})$ that kills it.

2.2 Persistence Diagram of Reeb Graph

Of particular interest to us are the persistence digram $\mathrm{Dg}_0(f)$ and extended persistence diagram $\mathrm{ExDg}_1(f)$. These two diagrams can be computed completely by considering the Reeb graph R_f. We give an intuitive explanation to this fact here, and we refer the reader to [4] for more details.

Note that pairing of critical points of a scalar function can be computed independent of the computation of Reeb graphs. However, the pairing is best described using Reeb graph since the structure of Reeb graph clearly reveals the topological feature associated to the pairing.

Before we describe the points in persistence digram $\mathrm{Dg}_0(f)$ and extended persistence diagram $\mathrm{ExDg}_1(f)$,we need to distinguish between two types of forks in the Reeb graph, namely the ordinary forks and the essential forks. Let R_f be a Reeb graph and let s be a down-fork such that $a = f(s)$. We say that the down-fork s is an *ordinary fork* if the lower branches of s are contained in disjoint connected components C_1 and C_2 of $(R_f)_{<a}$. The down-fork a is said to be *essential* if it is not ordinary. The ordinary and essential up-forks are defined similarly.

We now demonstrate the meaning of the persistence digram $\mathrm{Dg}_0(f)$ and extended persistence diagram $\mathrm{ExDg}_1(f)$ of Reeb graph.

Branching feature of a Reeb graph. Let $a \in \mathbb{R}$. We consider the changes that occur in $H_0((R_f)_{\leq a})$ as a increases. A connected component of $(R_f)_{\leq a}$ is created when a passes through a minimum of R_f. Let C be a connected component of $(R_f)_{\leq a}$. We say that a local minimum a of R_f *creates* C if a is the global minimum of C.

Every ordinary up-fork is paired with a local minimum to form one point in the persistence diagram $\mathrm{Dg}_0(f)$ as follows. Let s be an ordinary down-fork with $f(a) = s$ and let C_1 and C_2 be the connected components of $(R_f)_{<a}$. Let x_1 and x_2 be the creators of C_1 and C_2. Without loss of generality we assume that $f(x_1) < f(x_2)$. The homology class $[x_2]$ that is created at $f(x_2)$ and dies at $f(s)$ gives rise to a point (x_2, s) in the ordinary persistence diagram $\mathrm{Dg}_0(f)$. Note that such a pair occurs when the minimum is a branch in the Reeb graph, hence we name it *branching feature*.

Cycle feature of a Reeb graph. Let s be an essential down-fork. We call the down-fork s is a creator of a 1-cycle in the sublevel set $(R_f)_{\leq a}$. As shown in [1], s will be paired with an essential up-fork s' to form an *essential pair* (s', s), and hence a point (s', s) in the extended persistence diagram $\mathrm{ExDg}_1(f)$. The essential up-fork s' is determined as follows. Let Γ_s be the set of all cycles born at s and each cycle corresponds to a loop in the Reeb graph R_f. Let γ_s be an element of Γ_s with largest minimum value of f among these cycles born at s. The point s' is the point at which the function f achieves this minimum on the cycle γ_s.

3 Related Work

The first algorithm to compute Reeb graph on a triangulated surface was presented by Shinagawa and Kunii [32], with time complexity $O(n^2)$, where n is the number of triangles in the mesh. The efficient computation of Reeb graphs has been an active research topic for last two decades. Cole-McLaughlin et al. [11] improved the performance to $O(n\log(n))$. Pascucci et al. [28] presented an online method to computes Reeb graphs. Harvey et al. [21] deployed a randomized algorithm to compute Reeb graph on arbitrary simplicial complexes K in expected time $O(mlog(n))$, where m is the size of 2-skeleton of K (i.e., the total number of vertices, edges, and triangles), and n is the number of vertices. For the application of Reeb graphs, Hilaga et al. [22] provide a Multi-resolution Reeb Graph (MRG) representation of triangle meshes which is independent of rotation in topology matching.

By reducing the Reeb graph to contour tree via loop surgery, Tierny et al. [36] presented an algorithm to compute Reeb graph on a volumetric mesh in $\mathbb{R}^3$. The work by Doraiswamy and Natarajan [16] utilizes the union of contour trees to compute the Reeb graph. Other Reeb graph algorithms can be found in [14, 15, 26].

When Reeb graph is acyclic, it is also known as contour tree. Carr et al. [8] produced the first well-known algorithms for computing contour trees. Contour trees are used in volume rendering [38] and noise removal, while retaining important features in data [31].

Reeb graphs and contour trees have found numerous applications in graphics and visualization including data skeletonization [19], locus cut [12], data abstraction [25], retrieving topological information from point data, such as homology group computation [10, 13], volume rendering [39], and terrain applications [5, 20].

Branch decomposition was first used to provide a multiscale view of contour trees [27]. This provides the framework for pairing non-essential critical points in a Reeb graph. The first known description to pair critical points of a Morse function on a 2-manifold, including essential critical points, is given in [1]. However, the description is high level with no specific algorithm provided. Similar description of persistence pairing algorithm is also seen in [4].

Pairing of critical points of a scalar function has found multiple applications including segmentation of deformable shapes [34], hierarchical shape segmentation [30], description of protein shape [40], automatic extraction of surface structures [37], and 3D shape description and matching [6].

To the best of our knowledge, this paper is the first systematic development and implementation of two intuitive and efficient algorithms to pair the nodes of Reeb graphs by persistent features.

4 Conditioning the Graph

As mentioned in Section 2.2, our approach assumes that all point in the Reeb graph are either a minimum, maximum, up-forks with up-degree 2, or down-forks with down-degree 2. Fortunately, graphs that do not abide by these requirements can be conditioned to fit them. We define the $J+K$ degree of a node as the J up-degree and K down-degree. New nodes needed during processing are created with ε offset, where ε is a very small number.

There are 4 node conditions to be corrected:

- **1+1 nodes**: Nodes with both 1 up- and down-degree are non-critical. Therefore, they only need to be removed from the graph. This is done by removing the non-critical point and connecting the nodes above and below, as seen in Figure 3(a).

- **0+2 (and 2+0) nodes**: Nodes with 0 up-degree and 2 down-degree are degenerate maximum node, in that they are both down-fork and local maximum. As shown in Figure 3(b), this condition is corrected by added a new node for the local maximum ε higher value. This type of node does not usually occur in Reeb graphs, but it can occur in approximations of a Reeb graph, such as Mapper [33].

- **2+2 nodes**: Nodes with both 2 up- and down-degree are degenerate double forks, both down-fork and up-fork. Figure 3(c) shows how double forks can be corrected by splitting into 2 nodes ε distance apart.

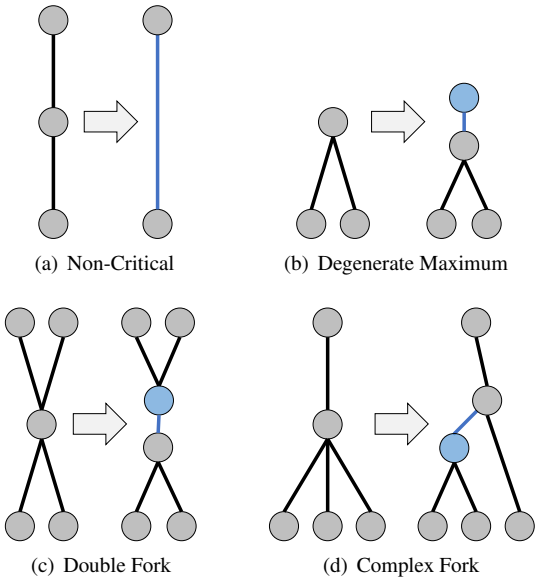


Fig. 3. Before pairing, the nodes of Reeb graph must be properly conditioned. There are 4 node configurations that require conditioning. New nodes and edges are shown in blue.

- **1+ N>2 (and N>2 +1) nodes**: Nodes with down-degree 3 or higher, are complex forks to pair. These are the forks corresponds to degenerate saddles in f, such as monkey saddles. A single critical point pairing to these forks just reduces the degree of down-fork by 1, requiring complicated tracking of pairs. To simplify this, as seen in Figure 3(d), complex forks can be split into 2 forks ε apart. The upper down-fork retains 1 of the original down edges. The new down-fork connects with the old and takes the remaining down-edges. For even higher-order forks, the operation can be repeated on the lower down-fork.

Beyond these requirements, we assume the Reeb graph is a single connected component. If the Reeb graph contains multiple connected components, each one can simply be extracted and processed in parallel.

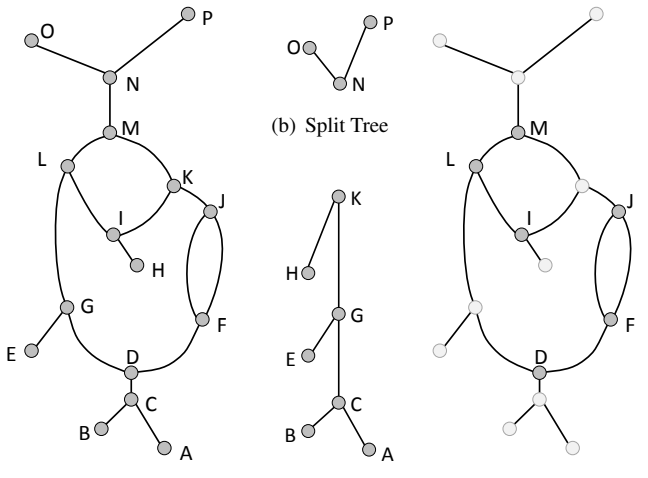
5 Multipass Approach

Roughly speaking the Reeb graph gives rise to two types of topological features: the branching features and cycle features. These features are precisely encoded in the zero persistence diagram $\mathrm{Dg}_0(f)$ and first extended persistence diagram $\mathrm{ExDg}_1(f)$ [4]. The persistence diagram $\mathrm{Dg}_0(f)$ can be obtained by pairing the non-essential fork nodes of the Reeb graph. On the other hand, the extended persistence diagram $\mathrm{ExDg}_1(f)$ can be obtained by pairing of essential fork nodes. We next demonstrate these two steps using Figure 4(a) as a running example.

5.1 Non-Essential Fork Pairing

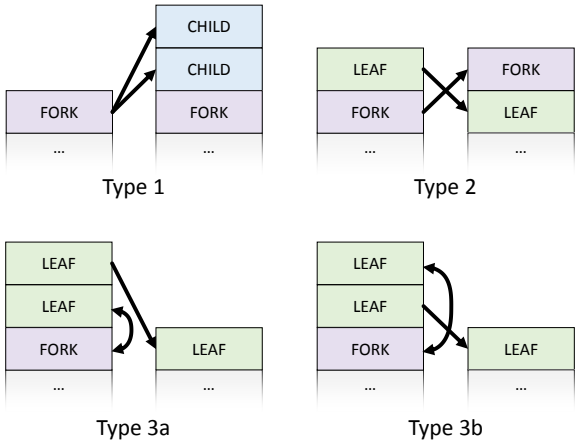
Identifying the non-essential forks can be reduced to calculating both a join and a split tree on the Reeb graph. In our implementation, this is done using Carr's et al.'s approach [8]. Then, a stack-based algorithm, based upon branch decomposition [27], can be executed to pair critical points. The algorithm operates as a depth first search that seeks out simply connected forks (i.e., forks connected to 2 leaves) and recursively pairs and collapses the tree.

The algorithm processes a stack that is initially seeded with the root of the graph. Assuming the graph has been properly conditioned, at each iteration, 1 of 3 operation types occurs, as seen in Figure 5(a). Operation Type 1 occurs when the top of the stack is a fork. In this case, the children of the fork are pushed onto the stack. Operation Type 2 occurs when the top of the stack is a leaf, but the next node is a fork. In this case, the leaf and fork have their orders swapped. Finally, operation Type 3 has 2 variants that occur when 2 leaf nodes sit atop the stack. In both variants, one leaf is paired with the fork, and the other leaf is pushed back onto the stack. The pairing occurs with the leaf that has a value *closer* to the value of the fork. The stack is processed until only a single leaf node remains on it.

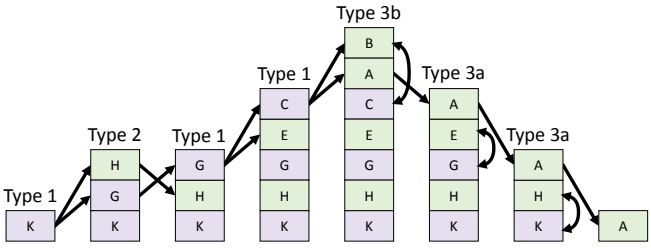


(a) Reeb graph (b) Split Tree (c) Join Tree (d) Essential Forks

Fig. 4. In the multipass approach, (a) the Reeb graph has (b) a split tree and (c) a join tree extracted for non-essential pairing. Then in a separate process, the (d) essential forks are paired one at a time. The persistence diagram for this Reeb graph is shown in Figure 2.



(a) The 4 cases of stack configurations and their result. Type 1 and 2 involve stack reorganization, while Type 3a and 3b are pairing operations.



(b) Example processing the join tree from Figure 4(c).

Fig. 5. Illustration of non-essential fork pairing. (a) The 4 cases for stack processing are illustrated with their resulting configurations. (b) An example pairing of the join tree from Figure 4(c) shows the stack at each processing step, from left to right.

The algorithm operates identically on both join and split trees. Finally, the unpaired global minimum and maximum left on the stacks created for the join tree and split tree, respectively, can be paired.

Figure 5(b) shows an example for the join tree in Figure 4(c). Initially the root K is placed onto the stack. A Type 1 operation pushes the children, G and H, onto the stack. Next, a Type 2 operation reorders the top of the stack. G, a down-fork, in now atop the stack, pushing its 2 children, E and C, onto the stack. Another Type 1 pushes C's children, A and B onto the stack. In the next 3 steps, a series of Type 3 operations occur. First B and C are paired, followed by E and G, and finally H and K. At the end, A, the global minimum, is the only point remaining on the stack. These assigned pairs, B/C, E/G, and H/K, appear in the $\mathrm{Dg}_0(f)$ in Figure 2(a), along with the split tree pairing, N/O, and the global min/max pairing, A/P.

5.2 Essential Forks Pairing

The remaining unpaired forks are essential forks, as seen in Figure 4(d). We extract an algorithms from the high-level description of [4] to pair them. The procedure processes up-forks one at a time.

The essential fork pairing algorithm can be treated as join tree problem. For a given up-fork, the node can be split into two nodes. A join tree can be computed by sweeping the superlevel set. At each step of the sweep, the connected components are calculated. The pairing for a selected up-fork occurs at the down-fork who merges the 2 generated nodes into a single connected component.

Figure 6 shows the sweeping process for the up-fork D. Initially (Figure 6(a)), D is split into D_L and D_R, which are each part of separate connected components, denoted by color. As the join tree is swept past

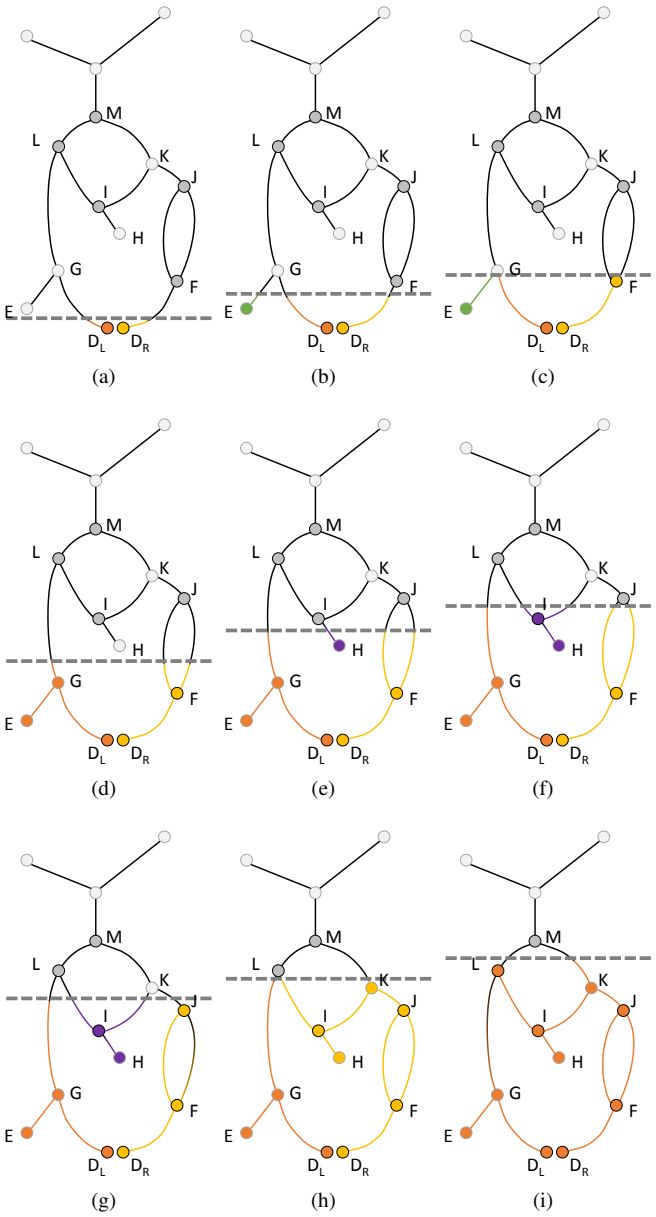


Fig. 6. The join tree-based essential fork pairing for up-fork D. (a) D is initially split into D_L and D_R. (b-h) The colors indicate different connected components as the join tree is swept up the superlevel set. (i) The pairing is found when D_L and D_R are contained in the same connected component.

E (Figure 6(b)), a new connected component is formed. In Figure 6(c), F is added to the connected component of D_R. As the join tree is swept past G (Figure 6(d)), the E and D_L connected components join. The process continues until Figure 6(g), where 3 connected components exist. The purple and yellow components join at K (Figure 6(h)). Finally at L (Figure 6(i)), both D_L and D_R are part of the same connected component. This indicates that D pairs with L.

Figure 7 shows the superlevel sets and associated join trees for the up-forks D, F, and I. The pairing partner L/D, J/F, and M/I can all be seen in the $\mathrm{ExDg}_1(f)$ in Figure 2(b).

6 Single-Pass Algorithm: Propagate and Pair

In the previous section, we showed that the critical point pairing problem could be broken down into a series of merge tree computations. For non-essential forks this was in the form of join and split trees, which are merge trees of the superlevel sets and sublevel sets, respectively. For essential saddles, it came in the form of a special join tree calculation for each essential up-fork. A natural question is whether these merge tree calculations can be combined into a single-pass operation, which is exactly what follows.

6.1 Basic Propagate and Pair

The algorithm operates by sweeping the Reeb graph from lowest to highest value. At each point, a list of points from the sublevel set is maintained. When a point is processed in the sweep, 2 possible operations occur on these lists: *propagate* and/or *pair*.

Propagate. The job of propagate is to push labels from unpaired nodes further up in the Reeb graph. 4 cases exist.

- For local minima, a label for the current critical point is propagated upward. In the examples of Figure 8(a) and 8(b), both A and B are propagated to C.
- For down-forks, all unpaired labels are propagated upwards. In the example of Figure 8(c), the critical points B and C are paired, thus only A is propagated to D.
- For up-forks, all unpaired labels are propagated upwards, along with labels for the fork that are each tagged with the specific branch of the that fork created them (in the examples with subscripts L and R). This tag is critical for closing essential cycles. In the example of Figure 8(d), the labels A and D_L are propagated to G, and labels A and D_R are propagated to F.
- Local maxima have no need to propagate.

Pair. The pairing operation searches the list of labels to determine an appropriate partner from the sublevel set. The pairing operation only occurs for local maxima and down-forks.

- For local maxima, the list labels is searched for the unpaired up-fork with the largest value. Those critical points are then paired.

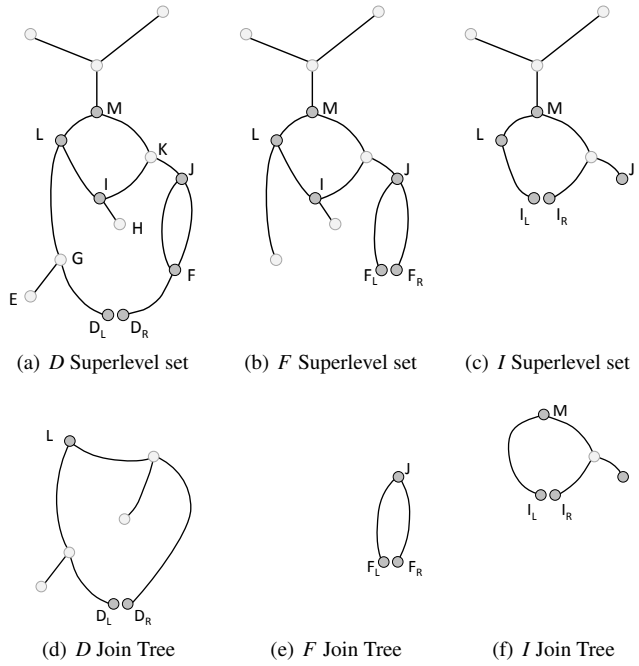
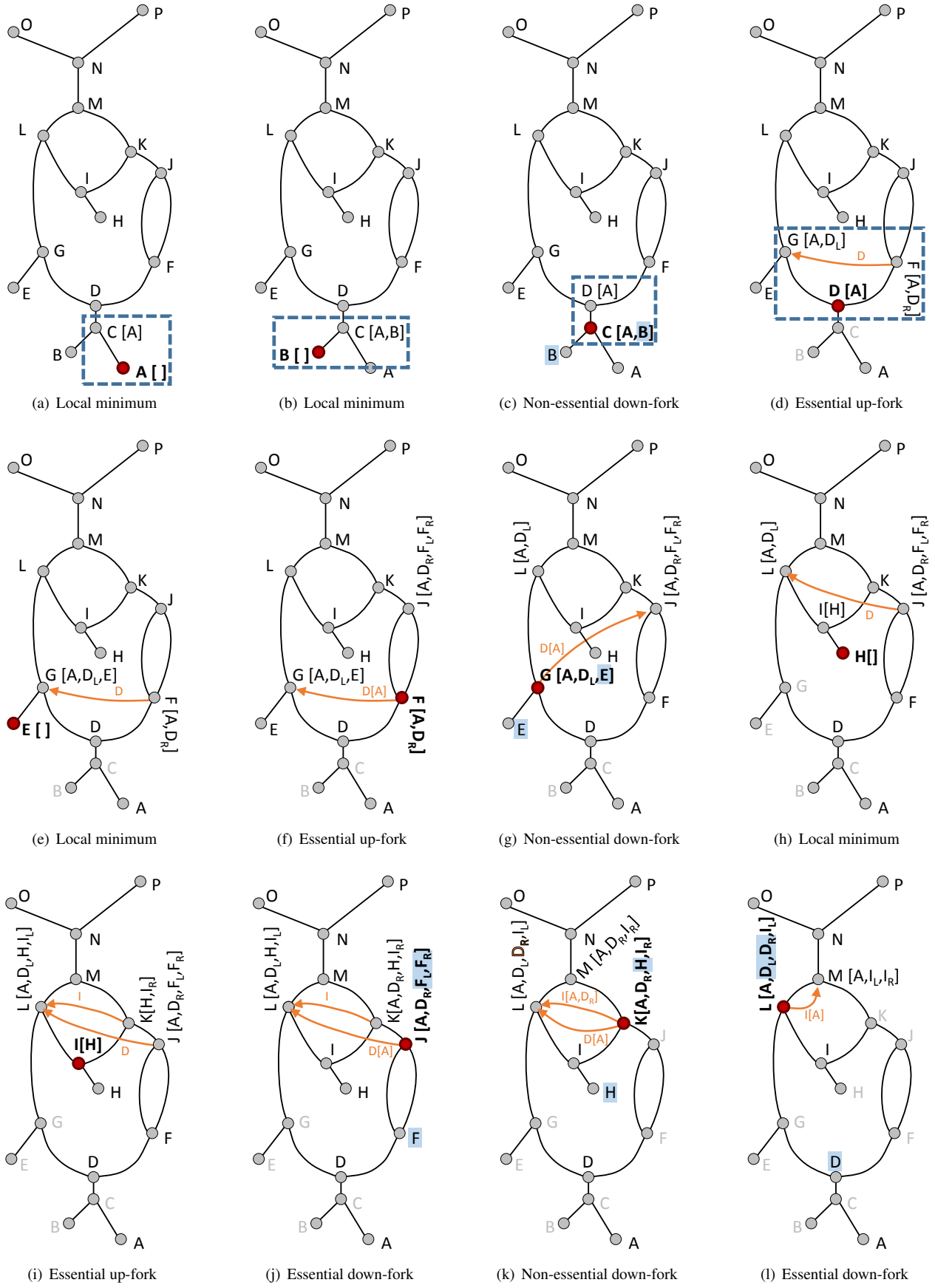


Fig. 7. Essential fork pairing for the example Reeb graph from Figure 4. Each up-fork (D, F, and I, respectively) is split into 2 pieces and (d-f) a join tree calculated from the (a-c) superlevel set to find the partner. Figure 6 shows a detailed calculation for D.

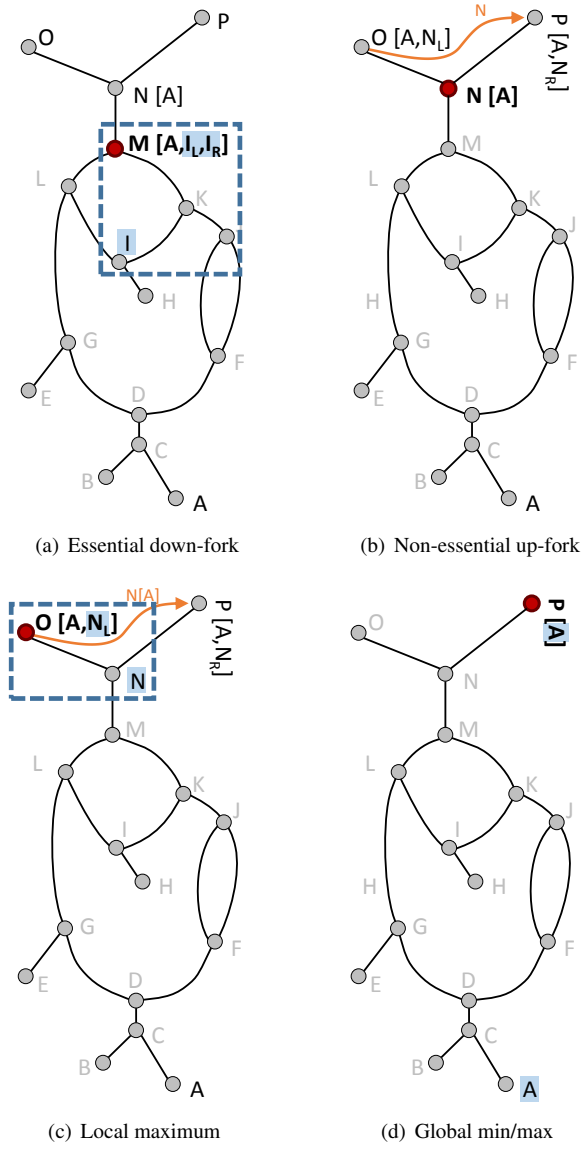


(a) Local minimum (b) Local minimum (c) Non-essential down-fork (d) Essential up-fork

(e) Local minimum (f) Essential up-fork (g) Non-essential down-fork (h) Local minimum

(i) Essential up-fork (j) Essential down-fork (k) Non-essential down-fork (l) Essential down-fork

Fig. 8. Propagate and Pair algorithm on the example Reeb graph from Figure 4. At each step, the node being processed is in bold; propagated edges are shown in brackets; pairing is shown in blue; and virtual edges are shown in orange. The example is continued in Figure 9.



(a) Essential down-fork (b) Non-essential up-fork

(c) Local maximum (d) Global min/max

Fig. 9. Continuation of Figure 8.

In the example from Figure 9(c) for local maximum O, the list is searched and N_L is determined to be the closest unpaired up-fork.

- For down-forks, two possible cases exist, essential or non-essential, which can be differentiated by searching the available labels. First, the list is searched for the largest up-fork with both legs. Both legs indicate that the current down-fork is essential, and it is paired with the associated up-fork. In the example, Figure 9(a), the list of M is searched and labels I_L and I_R found. If no such up-fork exists, then the down-fork is non-essential. In this case, the highest valued local minimum is selected from the list. In the example of Figure 8(c), no essential up-forks are found for C, and the largest local minimum, B is selected.

6.2 Virtual Edges for Propagate and Pair

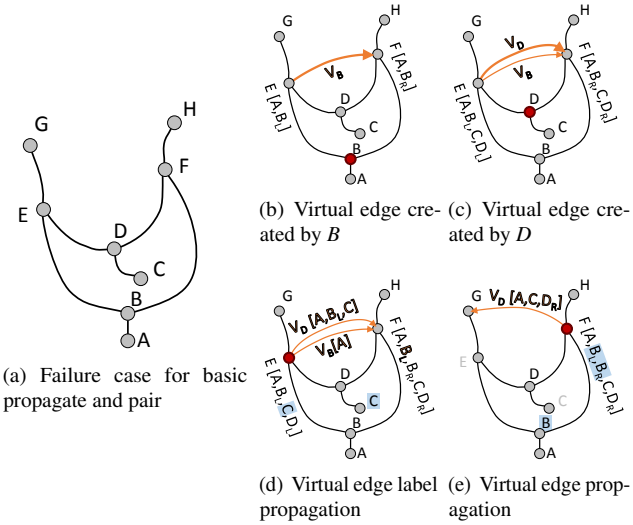
This propagate and pair approach succeeds in most case, but in certain cases, such as in Figure 10(a), it fails. The failure arises from the assumption that the superlevel set is the only thing needed to propagate labels. In this case, label information needs to be communicated between E and F, which are connected by the node D in the sublevel set. To resolve this communication issue, virtual edges are used. Virtual edges have 3 associated operations.

Creation. Virtual edges are created on all up-fork operations. For example in Figure 10(b), when processing B, the endpoints of the fork, E and F are connected with virtual edge V_B. Similarly, in Figure 10(c), when processing up-fork D, another virtual edge V_D is created connecting the endpoint, E and F.

Label Propagation. Propagating labels across virtual edges is similar to standard propagation with one additional condition. A label can only be propagated if its value is less than that of the up-fork that generated the virtual edge. In other words, for a given label X and a virtual edge V_Y, X is only propagated if $f(X) < f(Y)$. Looking at the example in Figure 10(d), for the virtual edge V_B, only A is propagated because $f(A) < f(B)$. For the virtual edge V_D, A, B_L, and C are all propagated, since they all have values smaller than D.

Virtual Edge Propagation. Finally, virtual edges themselves need to be propagated. For all critical points, any virtual edge endpoint attached to that critical point is propagated up outgoing edges. In the simple case of down-forks, the edge just gets pushed up, as we see in Figure 10(e). In the case of up-forks with virtual edges, additional virtual edges are created by the splitting action. During this virtual edge propagation phase, redundant virtual edges can also be culled. For example, the virtual edge V_D is a superlevel set of V_B. Therefore, V_B can be discarded.

The necessity of the virtual edge process can also be seen in Figure 8. In Figure 8(l), the pairing of L with D is only possible because of the virtual edge created by I in Figure 8(i).



(a) Failure case for basic propagate and pair

(b) Virtual edge created by B

(c) Virtual edge created by D

(d) Virtual edge label propagation

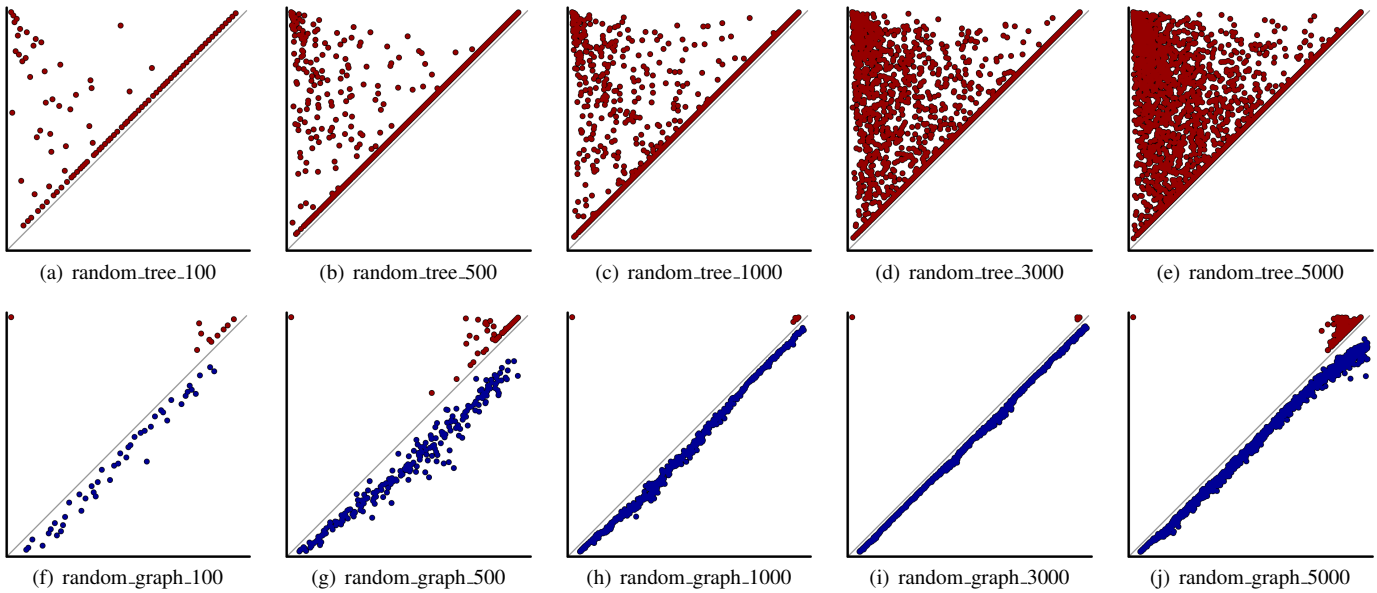
(e) Virtual edge propagation

Fig. 10. (a) An example case of where the basic propagate and pair algorithm fails. In this case, B and F should pair but will not. To overcome this limitation, (b-c) virtual edges are created as up-forks are processed. (d) Labels can then be propagated across virtual edges. (e) The virtual edges themselves are propagated and redundant edges removed.

7 Evaluation

We have implemented the described algorithms using Java. The source code will be made public upon publication of this manuscript. Reported performance was calculated on a 2017 MacBook Pro, 3.1 Ghz i5 CPU. Both the multipass and single-pass algorithms have worst case $O(nt)$ performance, where n is the number of nodes in the Reeb graph, and t is the number of up-forks.

We investigate the real performance of the algorithms using the Reeb graph from Figures 1 and 2, synthetically generated Reeb graphs in Figure 11, Reeb graphs calculated on publicly available meshes in Figure 12, and time-series of 120 Mapper graphs taken from the 2016 SciVis Contest in Figure 13.



(a) random_tree_100 (b) random_tree_500 (c) random_tree_1000 (d) random_tree_3000 (e) random_tree_5000

(f) random_graph_100 (g) random_graph_500 (h) random_graph_1000 (i) random_graph_3000 (j) random_graph_5000

Fig. 11. Persistence diagrams for random trees (top) and random graphs (bottom). The number indicates how many random iterations were used to generate the example. The $\mathrm{Dg}_0(f)$ and $\mathrm{ExDg}_1(f)$ have been combined into a single diagram with $\mathrm{Dg}_0(f)$ in red and $\mathrm{ExDg}_1(f)$ in blue.

Table 1. Performance for the datasets tested. Bold indicates the faster algorithm.

Data	Figure	Mesh		Reeb Graph Nodes		Cycles	Multipass Time (ms)	Single-pass Time (ms)
		Vertices	Faces	Initial	Conditioned			
running_example	1 / 2	—	—	16	16	3	1.06e-02	**5.06e-03**
random_tree_100	11(a)	—	—	401	204	0	**2.26e-02**	6.63e-02
random_tree_500	11(b)	—	—	2001	1004	0	**0.13**	1.62
random_tree_1000	11(c)	—	—	4001	2004	0	**0.28**	6.42
random_tree_3000	11(d)	—	—	12001	6004	0	**1.05**	84.72
random_tree_5000	11(e)	—	—	20001	10004	0	**1.94**	301.68
random_graph_100	11(f)	—	—	401	112	46	1.81e-02	**1.42e-02**
random_graph_500	11(g)	—	—	2001	542	231	**0.48**	1.84
random_graph_1000	11(h)	—	—	4001	1010	497	**0.53**	1.32
random_graph_3000	11(i)	—	—	12001	3014	1495	**1.63**	3.95
random_graph_5000	11(j)	—	—	20001	5204	2400	**14.98**	148.10
4_torus	12(c)	10401	20814	23	10	4	1.16e-03	**5.61e-04**
buddah	12(d)	10098	20216	33	14	6	1.44e-03	**6.65e-04**
david	12(f)	26138	52284	8	8	3	8.17e-04	**3.44e-04**
double_torus	12(a)	3070	6144	13	6	2	4.87e-04	**2.11e-04**
female	12(b)	8410	16816	15	8	0	1.63e-03	**1.82e-04**
flower	12(g)	4000	8256	132	132	65	2.63e-02	**2.08e-02**
greek	12(h)	39994	80000	23	10	4	8.09e-04	**3.23e-04**
topology	12(e)	6616	13280	28	28	13	4.19e-03	**3.09e-03**
scivis_contest_2016	13	194k (avg)	—	117 (avg)	178.2 (avg)	81.3 (avg)	3.76 (total)	**2.88** (total)

The synthetic Reeb graphs were generated by a Python script. Given a positive integer n, the script starts by creating a fork G_1 consisting of a node with valency 3 and three nodes with valency 1 linked to the 3-valence node. For each iteration $i < n$, another fork is generated, and one or two of its one valency nodes are glued to the nodes in G_{i-1} with valency one. If we constrain the choice of gluing a single node at each iteration the resulting graph will be a contour tree. The mesh data are provided by AIM@SHAPE Shape Repository. Reeb graphs of the mesh data were extracted using our own Reeb graph implementation in C++. The 2016 SciVis Contest data[1] is a large time-varying multi-run particle simulation. Our evaluation took one realization, smoothing length 0.44, run 50, and calculated the Mapper graph for all 120 time-steps of the variable of interest, concentration. Our included video shows the entire sequence. The Mapper graph was generated using a Python script that follows the generic Mapper algorithm [33].

The performance for the algorithms can be seen in Table 1. These values were obtained by running the test 1000 times and storing the average compute time. The persistence diagrams of both the single-pass and multipass algorithms were compared in order to verify correctness. For all synthetic examples but one, the multipass algorithm outperformed the single-pass algorithm. In all other cases (i.e., the running_example, all mesh examples, and scivis_contest_2016), the single-pass algorithm outperformed the multipass algorithm, albeit by a small amount. We had originally hypothesized that the performance between algorithms would be similar with a slight edge to the single-pass version. Further investigation is required into the precise cause of slowdown in the syn-

[1] https://www.uni-kl.de/sciviscontest/

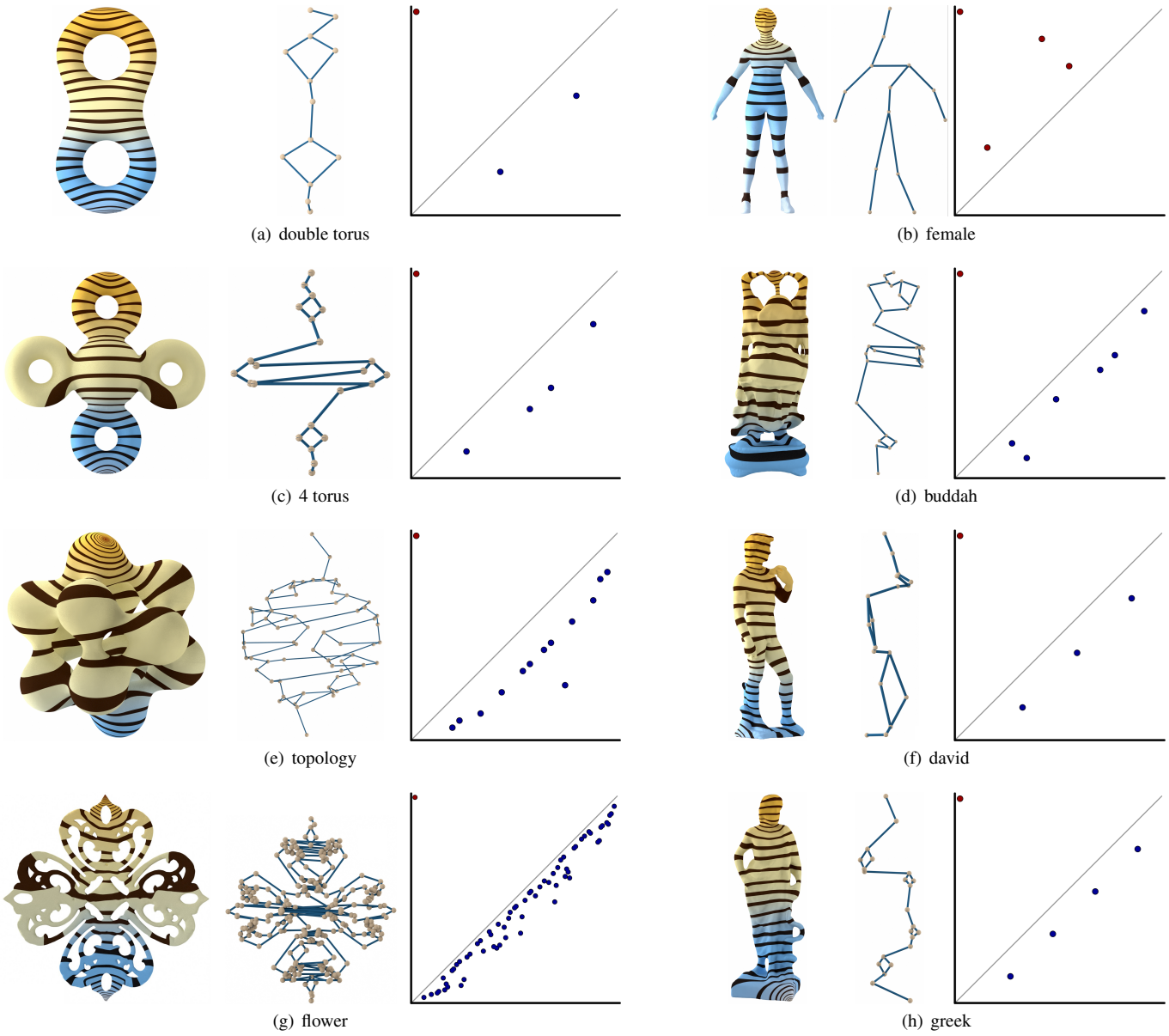


Fig. 12. Meshes, Reeb graphs, ad persistence diagrams for datasets used in evaluation. The mesh is colored by the scalar function applied to it. The $\mathrm{Dg}_0(f)$ and $\mathrm{ExDg}_1(f)$ have been combined into a single diagram with $\mathrm{Dg}_0(f)$ in red and $\mathrm{ExDg}_1(f)$ in blue.

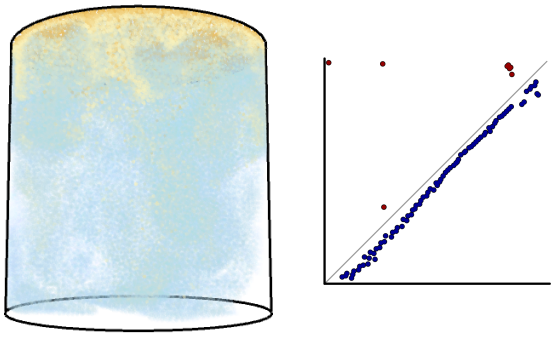


Fig. 13. One timestep (066) from the scivis_contest_2016 data. The spatial data with concentration mapped to the color channel is shown (left) along with the combined persistence diagram for the Mapper graph (right). The $\mathrm{Dg}_0(f)$ is in red, and the $\mathrm{ExDg}_1(f)$ is in blue.

thetic cases, but we believe it has to do with the cost and effects of the large number of virtual edges.

8 Conclusion

Pairing critical points is a key part of the TDA pipeline—the Reeb graphs capture complex structure, but direct representation is often impractical. Critical point pairing enables a compact visual representation in the form of a persistence diagram. Although previous works have laid out high-level descriptions of essential critical point pairing, none have previously provided an algorithm. Our approach leverages merge trees to provide both intuitive and efficient single-pass and multipass algorithms that can pair critical points in Reeb graphs, Mapper graphs, contour trees, join trees, and split trees.

Acknowledgments

This work was supported in part by a grant from XYZ (# 12345-67890). Our mesh data are provided by AIM@SHAPE Shape Repository

REFERENCES

[1] P. K. Agarwal, H. Edelsbrunner, J. Harer, and Y. Wang. Extreme elevation on a 2-manifold. *Discrete & Computational Geometry*, 36(4):553–572, 2006.

[2] M. Attene, S. Biasotti, and M. Spagnuolo. Shape understanding by contour-driven retiling. *The Visual Computer*, 19(2):127–138, 2003.

[3] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *Proceedings of the 8th IEEE Visualization*, pp. 167–ff, 1997.

[4] U. Bauer, X. Ge, and Y. Wang. Measuring distance between reeb graphs. In *Proceedings of the thirtieth annual symposium on Computational geometry*, p. 464. ACM, 2014.

[5] P. J. Besl and N. D. McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pp. 586–606. International Society for Optics and Photonics, 1992.

[6] S. Biasotti, B. Falcidieno, P. Frosini, D. Giorgi, C. Landi, S. Marini, G. Patané, and M. Spagnuolo. 3d shape description and matching based on properties of real functions. In *Eurographics (Tutorials)*, pp. 949–998, 2007.

[7] R. L. Boyell and H. Ruston. Hybrid techniques for real-time radar simulation. In *Proceedings of the November 12-14, 1963, fall joint computer conference*, pp. 445–458. ACM, 1963.

[8] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry: Theory and Applications*, 24(2):75–94, 2003. doi: 10.1016/S0925-7721(02)00093-7

[9] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. *Proceedings 15th IEEE Visualization*, pp. 497–504, 2004.

[10] F. Chazal and S. Y. Oudot. Towards persistence-based reconstruction in euclidean spaces. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pp. 232–241. ACM, 2008.

[11] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in reeb graphs of 2-manifolds. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pp. 344–350. ACM, New York, NY, USA, 2003. doi: 10.1145/777792.777844

[12] T. K. Dey and K. Li. Cut locus and topology from surface point data. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pp. 125–134. ACM, 2009.

[13] T. K. Dey, J. Sun, and Y. Wang. Approximating cycles in a shortest basis of the first homology group from point data. *Inverse Problems*, 27(12):124004, 2011.

[14] H. Doraiswamy and V. Natarajan. Efficient output-sensitive construction of reeb graphs. In *International Symposium on Algorithms and Computation*, pp. 556–567. Springer, 2008.

[15] H. Doraiswamy and V. Natarajan. Efficient algorithms for computing reeb graphs. *Computational Geometry*, 42(6):606–616, 2009.

[16] H. Doraiswamy and V. Natarajan. Computing reeb graphs as a union of contour trees. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):249–262, 2013. doi: 10.1109/TVCG.2012.115

[17] H. Edelsbrunner, J. Harer, A. Mascarenhas, and V. Pascucci. Time-varying reeb graphs for continuous space-time data. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 366–372. ACM, 2004.

[18] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pp. 454–463. IEEE, 2000.

[19] X. Ge, I. I. Safa, M. Belkin, and Y. Wang. Data skeletonization via reeb graphs. In *Advances in Neural Information Processing Systems*, pp. 837–845, 2011.

[20] S. K. Gupta, W. C. Regli, and D. S. Nau. Manufacturing feature instances: which ones to recognize? In *Proceedings of the third ACM symposium on Solid modeling and applications*, pp. 141–152. ACM, 1995.

[21] W. Harvey, Y. Wang, and R. Wenger. A randomized O (m log m) time algorithm for computing Reeb graphs of arbitrary simplicial complexes. *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pp. 267–276, 2010. doi: 10.1145/1810959.1811005

[22] M. Hilaga and Y. Shinagawa. Topology matching for fully automatic similarity estimation of 3D shapes. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 203–212, 2001. doi: 10.1145/383259.383282

[23] I. S. Kweon and T. Kanade. Extracting topographic terrain features from elevation maps. *CVGIP: image understanding*, 59(2):171–182, 1994.

[24] E. Munch. A users guide to topological data analysis. *Journal of Learning Analytics*, 4(2):47–61, 2017.

[25] M. Natali, S. Biasotti, G. Patanè, and B. Falcidieno. Graph-based representations of point clouds. *Graphical Models*, 73(5):151–164, 2011.

[26] S. Parsa. A deterministic $O(m \log m)$ time algorithm for the Reeb graph. In *ACM Sympos. Comput. Geom. (SoCG)*, pp. 269–276, 2012.

[27] 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*, pp. 452–290, 2004.

[28] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust online computation of Reeb graphs: Simplicity and speed. *ACM Transactions on Graphics*, 26(3):58.1–58.9, 2007.

[29] G. Reeb. Sur les points singuliers dune forme de pfaff completement intgrable ou dune fonction numrique. *CR Acad. Sci. Paris*, 222:847–849, 1946.

[30] M. Reuter. Hierarchical shape segmentation and registration via topological features of laplace-beltrami eigenfunctions. *International Journal of Computer Vision*, 89(2-3):287–308, 2010.

[31] P. Rosen, B. Wang, A. Seth, B. Mills, A. Ginsburg, J. Kamenetzky, J. Kern, and C. R. Johnson. Using contour trees in the analysis and visualization of radio astronomy data cubes. *arXiv preprint arXiv:1704.04561*, 2017.

[32] Y. Shinagawa and T. L. Kunii. Constructing a reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 11(6):44–51, 1991.

[33] G. Singh, F. Mémoli, and G. E. Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. In *Eurographics Symposium on Point-Based Graphics*, pp. 91–100, 2007.

[34] P. Skraba, M. Ovsjanikov, F. Chazal, and L. Guibas. Persistence-based segmentation of deformable shapes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 45–52. IEEE, 2010.

[35] S. Takahashi, Y. Takeshima, and I. Fujishiro. Topological volume skeletonization and its application to transfer function design. *Graphical Models*, 66(1):24–49, 2004.

[36] J. Tierny, J. P. Vandeborre, and M. Daoudi. Partial 3D shape retrieval by Reeb pattern unfolding. *Computer Graphics Forum*, 28(1):41–55, 2009. doi: 10.1111/j.1467-8659.2008.01190.x

[37] T. Várady, M. A. Facello, and Z. Terék. Automatic extraction of surface structures in digital shape reconstruction. *Computer-Aided Design*, 39(5):379–388, 2007.

[38] 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. doi: 10.1109/TVCG.2007.47

[39] G. H. Weber and G. Scheuermann. Topology-based transfer function design. In *Proceedings of the second IASTED international conference on visualization, imaging, and image processing*, pp. 527–532, 2002.

[40] L. Xie and P. E. Bourne. A robust and efficient algorithm for the shape description of protein structures and its application in predicting ligand binding sites. In *BMC bioinformatics*, vol. 8, p. S9. BioMed Central, 2007.