Available online at www.sciencedirect.com
ScienceDirect
Computational Geometry 41 (2008) 149–166
Computational Geometry
Theory and Applications
www.elsevier.com/locate/comgeo

Time-varying Reeb graphs for continuous space–time data ☆

Herbert Edelsbrunner [a], John Harer [b], Ajith Mascarenhas [c,*], Valerio Pascucci [c], Jack Snoeyink [d]

[a] *Department of Computer Science and Mathematics, Duke University, Durham, and Raindrop Geomagic, Research Triangle Park, NC, USA*
[b] *Department of Mathematics and Computer Science, Duke University, Durham, NC, USA*
[c] *Center for Applied Scientific Computing, Lawrence Livermore National Labs, Livermore, CA, USA*
[d] *Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA*

Received 15 December 2006; received in revised form 1 September 2007; accepted 15 November 2007

Available online 28 January 2008

Communicated by G. Rote

Abstract

The Reeb graph is a useful tool in visualizing real-valued data obtained from computational simulations of physical processes. We characterize the evolution of the Reeb graph of a time-varying continuous function defined in three-dimensional space. We show how to maintain the Reeb graph over time and compress the entire sequence of Reeb graphs into a single, partially persistent data structure, and augment this data structure with Betti numbers to describe the topology of level sets and with path seeds to assist in the fast extraction of level sets for visualization.

Keywords: Differential and computational topology; Morse functions; Critical points; Level sets; Reeb graph; Triangulations; Combinatorial algorithms

1. Introduction

Physical processes that are measured over time, or modeled and simulated on a computer, can produce large amounts of data that must be interpreted with the assistance of computational tools. Such data arises in a wide variety of studies, including computational fluid dynamics [9], oceanography [5], and climate modeling [19]. The data typically consists of finitely many points in space–time and a measured value for each. We can connect these points into a mesh and interpolate the values to obtain a continuous function over the entire domain. Piecewise linear interpolation is common for large amounts of data, because of its relative ease; multi-linear interpolation is also used for regular grids.

☆ Research of the first author was supported by NSF under grants EIA-99-72879 and CCR-00-86013. Research of the second author was supported by NSF under grant DMS-01-07621. Research of the third and last author was supported by NSF under grant 0128426 and by sub-contracts from Lawrence Livermore National Labs. Portions of this work was performed under the auspices of the US Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.
* Corresponding author.
E-mail address: ajmasci@gmail.com (A. Mascarenhas).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2007.11.001

Graphical visualization, often through level sets or iso-surfaces of a continuous function, is useful for interpreting the data. A level set consists of all points in the domain whose function values are equal to a chosen real number s. In $\mathbb{R}^3$, the level set is generically a surface that can be interactively visualized on a graphical display. By varying s, we can study the variation in the data. Topological features of the level sets, such as connected components, handles, and voids, can be important aids in interpreting the data. The Reeb graph encodes the evolution of these features and compactly represents topological information of all level sets. As we vary time, the Reeb graph goes through an evolution of its own, undergoing structural changes at birth-death points and at interchanges of critical points. The evolution of the Reeb graph thus represents a 2-parameter family of level sets. We suggest that this 2-parameter family, encoded in a compact data structure, is a useful representation of space–time data.

In this paper, we study how the Reeb graph of a smooth function on three-dimensional space evolves over time. We compactify the space to a closed manifold without boundary by adding the point at infinity, effectively creating the topology of the 3-sphere, denoted by $\mathbb{S}^3$. This compactification makes our analysis easier as we do not have to consider manifolds with boundaries. Our first contribution, described in Section 3, is a complete enumeration of the type of combinatorial changes the Reeb graph experiences:

- nodes disappear in pairs, contracting arcs to points (inversely, node appear in pairs, anti-contracting arcs);
- nodes swap their positions along the arcs of the graph.

The second type of change falls into more sub-types and is algorithmically more difficult to handle than the first type. Based on this classification, we develop an algorithm that maintains the Reeb graph through time and stores the evolution in a partially persistent data structure. The size of this data structure is proportional to the size of the initial Reeb graph at time zero, plus the number of changes it experiences through time. We describe the algorithm in an abstract setting in Section 4 and provide a concrete implementation for the piecewise linear case in Section 5. We also show how to augment the evolving Reeb graph with the Betti numbers of the level set components in Section 6, and with path seeds for efficient level set extraction in Section 7. We comment on the difficulties in extending the analysis and algorithm to general functions in Section 8. This paper is an extended version of an earlier conference paper [14]; Sections 6 through 8 are new.

Related prior work. In the interactive exploration of scientific data, Reeb graphs are used to select meaningful level sets [3] and to efficiently compute them [17]. An extensive discussion of Reeb graphs and related structures in geometric modeling and visualization applications can be found in [16].

All published algorithms for computing Reeb graphs take as input a function defined on a triangulated manifold. We express their running times as functions of n, the number of simplices in the triangulation. The first algorithm for functions on 2-manifolds due to Shinagawa and Kunii [24] takes time $O(n^2)$ in the worst case. Cole-McLaughlin et al. [10] give tight bounds on the number of loops in Reeb graphs of functions on 2-manifolds and describe an $O(n \log n)$ time algorithm to construct them.

Loop-free Reeb graphs, also known as contour trees, arise in simply-connected domains. They have received special attention because of the practical importance of these domains, and because the algorithms to compute them are simpler. An algorithm that constructs contour trees of functions on simply-connected manifolds of constant dimension in time $O(n \log n)$ has been suggested in [6], and improved to time $O(n + k \log k)$, where k is the number of critical points, in [8]. For the case of functions on 3-manifolds, an extension to the algorithm of [6] provides information about the genus of the level surfaces [22].

There has been some recent work on Reeb graphs of time-varying functions. Bajaj and Sohn [25] study the change in topology of level sets over time by computing the correspondence between contour trees of successive time steps. They assume that the function can change unpredictably between two successive time steps, and define temporal correspondence between the contour tree arcs using a notion of an overlap between level set components at time t with those at time $t + 1$. They compute this correspondence, and use it to track level set components and their topology over time. Szymczak [26] employs sub-domain aware contour trees to compute a mapping between contour trees of functions at two successive time-steps. Unlike the approach taken in this paper both Bajaj et al. and Szymczak ignore the individual structural changes to the trees in between the time-steps, but instead consider the cumulative effect of all these changes.

2. Mathematical background

We need some background from Morse theory [18,20] and from combinatorial and algebraic topology [2,21].

Smooth maps on manifolds. Let $\mathbb{M}$ be a smooth, compact d-manifold without boundary and $f : \mathbb{M} \to \mathbb{R}$ a smooth map. Assuming a local coordinate system in its neighborhood, $x \in \mathbb{M}$ is a *critical point* of f if all partial derivatives vanish at x. If x is a critical point, $f(x)$ is a *critical value*. Non-critical points and non-critical values are called *regular points* and *regular values*, respectively. The *Hessian* at x is the matrix of second-order partial derivatives. A critical point x is *non-degenerate* if the Hessian at x is non-singular. The *index* of a critical point x, denoted by $\operatorname{index} x$, is the number of negative eigenvalues of the Hessian. Intuitively, it is the number of mutually orthogonal directions at x along which f decreases. For $d = 3$ there are four types of non-degenerate critical points: the *minima* with index 0, the 1-*saddles* with index 1, the 2-*saddles* with index 2, and the *maxima* with index 3. A function f is *Morse* if

I. all critical points are non-degenerate;
II. $f(x) \neq f(y)$ whenever $x \neq y$ are critical.

We refer to I and II as Genericity Conditions as they prevent certain non-generic configurations of the critical points. This is so because Morse functions are dense in $C^\infty(\mathbb{M})$, the class of smooth functions on the manifold. In other words, for every smooth function there is an arbitrarily small perturbation that makes it a Morse function.

The critical points of a Morse function and their indices capture information about the manifold on which the function is defined. For example, the Euler characteristic of the manifold $\mathbb{M}$ equals the alternating sum of critical points, $\chi(\mathbb{M}) = \sum_x (-1)^{\operatorname{index} x}$. Another useful tool in the study of manifolds is the incremental construction by adding one cell at a time. Call a space homeomorphic to the k-dimensional ball a k-*cell*. Let $f : \mathbb{M} \to \mathbb{R}$ be a Morse function and define $\mathbb{M}_s = f^{-1}(-\infty, s]$. The boundary of $\mathbb{M}_s$ is the *level set*, $f^{-1}(s)$, defined by $s \in \mathbb{R}$. If s is a regular value then $f^{-1}(s)$ is a $(d-1)$-manifold. For positive ε consider $\mathbb{M}_{s+\varepsilon}$ and assume that f has a single critical point with function value in $(s, s+\varepsilon]$. If the index of that critical point is k then $\mathbb{M}_{s+\varepsilon}$ is homotopy equivalent to $\mathbb{M}_s$ with a single k-cell attached.

Reeb graph. A level set of f is not necessarily connected. We call two points $x, y \in \mathbb{M}$ *equivalent* when $f(x) = f(y)$ and x and y both belong to the same component of the level set. We obtain the *Reeb graph* as the quotient space in which every equivalence class is represented by a point and connectivity is defined in terms of the quotient topology [23]. Fig. 1 illustrates the definition for a 2-manifold of genus two. We call a point on the Reeb graph a *node* if the corresponding level set component passes through a critical point of f. The rest of the Reeb graph consists of arcs connecting the nodes. The *degree* of a node is the number of arcs incident to the node. A minimum creates and a maximum destroys a level set component and both correspond to degree-1 nodes. A saddle that splits one level set component in two or merges two to one corresponds to a degree-3 node. There are also saddles that alter the genus but do not affect the number of components, and they correspond to degree-2 nodes in the Reeb graph. Nodes of degree higher than three occur only for non-Morse functions.

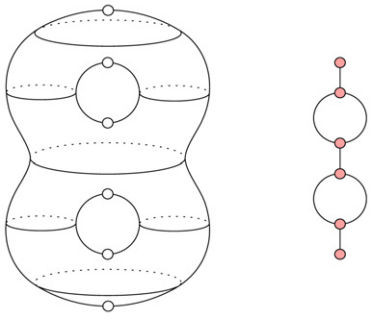

Fig. 1. The Reeb graph of the function f on a 2-manifold, which maps every point of the double torus to its distance above a horizontal plane below the surface.

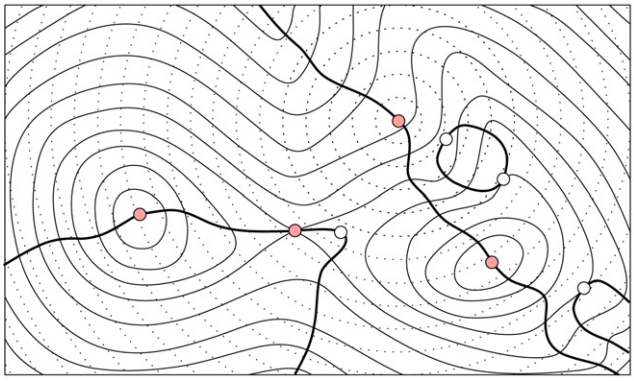

Fig. 2. The functions f and g are represented by their dotted and solid level curves. The Jacobi curve is drawn in bold solid lines. The birth–death points and the critical points of the two functions are marked by white and shaded dots, respectively.

In mathematics, the Reeb graph is often used to study the manifold $\mathbb{M}$ that forms the domain of the function. For example, the Reeb graph in Fig. 1 reveals that the function is defined on a double torus, assuming we know it is an orientable 2-manifold without boundary. In contrast, we use the Reeb graph to study the behavior of functions. The domain of interest is $\mathbb{R}^3$ but we compactify it and consider functions on the 3-sphere, $\mathbb{S}^3$. All our Reeb graphs reveal the (un-exciting) connectivity of $\mathbb{S}^3$ by being trees, but the structure of the tree tells us something about the chosen function f.

Jacobi curves. We use Reeb graphs to understand a function at moments in time, and Jacobi curves, as introduced in [13], to help track the evolution of the Reeb graphs through time. We define Jacobi curves of two Morse functions, $f, g : \mathbb{M} \to \mathbb{R}$, then specialize this definition to a time-varying function on the 3-sphere.

For a regular value $t \in \mathbb{R}$, we have the level set $g^{-1}(t)$ and the restriction of f to this level set, $f_t : g^{-1}(t) \to \mathbb{R}$. The *Jacobi curve* of f and g is the closure of the set of critical points of the functions f_t, for all $t \in \mathbb{R}$. The closure operation adds the critical points of f restricted to level sets at critical values, as well as the critical points of g, which form singularities in these level sets. Fig. 2 illustrates the definition by showing the Jacobi curve of two smooth functions on a piece of the two-dimensional plane.

We can now specialize the definition of Jacobi sets to a time-varying function. Consider a 1-parameter family of Morse functions on the 3-sphere, $f : \mathbb{S}^3 \times \mathbb{R} \to \mathbb{R}$, where the extra dimension in the domain is time. We can use the general definition of Jacobi sets described above by introducing an auxiliary function $g : \mathbb{S}^3 \times \mathbb{R} \to \mathbb{R}$ defined by $g(x, t) = t$. A level set has the form $g^{-1}(t) = \mathbb{S}^3 \times t$, and the restriction of f to this level set is $f_t : \mathbb{S}^3 \times t \to \mathbb{R}$. Generically, the function f_t is Morse, but there are discrete values of t at which it violates one or both Genericity conditions of Morse functions. In the next section, we will see that the Reeb graph of f_t undergoes combinatorial changes at these values. The Jacobi curve of f and g may consist of several components, and in the assumed generic case each is a closed 1-manifold.

We can identify the *birth–death points* where the level sets of f and g and the Jacobi curve have a common normal direction. To understand these points, imagine a level set in the form of a (two-dimensional) sphere deforming, sprouting a bud, as we go forward in time. The bud has two critical points, one a maximum and the other a 2-saddle. At the time when the bud just starts sprouting there is a point on the sphere, a birth point, where both these critical points are born. Run this in reverse order to understand a death point. We decompose the Jacobi curve into *segments* by cutting it at the birth–death points. The index of the critical point tracing a segment is the same everywhere along the segment. The indices within two segments that meet at a birth–death point differ by one:

Index Lemma. *Let $f : \mathbb{M} \times \mathbb{R} \to \mathbb{R}$ be a 1-parameter family of Morse functions. The indices of two critical points created or destroyed at a birth–death point differ by exactly one.*

Proof. At time t, let f_t have a single birth point. We can choose a small positive ε such that there are no other birth–death points with time in $[t - \varepsilon, t + \varepsilon]$. Denote by x and y the two newly created critical points in $f_{t+\varepsilon}$ and

Table 1
Classification of vertices into regular and simple critical points using the reduced Betti numbers of the lower link

	$\tilde{\beta}_{-1}$	$\tilde{\beta}_0$	$\tilde{\beta}_1$	$\tilde{\beta}_2$
regular	0	0	0	0
minimum	1	0	0	0
1-saddle	0	1	0	0
2-saddle	0	0	1	0
maximum	0	0	0	1

let $k = \operatorname{index} x \leqslant \operatorname{index} y$. The point x either destroys a homology class of dimension $k-1$ or it introduces one of dimension k. The former case is ruled out as $\operatorname{index} y \geqslant k$, and a cell of dimension larger than or equal to k cannot compensate for the destroyed dimension $k-1$ class. In the latter case, when x creates a homology class of dimension k, we need a $(k+1)$-cell to cancel the homology class, which implies that $\operatorname{index} y = k+1$. The claim follows. □

Piecewise linear functions. A *triangulation* of a manifold $\mathbb{M}$ is a simplicial complex, K, whose underlying space is homeomorphic to $\mathbb{M}$ [2]. Given values at the vertices, we obtain a continuous function on $\mathbb{M}$ by linear interpolation over the simplices of the triangulation. We need some definitions to talk about the local structure of the triangulation and the function. The *star* of a vertex u consists of all simplices that share u, including u itself, and the *link* consists of all faces of simplices in the star that are disjoint from u. The *lower link* is the subset of the link induced by vertices with function value less than u:

$$\begin{aligned}
&\operatorname{St} u = \{\sigma \in K \mid u \subseteq \sigma\},\\
&\operatorname{Lk} u = \{\tau \in K \mid \tau \subseteq \sigma \in \operatorname{St} u,\ u \notin \tau\},\\
&\operatorname{Lk}_- u = \big\{\tau \in \operatorname{Lk} u \mid v \in \tau \Rightarrow f(v) \leqslant f(u)\big\}.
\end{aligned}$$

Critical points of piecewise linear functions have been introduced by Banchoff [4] as the vertices whose lower links have Euler characteristic different from 1. Our classification is finer than Banchoff's and based on the reduced Betti numbers of the lower link. The *k-th reduced Betti number*, denoted as $\tilde{\beta}_k$, is the rank of the k-th reduced homology group of the lower link: $\tilde{\beta}_k = \operatorname{rank} \tilde{\mathsf{H}}_k$. The reduced Betti numbers are the same as the usual (un-reduced) Betti numbers, except that $\tilde{\beta}_0 = \beta_0 - 1$ for non-empty lower links, and $\tilde{\beta}_{-1} = 1$ for empty lower links [21]. When the link is a 2-sphere only $\tilde{\beta}_{-1}$ through $\tilde{\beta}_2$ can be non-zero. Simple critical points have exactly one non-zero reduced Betti number, which is equal to 1; see Table 1. The first case in which this definition differs from Banchoff's is a double saddle obtained by combining a 1- and a 2-saddle into a single vertex. The Euler characteristic of the lower link is unity, which implies that Banchoff's definition does not recognize it as critical. A *multiple saddle* is a critical point that falls outside the classification of Table 1 and therefore satisfies $\tilde{\beta}_{-1} = \tilde{\beta}_2 = 0$ and $\tilde{\beta}_0 + \tilde{\beta}_1 \geqslant 2$. It can be unfolded into simple 1-saddles and 2-saddles by modifying the simplicial complex, as explained in [6,15].

3. Time-varying Reeb graphs

In this section, we characterize how the Reeb graph of a function changes with time. Specifically, we give a complete enumeration of the combinatorial changes that occur for a Morse function on $\mathbb{S}^3$.

Jacobi curves connect Reeb graphs. Let R_t be the Reeb graph of f_t, the function on $\mathbb{S}^3$ at time t. The nodes of R_t correspond to the critical points of f_t, and as we vary t, they trace out the segments of the Jacobi curve. The segments connect the family through time, giving us a mechanism for identifying nodes in different Reeb graphs. We illustrate this idea in Fig. 3.

Generically, the function f_t is Morse. However, there are discrete moments in time at which f_t violates one or both Genericity Conditions of Morse functions and the Reeb graph of f_t experiences a combinatorial change. Since we have only one varying parameter, namely time, we may assume that there is only a single violation of the Genericity Conditions at any of these discrete moments, and there are no violations at all other times. Condition I is violated

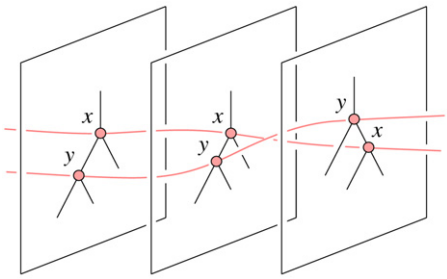

Fig. 3. Reeb graphs at three moments in time whose nodes are connected by two segments of the Jacobi curve.

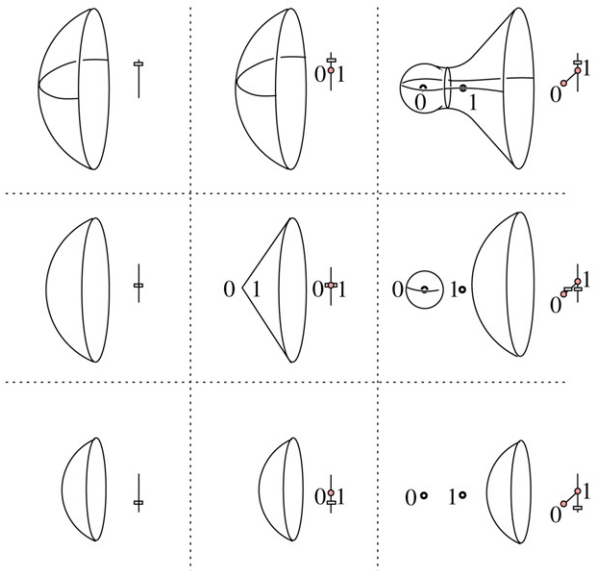

Fig. 4. Level sets and Reeb graphs around a 0–1 birth point. Time increases from left to right and the level set parameter, indicated by a rectangular slider bar, increases from bottom to top. Going forward in time, we see an arc sprouting in the Reeb graph, while going backward in time we see it retracting.

whenever f_t has a birth–death point at which a cancellation annihilates two converging critical points or an anti-cancellation gives birth to two diverging critical points. Condition II is violated whenever f_t has two critical points $x \neq y$ with $f_t(x) = f_t(y)$ that form an *interchange*. The two critical points may be independent and have no effect on the Reeb graph, or they may belong to the same level set component of f_t and correspond to two nodes that swap their positions along the Reeb graph. We now analyze the changes caused by birth–death points and by interchanges in detail. We refer to the moments in time when these changes occur as birth–death events and interchange events, respectively.

Nodes appear and disappear. When time passes the moment of a birth point, we get two new critical points and correspondingly two new nodes connected by an arc in the Reeb graph. By the Index Lemma, the indices of the two critical points differ by one, leaving three possibilities: 0–1, 1–2, and 2–3. Consider first the 0–1 case in which a minimum and a 1-saddle are born; see Fig. 4. In the Reeb graph, we get a new degree-1 node that starts an arc ending at a new degree-3 node. In other words, the Reeb graph sprouts a new arc downward from an existing branch. The 2–3 case is upside-down symmetric to the 0–1 case, with the Reeb graph sprouting a new arc upward from an existing branch.

Consider second the 1–2 case in which a 1-saddle and a 2-saddle are born. As illustrated in Fig. 5, this event corresponds to the appearance of a short-lived handle in the evolution of level sets. In the Reeb graph we get two new degree-2 nodes that effectively refine an arc by decomposing it into three arcs. Turning the picture upside-down does

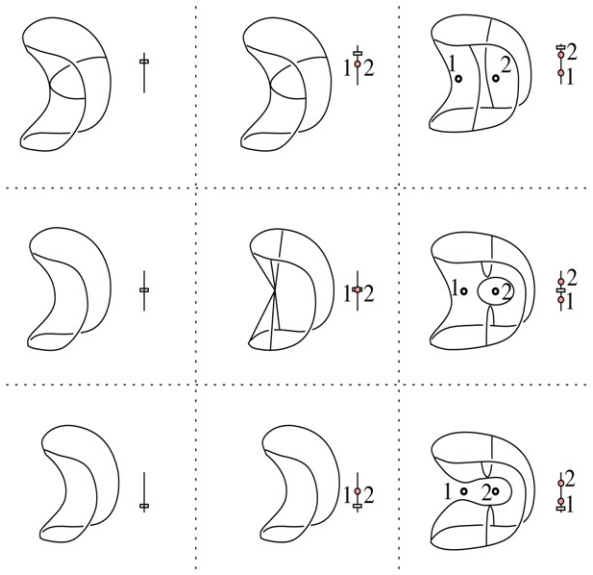

Fig. 5. Level sets and Reeb graphs around a 1–2 birth point. Time increases from left to right and the level set parameter increases from bottom to top. Going forward in time, we see a refinement of an arc in the Reeb graph and going backward we see a coarsening.

not change anything, which shows that the case is symmetric to itself. We have three similar cases when time passes the moment of a death point. Two critical points of f_t converge and annihilate when they collide, and correspondingly an arc of the Reeb graph contracts to a point, effectively removing its two nodes. The 0–1 and 2–3 cases are illustrated in Fig. 4, which we now read from right to left, and the 1–2 case is illustrated in Fig. 5, which we also read backward, from right to left.

Nodes swap. Reeb graph nodes swap position in the Reeb graph when the corresponding critical points, x and y, form an interchange and, at that moment, belong to the same level set component. Assume without loss of generality that $f_{t-\varepsilon}(x) < f_{t-\varepsilon}(y)$ and $f_{t+\varepsilon}(x) > f_{t+\varepsilon}(y)$. We have four choices for each of x and y depending on whether they add or remove a handle, merge two level set components or split a level set component. This gives a total of sixteen configurations. We analyze possible before and after combinations and pair them, giving us the cases illustrated in Fig. 6. It is convenient to group the cases with similar starting configurations together. We use $+$, $-$, M, S to denote 'handle addition', 'handle deletion', 'component merge', and 'component split', respectively, for each of x and y.

`Case 1` ($++, +-, -+, --$) Both x and y change the genus and their corresponding nodes simply swap their positions in the Reeb graph. [We pair $++$ with itself to get `Case 1a`, $+-$ with $-+$ to get `Case 1b`, and $--$ with itself to get `Case 1c`.]

`Case 2` (+M, M+, −M, M−) We consider two sub-cases.

- (M+) Before the swap, x merges two components and y adds a handle. Either y involves the two components that were merged by x, so x and y just swap, or y involves only one of the two components, so y goes down one of the branches at x. [In the first configuration, we pair M+ with itself to get `Case 2a`, and in the second we pair M+ with +M to get `Case 2b`.]
- (M−) Before the swap, x merges two components and y removes a handle. After the swap, y moves down one of the branches at x. [We pair M− with −M to get `Case 2c`.]

`Case 3` (−S, S−, +S, S+) We consider two sub-cases.

- (−S) Before the swap, x deletes a handle and y splits the component. Either y breaks a handle and x splits the component into two, so the nodes x and y swap, or x involves only one of the two components

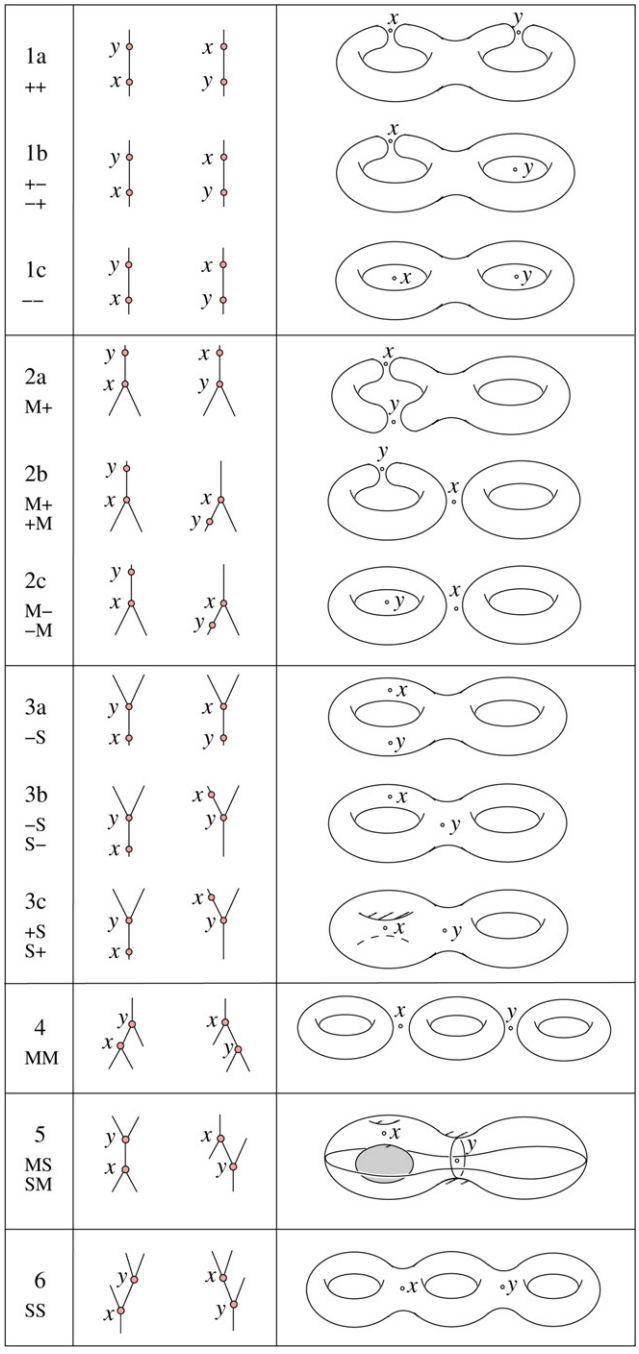

Fig. 6. On the left, Reeb graph portions before and after the interchange x and y. On the right, level sets at a value just below the function value of x and y. In each case, the index of a critical point can be inferred from whether the level set merges (index 1) or splits (index 2) locally at the critical point.

split by y, so node x goes up one of the branches at node y. [In the first configuration, we pair $-$S with itself to get `Case 3a`, and in the second we pair $-$S with S$-$ to get `Case 3b`.]

(+S) Before the swap, x adds a handle and y splits the component. After the swap, x moves up one of the branches at y. [We pair +S with S+ to get `Case 3c`.]

`Case 4` (MM) Three components merge into one, and the only change between before and after is the order of merging. [We pair MM with itself.]

`Case 5` (MS, SM) Before the swap, x merges two components and y splits the merged component. After the swap, y splits one of the components which merge at x before the swap, and x merges one of the split components with the remaining component. [We pair MS with SM.]

`Case 6` (SS) A component splits into three, and the only change between before and after is the order of splitting. [We pair SS with itself.]

The pairing of cases indicates a symmetry between before and after configurations. There is also the symmetry we observe when we exchange inside with outside. Equivalently, we substitute $-f$ for f, which turns the Reeb graph upside-down, exchanging minima with maxima and 1-saddles with 2-saddles.

4. Algorithm for time-varying Reeb graphs

In this section, we introduce an algorithm for maintaining a Reeb graph through time. The algorithm is explained at the abstract level without going into implementation details. Section 5 will describe an adaptation of the algorithm to the piecewise linear case.

Data types. We represent **time** by a conventional priority queue storing *birth–death* and *interchange events* prioritized by the moments in time they occur. At a given moment, t, the time data type supports the following operations:

INSERT(e) : add the future event e (it occurs after time t);
NEXTEVENT : return the earliest, top priority event and delete it from the queue;
DELETE(e) : delete the event e from the queue.

We maintain the **Reeb graph** as a collection of nodes, and arcs that connect the nodes. Each node knows about its incident arcs and about the segment of the Jacobi curve that contains the corresponding critical point. Each arc knows its start-node and end-node and the time when they will die at a death point or swap at an interchange. At a given moment in time, t, the Reeb graph data type supports the following operations:

SEGMENT(x) : return the segment of the Jacobi curve that contains the critical point that corresponds to node x;
NODES(a) : return the start-node and the end-node of arc a;
REMOVEARC(a) : remove the arc a from the Reeb graph;
ADDARC(x, y) : add an arc connecting the nodes x and y in the Reeb graph.

We have similar operations for removing and adding nodes, which are invoked whenever we remove or add arcs. The **Jacobi curve** is stored as a collection of segments joined at shared birth–death points. Each segment knows its endpoints, the index of its critical point, and the corresponding node in the Reeb graph, if any. Each birth–death point knows its incident segments. At a given moment in time, t, the data type for the Jacobi curve supports the following operations:

NODE(γ) : return the node in the Reeb graph that corresponds to the critical point on the segment γ;
NEXTXING(γ, γ', t) : return the next interchange (after time t) of the critical points tracing the segments γ and γ'.

Finally, the Jacobi curve data type supports the operation BDPOINTS that returns all birth–death points of the function f.

Sweeping time. We use the operations provided by the various data types to maintain the Reeb graph of f_t through time. We assume that data is available in a finite range, from time 0 to 1. Starting with the Reeb graph R_0 at time $t = 0$, we maintain R_t by sweeping forward in time, using the Jacobi curve as a path for its nodes. The time data type is initialized by inserting all birth–death points provided by BDPOINTS. Interchange events are inserted and deleted as

arcs appear and disappear in the Reeb graph. Events are processed in the order they are returned by repeated execution of the NEXTEVENT operation.

`Case` birth event. The type is 0–1, 1–2, or 2–3 and can be determined from the indices of the two segments that meet at the birth point u on the Jacobi curve. Next, we determine the arc a to modify. Arc a contains the representative point of the level set through birth point u. We defer the description of how to determine this arc till the next section. Finally, for cases 0–1, and 2–3 we refine a and sprout a bud, and for case 1–2 we refine a by decomposing it into three arcs.

`Case` death event. We retrieve the nodes in the Reeb graph that correspond to the two segments γ and γ' that share the death point on the Jacobi curve: $x = \text{NODE}(\gamma)$ and $y = \text{NODE}(\gamma')$. Then we contract the arc connecting x and y to a point and finally delete this point by removing three arcs and adding one.

`Case` interchange event. We swap the two nodes x and y that correspond to the critical points of the interchange by removing and adding arcs as indicated in Fig. 6. Determining which arcs below x or above y to remove is equivalent to deciding between sub-cases of the interchange event. Such a decision is needed in `Cases 2` to `6`; We defer the description of how to make these decisions to the next section.

As mentioned earlier, each arc removal implies the deletion of an interchange event, and each arc addition implies the insertion of one into the time data type. Determining arcs to modify during birth events and interchange events requires global information on level set connectivity. In the next section, we will use a PL mesh to specify the PATH operation required to determine these arcs.

5. PL implementation

In this section, we describe how to implement the algorithm of Section 4 for a piecewise linear function defined on a triangulation of the 3-sphere cross time.

Data structures. We can now describe specific data structures implementing the three abstract data types: time, Reeb graph, and Jacobi curve. A standard implementation of the priority queue, such as a binary heap, will do for time, and a standard graph representation will do for the Reeb graph [1]. The Jacobi curve is represented by cyclic lists of edges in the input triangulation, K. Each cycle is decomposed into segments of maximal linear lists of edges that are monotonic in time.

Let K_t denote the three-dimensional slice at time t of the four-dimensional triangulation K. The vertices of K_t are points on edges of K, the edges are slices of triangles, etc. We require the following operation to determine arcs to modify during birth events and to distinguish between the various configurations at an interchange event.

PATH(u) : return a monotone path connecting leaves in the Reeb graph. The path contains the point representing the level set component of f_t passing through the vertex u, which is either on the Jacobi curve or in the link of such a point.

To compute this path, we walk in the 1-skeleton of K_t, in the direction of increasing f_t, until we reach a critical vertex x. Similarly, we walk in the direction of decreasing f_t until we reach another critical vertex y. Observe that x and y are also nodes in the Reeb graph, R_t, and delimit the desired path.

Determining arcs to modify at birth and interchange events. A birth event modifies an arc a which we can find on PATH(u), where u is the birth point. Interchange events have various subcases which we can distinguish using the PATH operation in the following manner: we consider `Case 2` illustrated in Fig. 6. we distinguish between `Case 2a` and `2b` using the PATH operation; Case 2c can be distinguished using the index of y. In K_t, points x and y have the same function value and the lower link of y has two components. Letting u and v be a vertex in each, we compute PATH(u) and PATH(v). Next, we compute arc a in PATH(u), and arc b in PATH(v), both arcs incident to and below x. We have `Case 2a` iff $a \neq b$. We distinguish between `Cases 2b` and `2c` using the index of y and identify $a = b$ as the arc below x to be refined by y in the new Reeb graph. `Case 4` is similar, except that the only decision to be made is which arc below x gets refined by y. `Cases 3` and `6` are upside-down symmetric to `Cases 2` and `4` and

are distinguished by calling PATH for vertices in the upper link components of x. Finally, `Case 5` is slightly different and decided by calling PATH for x and for y. We determine arc a in PATH(x) incident to and above y, and arc b in PATH(y) incident to and below x, to modify at the interchange.

Initialization, sweep, and construction. We begin by constructing the Jacobi curve as a collection of edges in K using the algorithm in [13]. This provides the collection of birth–death points, which we use to initialize the priority queue representation of time. We also construct the Reeb graph at time zero from scratch, using the algorithm in [6], which is similar to the forward-backward sweep algorithm for computing Betti numbers in [11]. The latter algorithm also detects when 1-cycles are created and destroyed, which is the information we need to add the degree-2 nodes to the Reeb graph, which is not part of the former algorithm. The last step in preparation for the sweep through time inserts the interchange events that correspond to arcs in the Reeb graph into the priority queue. Specifically, for each arc a in R_0, we get $(x, y) = \text{NODES}(a)$, $\gamma = \text{SEGMENT}(x)$, $\gamma' = \text{SEGMENT}(y)$ and we insert the interchange returned by $\text{NEXTXING}(\gamma, \gamma', 0)$ into the priority queue.

The sweep is now easy, repeatedly retrieving the next event, updating the Reeb graph, and deleting and inserting interchange events as arcs are removed and added. We think of the sequence as the evolution of a single Reeb graph. Following Driscoll et al. [12], we accumulate the changes to form a single data structure representing the entire evolution, which we refer to as the *partially persistent Reeb graph*. We adhere to the general recipe, using a constant number of data fields and pointers per node and arc to store time information and keep track of the changes caused by an update. In addition, we construct an array of access pointers that can be used to retrieve the Reeb graph at any moment in time proportional to its size.

Analysis. The running time of the algorithm can be expressed in terms the following parameters

$N =$ number of simplices in K, the triangulation of the space–time data;

$n =$ upper bound on the number of simplices in a slice K_t of K;

$E =$ number of birth–death and interchange events;

$k =$ number of edges of the Jacobi curve.

We have $n \leqslant N$, $k \leqslant N$, and $E \leqslant k^2$, assuming the triangulation is fine enough to resolve the Jacobi curve as a disjoint collection of simple cycles. For typical input data, the left side will be significantly smaller than the right side in all three inequalities. To construct the Jacobi curve, we compute the reduced Betti numbers of the lower link of each edge in time $O(\ell)$, where ℓ is the size of the link. The total size of all links is proportional to N, which implies a running time of $O(N)$. The birth–death points are inserted into the priority queue in $O(\log n)$ time each. The initial Reeb graph is constructed in time $O(n \log n)$, inserting the initial batch of interchange events in time $O(n \log n)$ also. The sweep iterates through E events, each in time $O(n)$ needed to determine the after configuration of the event. In addition, we use time $O(k)$ to move the nodes of the Reeb graph along the chains of edges representing the segments of the Jacobi curve. In total, the running time is $O(N + En)$. We construct the partially persistent data structure representing the evolution of the Reeb graph in the same time. The size of that data structure is proportional to the size of the initial Reeb graph plus the number of events, which is $O(n + E)$.

An obvious place to improve the running time is to improve the time needed to do a PATH operation. Is there a data structure that can return (the endpoints of a) path in time $O(\log n)$? If so then the total running time would improve to $O(N + E \log n)$.

6. Betti numbers of level sets

The Betti numbers of a space count various topological features. For the level sets of regular values of f_t, which are 2-manifolds, β_0 is the number of connected components, β_1 is the number of tunnels, and β_2 is the number of voids. (For a d-dimensional space, only β_0 through β_d may be non-zero.)

There is coherence for the set of Betti numbers along an arc of R_t. There is also coherence for how these numbers change as the Reeb graph evolves over time. Because level set components change topology only at critical points, all level set components in the family represented by an open Reeb graph arc (excluding its end-nodes) of R_t have the same set of Betti numbers. If we equip each arc of the time-varying Reeb graph with the set of Betti numbers of its

family of level set components, then we can obtain Betti numbers for any level set component at any time. In other words, the Betti numbers can be parametrized by time t and level value s as $\beta_i(s,t)$. The arc of Reeb graph R_t that represents a component of the level set at level value s contains the set of Betti numbers for that component. We will however continue to use the unparametrized notation β_i for simplicity.

For a regular value of f_t each level set component is a closed 2-manifold, giving $\beta_0 = \beta_2 = 1$ for each arc; only the value of β_1 must be computed. Before we compute the value of β_1 for each arc of the time-varying Reeb graph, we first consider the function f_t and classify the effect of its critical points on level sets, and their β_1 value. We can then vary t and understand how each birth–death, and interchange event that modifies the Reeb graph changes the β_1 value of its arcs.

Critical points change the β_1 value of level set components. We study the action of a critical point x on the topology of level set components as we sweep function value f_t past its value from below. This tells us how the β_1 value of the arcs incident to and above x are related to the β_1 value of the arcs incident to and below x.

If x is a minimum it creates a new component homeomorphic to $\mathbb{S}^2$; the arc incident to and above it has Betti number $\beta_1 = 0$.

If x has index-1 then it can merge two level set components into one, or add a handle to a single level set component. When two level set components merge, we have the connected sum of two 2-manifolds. We derive the relationship between their Betti numbers before and after the merge.

Connected sum Lemma. *The Betti number β_1 of the connected sum of 2-manifolds $\mathbb{M}$, and $\mathbb{N}$, without boundary, is the sum of their Betti number β_1:* $\beta_1^{\mathbb{M}\#\mathbb{N}} = \beta_1^{\mathbb{M}} + \beta_1^{\mathbb{N}}$.

Proof. The Euler characteristic of the connected sum is $\chi(\mathbb{M}\#\mathbb{N}) = \chi(\mathbb{M}) + \chi(\mathbb{N}) - 2$. Expressing each of the Euler characteristics in terms of Betti numbers, and noting that $\beta_0 = \beta_1 = 1$ for a 2-manifold without boundary yields the formula $\beta_1^{\mathbb{M}\#\mathbb{N}} = \beta_1^{\mathbb{M}} + \beta_1^{\mathbb{N}}$. □

If x merges two components then according to the connected sum lemma the β_1 value of the arc incident to and above it is the sum of the β_1 value of the arcs incident to and below it.

If x adds a handle we can understand the change in the β_1 value using the relationship between the value of β_1 and the genus g: $\beta_1 = 2g$. Adding a handle increases the value of β_1 by 2; the β_1 value of the arc incident to and above x is larger by 2 than the β_1 value of the arc incident to and below it.

If x is a maximum or an index-2 critical point its action on value of β_1 is upside down symmetric to that when it is a minimum, or an index-1 critical point respectively.

With this understanding of the action of critical points on the value of β_1, we investigate how to update the Reeb graph over time at birth–death and interchange events.

The β_1 value changes over time. At a 0–1 birth event, set the β_1 value of the new arc to 0 according to our classification; the β_1 value of the other arcs incident to the new index-1 saddle are the same as that of the deleted arc. Handle a 2–3 birth similarly. Fig. 7 shows β_1 value at a 0–1 birth on the left; the 2–3 birth is upside down symmetric to it.

At a 1–2 birth event, set the β_1 value of the new arc to the β_1 value of the arc incident to and below its index-1 node, and increment it by 2; the β_1 value of the other arcs incident to the new nodes are the same as that of the deleted arc. Fig. 7 shows the β_1 value at a 1–2 birth on the right.

Handle a death event as if running the corresponding birth event in reverse. Set the β_1 value of the new arc to the β_1 value of one of the deleted arcs that did not connect the destroyed pair of nodes. Going backwards in time in Fig. 7 shows the β_1 value at death events.

We now consider updates during interchange events. We can analyze each interchange case using the action of critical points on the value of β_1 developed earlier, to yield the updates to value of β_1 shown in Fig. 8.

Consider `case 1b` in Fig. 8 and observe how the β_1 value of the level set component changes as we sweep up the level value from below. Before the interchange, node '+' adds a handle increasing the β_1 value by 2, and node '−' deletes a handle restoring the β_1 value to the start value. After the interchange, the effects are reversed; node '−' deletes a handle, following which node '+' adds a handle. For a more complicated case, consider `case 2b`. Before the interchange, the merge node first creates a connected sum of two components with their β_1 values summed,

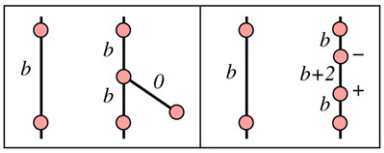

Fig. 7. Updating the value of β_1 at a birth–death event. As described in Section 3, a '+' means handle addition, and a '−' means handle deletion. For each picture time increases to the right. The β_1 value of arcs at a 0–1 birth event at left, and at a 1–2 birth event at right. A 2–3 birth is upside down symmetric to the 0–1 birth. Running in reverse gives us the corresponding death cases.

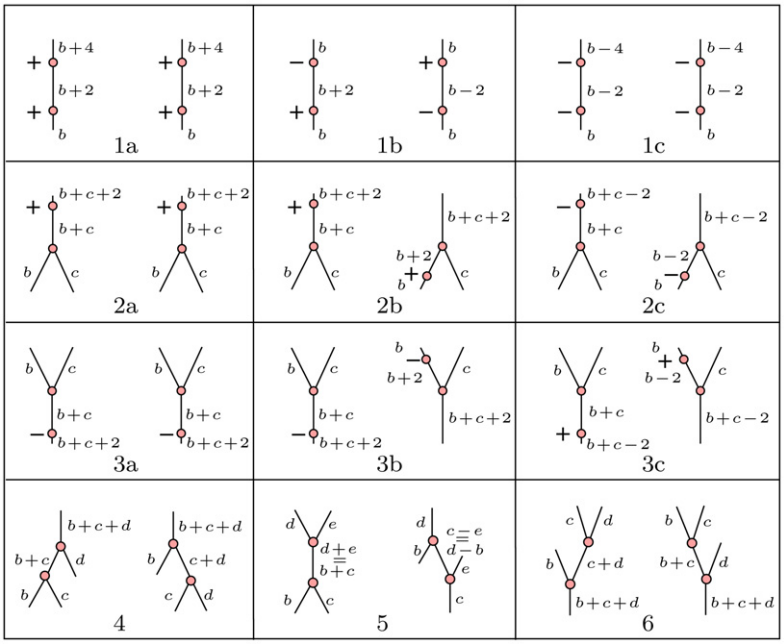

Fig. 8. Updates to β_1 value for each interchange case. As described in Section 3, a '+' means handle addition, and a '−' means handle deletion. For clarity, we omit the x, y labels of the interchanging nodes. The letters b, c, d, e denote the β_1 values of the arcs.

following which node '+' adds a handle. After the interchange, because node '+' slides down one of the arcs incident to the merge node, it adds a handle to one of the merging components first, followed by the action of the merge node.

We also observe upside-down symmetry in several cases; `case 1a` with `1c`, `case 2` with `3`, and `case 4` with `6`.

We are now ready to compute the β_1 value for each arc of the time-varying Reeb graph.

Computing the β_1 value over time. We can compute β_1 value for each arc during the forward sweep in time that constructs the time-varying Reeb graph. Equip each arc of the Reeb graph R_0 at time $t = 0$ with its β_1 value using the algorithm by Pascucci et al. [22]. Maintain β_1 value for each arc of R_t during every birth–death and interchange event of the sweep according to the update policy developed before.

7. Path seeds over time

Path seeds, developed by Carr et al. [7], can be used for fast level set extraction from a static dataset; we wish to compute path seeds for fast level set extraction from time-varying data.

7.1. Static path seeds

We define static path seeds and compute them; our description enables us to define and compute time-varying path seeds.

Definition 7.1 *(Descending path).* A *descending path* beginning at vertex u of the slice K_t is an edge connected sequence of vertices with decreasing value of f_t.

For the next definition, recall that each node x of the Reeb graph R_t maps to a critical point of f_t, which we also call x.

Definition 7.2 *(Path seed).* A *path seed* of an arc of R_t with upper node x is a vertex $v \in K_t$ such that every descending path which begins at x and contains u intersects all the level set components that the arc represents.

This definition does not suggest either how such a vertex u may be found, or how such a descending path that contains both x and u may be found. As will be explained soon, vertex u turns out to be either x itself or a vertex in its lower link; finding a descending path through x and u becomes straightforward.

Definition 7.3 *(Seed edge).* A *seed edge* of a level set component is an edge in K_t that intersects it.

If each arc of the Reeb graph is equipped with a path seed, we can find a seed edge for any level set component it represents in any descending path which begins at its upper vertex x and contains its path seed. This level set component can be extracted by visiting only those simplices that intersect it, starting from the simplices incident to the seed edge.

A path seed maps an individual arc to its level sets' component. In their flexible isosurface interface, Carr et al. use this mapping to selectively extract user specified components for one, or several level values. Thus path seeds have an advantage over conventional seed set methods [17] as the latter do not maintain the required mapping.

Choosing path seeds. We describe how to choose a path seed for an arc a of a static Reeb graph depending on the type of the arc's upper node x.

`Case` S1: If x is a maximum, a split node, a '+' node, or a '−' node, it has one arc incident to and below it; choose vertex x itself as the path seed for arc a because all descending paths that begin at x intersect all level set components that arc a represents.

`Case` S2: If x is a merge node, it has two arcs incident to and below it, and vertex x has two lower link components. A descending path that starts at x and contains a vertex u of one of its lower link components intersects all the level set components represented by one of the two arcs; a descending path from x that contains a vertex of the other lower link component intersects all the level set components of the remaining arc; match a descending path to its arc and use the lower link vertex in that path as the arc's seed path. Thus path seeds also map each lower link component of x to its two arcs below.

We can choose path seeds in case S2 using the PATH operation described in Section 5 and restated here.

PATH(u) : return a monotone path connecting leaves in the Reeb graph. The path contains the point representing the level set component of f_t passing through the vertex u, which is either on the Jacobi curve or in the link of such a point.

Letting u and v be a vertex in each lower link component of x, compute the arcs a and b incident to and below x in PATH(u) and PATH(v); choose u as the path seed for a, and v as the path seed for b.

7.2. Time-varying path seeds

We are ready to extend the definition of path seeds to time-varying Reeb graphs. Define the *lifespan* of an arc in the time-varying Reeb graph as the time interval between its creation and its deletion.

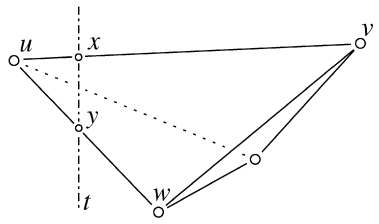

Fig. 9. In slice K_t, vertex y belongs to the link of x.

Definition 7.4 *(Time-varying Path Seeds).* A *time-varying path seed* of an arc a of the time-varying Reeb graph is an edge in the triangulation K such that it contains a path seed for arc a in every Reeb graph R_t in its lifespan.

If we equip each arc of the time-varying Reeb graph with a time-varying path seed then, we can find a seed edge for any level set component at any time. But first, we must choose the time-varying path seed of an arc a depending on its upper node x just like we did in the static case; we prove a lemma on the link structure of a vertex in the slice K_t to assist us in our choice. This analysis enables us to choose the time-varying path seed of new arcs created at birth–death and interchange events, and we can compute time-varying path seeds for each arc of the time-varying Reeb graph.

Choosing time-varying path seeds. For the slice K_t shown in Fig. 9, We analyze the condition under which vertex y belongs to the lower link of vertex x. In the proof, recall that the function g represents the time.

Lower Link Lemma. *Let vertices x and $y \in K_t$ lie on edges uv and $uw \in K$ respectively. Vertex y is in the lower link of critical vertex x whenever vertex w is in the lower link of Jacobi edge uv.*

Proof. Vertex y is in the lower link of x if and only if the following condition holds.

$$\frac{f(w)-f(u)}{g(w)-g(u)} < \frac{f(v)-f(u)}{g(v)-g(u)}.$$

Letting $\lambda = (f(u)-f(v))/(g(v)-g(u))$, this reduces to $f(w)+\lambda g(w) < f(u)+\lambda g(u)$. This is exactly the condition for w to be in the lower link of uv when it is a Jacobi edge [13]. □

As described in the analysis next, this lemma helps us choose time-varying path seeds for merge nodes.

`Case` T1: If x is a maximum node, a split node, a '+' node, or a '−' node, it has one arc incident to and below it for its lifespan; choose the Jacobi edge traced by x itself as the time-varying path seed for a because taking slices in its lifespan gives us case S1 discussed before.

`Case` T2: If x is a merge node, we can use the lower link lemma to choose time-varying path seeds for arcs incident to and below it. For the slice K_t shown in Fig. 9, one vertex y from each lower link component of x can be a path seed for each of its arcs in R_t according to case S2. The lower link lemma implies that vertex y is in the lower link of x for the time interval $(g(u), g(w))$ spanned by edge uw. From each lower link component of uv choose a vertex w and match edge uw to the arc incident to and below x if vertex y is a path seed for that arc in any slice K_t in this time interval. (This matching can be computed as explained in the static case using the PATH operation.) Edge wv is the time-varying path seed for that arc for the time interval $[g(w), g(v))$.

We can choose time-varying path seeds for each new arc created at a birth–death and interchange event; simply determine the case based on the upper node of the new arc and choose as described.

Computing path seeds for the time-varying Reeb graph. We now describe how to compute time-varying path seeds for every arc of the time-varying Reeb graph. The computation is done during the forward sweep in time that constructs the time-varying Reeb graph. Equip each arc of the Reeb graph R_0 with a time-varying path seed according to case T1,

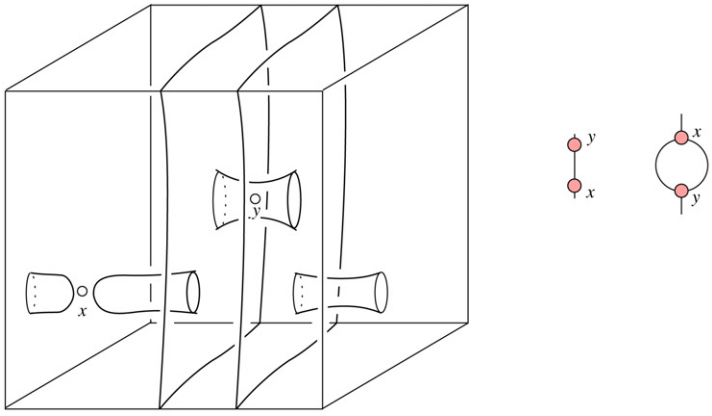

Fig. 10. An interchange that creates a loop. The function f_t is defined on a 3-torus shown here as a cube with opposite faces identified. The level set (bold) at a value below that at x is a pair of nested 2-tori connected by a tunnel around y. Before the interchange, x adds a handle and then y removes the tunnel around it. After the interchange, y removes the tunnel disconnecting the level set and x adds a handle reconnecting the level set. As shown on the right, the Reeb graph has a loop after the interchange.

or T2. Update the time-varying path seeds of new arcs created at each birth–death and interchange event as described before.

8. Analysis for unrestricted functions f, g

In this section, we comment on the difficulties in extending our analysis to the case when f_t is defined on a general 3-manifold, and to the case when function f is restricted to the level sets of another function g defined on the same 4-manifold.

Time-varying function f_t defined on a general 3-manifold. The Reeb graph of a function defined on a general 3-manifold can have loops. The merge and split saddles that define the top and bottom of a loop can interchange over time and one expects the loop to be conserved past an interchange but the saddles to be swapped. Indeed, this is the case for time-varying functions defined on a general 2-manifold because the number of loops remain invariant [10]. However, this is not the case for general 3-manifolds. Fig. 10 shows an example of such an interchange. In the example, we have a time-varying function defined on a 3-torus obtained by identifying the opposite faces of a cube. Critical points x and y will interchange, and the level set at a value just below that of x is a pair of nested 2-tori connected by a tunnel around critical point y. Before the interchange, critical point x adds another handle to the level set by connecting the nested tori, and critical point y removes the tunnel around it. After the interchange, critical point y removes the tunnel connecting the tori to disconnect the level set, and critical point x adds a handle to reconnect the level set, creating a loop in the Reeb graph. Locally, around critical points x and y, this case is similar to `case 1b` in Fig. 6.

Loops also present an algorithmic difficulty because the PATH function cannot be used to distinguish between several possible configurations after an interchange as there can be several paths between leaves in the Reeb graph which is no longer a tree. Thus it is not clear how to distinguish between the two cases shown in Fig. 10 and `case 1b` in Fig. 6.

Unrestricted generic functions f, g defined on a common 4-manifold. In this general setting the 3-manifold $f^{-1}(t)$ undergoes topological change at the critical points of function g. These critical points can have index 0 (minimum), index 1 through 3 (saddles), and index 4 (maximum). An index k critical point adds a k-handle to $f^{-1}(t)$ along the product of a $(k-1)$-sphere and a $(4-k)$-ball. Thus, a minimum of g adds a new disjoint component to $f^{-1}(t)$ that grows from a point into a component homeomorphic to $\mathbb{S}^3$; the Reeb graph gains a new arc connecting the maximum and minimum of f_t on this new component. Similarly, a maximum of g removes a component from $f^{-1}(t)$; the Reeb graph loses the arc corresponding to the destroyed component. An index 1 saddle of g adds a 1-handle along the product of $\mathbb{S}^0$ and a 3-ball, which is two disjoint 3-balls, locally merging $f^{-1}(t)$. If the critical point acts on two

disconnected components of $f^{-1}(t)$ then two disconnected graphs are connected in the Reeb graph. If the critical point acts on a single component of $f^{-1}(t)$ then the Reeb graph gains a loop. An index 3 saddle of g acts in a manner opposite to an index 1 saddle; it locally splits $f^{-1}(t)$. Thus the Reeb graph gets a new connected component (by a single component splitting off) or it loses a loop. An index 2 saddle of g adds a 2-handle along the product of $\mathbb{S}^1$ and a 2-ball, which is a solid torus. It is not clear how the action of an index 2 critical point affects the Reeb graph. We leave this as future work.

9. Conclusion

The main contribution of this paper is the classification of the combinatorial changes in the evolution of the Reeb graph of a generic time-varying Morse function on $\mathbb{S}^3$. We establish a connection between the time-series of Reeb graphs and the Jacobi curve defined by the time-varying function. Using this connection, we describe an algorithm that maintains the Reeb graph for piecewise linear data. Letting n be the upper bound on the number of simplices in the triangulation of $\mathbb{S}^3$, this algorithm takes time $O(n)$ per combinatorial change in the Reeb graph. While maintaining the Reeb graph, we construct a partially persistent data structure of size proportional to the initial Reeb graph plus the number of events that represent the entire evolution. Given a moment in time, t, we can use this data structure to retrieve the Reeb graph R_t in time logarithmic in the number of events plus linear in its size. We can also augment the time-varying Reeb graph with the Betti numbers of level set components, and with path seeds for fast level set extraction.

Both our case analysis of events and our algorithm are limited to $\mathbb{S}^3$ and to a function on $\mathbb{S}^3$ that varies with time. We comment on the difficulties in extending the analysis and the algorithm to a time-varying function on a general 3-manifold, and to a function f restricted to the level sets of another function g defined on the same 4-manifold. Additional problems can be formulated by increasing the number of dimensions and the number of functions. Future research includes algorithms to augment the time-varying Reeb graph with quantitative information about level set geometry such as surface area and enclosed volume.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.
[2] P.S. Alexandrov, Combinatorial Topology, Dover, Mineola, NY, 1998.
[3] C.L. Bajaj, V. Pascucci, D.R. Schikore, The contour spectrum, in: Proc. IEEE Conf. Visualization, 1997, pp. 167–175.
[4] T.F. Banchoff, Critical points for embedded polyhedral surfaces, Amer. Math. Monthly 77 (1970) 457–485.
[5] K.G. Bemis, D. Silver, P.A. Rona, C. Feng, Case study: a methodology for plume visualization with application to real-time acquisition and navigation, in: Proc. IEEE Conf. Visualization, 2000, pp. 481–494.
[6] H. Carr, J. Snoeyink, U. Axen, Computing contour trees in all dimensions, Comput. Geom. 24 (2) (2003) 75–94.
[7] H. Carr, J. Snoeyink, Path seeds and flexible isosurfaces: Using topology for exploratory visualization, in: Proc. of Eurographics Vis. Sympos. 2003, 2003, pp. 49–58.
[8] Y.-J. Chiang, T. Lenz, X. Lu, G. Rote, Simple and optimal output-sensitive construction of contour trees using monotone paths, Comp. Geom. Theory Appl. 30 (2) (2005) 165–195.
[9] T. Chiueh, K.-L. Ma, A parallel pipelined renderer for time-varying volume data, in: Proc. Parallel Arch., Alg., Networks, 1997, pp. 9–15.
[10] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, V. Pascucci, Loops in Reeb graphs of 2-manifolds, in: Proc. 14th Ann. Sympos. Comput. Geom., 2003, pp. 344–350.
[11] C.J.A. Delfinado, H. Edelsbrunner, An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere, Comput. Aided Geom. Design 12 (1995) 771–784.
[12] J.R. Driscoll, N. Sarnak, D.D. Sleator, R.E. Tarjan, Making data structures persistent, J. Comput. Sys. Sci. 38 (1989) 86–124.
[13] H. Edelsbrunner, J. Harer, Jacobi sets of multiple Morse functions, in: F. Cucker (Ed.), Foundations of Computational Mathematics, Cambridge Univ. Press, England, 2004, pp. 37–57.
[14] H. Edelsbrunner, J. Harer, A. Mascarenhas, V. Pascucci, Time-varying Reeb graphs for continuous space–time data, in: SCG '04: Proc. of the 20th Ann. Sympos. Comput. Geom., 2004, pp. 366–372.
[15] H. Edelsbrunner, J. Harer, V. Natarajan, V. Pascucci, Morse–Smale complexes for piecewise linear 3-manifolds, in: Proc. 19th Ann. Sympos. Comput. Geom., 2003, pp. 361–370.
[16] A.T. Fomenko, T.L. Kunii (Eds.), Topological Methods for Visualization, Springer-Verlag, Tokyo, Japan, 1997.
[17] M. van Kreveld, R. von Oostrum, C.L. Bajaj, V. Pascucci, D.R. Schikore, Contour trees and small seed sets for iso-surface traversal, in: Proc. 13th Ann. Sympos. Comput. Geom., 1997, pp. 212–220.
[18] Y. Matsumoto, An Introduction to Morse Theory, Translated from Japanese by K. Hudson and M. Saito, Amer. Math. Soc., 2002.
[19] N. Max, R. Crawfis, D. Williams, Visualization for climate modeling, IEEE Comput. Graphics Appl. 13 (1993) 34–40.
[20] J. Milnor, Morse Theory, Princeton Univ. Press, NJ, 1963.

[21] J.R. Munkres, Elements of Algebraic Topology, Addison-Wesley, Redwood City, CA, 1984.
[22] V. Pascucci, K. Cole-McLaughlin, Parallel computation of the topology of level sets, Algorithmica 38 (2) (2003) 249–268.
[23] G. Reeb, Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique, Comptes Rendus de L'Académie ses Séances, Paris 222 (1946) 847–849.
[24] Y. Shinagawa, T.L. Kunii, Constructing a Reeb graph automatically from cross sections, IEEE Comput. Graphics Appl. 11 (1991) 44–51.
[25] B.S. Sohn, C.L. Bajaj, Time-varying contour topology, IEEE Transactions on Visualization and Computer Graphics 12 (1) (2006) 14–25.
[26] A. Szymczak, Subdomain aware contour trees and contour evolution in time-dependent scalar fields, in: SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005 (SMI' 05), 2005, pp. 136–144.