An Almost Optimal Algorithm for Dynamically Updating the Reeb Graph

Salman Parsa *

This work is about an efficient algorithm for dynamic updates of the Reeb graph. Given a simplicial complex K and a simplex-wise linear map $f : |K| \to \mathbb{R}$ the Reeb graph $R(f, K)$ is defined as follows. Define two points $x, y \in |K|$ equivalent, $x \sim y$, if $f(x) = f(y)$ and x and y are connected in $f^{-1}(f(x))$. Then the Reeb graph is the space $|K|/\sim$. Intuitively, it is the space whose points are connected components of the various level-sets of f. The Reeb graph is the 1-dimensional structure which sometimes can be used in place of the complex in applications, see [1] for a survey. Let m denote the size of the complex, which is the total number of its simplices. In [4] an algorithm is given for computing the Reeb graph in $O(m \log m)$ time. This algorithm uses dynamic trees data structures and builds on previous work in [2, 5]. They also can be computed in the same time-bound by a randomized algorithm [3].

The Reeb graph depends on the function defined on K. In this work we are interested in the changes in the Reeb graph when the function changes dynamically. The map f is defined by assigning real values to the vertices of K. We assume these real values are distinct. It then follows easily that the Reeb graph depends only on the ordering of vertices by their function values. Therefore, we only need to update the Reeb graph when this ordering is changed. The basic event is when two neighboring vertices exchange their place in this ordering. This is called an *interchange* event. Our task is to update the Reeb graph when such an interchange happens. The difficulty here is to decide global connectivity of the level-sets that change dynamically. Figure 1 below shows when a loop appears or disappears in the Reeb graph by a single interchange event. Part a of the figure depicts a complex which is the boundary of a cube but with two triangles removed, namely $v_3v_4v_9$ and $v_5v_6v_{10}$. The front face is not shown. The function on the complex assigns the usual perceived height to the vertices. The Reeb graph is depicted in part b. The loop will disappear when v_9 and v_{10} exchange their place in the ordering.

Now let us take l copies of the complex in the figure and identify the vertices which correspond to v_9, and similarly identify those corresponding to v_{10}. Then the same change happens in each copy. The Reeb graphs are drawn in parts c and d of the figure. This example shows that in general a single update can change the Reeb graph by $\Omega(m)$ combinatorial changes, where m is size of the complex. Therefore, in the worst-case there cannot be an efficient algorithm in terms of m essentially better than reconstructing the Reeb graph. However, note that a single interchange can change the Reeb graph by $O(l)$ combinatorial changes, where l is an upper bound on size of the star of the vertices. Therefore, we have to write the running time of an update as a function of l.

We announce an algorithm which updates the Reeb graph in $O(l \log^2 m)$ worst-case time. We are not aware of any other algorithm for this problem other than using off-the-shelf graph connectivity for level-set graphs. In these algorithms the running time is $O(lu(n))$ where $u(n)$ is the worst-case deterministic run-time of the dynamic graph connectivity structure, hence, $u(n) = \Theta(\sqrt{n_1})$.

*Duke University, Durham, NC, USA and IST Austria, Klosterneuburg, Austria. email : salparsa@cs.duke.edu

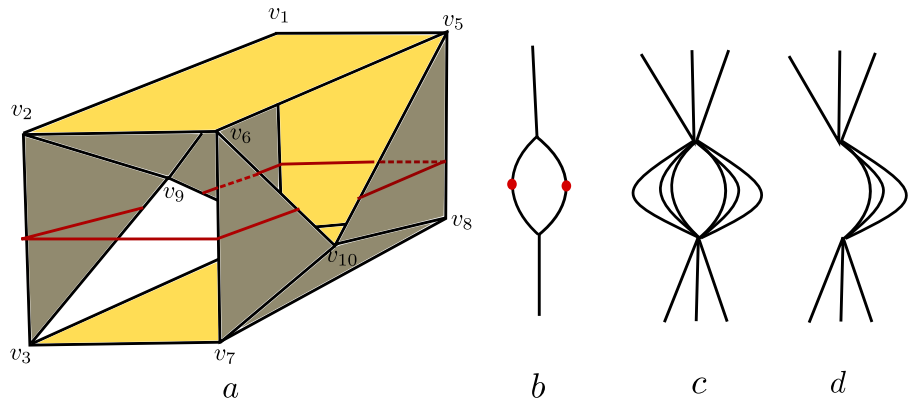

Figure 1: Dependency on the size of the stars

Here we give an overview of the approach. It is easy to see that the Reeb graph only depends on the 2-skeleton of the complex, hence we can assume K is 2-dimensional. A level-set of f will be a graph which we call a *level-set graph*. There exist n_0 level-set graphs which do not include a vertex, where by n_i we denote the number of i-simplices in K. Thus, the size of K is $m = n_0 + n_1 + n_2$. The level-sets are over the same node set, which is the set of edges of K. Arcs of each level-set graph correspond to triangles in K.

Each interchange changes one of these level-set graphs by inserting and deleting some arcs. There will be $O(l)$ of these operations. If we can maintain these level-sets in a data structure such that insert, delete and connectivity queries can be performed on it efficiently, we can then update the Reeb graph efficiently. Here one can use off-the-shelf dynamic graph connectivity structures. However, there are two problems with them. First, the best worst-case time for dynamic connectivity is $O(\sqrt{n})$ for a graph with n vertices. Second, we do not have enough operations on the same data structure to use the data structures with amortized time bounds. In our approach, we maintain a spanning forest of each level-set graph. It turns out that using the forests before and after the current level-set graph, one can always maintain the connectivity in $O(\log^2 n)$ worst-case time per operation on a level-set graph. There is a preprocessing cost which amounts to constructing a dynamic tree data structure for spanning forest of each level set. These data structures also are the required space for the algorithm, which is $O(n_0 n_1)$ in the worst-case.

References

[1] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theo. Comp. Sci.*, 392(1-3):5 – 22, 2008.

[2] Harish Doraiswamy and Vijay Natarajan. Efficient algorithms for computing Reeb graphs. *Comput. Geom.*, 42(6-7):606 – 616, 2009.

[3] William Harvey, Yusu Wang, and Rephael Wenger. A randomized $O(m \log m)$ time algorithm for computing Reeb graphs of arbitrary simplicial complexes. In *Proc. 2010 Ann. Sympos. Comput. Geom*, pages 267–276.

[4] Salman Parsa. A deterministic $O(m \log m)$ time algorithm for the Reeb graph. *Discrete and Computational Geometry*, 49(4):864–878, 2013.

[5] Yoshihisa Shinagawa and Tosiyasu L. Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Comput. Graphics Appl.*, 11:44–51, 1991.