Journal of Symbolic Computation 78 (2017) 76–90

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

PHAT – Persistent Homology Algorithms Toolbox

Ulrich Bauer [a], Michael Kerber [b], Jan Reininghaus [c], Hubert Wagner [d]

[a] Technische Universität München (TUM), Munich, Germany
[b] Graz University of Technology, Graz, Austria
[c] CD-Adapco Inc., Vienna, Austria
[d] Institute of Science and Technology (IST) Austria, Klosterneuburg, Austria

ARTICLE INFO

Article history:
Received 31 January 2015
Accepted 29 November 2015
Available online 29 March 2016

Keywords:
Persistent homology
Topological data analysis
Matrix reduction
Algorithm engineering

ABSTRACT

PHAT is an open-source C++ library for the computation of persistent homology by matrix reduction, targeted towards developers of software for topological data analysis. We aim for a simple generic design that decouples algorithms from data structures without sacrificing efficiency or user-friendliness. We provide numerous different reduction strategies as well as data types to store and manipulate the boundary matrix. We compare the different combinations through extensive experimental evaluation and identify optimization techniques that work well in practical situations. We also compare our software with various other publicly available libraries for persistent homology.

1. Introduction

1.1. Motivation and related work

Persistent homology is one of the most widely applicable tools in the emerging field of computational topology. Intuitively, persistent homology tracks the topological features in a growing sequence of shapes; this includes the Betti numbers of each shape in the sequence, but also how homology classes appear and disappear in the process. This information can be summarized into a two-dimensional point plot summary (the *persistence diagram*) which has shown to be stable under

E-mail addresses: mail@ulrich-bauer.org (U. Bauer), kerber@tugraz.at (M. Kerber), hub.wag@gmail.com (H. Wagner).

http://dx.doi.org/10.1016/j.jsc.2016.03.008
0747-7171/

perturbations of the shape. For a comprehensive introduction to the theory and some applications, see Edelsbrunner and Harer (2008, 2010).

The computation of a persistence diagram usually includes two steps: the first step is the construction of a *filtered cell complex*, i.e., an ordered list of cells such that every prefix forms a combinatorial subcomplex. The filtered cell complex is often represented by its *boundary matrix*, a square matrix whose indices correspond to the ordering of the cells, and whose entries encode the boundary relation of the complex. We currently only consider homology with $\mathbb{Z}_2$-coefficients throughout, so that the boundary matrix has entries in $\{0, 1\}$. Given a boundary matrix, the second step is to compute the persistent homology itself. One approach is to transform the boundary matrix in *reduced form* using elementary column operations, similar to Gaussian elimination. A boundary matrix is called *reduced* if different columns have different pivots. The pivot of a column is the maximal index of the nonzero column entries. While alternative reduction methods based on matrix multiplication (Milosavljevic et al., 2011) and rank computations (Chen and Kerber, 2013) with superior asymptotic complexity have been presented, reduction by column operations is the basis of all efficient approaches for persistence computation to date.

For the first reduction algorithms (Edelsbrunner et al., 2002; Zomorodian and Carlsson, 2005), a quasi-linear complexity on many practical instance has been observed. However, the success of persistent homology has triggered the need of computing persistence on more and more complicated and larger datasets. In the last years, several heuristics with a tremendous effect on the performance of the algorithm have been proposed: replacing homology with cohomology (de Silva et al., 2011; Boissonnat et al., 2013), the usage of Discrete Morse Theory (Günther et al., 2011; Mischaikow and Nanda, 2013), exploiting the special structure of boundary matrices during the reduction (Chen and Kerber, 2011), and tuning the reductions towards parallelizable algorithms (Bauer et al., 2014a, 2014b; Lewis and Zomorodian, 2014; Lipsky et al., 2011). While some approaches also show favorable asymptotic bounds in special cases, the worst-case performance remains cubic in the number of cells, as in the original reduction algorithm.

The plethora of heuristics for persistence computations asks for a qualitative comparison of these approaches: previous comparisons show no clear "winner" among the approaches (e.g., Chen and Kerber, 2011; Boissonnat et al., 2013). While such experimental cross-evaluations are indisputably an important quality criterion, comparing two algorithms embedded in different software libraries reduces the informative value of such results, because the outcome is influenced by other factors than the algorithmic approach, for instance, programming language, implementation of low-level operations, and employed data structures.

1.2. Contributions

This paper introduces the PHAT library[1] as a platform for comparative evaluation of new and existing algorithms and data structures for matrix reduction. More precisely, PHAT provides a slim generic framework for reducing a boundary matrix and we have realized several of the aforementioned heuristics in this framework (see Section 3 for more details). Moreover, each algorithm also comes as a cohomology version by just running it on the anti-transposed matrix. We make the following contributions:

- We show by exhaustive experimental evaluation the tremendous impact of the *clearing* optimization in general, and of using cohomology on wide classes of inputs, confirming earlier reports (Chen and Kerber, 2011; Bauer et al., 2014a; de Silva et al., 2011) in a unified and easily reproducible software framework.
- PHAT provides several data structures to store matrix columns during the reduction process. Other libraries for persistent homology neglect the effect of choosing such a column representation (an exception is the *simplex tree* (Boissonnat and Maria, 2012) in the GUDHI library (Maria et al., 2014)). We implement various data structures in PHAT (Section 4) and provide the first systematic

[1] http://bitbucket.org/phat-code.