GAP persistence – a computational topology package for GAP

Mikael Vejdemo-Johansson*

Abstract

We introduce a recently built package for computing persistent homology within the computer algebra system GAP. A native GAP implementation of the persistent homology algorithm allows for easier access to the group-theoretic primitives in GAP while simultaneously working on computational topology questions – enabling for instance the use of symmetry groups to study symmetric point clouds. The strongest innovation here is in the combination of persistent homology and point cloud algorithms with the group theoretic tools in GAP.

This package is different from the several existing homological algebra and simplicial topology packages already in GAP: `simpcomp`, `homalg`, `Hap`, et c., in that the focus here is on point cloud topology, and not combinatorial complexes or homological algebra.

1 Introduction

We introduce a new package for persistent homology, written entirely in native GAP. The chosen platform has several advantages over the spectrum of available package for persistent homology and topological data analysis:

- GAP has an old, strong, and productive user base within group theory and its various applications – with many uses within research into group representation theory, crystallography, et c.
- GAP is a full-scaled computational algebra package, with strong support for combinatorics and group theory. This enables future projects that use symmetry groups to draw on inherent symmetries in point clouds, and enables persistence methods for studying objects in pure mathematics.
- The package has been developed in the project HPC-GAP, and is part of an effort to produce comfortable parallelism primitives in GAP. As such, the platform choice serves as a foundation for a future implementation of the parallelism proposed by Lipsky, Skraba and Vejdemo-Johansson in [2].

This package is a genuine new addition to the computational topology landscape. The fundamental novelty is the implementation of persistent homology routines on a platform strongly optimized for computational group theoretic investigations. The platform choice is the strongest distinguishing feature – most strong persistent homology packages are not as tightly integrated in a full-scale computational algebra system; and the computational topology packages in GAP– notably `simpcomp`, `Hap`, and `homalg` are all concerned with different aspects of topology and homological algebra than our focus.

2 Current capabilities: nearest neighbours and topology

The currently available version of the package is focused on implementing methods from point cloud topology. While the capabilities of the system are ever increasing, the current implemented code is driven by immediate research requirements.

The implemented software can be divided into three main parts:

2.1 Metric geometry

GAP `persistence` comes with support for enumerating the k nearest neighbours of a point using an arbitrary metric – the Euclidean metric is provided as a helper function – and for enumerating all points in a particular ϵ-ball around a given center point, again using an arbitrary metric, though the Euclidean metric is provided.

In its current shape, the code uses brute force searches for these enumerations. While there are many much faster methods available in low ambient dimensions – using spatial trees for speeding up lookups, there do not seem to be fast methods that scale well with ambient dimension.

*School of Computer Science, University of St Andrews, `mik@mcs.st-andrews.ac.uk`

This is an abstract related to a tutorial or demo given at GTS 2012. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear in a conference with formal proceedings and/or in a journal.

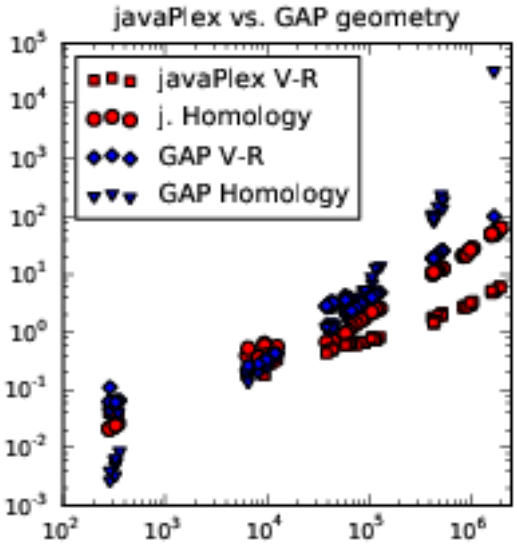

Figure 1: Runtimes in seconds against simplex counts for `javaPlex` and for GAP `persistence`. Up to about several hundred thousand simplices, GAP `persistence` stays about 3x-5x the time needed for `javaPlex` for comparable computations, while for a few million simplices, the runtime needed for GAP `persistence` increases over the corresponding `javaPlex` computation with up to a few orders of magnitude.

2.2 Neighbourhood graphs

The GAP `persistence` package also provides some graph theoretic primitives – capable of creating graphs with both vertex and edge decorations. These come with functionality that relies on the metric geometry above to generate neighbourhood graphs from point clouds.

Internally, given the needs of clique finding algorithms and persistent homology as a main use case, the package uses a neighbourhood list representation of graphs.

2.3 Persistent homology

The package includes a full implementation of the Incremental method for clique complex construction described by Zomorodian in [3], and a full implementation of the persistence algorithm as described by Edelsbrunner–Letscher–Zomorodian and by Zomorodian-Carlsson [1, 4], both with and without retention of representative cycles for homology classes.

3 Performance comparisons

The package is implemented entirely in native GAP, with the only crucial external dependency being on the high-speed hash table implementations used in the GAP package `orb`, included in standard GAP distributions. We have performed comparative benchmarks with the performance of GAP `persistence` as compared to that of `javaPlex`, and present some of these results in Figure 1. We would like to draw the reader's attention to the fact that while an entirely interpreted language, GAP stays within about one order of magnitude slowdown compared to `javaPlex` until we reach about a million simplices. At this point, asymptotic complexity overtakes GAP's performance, yielding a slowdown of several orders of magnitude for problems on a few million simplices.

While GAP `persistence` is slower than `javaPlex`, something that is understandable given that it is written in an entirely interpreted language, the package shows usable performance characteristics. The real power in the comparison comes in memory usage – GAP `persistence` stays very lean compared to `javaPlex`. Due to difficulties in accurately monitoring memory usage on two different platforms, both with significant internal memory managment, we show no data for this – but observations during benchmarks would regularly show `javaPlex` significantly below 1G of consumed memory, even for problem sizes of several hundred thousand simplices, while `javaPlex` would comfortably use up to 4-5G for corresponding computations.

4 Usage example

See Example 4.1 for the GAP session described here. The language is interpreted – and the intermediate stages can be observed to get rough estimates of the computational complexity.

As an example of how to use the package, we use GAP to generate an orbit of a group action with non-trivial topology. The dihedral group D_{2k}, generated by an element c of order k and an element r of order 2, acts on $\mathbb{R}^3$ by rotation by $2\pi/k$ around the z-axis for the action of c and reflection through the xy-plane for the action of r.

Any non-zero point $p \in \mathbb{R}^3$ will have an orbit consisting of the vertices of a prism of a k-gon. If p is close to the xy-plane, the resulting point cloud will be close to a single circle. If p is close to the z-axis, the point cloud will be close to a pair of clusters. Otherwise, p will trace out a pair of circles – corresponding to the top and the bottom k-gons.

We do this computation for $k = 1000$, starting with the point $(1, 1, 1)$ to get two circles – each with 1000 points. We compute persistent homology up to ambient dimension, and filter the resulting barcode to highlight the interesting parts of the barcode. Since we compute with a 3-skeleton, we can expect significant noise in H_3 since the 4-simplices that could kill homology classes will not show up. We filter these classes away together with shortlived classes.

Starting instead with the point $(1, 1, 0.001)$, we get a point cloud that is almost indistinguishable from a single circle. The barcode brings up one generator each of H_0 and H_1.

Timing prints are in milli-seconds for the entire log file.

Example 4.1 A session with GAP and the `persistence` package

```
Lachesis% gap.dev -b
#I  Package ‘‘nq’’: The executable program is not available
gap> LoadPackage("persistence");
true
gap> r := RealPart(E(1000));;
gap> i := ImaginaryPart(E(1000));;
gap> m1 := [[1,0,0],[0,1,0],[0,0,-1]];;
gap> m2 := [[r,i,0],[-i,r,0],[0,0,1]];;
gap> d2000 := Group(m1,m2);;
gap> Length(Elements(d2000)); # Size will test for finiteness first; slower.
2000
gap> cPts := Orbit(d2000, [1,1,1], OnRight);;
gap> pts := List(cPts, v -> List(v, Float));;
gap> eps := 0.01;;
gap> vr := VietorisRips(pts, eps, 3);; time;
29910
gap> int := PersistentHomology(vr, GF(3));; time;
44
gap> FilteredBarcode(int, 2, 0.005); time;
[ [ 0, 0., infinity ], [ 0, 0., infinity ], [ 1, 7.89566e-05, infinity ],
  [ 1, 7.89566e-05, infinity ] ]
2
gap> cPts := Orbit(d2000, [1,1,1/1000], OnRight);;
gap> pts := List(cPts, v -> List(v, Float));;
gap> eps := 0.01;;
gap> vr := VietorisRips(pts, eps, 3);; time;
29239
gap> int := PersistentHomology(vr, GF(3));; time;
278
gap> FilteredBarcode(int, 2, 0.005); time;
[ [ 0, 0., infinity ], [ 1, 7.89566e-05, infinity ] ]
4
```

The resulting barcode is a list of triples with the format `[dimension, birth, death]`. It is worth noticing that the datastructure `vr` is consumed during the process, and thus should be reconstructed from `vrg` and `vrc` for each renewed or halfway aborted run of the algorithm. Not surprisingly, given the input, the computation does recover the topology of the two circles.

The package comes with the capacity to load comma-separated values from a file, allowing the separation of data generation and analysis. Furthermore, one can work in the group representation theory setting of GAP and

use the command `Float` to convert vectors of rationals into vectors of machine float acceptable to the homology computation framework at the last moment. For cyclotomics, the `Float` command will be adapted in the future in all of GAP: in the meantime, we are providing a command in the library to deal with conversions from cyclotomic integers to floating point values.[1]

References

[1] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 454–463, Washington, DC, USA, 2000. IEEE Computer Society. DOI: 10.1109/SFCS.2000.892133

[2] D. Lipsky, P. Skraba, and M. Vejdemo-Johansson. A spectral sequence for parallelized persistence. arXiv:1112.1245v1 [cs.CG], December 2011.

[3] A. Zomorodian. Technical section: Fast construction of the Vietoris-Rips complex. *Comput. Graph.*, 34(3):263–271, June 2010. DOI: 10.1016/j.cag.2010.03.007

[4] A. Zomorodian and G. Carlsson. Computing persistent homology. In *Proceedings of the twentieth annual symposium on Computational geometry*, SCG '04, pages 347–356, New York, NY, USA, 2004. ACM. DOI: 10.1145/997817.997870.

[1] Floating point arithmetic is a very new addition to GAP– and as of writing this extended abstract, the final release for version 4.5 was not yet completed.