Highly Symmetric Point Clouds

Mikael Vejdemo-Johansson
CUNY College of Staten Island
CUNY Graduate Center
(visited) Queen Mary University of London
(visiting) TU München

Jordan Matuszewski
CUNY Graduate Center

Rising Interest in Symmetric Point Clouds

- Adams, Virk: Lower bounds on the homology of Vietoris-Rips complexes of hypercube graphs (2023)
- Vargas-Rosario: Persistent Homology of Products and Gromov-Hausdorff Distances Between Hypercubes and Spheres (2023)
- Adams, Shukla, Singh: Čech complexes of hypercube graphs (2022)
- Adamaszek, Adams: On Vietoris–Rips complexes of hypercube graphs (2022)
- Adams, Coldren, Willmot: The persistent homology of cyclic graphs (2022)
- Adamaszek, Adams, Reddy: On Vietoris-Rips complexes of ellipses (2019)
- Adams, Chowdhury, Jaffe, Sibanda: Vietoris-Rips complexes of regular polygons (2018)
- Adamaszek, Adams: The Vietoris-Rips complexes of a circle (2017)
- Adamaszek, Adams, Frick, Peterson, Previte-Johnson: Nerve complexes of circular arcs (2016)

New software development

- We are **finally** reworking JavaPlex to create a modern TDA-library on the JVM with support for easy access from Matlab, Mathematica, Scala, Kotlin, Java, and a range of software platforms.

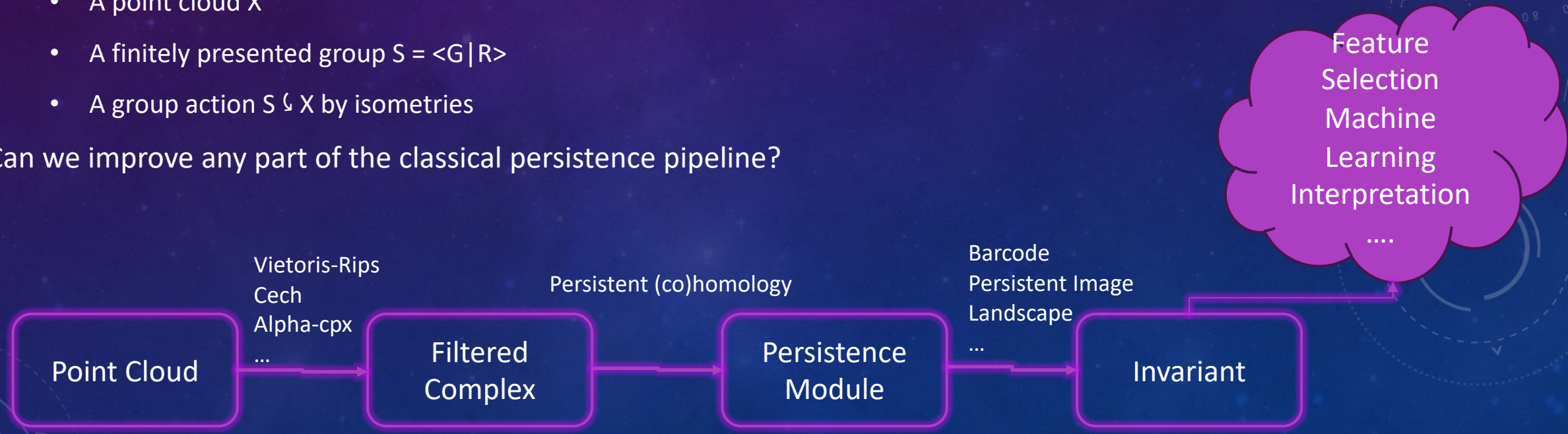
- We have started developing: TDA4j – a JVM library for persistent (co)homology.

Question:
Does knowing the symmetry of a point cloud help?

Suppose we know:

- A point cloud X
- A finitely presented group S = <G|R>
- A group action S ↷ X by isometries

Can we improve any part of the classical persistence pipeline?



A First Study Object:
Vietoris-Rips Complex of the Hypercube with Zomorodian's Incremental Algorithm

Hypercube of n-bit words

- Points: length n words in the alphabet {0,1}
- Distance: Hamming distance

Group Action

- Symmetric Group S_n acts by permuting the bits

Zomorodian's Incremental Algorithm

- Modification of Bron-Kerbosch
- Maintain stack of candidate tasks
- Generate candidate simplices

Can be adapted to create Cech-complexes by evaluating each candidate simplex.

Generating new task can be separated from emitting new simplices.

Quick Primer on Group Actions for Symmetry

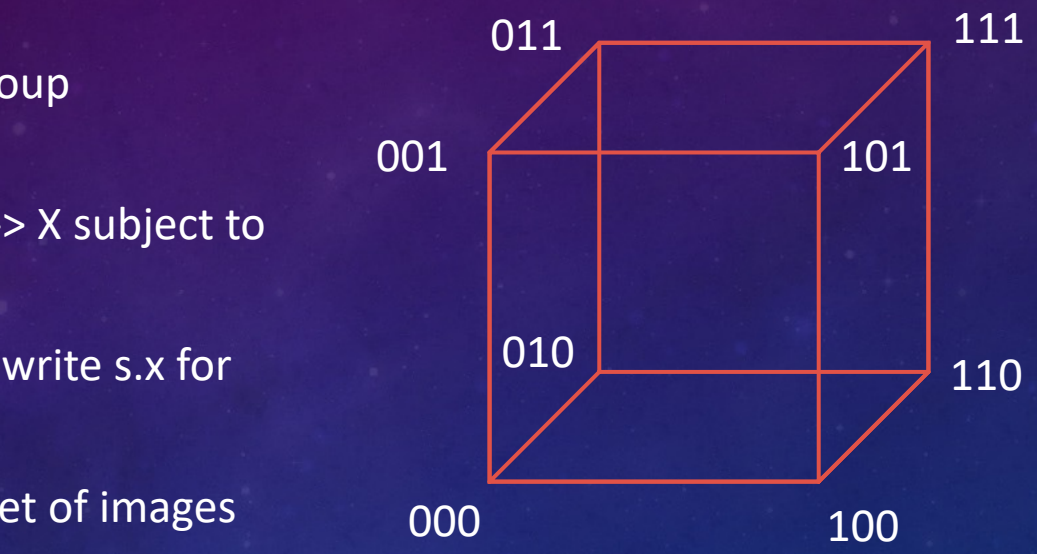
Group S acts on a set X if there is a group homomorphism: S -> Bij(X,X).

Equivalently, there is a set map S x X -> X subject to associativity conditions.

Either way, we can abuse notation to write s.x for the result of using s to modify x.

The **orbit** of an element x in X is the set of images across the group: { s.x | s in S}

I will use **section** of the group action to refer to the choice of one element in each orbit.



Using Total Orders

We will want some total order of simplices in the end anyway (so we get a stream of simplices)

We can induce a total order on simplices if we have a total order on vertices: lexicographic / co-lexicographic (with or without reversing…)

Because everything is finite and tame, this means each orbit will have **a unique smallest element**.

The action $S \curvearrowright X$ lifts to an action $S \curvearrowright VR(X)$: $s.[v_0,\ldots,v_d] = [s.v_0,\ldots,s.v_d]$

Because we are acting by isometries, the entire orbit of a simplex enters the stream at once.

Creating a new candidate in Zomorodian's algorithm is fast (…with the right data structures…)

Two options for using the orbit structures:

1. Emit the entire orbit when we see the first element to get generated.
2. Emit the entire orbit when we see the unique minimal element of the orbit get generated.

Option 1. would require us to be able to recognize when a simplex has already been emitted, which implies storing too much.

Option 2 could speed things up without adding storage if we can easily recognize minimal elements of orbits.

Using the Finitely Generated Group Structure

Wouldn't it be nice if:

Conjecture:
An element x in X is orbit-minimal if for each generator g in G: $g.x > x$ and $g^{-1}.x > x$.

Unfortunately, this is not true – counterexamples show up almost immediately when generating a Vietoris-Rips complex.

However:

Certainly, if the criterion does **not** hold, then x could not possibly be orbit-minimal.

The criterion might be much faster to check than generating the entire orbit repeatedly.

So we can prune the search space more aggressively.

We have implementations of Zomorodian's Incremental Algorithm that use orbit-minimal elements

4-bit Hypercube, max dist=10, max dim=10

- $|S_4| = 24$
- $|X| = 16$
- $|VR(X)| = 63\,018$
- Naïve Bron-Kerbosch:
  52.0s / 7ms / 0ms
- Zomorodian:
  51.4s / 5.2s / 53ms
- Zomorodian with orbit-minimal elements:
  32.2s / 5.2s / 207ms
- Zomorodian with group generator checks:
  15.4s / 6.0s / 53ms

We measure:
Creation / Traversing in order / Visiting every 100^{th}

We have not yet tried very hard to be competitive with other libraries.

After a week of computation, the 5-bit hypercube has not finished using any of the algorithms.

However: it has also not run into out-of-memory errors.

Computations were performed on a 32-core compute server with Ubuntu and 256G RAM.

We have implementations of Zomorodian's Incremental Algorithm that use orbit-minimal elements

5-bit Hypercube, max dist=10, max dim=10

- $|S_5| = 120$
- $|X| = 32$
- $|VR(X)| = 107\ 594\ 212$
- Naïve Bron-Kerbosch:
  > 1 week
- Zomorodian:
  > 1 week
- Zomorodian with orbit-minimal elements:
  2d17h / ?? / ??
- Zomorodian with group generator checks:
  3h57m / ?? / ??

We measure wall clock time for the computation:
Creation / Traversing in order / Visiting every 100^{th}

We have not yet tried very hard to be competitive with other libraries.

After a week of computation, the 5-bit hypercube has not finished using any of the algorithms.

However: it has also not run into out-of-memory errors.

Computations were performed on a 32-core compute server with Ubuntu and 256G RAM.

Our intended next steps

- Implement the Ripser strategy with addressing each simplex with binomial coefficients.
- Explore both the classical persistent homology algorithm and the Ripser algorithm for places where the symmetries can be used.
- Optimize the code and benchmark against other libraries.

And Also

- Extend the current draft TDA4j library to a fully useable successor to JavaPlex and library for the JVM
- Provide user and developer documentation to make TDA4j a viable platform for algorithmic research in persistence.

Thank you for your attention
And **Thank you** to my research group and collaborators

Jordan Matuszewski, CUNY GC

Daniel Hope, CUNY

Martin Bendersky, CUNY
Yosef Berman, CUNY GC

This work was supported by a grant from the Simons Foundation (961833, MVJ)