A roadmap for the computation of persistent homology

Nina Otter, Mason A. Porter, Ulrike Tillmann,
Peter Grindrod, Heather A. Harrington

Mathematical Institute, University of Oxford

Aberdeen, 8 June 2015

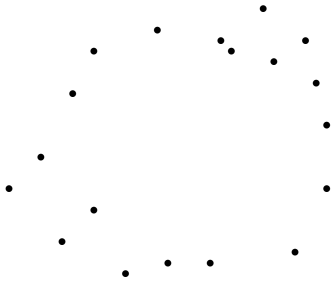
What is persistent homology?

Persistent homology is a method from algebraic topology used to study *topological features* of *data*.

- *Topological features*: e.g. connected components, holes, voids, etc.
- *Data*: e.g. a finite set X together with a distance d, called a **point cloud**

Example

What is the topology of this point cloud?



Here X is a subset of $\mathbb{R}^2$ and d is the Euclidean distance.

Idea: build a simplicial complex on the points.

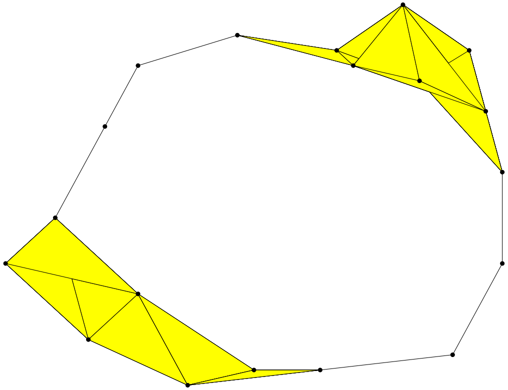
Idea: build a simplicial complex on the points.

Choose a distance ϵ. Draw a k-simplex on $x_0, \ldots, x_k$ if and only if the points have pairwise distance smaller then or equal to ϵ.

Idea: build a simplicial complex on the points.

Choose a distance ϵ. Draw a k-simplex on $x_0, \ldots, x_k$ if and only if the points have pairwise distance smaller then or equal to ϵ.

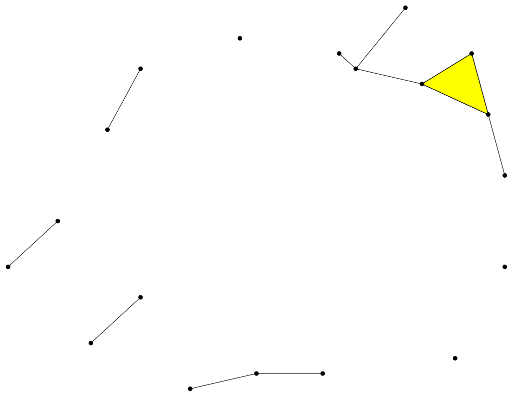
For example:



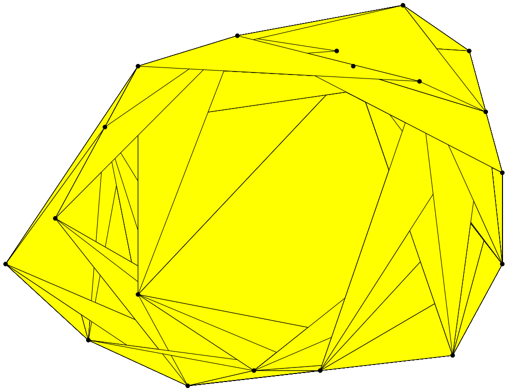
Problem: how do we choose ϵ?

Problem: how do we choose ϵ?

If ϵ is too small:



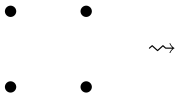
If ϵ is too large:



Solution: look at *all* possible values for the distance and obtain a sequence of simplicial complexes $\{K_\epsilon\}_{\epsilon \geq 0}$ with $K_{\epsilon_1} \subseteq K_{\epsilon_2}$ for $\epsilon_1 \leq \epsilon_2$. We call this a **filtered simplicial complex**.

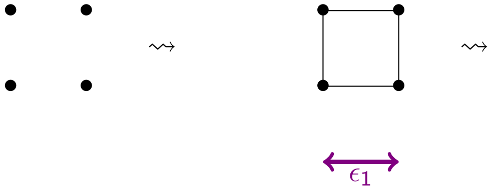
Solution: look at *all* possible values for the distance and obtain a sequence of simplicial complexes $\{K_\epsilon\}_{\epsilon \geq 0}$ with $K_{\epsilon_1} \subseteq K_{\epsilon_2}$ for $\epsilon_1 \leq \epsilon_2$. We call this a **filtered simplicial complex**.

Each feature, e.g. hole, appears at a certain distance ϵ_1 and disappears at another distance ϵ_2:



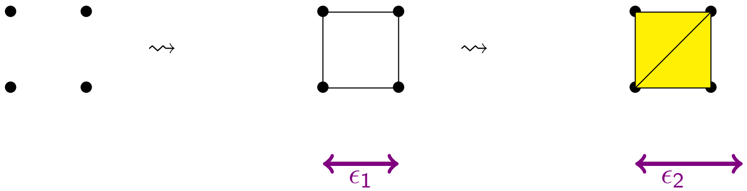
Solution: look at *all* possible values for the distance and obtain a sequence of simplicial complexes $\{K_\epsilon\}_{\epsilon \geq 0}$ with $K_{\epsilon_1} \subseteq K_{\epsilon_2}$ for $\epsilon_1 \leq \epsilon_2$. We call this a **filtered simplicial complex**.

Each feature, e.g. hole, appears at a certain distance ϵ_1 and disappears at another distance ϵ_2:

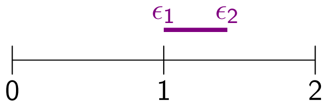


Solution: look at *all* possible values for the distance and obtain a sequence of simplicial complexes $\{K_\epsilon\}_{\epsilon \geq 0}$ with $K_{\epsilon_1} \subseteq K_{\epsilon_2}$ for $\epsilon_1 \leq \epsilon_2$. We call this a **filtered simplicial complex**.

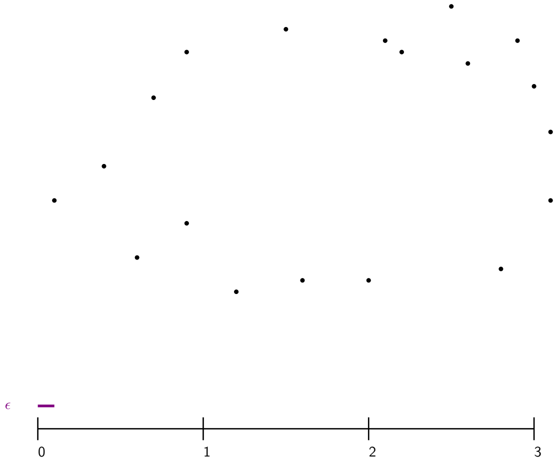
Each feature, e.g. hole, appears at a certain distance ϵ_1 and disappears at another distance ϵ_2:



We say that the **persistence** of this feature is the interval $[\epsilon_1, \epsilon_2)$ and we represent it by a bar



Barcode



How do we compute the barcode?

Step 1: Order the simplices so that the total order is compatible with the filtration.

Step 2: Construct the boundary matrix.
Let n be the number of simplices. The **boundary matrix** B is an $n \times n$-matrix defined by:

$$B(i,j) = \begin{cases} 1 & \sigma_i \subset \sigma_j \text{ and } dim(\sigma_i) = dim(\sigma_j) - 1 \\ 0 & \text{otherwise} \end{cases}$$

Step 3: Reduce the matrix using column additions from left to right.

Step 4: Read the endpoints of the persistence intervals from the matrix to get the barcode.

Step 1. Given a filtered simplicial complex

$$\emptyset = K_0 \subset K_1 \subset \cdots \subset K_m = K$$

put an order on its simplices such that:

- A face of a simplex precedes the simplex.
- A simplex in K_i precedes simplices in $K \setminus K_i$.

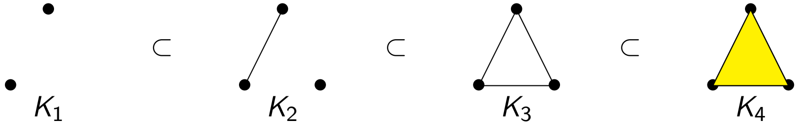
Step 1. Given a filtered simplicial complex

$$\emptyset = K_0 \subset K_1 \subset \cdots \subset K_m = K$$

put an order on its simplices such that:

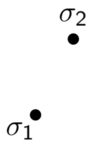
- A face of a simplex precedes the simplex.
- A simplex in K_i precedes simplices in $K \setminus K_i$.

Example:



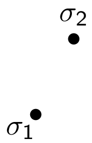
Put an order on the simplices such that:

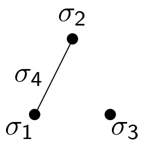
- A face of a simplex precedes the simplex.
- A simplex in K_i precedes simplices in $K \setminus K_i$.



Put an order on the simplices such that:

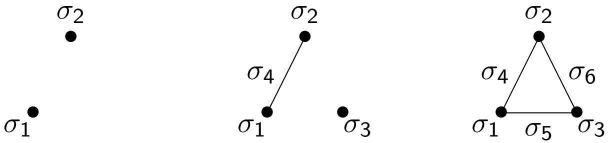
- A face of a simplex precedes the simplex.
- A simplex in K_i precedes simplices in $K \setminus K_i$.





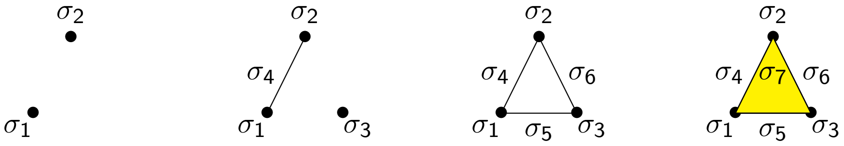
Put an order on the simplices such that:

- A face of a simplex precedes the simplex.
- A simplex in K_i precedes simplices in $K \setminus K_i$.



Put an order on the simplices such that:

- A face of a simplex precedes the simplex.
- A simplex in K_i precedes simplices in $K \setminus K_i$.



Step 2. We obtain the boundary matrix:

	1	2	3	4	5	6	7
1				1	1		
2				1		1	
3					1	1	
4							1
5							1
6							1
7							

For $j = 1, \dots, 7$ define $low(j) = i$ if i is the position of the lowest 1 in column j.

If the column is zero leave $low(j)$ undefined.

e.g. $low(4) = 2$

Step 3. Reduce the boundary matrix.

Let n be the number of simplices.

Algorithm:

```
for $j = 1$ to $n$ do
    while there exists $i < j$ with
    $low(i) = low(j)$ do
        add column $i$ to column $j$
    end while
end for
```

	1	2	3	4	5	6	7
1				1	1		
2				1		1	
3					1	1	
4							1
5							1
6							1
7							

Step 3. Reduce the boundary matrix.

Algorithm:

```
for $j = 1$ to $n$ do
    while there exists $i < j$ with
    $low(i) = low(j)$ do
        add column $i$ to column $j$
    end while
end for
```

Add column 5 to column 6:

	1	2	3	4	5	6	7
1				1	1	1	
2				1		1	
3					1		
4							1
5							1
6							1
7							

Step 3. Reduce the boundary matrix.

Algorithm:

```
for $j = 1$ to $n$ do
    while there exists $i < j$ with
    $low(i) = low(j)$ do
        add column $i$ to column $j$
    end while
end for
```

Add column 4 to column 6:

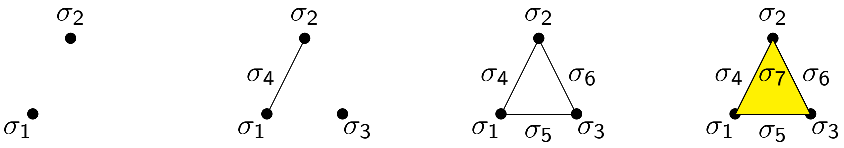
	1	2	3	4	5	6	7
1				1	1		
2				1			
3					1		
4							1
5							1
6							1
7							

Step 4. Read the persistence pairs.

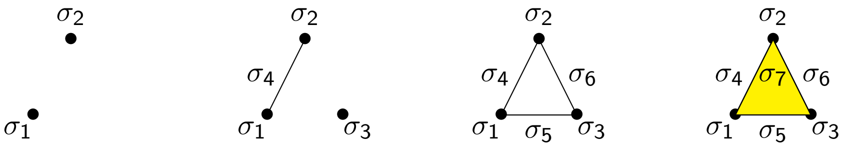
- If $low(j) = i$ then σ_j is negative and paired with the positive σ_i.
- If $low(j)$ is undefined then σ_j is positive.
  If there exists k such that $low(k) = j$ then σ_j is paired with the negative simplex σ_k.
  If no such k exists σ_j is unpaired.

	1	2	3	4	5	6	7
1				1	1		
2				1			
3					1		
4							1
5							1
6							1
7							

σ_1 positive, unpaired
σ_2 positive, paired with σ_4
σ_3 positive, paired with σ_5
σ_4 negative, paired with σ_2
σ_5 negative, paired with σ_3
σ_6 positive, paired with σ_7
σ_7 negative, paired with σ_6

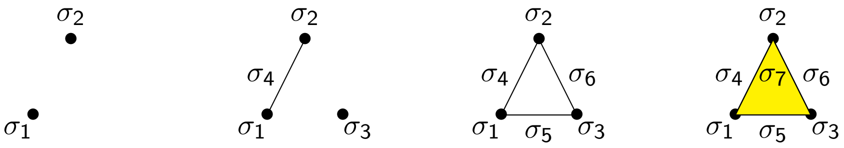


σ_1 positive, unpaired $\rightsquigarrow$ interval $[1, \infty)$ in H_0.



σ_1 positive, unpaired $\rightsquigarrow$ interval $[1, \infty)$ in H_0.

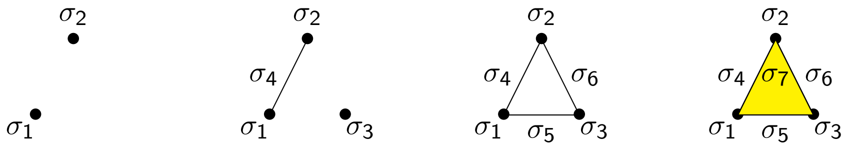
σ_2 positive, paired with σ_4 $\rightsquigarrow$ no interval, since σ_2 and σ_4 enter at the same time in the filtration



σ_1 positive, unpaired $\rightsquigarrow$ interval $[1, \infty)$ in H_0.

σ_2 positive, paired with σ_4 $\rightsquigarrow$ no interval, since σ_2 and σ_4 enter at the same time in the filtration

σ_3 positive, paired with σ_5 $\rightsquigarrow$ interval $[2, 3)$ in H_0.



σ_1 positive, unpaired $\rightsquigarrow$ interval $[1, \infty)$ in H_0.

σ_2 positive, paired with σ_4 $\rightsquigarrow$ no interval, since σ_2 and σ_4 enter at the same time in the filtration

σ_3 positive, paired with σ_5 $\rightsquigarrow$ interval $[2, 3)$ in H_0.

σ_6 positive, paired with σ_7 $\rightsquigarrow$ interval $[3, 4)$ in H_1.

PH computation pipeline

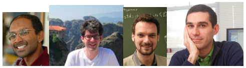
$$\begin{tikzcd}
\boxed{\text{Data}} \arrow[r] & \boxed{\text{Filtered complex}} \arrow[r] & \boxed{\text{Boundary matrix}} \arrow[r] & \boxed{\text{Barcodes}}
\end{tikzcd}$$

Softwares

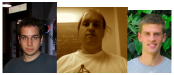
- **Perseus** `http://www.sas.upenn.edu/~vnanda/perseus/`
- **JavaPlex** `http://appliedtopology.github.io/javaplex/`
- **jHoles** `http://cuda.unicam.it/jHoles/`
- **Dionysus** `http://www.mrzv.org/software/dionysus/`
- **phom** `http://cran.r-project.org/web/packages/phom/`
- **PHAT** `https://code.google.com/p/phat/`
- **DIPHA** `https://code.google.com/p/dipha/`
- **GUDHI** `https://project.inria.fr/gudhi/software/`

A brief (biased) history of PH softwares

2004 PH algorithm *G. Carlsson and A. Zomorodian*



2005 **Plex** *V. de Silva, P. Perry, L. Kettner, A. Zomorodian*



2011 **JavaPlex** *A. Tausz, M. Vejdemo-Johansson, H. Adams*

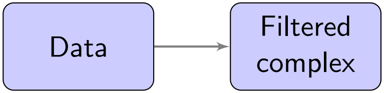
2012 **Perseus** *V. Nanda*

2013 **PHAT** *M. Kerber, J. Reininghaus, U. Bauer, H. Wagner*

2014 **DIPHA** *M. Kerber, J. Reininghaus, U. Bauer*

2014 **GUDHI** *C. Maria, J.-D. Boissonnat, M. Glisse, M. Yvinec*

Optimisations



Optimisations

$$\begin{tikzcd}\boxed{\text{Data}} \arrow[r] & \boxed{\text{Filtered complex}}\end{tikzcd}$$

Simple complexes:

- Witness complex, [*de Silva, Carlsson 2004*]
- Linear size approximations of VR complex, [*Sheehy 2013*]

Simplification of a given filtered complex:

- Discrete Morse theory, [*Mischaikov, Nanda 2013*]
- The tidy set, [*Zomorodian 2010*]

Efficient data structure:

- Simplex tree, [*Boissonnat, Maria 2012*]

Optimisations

$$\begin{tikzcd}
\boxed{\text{Data}} \arrow[r] & \boxed{\text{Filtered complex}} \arrow[r] & \boxed{\text{Boundary matrix}} \arrow[r] & \boxed{\text{Barcodes}}
\end{tikzcd}$$

Simple complexes:

- Witness complex, [*de Silva, Carlsson 2004*]
- Linear size approximations of VR complex, [*Sheehy 2013*]

Simplification of a given filtered complex:

- Discrete Morse theory, [*Mischaikov, Nanda 2013*]
- The tidy set, [*Zomorodian 2010*]

Efficient data structure:

- Simplex tree, [*Boissonnat, Maria 2012*]

Optimisations

$$\begin{tikzcd}\boxed{\text{Data}} \arrow[r] & \boxed{\text{Filtered complex}} \arrow[r] & \boxed{\text{Boundary matrix}} \arrow[r] & \boxed{\text{Barcodes}}\end{tikzcd}$$

Simple complexes:

- Witness complex, [*de Silva, Carlsson 2004*]
- Linear size approximations of VR complex, [*Sheehy 2013*]

Simplification of a given filtered complex:

- Discrete Morse theory, [*Mischaikov, Nanda 2013*]
- The tidy set, [*Zomorodian 2010*]

Efficient data structure:

- Simplex tree, [*Boissonnat, Maria 2012*]

Sequential optimization:

- Dual algorithm, [*de Silva, Morozov, Vejdemo-Johansson 2011*]
- Twist algorithm, [*Chen, Kerber 2011*]

Parallel optimizations:

- Spectral sequence algorithm [*Edelsbrunner, Harer 2008*]
- Chunk algorithm [*Bauer, Kerber, Reininghaus 2013*]
- Distributed computation [*Bauer, Kerber, Reininghaus 2014*]

Efficient data structures:

- Bit tree pivot column [*Bauer, Kerber, Reininghaus, Wagner 2013*]
- Compressed annotation matrix [*Boissonnat, Dey, Maria 2013*]

Sequential optimisation: example

Twist algorithm

Note:

- If column j is reduced and $low(j) = i > 0$ then simplex σ_i is positive and reducing column i will result in setting column i to zero.
- σ_i is a codimension 1 face of σ_j

Optimisation:

- Reduce columns from right to left
- If column j is reduced and $low(j) = i > 0$ then set column i to zero.

Previous example:

	1	2	3	4	5	6	7
1				1	1		
2				1		1	
3					1	1	
4							1
5							1
6							1
7							

Column 7 is reduced and $\mathit{low}(7) = 6$, so set column 6 to zero.

Previous example:

	1	2	3	4	5	6	7
1				1	1		
2				1		1	
3					1	1	
4							1
5							1
6							1
7							

Column 7 is reduced and $low(7) = 6$, so set column 6 to zero.

	1	2	3	4	5	6	7
1				1	1		
2				1			
3					1		
4							1
5							1
6							1
7							

Now we are done.

Parallel optimisation: example

Spectral sequence algorithm

Let k_j denote the number of simplices in subcomplex K_j.

Let B^j denote the columns numbered $k_{j-1}+1$ to k_j.

Let B_i denote the rows numbered $k_{i-1}+1$ to k_i.

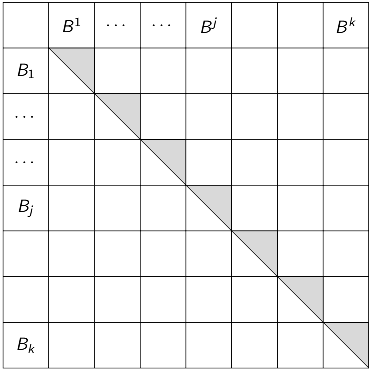
Idea:

- Reduce the matrix in phases: in each phase r, reduce columns in B^j by adding columns in the blocks from B^{j-r+1} to B^j.

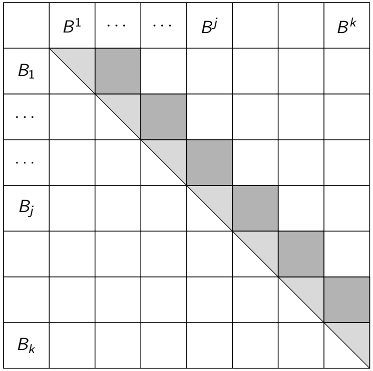
Optimisation:

- The reduction in each block and each phase is independent, and can be executed in parallel.

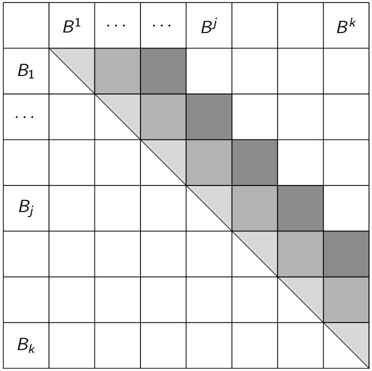
Phase $r = 1$:



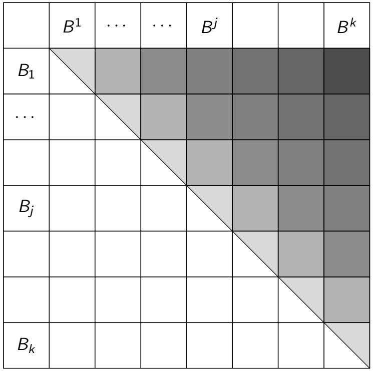
Phase $r = 2$:



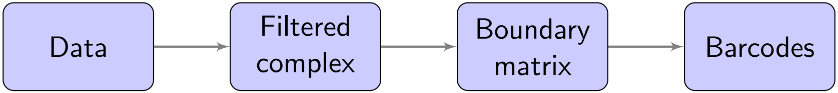
Phase $r = 3$:



Phase $r = k$:



Optimisations



Simple complexes:

- Witness complex → JavaPlex, phom
- Linear size approximations of VR complex → no implementation (?)

Simplification of a given filtered complex:

- Discrete Morse theory → Perseus
- The tidy set → not open source

Efficient data structure:

- Simplex tree → GUDHI

Sequential optimization:

- dual algorithm → Dionysus, PHAT, DIPHA, GUDHI, JavaPlex
- Twist algorithm → PHAT, DIPHA, Dionysus

Parallel optimizations:

- Spectral sequence algorithm → PHAT
- Chunk algorithm → PHAT
- Distributed computation → DIPHA

Efficient data structures:

- Bit tree pivot column → PHAT, DIPHA
- Compressed annotation matrix → GUDHI

Software	Precomp.	filt. Compl. [1]	Parallel	Visualiz.	PH algorithms
javaPlex	✓	VR, LW, W, CW	✗	✓	standard, dual zig zag
Perseus	✓	VR	✗	✓	Morse reductions
Dionysus	✗	α, VR, Čech	✗	✗	standard, dual twist
jHoles	✓	WRCF	✓ shared	✓[2]	standard (javaPlex)
phom	✓	VR,LW	✗	✓	standard, dual
PHAT	✗	✗	✓shared	✗	standard, dual, twist chunk, spectral seq. dual, spectral seq.
DIPHA	✗	VR, lower star	✓distr.	✓	dual, twist distributed
GUDHI	✗	VR	✗	✗	multifield dual

[1] VR=Vietoris Rips complex, W=witness complex, LW=lazy witness complex, CW=CW complex, WRCF= weight rank clique filtration

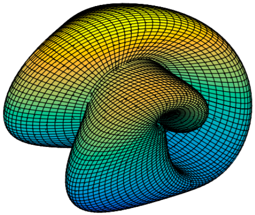
Data

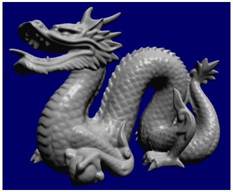
Synthetic data

- Points sampled from Klein bottle
- Geometric random graphs

Real world data

- Genomic sequence of HIV virus
- Points sampled from 3D scans of Stanford dragon
- C. Elegans neuronal network
- Human genome network





Machines

- Cluster[3]: 1728 (180*16) cores of 2.0GHz
  RAM : 64 GiB x80 nodes, 128 GiB x 4 nodes
- Shared memory system[3]: 64 cores of 2.67GHz
  RAM: 1 TB

[3] Advanced Research Computing (ARC), Oxford

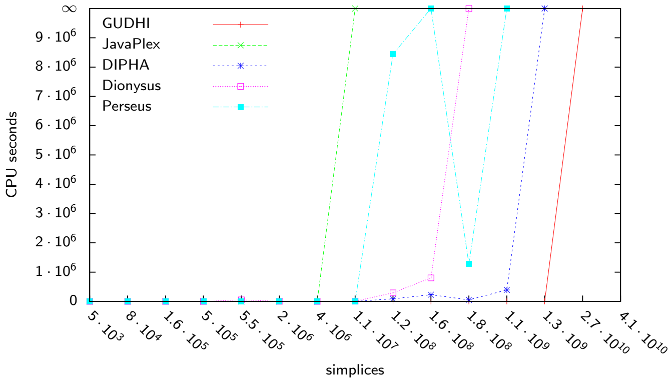
Methods

We study the softwares from four different points of view:

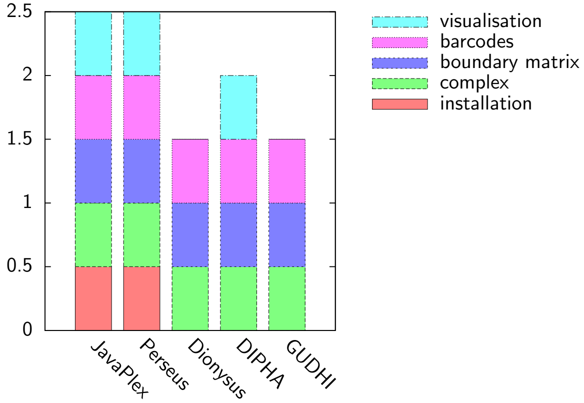
1. Performance measured in CPU and real time
2. Memory usage
3. Maximum size of simplicial complex allowed by the software
4. User-friendliness: phases of computation of PH supported by software

Performance

Performance for VR complex for point clouds created from subsample of Klein bottle, HIV genome and the 3D scans of the Standford dragon.



User-friendliness



Maximal size of simplicial complex

Maximal size of simplicial complex supported by the software[4]:

Software	JavaPlex	Perseus	Dionysus	DIPHA	GUDHI
maximal size	$4 \cdot 10^6$	$2 \cdot 10^8$	$1.6 \cdot 10^8$	$1 \cdot 10^{10}$	$2 \cdot 10^{10}$

[4] Thus far, computations still in progress.

Challenges

- Creation of a computational topology library.
- Definition and construction of benchmarking datasets for the test of new algorithms and data structures.
- Uniformization of input type across different implementations.
- Efficient storage and constrution of complexes: recent progress in the computation of barcodes from the boundary matrix is hindered by the complexity of the computation of the complex.
- Stream Processing: new techniques are needed to compute PH for streams of data.

References

- Witness complex: Topological estimation using witness complexes, V. de Silva, G. Carlsson 2004
- Approximation of VR complex: Linear-Size Approximations to the Vietoris-Rips Filtration. D. R. Sheehy 2013
- Morse theoretic reductions: Morse Theory for Filtrations and Efficient Computation of Persistent Homology, K. Mischaikov and V. Nanda 2013
- The Tidy Set: A Minimal Simplicial Set for Computing Homology of Clique Complexes, A. Zomorodian 2010
- The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes. J.-D. Boissonnat, C. Maria 2012
- Dual algorithm: Dualities in Persistent (Co)Homology. V. de Silva, D. Morozov, M. Vejdemo-Johansson 2011
- Twist algorithm: Persistent Homology Computation with a Twist. C. Chen and M. Kerber 2011
- Spectral sequence algorithm: Persistent homology - A Survey. H. Edelsbrunner, J. Harer 2008
- Chunk algorithm: Clear and Compress: Computing Persistent Homology in Chunks. U. Bauer, M. Kerber, J. Reininghaus 2013
- Distributed computation: Distributed computation of persistent homology. U. Bauer, M. Kerber, J. Reininghaus 2014
- Bit-tree: PHAT Persistent Homology Algorithms Toolbox. U. Bauer, M. Kerber, J. Reininghaus, H. Wagner 2013
- The Compressed Annotation Matrix: An Efficient Data Structure for Computing Persistent Cohomology. J-D. Boissonnat, T. K. Dey, Maria 2013