Multi starts at one

Efficient Computation of Vietoris–Rips Persistence Barcodes

Ulrich Bauer

TUM

August 8, 2018

Workshop *Multiparameter Persistent Homology*
Casa Matemática Oaxaca

Vietoris–Rips persistent homology

Vietoris–Rips filtrations

Consider a finite metric space (X, d).
The *Vietoris–Rips complex* is the simplicial complex

$$\mathrm{Rips}_t(X) = \{S \subseteq X \mid \operatorname{diam} S \leq t\}$$

- 1-skeleton: all edges with pairwise distance $\leq t$
- all possible higher simplices (flag complex)

Vietoris–Rips filtrations

Consider a finite metric space (X, d).
The *Vietoris–Rips complex* is the simplicial complex

$$\mathrm{Rips}_t(X) = \{S \subseteq X \mid \operatorname{diam} S \leq t\}$$

- 1-skeleton: all edges with pairwise distance $\leq t$
- all possible higher simplices (flag complex)

Goal:

- compute persistence barcodes for $H_d(\mathrm{Rips}_t(X))$
  (in dimensions $0 \leq d \leq k$)

Demo: Ripser

Example data set:

- 192 points on $\mathbb{S}^2$
- persistent homology barcodes up to dimension 2
- over 56 mio. simplices in 3-skeleton

Demo: Ripser

Example data set:

- 192 points on $\mathbb{S}^2$
- persistent homology barcodes up to dimension 2
- over 56 mio. simplices in 3-skeleton

Comparison with other software:

- javaplex: 3200 seconds, 12 GB
- Dionysus: 533 seconds, 3.4 GB
- GUDHI: 75 seconds, 2.9 GB
- DIPHA: 50 seconds, 6 GB
- Eirene: 12 seconds, 1.5 GB

Demo: Ripser

Example data set:

- 192 points on $\mathbb{S}^2$
- persistent homology barcodes up to dimension 2
- over 56 mio. simplices in 3-skeleton

Comparison with other software:

- javaplex: 3200 seconds, 12 GB
- Dionysus: 533 seconds, 3.4 GB
- GUDHI: 75 seconds, 2.9 GB
- DIPHA: 50 seconds, 6 GB
- Eirene: 12 seconds, 1.5 GB

Ripser: 1.2 seconds, 152 MB

Ripser

A software for computing Vietoris–Rips persistence barcodes

- about 1000 lines of C++ code, no external dependencies
- support for
  - coefficients in a prime field $\mathbb{F}_p$
  - sparse distance matrices for distance threshold
- open source (`http://ripser.org`)
  - released in July 2016
- online version (`http://live.ripser.org`)
  - launched in August 2016
- most efficient software for Vietoris–Rips persistence
  - computes H^2 barcode for $50\,000$ random points on a torus in 136 seconds / 9 GB (using distance threshold)
- 2016 ATMCS Best New Software Award (jointly with RIVET)

Design goals

Goals for previous persistence software projects:

- PHAT [B, Kerber, Reininghaus, Wagner 2013]:
  fast persistence computation (matrix reduction only), comparing different algorithms and data sturctures
- DIPHA [B, Kerber, Reininghaus 2014]:
  distributed persistence computation, based on spectral sequence algorithm

Design goals

Goals for previous persistence software projects:

- PHAT [B, Kerber, Reininghaus, Wagner 2013]:
  fast persistence computation (matrix reduction only),
  comparing different algorithms and data sturctures
- DIPHA [B, Kerber, Reininghaus 2014]:
  distributed persistence computation, based on spectral sequence algorithm

Goals for Ripser:

- Use as little memory as possible
- Be reasonable about computation time

The four special ingredients

The improved performance is based on 4 insights:

- Clearing inessential columns [Chen, Kerber 2011]
- Computing cohomology [de Silva et al. 2011]
- Implicit matrix reduction
- Apparent and emergent pairs

The four special ingredients

The improved performance is based on 4 insights:

- Clearing inessential columns [Chen, Kerber 2011]
- Computing cohomology [de Silva et al. 2011]
- Implicit matrix reduction
- Apparent and emergent pairs

Lessons from PHAT:

- Clearing and cohomology yield considerable speedup,
- but only when *both* are used in conjuction!

Filtrations and refinements

Simplexwise filtrations

Call a filtration $(K_i)_{i \in I}$ of simplicial complexes (I totally ordered)

- *essential* if $i \neq j$ implies $K_i \neq K_j$,
- *simplexwise* if for all $i \in I$ with $K_i \neq \emptyset$ there is some index $j < i \in I$ and some simplex $\sigma \in K$ such that $K_i \smallsetminus K_j = \{\sigma\}$.

Note:

- These properties are natural assumptions for computation.
- The *Vietoris–Rips filtration* (indexed by $\mathbb{R}$) is not simplexwise (and not essential).
- To compute Vietoris–Rips persistence, we will reindex and refine.

Reindexing and refinement

A filtration $K : I \to \mathbf{Simp}$ is a *reindexing* of another filtration $F : R \to \mathbf{Simp}$ if $F = K \circ r$ for some monotonic $r : R \to I$.

- If r is injective, we call K a *refinement* of F;
- if r is surjective, we call K a *reduction* of F.

Reindexing and refinement

A filtration $K : I \to \mathbf{Simp}$ is a *reindexing* of another filtration $F : R \to \mathbf{Simp}$ if $F = K \circ r$ for some monotonic $r : R \to I$.

- If r is injective, we call K a *refinement* of F;
- if r is surjective, we call K a *reduction* of F.

The Vietoris–Rips filtration can be reindexed:

- reduced to an essential filtration (over the set of pairwise distances), and further
- refined to a simplexwise filtration

Reindexing and refinement

A filtration $K : I \to \mathbf{Simp}$ is a *reindexing* of another filtration $F : R \to \mathbf{Simp}$ if $F = K \circ r$ for some monotonic $r : R \to I$.

- If r is injective, we call K a *refinement* of F;
- if r is surjective, we call K a *reduction* of F.

The Vietoris–Rips filtration can be reindexed:

- reduced to an essential filtration (over the set of pairwise distances), and further
- refined to a simplexwise filtration

Ripser use the *lexicographic refinement*: simplices ordered by

- diameter, then
- dimension, then
- (decreasing) lexicographic order of vertices $\{v_{i_k}, \ldots, v_{i_0}\}$ (induced by fixed total order $v_0 < \cdots < v_{n-1}$ on vertices)

Enumerating (co)faces in lexicographic order

There is an order-preserving bijection

$$(v_{i_k}, \dots, v_{i_0}) \mapsto \sum_{j=0}^{k} \binom{i_j}{j+1}$$

from k-simplices (ordered tuples of vertices) to $\{0, \dots, \binom{n}{k+1} - 1\}$ (called *combinatorial number system*).

- Using this, faces and cofaces can be efficiently enumerated in (decreasing) lexicographic order

Persistent homology

Consider a filtration $F : \mathbb{R} \to \mathbf{Simp}$ with reindexing $F = K \circ r$, $K : I \to \mathbf{Simp}$, $r : \mathbb{R} \to [n]$. The persistence barcode of $K_\bullet$ determines the persistence barcode of $F_\bullet$:

$$B(H_*(F_\bullet)) = \{r^{-1}(I) \neq \varnothing \mid I \in B(H_*(K_\bullet))\}.$$

Matrix reduction

Computing homology

Computing homology $H_* = Z_*/B_*$:

- compute basis for boundaries $B_* = \operatorname{im} \partial_*$
- extend to basis for cycles $Z_* = \ker \partial_*$
- new (non-boundary) basis cycles generate quotient Z_*/B_*

Computing homology

Computing homology $H_* = Z_*/B_*$:

- compute basis for boundaries $B_* = \operatorname{im} \partial_*$
- extend to basis for cycles $Z_* = \ker \partial_*$
- new (non-boundary) basis cycles generate quotient Z_*/B_*

Computing *persistent* homology $H_* = Z_*/B_*$ (for a simplexwise filtration $K_i \subseteq K$):

- compute *filtered* basis for boundaries $B_* = \operatorname{im} \partial_*$
- extend to basis for cycles $Z_* = \ker \partial_*$
- all basis cycles generate *persistent* homology

Persistence by matrix reduction

Given:

- D: matrix of boundary ∂_d for a simplexwise filtration $(K_i)_i$ (for canonical basis in filtration order, indexed by $I \times J$)

Wanted:

- persistence barcode of homology $H_d(K_i; \mathbb{F})$ for some (prime) field $\mathbb{F}$, in dimensions $d = 0, \dots, k$

Notation:

- m_j: jth column of M, m_{ij}: entry in ith row and jth column
- $\operatorname{Pivot} m_j = \min\{i \in I : m_{kj} = 0 \text{ for all } k > i\}$.

Persistence by matrix reduction

Given:

- D: matrix of boundary ∂_d for a simplexwise filtration $(K_i)_i$ (for canonical basis in filtration order, indexed by $I \times J$)

Wanted:

- persistence barcode of homology $H_d(K_i;\mathbb{F})$ for some (prime) field $\mathbb{F}$, in dimensions $d = 0, \ldots, k$

Notation:

- m_j: jth column of M, m_{ij}: entry in ith row and jth column
- $\operatorname{Pivot} m_j = \min\{i \in I : m_{kj} = 0$ for all $k > i\}$.

Computation: barcode is obtained by *matrix reduction* of D

- $R = D \cdot V$ *reduced* (non-zero columns have distinct pivots)
- V is regular upper triangular

Compatible basis cycles

For a reduced boundary matrix $R = D \cdot V$, call

$$I_b = \{i : r_i = 0\} \qquad \textit{birth indices},$$

$$I_d = \{j : r_j \neq 0\} \qquad \textit{death indices},$$

$$I_e = I_b \setminus \operatorname{Pivots} R \qquad \textit{essential indices}$$

(note: $i = \operatorname{Pivot} r_j$ implies $r_i = 0$, thus $\operatorname{Pivots} R \subseteq I_b$). Then

Compatible basis cycles

For a reduced boundary matrix $R = D \cdot V$, call

$$I_b = \{i : r_i = 0\}$$ *birth* indices ,

$$I_d = \{j : r_j \neq 0\}$$ *death* indices,

$$I_e = I_b \setminus \operatorname{Pivots} R$$ *essential* indices

(note: $i = \operatorname{Pivot} r_j$ implies $r_i = 0$, thus $\operatorname{Pivots} R \subseteq I_b$). Then

$$\Sigma_B = \{r_j \mid j \in I_d\}$$ is a basis of B_*,

Compatible basis cycles

For a reduced boundary matrix $R = D \cdot V$, call

$I_b = \{i : r_i = 0\}$ — *birth* indices ,

$I_d = \{j : r_j \neq 0\}$ — *death* indices,

$I_e = I_b \setminus \operatorname{Pivots} R$ — *essential* indices

(note: $i = \operatorname{Pivot} r_j$ implies $r_i = 0$, thus $\operatorname{Pivots} R \subseteq I_b$). Then

$\Sigma_B = \{r_j \mid j \in I_d\}$ is a basis of B_*,

$\widetilde{\Sigma}_Z = \{v_i \mid i \in I_b\}$ is a basis of Z_*,

Compatible basis cycles

For a reduced boundary matrix $R = D \cdot V$, call

$$I_b = \{i : r_i = 0\} \qquad \textit{birth} \text{ indices ,}$$

$$I_d = \{j : r_j \neq 0\} \qquad \textit{death} \text{ indices,}$$

$$I_e = I_b \setminus \operatorname{Pivots} R \qquad \textit{essential} \text{ indices}$$

(note: $i = \operatorname{Pivot} r_j$ implies $r_i = 0$, thus $\operatorname{Pivots} R \subseteq I_b$). Then

$$\Sigma_B = \{r_j \mid j \in I_d\} \qquad \text{is a basis of } B_*,$$

$$\widetilde{\Sigma}_Z = \{v_i \mid i \in I_b\} \qquad \text{is a basis of } Z_*,$$

$$\Sigma_Z = \Sigma_B \cup \{v_i \mid i \in I_e\} \qquad \text{is } \textit{another} \text{ basis of } Z_*.$$

Compatible basis cycles

For a reduced boundary matrix $R = D \cdot V$, call

$I_b = \{i : r_i = 0\}$	*birth* indices ,
$I_d = \{j : r_j \neq 0\}$	*death* indices,
$I_e = I_b \setminus \operatorname{Pivots} R$	*essential* indices

(note: $i = \operatorname{Pivot} r_j$ implies $r_i = 0$, thus $\operatorname{Pivots} R \subseteq I_b$). Then

$\Sigma_B = \{r_j \mid j \in I_d\}$	is a basis of B_*,
$\widetilde{\Sigma}_Z = \{v_i \mid i \in I_b\}$	is a basis of Z_*,
$\Sigma_Z = \Sigma_B \cup \{v_i \mid i \in I_e\}$	is *another* basis of Z_*.

- Persistent homology is generated by the basis cycles Σ_Z.
- Persistence intervals: $\{[i, j) \mid i = \operatorname{Pivot} r_j\} \cup \{[i, \infty) \mid i \in I_e\}$
- Matrix V not used for barcode
- Columns with indices $\operatorname{Pivots} R$ not used at all

Matrix reduction algorithm

Matrix reduction algorithm (variant of Gaussian elimination):

Require: D: $I \times J$ matrix

Ensure: $R = D \cdot V$: reduced, V: regular upper triangular,
P: persistence pairs

$R := D$

$V := \mathrm{Id}(J \times J)$

for $j \in J$ in increasing order **do**

  while $\exists k < j$ with $i := \mathrm{Pivot}\, r_k = \mathrm{Pivot}\, r_j$ **do**

    $r_j := r_j - \frac{r_{ij}}{r_{ik}} \cdot r_k$ ▷ eliminate pivot entry r_{ij}

    $v_j := v_j - \frac{r_{ij}}{r_{ik}} \cdot v_k$

  if $i := \mathrm{Pivot}\, r_j \neq 0$ **then**

    append (i, j) to P

return R, V

Matrix reduction algorithm

Matrix reduction algorithm (variant of Gaussian elimination):

Require: D: $I \times J$ matrix

Ensure: $R = D \cdot V$: reduced, V: regular upper triangular,

P: persistence pairs

$R := D$

$V := \mathrm{Id}(J \times J)$

for $j \in J$ in increasing order **do**

  while $\exists k < j$ with $i := \operatorname{Pivot} r_k = \operatorname{Pivot} r_j$ **do**

    $r_j := r_j - \frac{r_{ij}}{r_{ik}} \cdot r_k$ ▷ eliminate pivot entry r_{ij}

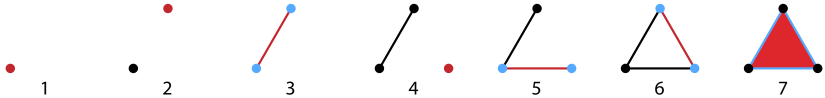
    $v_j := v_j - \frac{r_{ij}}{r_{ik}} \cdot v_k$

  if $i := \operatorname{Pivot} r_j \neq 0$ **then**

    append (i, j) to P

return R, V

Example: matrix reduction



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

R

$= D \cdot$

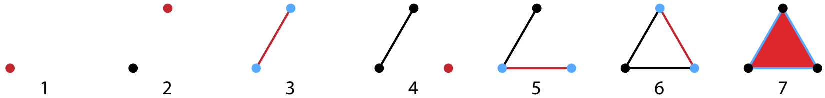
	1	2	3	4	5	6	7
1	1						
2		1					
3			1				
4				1			
5					1		
6						1	
7							1

V

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with $\operatorname{pivot} r_k = \operatorname{pivot} r_j$
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction



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

R

$= D \cdot$

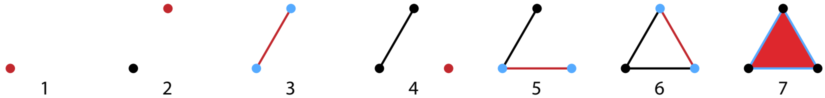
	1	2	3	4	5	6	7
1	1						
2		1					
3			1				
4				1			
5					1		
6						1	
7							1

V

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with pivot r_k = pivot r_j
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction

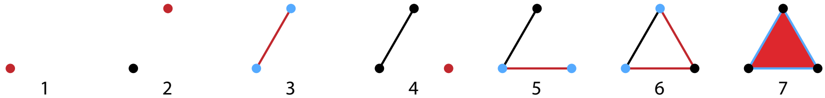


$$\underbrace{\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & & & 1 & & 1 & & \\ 2 & & & 1 & & & 1 & \\ 3 & & & & & & & 1 \\ 4 & & & & & \color{red}{1} & \color{red}{1} & \\ 5 & & & & & & & 1 \\ 6 & & & & & & & 1 \\ 7 & & & & & & & \end{array}}_{R} = D \cdot \underbrace{\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 1 & & & & & & \\ 2 & & 1 & & & & & \\ 3 & & & 1 & & & & \\ 4 & & & & 1 & & & \\ 5 & & & & & 1 & & \\ 6 & & & & & & 1 & \\ 7 & & & & & & & 1 \end{array}}_{V}$$

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with $\operatorname{pivot} r_k = \operatorname{pivot} r_j$
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction

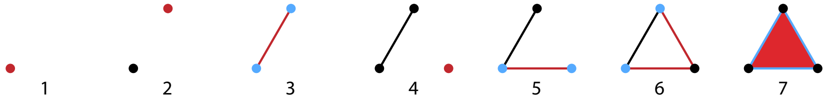


$$\underbrace{\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline
1 & & & 1 & & 1 & 1 & \\
2 & & & 1 & & & 1 & \\
3 & & & & & & & 1 \\
4 & & & & & 1 & 0 & \\
5 & & & & & & & 1 \\
6 & & & & & & & 1 \\
7 & & & & & & &
\end{array}}_{R} = D \cdot \underbrace{\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline
1 & 1 & & & & & & \\
2 & & 1 & & & & & \\
3 & & & 1 & & & & \\
4 & & & & 1 & & & \\
5 & & & & & 1 & 1 & \\
6 & & & & & & 1 & \\
7 & & & & & & & 1
\end{array}}_{V}$$

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with $\operatorname{pivot} r_k = \operatorname{pivot} r_j$
    - add r_k to r_j, add v_k to v_j

Example: matrix reduction



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

R

$= D \cdot$

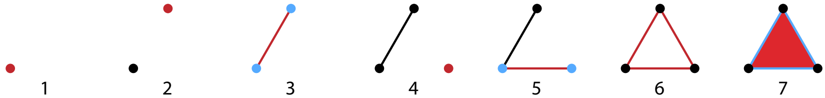
	1	2	3	4	5	6	7
1	1						
2		1					
3			1				
4				1			
5					1	1	
6						1	
7							1

V

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with $\operatorname{pivot} r_k = \operatorname{pivot} r_j$
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction

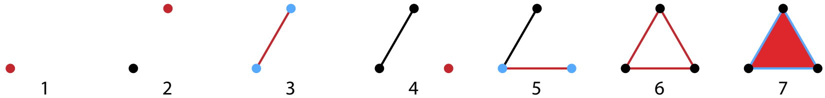


$$\underbrace{\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline
1 & & & 1 & & 1 & \color{red}{0} & \\
2 & & & 1 & & & \color{red}{0} & \\
3 & & & & & & & 1 \\
4 & & & & & 1 & & \\
5 & & & & & & & 1 \\
6 & & & & & & & 1 \\
7 & & & & & & &
\end{array}}_{R} = D \cdot \underbrace{\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline
1 & 1 & & & & & & \\
2 & & 1 & & & & & \\
3 & & & 1 & & & \color{red}{1} & \\
4 & & & & 1 & & & \\
5 & & & & & 1 & 1 & \\
6 & & & & & & 1 & \\
7 & & & & & & & 1
\end{array}}_{V}$$

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with pivot r_k = pivot r_j
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction



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

R

$= D \cdot$

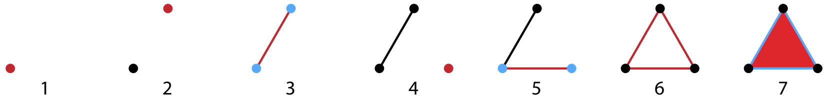
	1	2	3	4	5	6	7
1	1						
2		1					
3			1			1	
4				1			
5					1	1	
6						1	
7							1

V

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with pivot $r_k =$ pivot r_j
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction

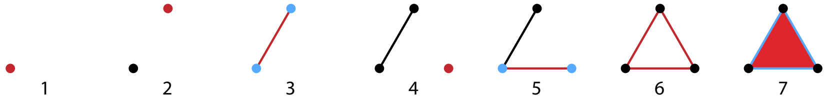


$$\underbrace{\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & & & 1 & & 1 & & \\ 2 & & & 1 & & & & \\ 3 & & & & & & & 1 \\ 4 & & & & & \color{red}{1} & & \\ 5 & & & & & & & 1 \\ 6 & & & & & & & 1 \\ 7 & & & & & & & \end{array}}_{R} = D \cdot \underbrace{\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 1 & & & & & & \\ 2 & & 1 & & & & & \\ 3 & & & 1 & & & 1 & \\ 4 & & & & 1 & & & \\ 5 & & & & & 1 & 1 & \\ 6 & & & & & & 1 & \\ 7 & & & & & & & 1 \end{array}}_{V}$$

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with pivot r_k = pivot r_j
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction

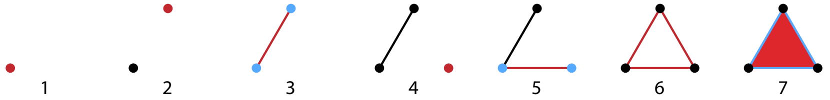


$$\underbrace{\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & \color{red}{5} & 6 & 7 \\ \hline 1 & & & 1 & & 1 & & \\ 2 & & & 1 & & & & \\ 3 & & & & & & & 1 \\ 4 & & & & & \color{red}{1} & & \\ 5 & & & & & & & 1 \\ 6 & & & & & & & 1 \\ 7 & & & & & & & \end{array}}_{R} = D \cdot \underbrace{\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 1 & & & & & & \\ 2 & & 1 & & & & & \\ 3 & & & 1 & & & 1 & \\ 4 & & & & 1 & & & \\ 5 & & & & & 1 & 1 & \\ 6 & & & & & & 1 & \\ 7 & & & & & & & 1 \end{array}}_{V}$$

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with $\operatorname{pivot} r_k = \operatorname{pivot} r_j$
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction



R:

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

$= D \cdot$

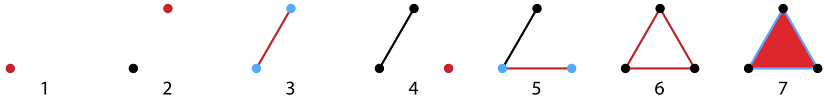
V:

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

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with pivot r_k = pivot r_j
  - add r_k to r_j, add v_k to v_j

Example: matrix reduction



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

R

$= D \cdot$

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

V

Algorithm (over $\mathbb{F}_2$):

- while $\exists k < j$ with pivot r_k = pivot r_j
  - add r_k to r_j, add v_k to v_j

Clearing

Clearing non-essential positive columns

Idea [Chen, Kerber 2011]:

- Don't reduce at non-essential birth indices
- Use the fact that $i = \operatorname{Pivot} r_j$ implies $r_i = 0$
- Reduce boundary matrices of $\partial_d : C_d \to C_{d-1}$ in decreasing dimension $d = k+1, \dots, 1$
- Whenever $i = \operatorname{Pivot} r_j$ (in matrix for ∂_d)
  - Set r_i to 0 (in matrix for ∂_{d-1})

Clearing non-essential positive columns

Idea [Chen, Kerber 2011]:

- Don't reduce at non-essential birth indices
- Use the fact that $i = \operatorname{Pivot} r_j$ implies $r_i = 0$
- Reduce boundary matrices of $\partial_d : C_d \to C_{d-1}$ in decreasing dimension $d = k+1, \dots, 1$
- Whenever $i = \operatorname{Pivot} r_j$ (in matrix for ∂_d)
  - Set r_i to 0 (in matrix for ∂_{d-1})
  - Set v_i to r_j (if V is needed)

Clearing non-essential positive columns

Idea [Chen, Kerber 2011]:

- Don't reduce at non-essential birth indices
- Use the fact that $i = \operatorname{Pivot} r_j$ implies $r_i = 0$
- Reduce boundary matrices of $\partial_d : C_d \to C_{d-1}$ in decreasing dimension $d = k+1, \ldots, 1$
- Whenever $i = \operatorname{Pivot} r_j$ (in matrix for ∂_d)
  - Set r_i to 0 (in matrix for ∂_{d-1})
  - Set v_i to r_j (if V is needed)
- Still yields $R = D \cdot V$ reduced, V regular upper triangular

Note:

- reducing *birth* columns typically harder than death columns: $O\big((i-1)^2\big)$ (birth) vs. $O\big((j-i)^2\big)$ (death)
- with clearing: need only reduce *essential* columns

Clearing boundary matrix reduction

Require: D: $I \times J$ filtration d-boundary matrix,
$\widetilde{P}$: persistence pairs of dimension $(d, d+1)$
Ensure: V: regular upper triangular, $R = D \cdot V$: reduced,
P: persistence pairs of dimension $(d-1, d)$

$\widehat{J} = \{j \in J \mid j$ is not a birth index in $\widetilde{P}\}$
$\widehat{D} = I \times \widehat{J}$ submatrix of D
reduce $\widehat{D}$ to $\widehat{R} = \widehat{D} \cdot \widehat{V}$, yielding persistence pairs P
$R = \operatorname{Expand}(\widehat{R}, I \times J)$ ▷ fill in zeros
$V = \operatorname{Expand}(\widehat{V}, J \times J)$
for $(i, j) \in \widetilde{P}$ **do**
    $v_i = \widetilde{r}_j$
return V, R, P

Counting homology column reductions

Consider K: $k+1$-skeleton of $n-1$-simplex, Rips filtration
Number of columns in coboundary matrix:

$$\underbrace{\sum_{d=1}^{k+1}\binom{n}{d+1}}_{\text{total}} = \underbrace{\sum_{d=1}^{k+1}\binom{n-1}{d}}_{\text{death}} + \underbrace{\sum_{d=1}^{k+1}\binom{n-1}{d+1}}_{\text{birth}}$$

$$= \underbrace{\sum_{d=1}^{k+1}\binom{n-1}{d}}_{\text{death}} + \underbrace{\binom{n-1}{k+2}}_{\text{essential}} + \underbrace{\sum_{d=1}^{k}\binom{n-1}{d+1}}_{\text{cleared}}$$

Counting homology column reductions

Consider K: $k+1$-skeleton of $n-1$-simplex, Rips filtration

Number of columns in coboundary matrix:

$$\underbrace{\sum_{d=1}^{k+1}\binom{n}{d+1}}_{\text{total}} = \underbrace{\sum_{d=1}^{k+1}\binom{n-1}{d}}_{\text{death}} + \underbrace{\sum_{d=1}^{k+1}\binom{n-1}{d+1}}_{\text{birth}}$$

$$= \underbrace{\sum_{d=1}^{k+1}\binom{n-1}{d}}_{\text{death}} + \underbrace{\binom{n-1}{k+2}}_{\text{essential}} + \underbrace{\sum_{d=1}^{k}\binom{n-1}{d+1}}_{\text{cleared}}$$

$k=2, n=192$: $\quad 56\,050\,096 = 1\,161\,471 + 53\,727\,345 + 1\,161\,280$

- Clearing didn't help much!

Cohomology

Persistent (co)homology

How is persistent cohomology related to persistent homology?

- Cohomology (over $\mathbb{F}$) is vector space dual to homology:

$$H^p(K;\mathbb{F}) \cong H_p(K;\mathbb{F})^* = \operatorname{Hom}(H_p(K;\mathbb{F}),\mathbb{F}).$$

- Duality preserves ranks of linear maps (in finite dimensions): for $f : V \to W$ with dual $f^* : W^* \to V^*$,

$$\operatorname{rank} f^* = \operatorname{rank} f.$$

- The persistence barcode of a persistence module is uniquely determined by the ranks of the internal maps.

Thus, persistent homology and persistent cohomology have the same barcode [de Silva et al 2011].

Persistent (relative) homology

How is persistent relative homology related to persistent homology?

- The short exact sequence of filtered chain complexes (with coefficients in $\mathbb{F}$)

$$0 \to C_p(K_\bullet) \to C_p(K) \to C_p(K, K_\bullet) \to 0$$

induces a long exact sequence in persistent homology

$$\cdots \to H_{p+1}(K, K_\bullet) \to H_p(K_\bullet) \to H_p(K) \to H_p(K, K_\bullet) \to \cdots$$

and analogously for persistent cohomology.

Persistent (relative) (co)homology

Consider the long exact sequence

$$\cdots \to H_{p+1}(K) \xrightarrow{r} H_{p+1}(K, K_\bullet) \xrightarrow{\partial} H_p(K_\bullet) \xrightarrow{i} H_p(K) \to \cdots$$

We have:

- $B(\ker(i)) = \{[b, d) \in B(H_p(K_\bullet))\} = B(\operatorname{im}(\partial))$
- $B(\operatorname{im}(i)) = \{[b, \infty) \in B(H_p(K_\bullet))\}$
- $B(\operatorname{coker}(i)) = \{(-\infty, b) \mid [b, \infty) \in B(H_p(K_\bullet))\} = B(\operatorname{im}(r))$
- $H_p(K_\bullet) \cong \operatorname{im}(\partial) \oplus \operatorname{im}(i)$
- $H_{p+1}(K, K_\bullet) \cong \operatorname{im}(\partial) \oplus \operatorname{im}(r)$

Thus, the barcodes of persistent (co)homology and of persistent relative (co)homology determine each other [de Silva et al 2011; Schmahl 2018].

Cohomology and clearing

- Cohomology allows for clearing starting from dimension 0 (avoiding the initial overhead)
- Persistence in dimension 0 has special algorithms (Kruskal's algorithm for MST, union-find data structure)
- Persistent cohomology arises from a *reverse* filtration

$$C^*(K_0) \twoheadleftarrow C^*(K_1) \twoheadleftarrow \cdots \twoheadleftarrow C^*(K_n)$$

- Persistent *relative* cohomology arises from a filtration

$$C^*(K, K_0) \hookleftarrow C^*(K, K_1) \hookleftarrow \cdots \hookleftarrow C^*(K, K_n)$$

  - Can be computed by reduction of coboundary matrix: transpose of boundary matrix, with reversed index orders

Counting cohomology column reductions

Consider K: $k+1$-skeleton of $n-1$-simplex, Rips filtration

Number of columns in boundary matrix:

$$\underbrace{\sum_{d=0}^{k} \binom{n}{d+1}}_{\text{total}} = \underbrace{\sum_{d=0}^{k} \binom{n-1}{d+1}}_{\text{death}} + \underbrace{\sum_{d=0}^{k} \binom{n-1}{d}}_{\text{birth}}$$

$$= \underbrace{\sum_{d=0}^{k} \binom{n-1}{d+1}}_{\text{death}} + \underbrace{\binom{n-1}{0}}_{\text{essential}} + \underbrace{\sum_{d=1}^{k} \binom{n-1}{d}}_{\text{cleared}}$$

Counting cohomology column reductions

Consider K: $k+1$-skeleton of $n-1$-simplex, Rips filtration

Number of columns in boundary matrix:

$$\underbrace{\sum_{d=0}^{k}\binom{n}{d+1}}_{\text{total}} = \sum_{d=0}^{k}\underbrace{\binom{n-1}{d+1}}_{\text{death}} + \sum_{d=0}^{k}\underbrace{\binom{n-1}{d}}_{\text{birth}}$$

$$= \sum_{d=0}^{k}\underbrace{\binom{n-1}{d+1}}_{\text{death}} + \underbrace{\binom{n-1}{0}}_{\text{essential}} + \sum_{d=1}^{k}\underbrace{\binom{n-1}{d}}_{\text{cleared}}$$

$k = 2, n = 192$: $\quad 1\,179\,808 = 1\,161\,471 + 1 + 18\,336$

Observations

For a typical input:

- V has very few off-diagonal entries
- most death columns of D are already reduced

Observations

For a typical input:

- V has very few off-diagonal entries
- most death columns of D are already reduced

Previous example ($k = 2, n = 192$):

- Only $79 + 42 + 1 = 122$ out of $192 + 18\,145 + 1\,143\,135 = 1\,161\,471$ columns are actually modified in matrix reduction

Implicit matrix reduction

Implicit matrix reduction

Standard approach:

- Boundary matrix D for filtration-ordered basis
  - Explicitly generated and stored in memory

Implicit matrix reduction

Standard approach:

- Boundary matrix D for filtration-ordered basis
  - Explicitly generated and stored in memory
- Matrix reduction: store only reduced matrix R
  - transform D into R by column operations

Implicit matrix reduction

Standard approach:

- Boundary matrix D for filtration-ordered basis
  - Explicitly generated and stored in memory
- Matrix reduction: store only reduced matrix R
  - transform D into R by column operations

Approach for Ripser:

- Boundary matrix D for lexicographically ordered basis
  - Implicitly defined and recomputed when needed

Implicit matrix reduction

Standard approach:

- Boundary matrix D for filtration-ordered basis
  - Explicitly generated and stored in memory
- Matrix reduction: store only reduced matrix R
  - transform D into R by column operations

Approach for Ripser:

- Boundary matrix D for lexicographically ordered basis
  - Implicitly defined and recomputed when needed
- Matrix reduction in Ripser: store only coefficient matrix V
  - recompute previous columns of $R = D \cdot V$ when needed
  - Typically, V is much sparser and smaller than R

Implicit boundary matrix reduction algorithm

Require: D: $I \times J$ matrix

Ensure: $R = D \cdot V$: reduced, V: regular upper triangular,
P: persistence pairs

for $j \in J$ in increasing order **do**
  $v_j := e_j$
  $r_j := d_j$
  while $\exists k < j$ with $i := \operatorname{Pivot} r_j$ and $(i, k) \in P$ **do**
    $r_j := r_j - r_{ij} \cdot D \cdot v_k$ ▷ eliminate pivot entry r_{ij}
    $v_j := v_j - r_{ij} \cdot v_k$
  if $i := \operatorname{Pivot} r_j \neq 0$ **then**
    $v_j = r_{ij}^{-1} \cdot v_j$ ▷ make pivot entry $r_{ij} = 1$
    append (i, j) to P

return V

Implementation details

Current (working) columns v_j, r_j:

- priority queue (heap), comparison-based
- representing a linear combination of row basis elements
- elements: tuples consisting of
  - coefficient
  - row index
  - diameter of corresponding simplex
- pivot entry lazily evaluated (when extracting top element)

Previous (finalized) columns v_k $(k < j)$:

- sparse matrix data structure

Apparent and emergent pairs

Natural filtration settings

Typical assumptions on the filtration:

• general filtration	persistence (in theory)
• filtration by singletons or pairs	discrete Morse theory
• simplexwise filtration	persistence (computation)

Natural filtration settings

Typical assumptions on the filtration:

• general filtration	persistence (in theory)
• filtration by singletons or pairs	discrete Morse theory
• simplexwise filtration	persistence (computation)

Discrete Morse theory sits between persistence and persistence (!)

Morse pairs and persistence pairs

Consider a *Morse filtration* (one or two simplices at a time).

Morse pair (σ, τ):

- inserting σ and τ simultaneously does not change the *homotopy type*

Morse pairs and persistence pairs

Consider a *Morse filtration* (one or two simplices at a time).
Morse pair (σ, τ):

- inserting σ and τ simultaneously does not change the *homotopy type*

Consider a *simplexwise filtration* (one simplex at a time).
Persistence pair (σ, τ):

- inserting simplex σ creates a new *homological* feature
- inserting τ destroys that feature again

Apparent pairs

Definition

In a simplexwise filtration, (σ, τ) is an *apparent* pair if

- σ is the youngest face of τ
- τ is the oldest coface of σ

Apparent pairs

Definition

In a simplexwise filtration, (σ, τ) is an *apparent* pair if

- σ is the youngest face of τ
- τ is the oldest coface of σ

Lemma

The apparent pairs are persistence pairs.

Lemma

The apparent pairs form a discrete gradient.

- Generalizes a construction proposed by [Kahle 2011] for the study of random Rips filtrations
- Apparent pairs also appear (independently) in Eirene [Henselmann 2016] and in [Mendoza-Smith, Tanner 2017]

From Morse theory to persistence and back

Proposition (from Morse to persistence)

The pairs of a Morse filtration are apparent 0*-persistence pairs for the canonical simplexwise refinement of the filtration.*

From Morse theory to persistence and back

Proposition (from Morse to persistence)

The pairs of a Morse filtration are apparent 0*-persistence pairs for the canonical simplexwise refinement of the filtration.*

Proposition (from persistence to Morse)

Consider an arbitrary filtration with a simplexwise refinement.
The apparent 0*-persistence pairs yield a Morse filtration*

- *refining the original one, and*
- *refined by the simplexwise one.*

Emergent persistent pairs

A persistence pair (σ, τ) for a simplexwise filtration is

- an *emergent face pair* if σ is the youngest proper face of τ,
- an *emergent coface pair* if τ is the oldest proper coface of σ.

Emergent persistent pairs

A persistence pair (σ, τ) for a simplexwise filtration is

- an *emergent face pair* if σ is the youngest proper face of τ,
- an *emergent coface pair* if τ is the oldest proper coface of σ.

Lemma

Consider the lexicographically refined Rips filtration. Assume that

- *τ is the lexicographically minimal proper coface of σ with $\operatorname{diam}(\tau) = \operatorname{diam}(\sigma)$, and*
- *τ is not already in a persistence pair (ρ, τ) with $\rho \succ \sigma$.*

Then (σ, τ) is a 0-persistence emergent coface pair.

Emergent persistent pairs

A persistence pair (σ, τ) for a simplexwise filtration is

- an *emergent face pair* if σ is the youngest proper face of τ,
- an *emergent coface pair* if τ is the oldest proper coface of σ.

Lemma

Consider the lexicographically refined Rips filtration. Assume that

- τ *is the lexicographically minimal proper coface of* σ *with* $\operatorname{diam}(\tau) = \operatorname{diam}(\sigma)$, *and*
- τ *is not already in a persistence pair* (ρ, τ) *with* $\rho \succ \sigma$.

Then (σ, τ) *is a* 0*-persistence emergent coface pair.*

- Includes all apparent pairs with persistence 0
- Can be identified *without* enumerating all cofaces of σ (shortcut for computation)

Speedup from emergent pairs shortcut

Previous example ($k = 2, n = 192$):

- Only $36\,672 + 164\,214 + 3\,392\,039 = 3\,592\,925$ out of $36\,672 + 3\,447\,550 + 216\,052\,515 = 219\,536\,737$ nonzero entries of the coboundary matrix are actually visited

Speedup from emergent pairs shortcut

Previous example ($k = 2, n = 192$):

- Only $36\,672 + 164\,214 + 3\,392\,039 = 3\,592\,925$ out of $36\,672 + 3\,447\,550 + 216\,052\,515 = 219\,536\,737$ nonzero entries of the coboundary matrix are actually visited

Using implicit reduction (boundary matrix columns may be revisited multiple times):

- Only $155\,474 + 253\,134 + 7\,500\,332 = 7\,908\,940$ out of $155\,474 + 3\,536\,470 + 220\,160\,808 = 223\,852\,752$ nonzero entries are actually visited

Speedup: $1.2\,s$ vs. $5.6\,s$ (factor 4.7)

Ripser Live: users from 567 different cities