Clear and Compress: Computing Persistent Homology in Chunks

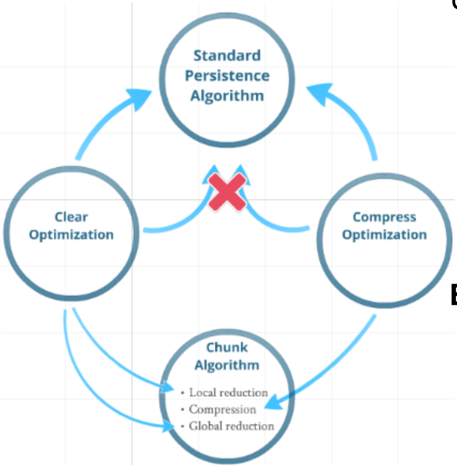
Ulrich Bauer[1] Michael Kerber[2] Jan Reininghaus[1]

[1] Institute of Science and Technology (IST) Austria

[2]Stanford University and Max Planck Center for Visual Computing and Communication, Saarbrücken, Germany

SIAM AG 2013

Overview



Complexity analysis

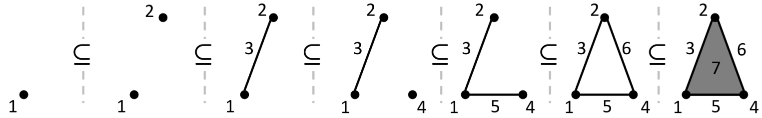
$O(m\ell^3 + g\ell n + g^3)$

- $n = \#$simplices
- $\ell =$ max. size of chunk
- $m = \#chunks$
- $g = \#$non-local pairs

Experimental results

- Parallelized version
- Outperforms standard algorithm, Dionysus
- Faster than clear optimization

Standard Persistence Algorithm (reminder)



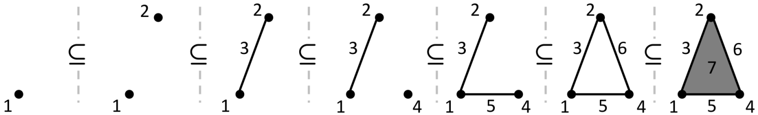
	1	2	3	4	5	6	7
1			1		1		
2			1			1	
3							1
4					1	1	
5							1
6							1
7							

Algorithm:

for i from 1 to n:

- while pivot[j]=pivot[i] for some $j < i$: add column j to column i

Standard Persistence Algorithm (reminder)



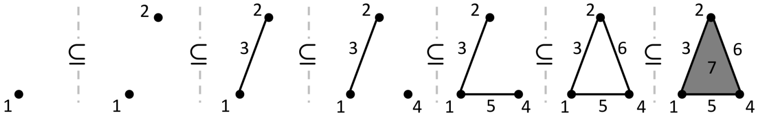
	1	2	3	4	5	6	7
1			1		1		
2			1			1	
3							1
4					1	1	
5							1
6							1
7							

Algorithm:

for i from 1 to n:

- while pivot[j]=pivot[i] for some $j < i$: add column j to column i

Standard Persistence Algorithm (reminder)



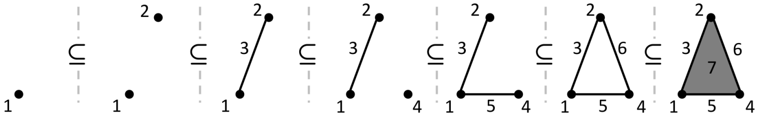
	1	2	3	4	5	6	7
1			1		1	1	
2			1			1	
3							1
4					1	0	
5							1
6							1
7							

Algorithm:

for i from 1 to n:

- while pivot[j]=pivot[i] for some $j < i$: add column j to column i

Standard Persistence Algorithm (reminder)



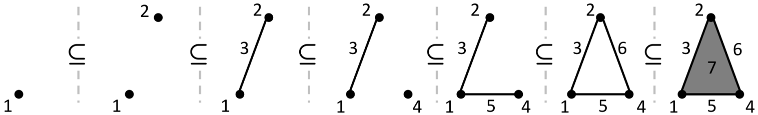
	1	2	3	4	5	6	7
1			1		1	0	
2			1			0	
3							1
4					1		
5							1
6							1
7							

Algorithm:

for *i* from 1 to *n*:

- while pivot[*j*]=pivot[*i*] for some $j < i$: add column *j* to column *i*

Standard Persistence Algorithm (reminder)



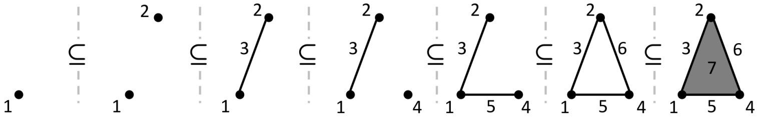
	1	2	3	4	5	6	7
1			1		1		
2			1				
3							1
4					1		
5							1
6							1
7							

Algorithm:

for i from 1 to n:

- while pivot[j]=pivot[i] for some $j < i$: add column j to column i

Standard Persistence Algorithm (reminder)



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

- positive inessential, negative, and essential simplices/columns
- Column i is zero $\Leftrightarrow$ i-th simplex is positive
- All column additions within the same dimension
- Pivots do not dependent on the order of operations

Clear Optimization [Chen, K. 2011]

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

(i, j) is persistence pair

$\Rightarrow$ i-th simplex is positive

$\Rightarrow$ column i reduces to 0

Clear Optimization [Chen, K. 2011]

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

Algorithm:

for δ from dim K downto 0:

- For simplices of dim. δ, left to right:
  - while pivot[j]=pivot[i] for some $j < i$, add column j to column i
- For any column i with pivot[i]=j: set column j to zero

Compress Optimization [Zomorodian, Carlsson 2004]

	1	2	3	4	5	6	7	8	9	10	11	12	13
1			1			1			1				
2			1		1								
3							1			1		1	1
4					1	1							
5							1			1			1
6							1					1	1
7													
8									1				
9										1		1	

(i, j) is persistence pair

$\Rightarrow$ j is not the pivot of any column

$\Rightarrow$ Setting row j to zero is fine!

Compress Optimization [Zomorodian, Carlsson 2004]

	1	2	3	4	5	6	7	8	9	10	11	12	13
1			1			1			1				
2			1		1								
3							1			1		1	1
4					1	1							
5							1			1			1
6							1					1	1
7													
8									1				
9										1		1	

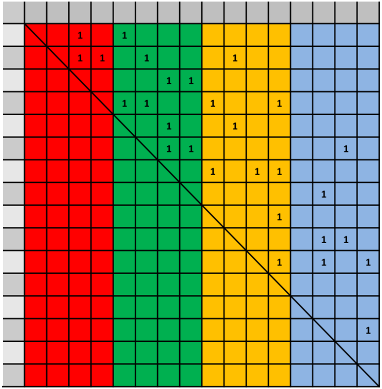
Algorithm:

for *i* from 1 to *n*:

- Remove row indices for negative simplices (compression)
- while pivot[*j*]=pivot[*i*] for some $j < i$, add column *j* to column *i*

Chunk Algorithm: Local Reduction

Local simplex: Paired with simplex in same or adjacent chunk
Global simplex: non-local (must have persistence $\geq \ell$)



Local reduction algorithm:

- Reduce columns using only columns from same chunk
- Reduce columns using only columns from same or left-neighboring chunk

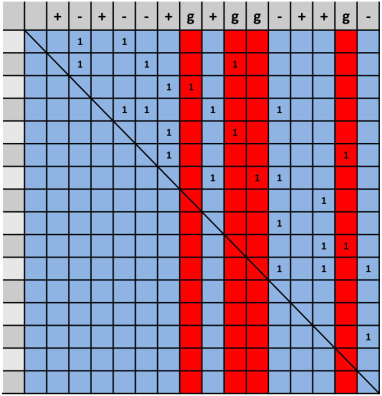
Main observation:

- Local columns are reduced

Running time: $O(m\ell^3)$

Chunk Algorithm: Local Reduction

Local simplex: Paired with simplex in same or adjacent chunk
Global simplex: non-local (must have persistence $\geq \ell$)



Local reduction algorithm:

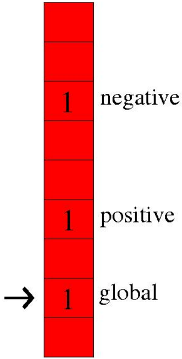
- Reduce columns using only columns from same chunk
- Reduce columns using only columns from same or left-neighboring chunk

Main observation:

- Local columns are reduced

Running time: $O(m\ell^3)$

Chunk Algorithm: Compression



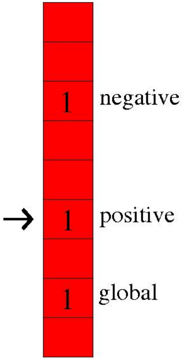
For each global column:
iterate over entries from bottom to top

- Global index: skip
- Local positive: add (local) column with same pivot
- Local negative: set to zero (compress)

Running time: $O(g(n + n\ell)) = O(gn\ell)$

Result: Only global indices remain

Chunk Algorithm: Compression



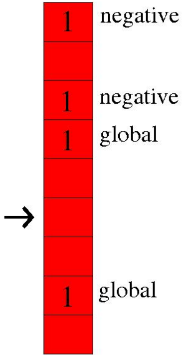
For each global column:
iterate over entries from bottom to top

- Global index: skip
- Local positive: add (local) column with same pivot
- Local negative: set to zero (compress)

Running time: $O(g(n + n\ell)) = O(gn\ell)$

Result: Only global indices remain

Chunk Algorithm: Compression



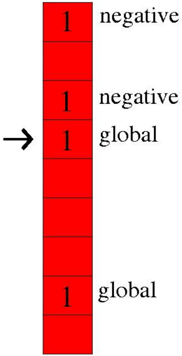
For each global column:
iterate over entries from bottom to top

- Global index: skip
- Local positive: add (local) column with same pivot
- Local negative: set to zero (compress)

Running time: $O(g(n + n\ell)) = O(gn\ell)$

Result: Only global indices remain

Chunk Algorithm: Compression



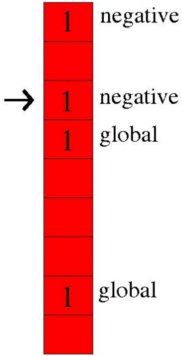
For each global column:
iterate over entries from bottom to top

- Global index: skip
- Local positive: add (local) column with same pivot
- Local negative: set to zero (compress)

Running time: $O(g(n + n\ell)) = O(gn\ell)$

Result: Only global indices remain

Chunk Algorithm: Compression



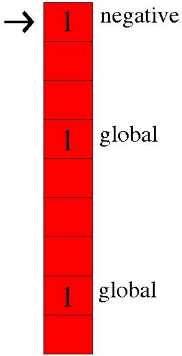
For each global column:
iterate over entries from bottom to top

- Global index: skip
- Local positive: add (local) column with same pivot
- Local negative: set to zero (compress)

Running time: $O(g(n + n\ell)) = O(gn\ell)$

Result: Only global indices remain

Chunk Algorithm: Compression



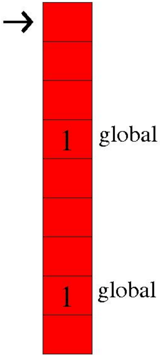
For each global column:
iterate over entries from bottom to top

- Global index: skip
- Local positive: add (local) column with same pivot
- Local negative: set to zero (compress)

Running time: $O(g(n + n\ell)) = O(gn\ell)$

Result: Only global indices remain

Chunk Algorithm: Compression



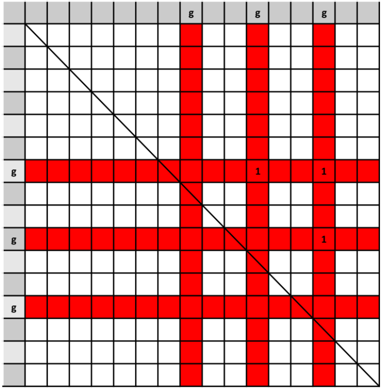
For each global column:
iterate over entries from bottom to top

- Global index: skip
- Local positive: add (local) column with same pivot
- Local negative: set to zero (compress)

Running time: $O(g(n + n\ell)) = O(gn\ell)$

Result: Only global indices remain

Chunk Algorithm: Global Reduction



Reduce $g \times g$ submatrix: $O(g^3)$

Further optimizations:

- Avoid additions during compression
  - *inactive indices:* either
    - negative, or
    - positive and pivot of column with otherwise only inactive entries
  - inactive indices can be set to zero
- Avoid compression of global positive columns (clear optimization)

Complexity Analysis

$$O(m\ell^3 + gn\ell + g^3)$$

- $n = \#$simplices
- $\ell =$ max. size of chunk
- $m = \#chunks$
- $g = \#$non-local pairs

$\sqrt{n}$ chunks of size $\sqrt{n}$:

$$O(n^2 + g_1 n\sqrt{n} + g_1^3)$$

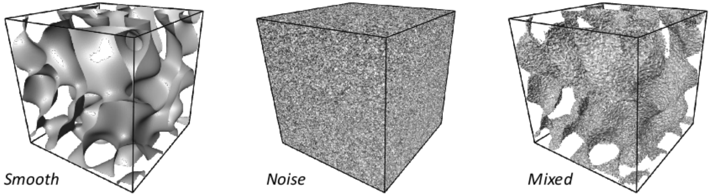
$\frac{n}{\log n}$ chunks of size $\log n$:

$$O(n \log^2 n + g_2 n \log n + g_2^3)$$

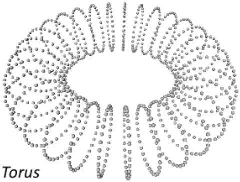
Special case: d-dimensional image + lower star filtration:

$$O(n + gn + g^3) = O(gn + g^3)$$

Experimental results



Morse filtrations



Alpha filtration

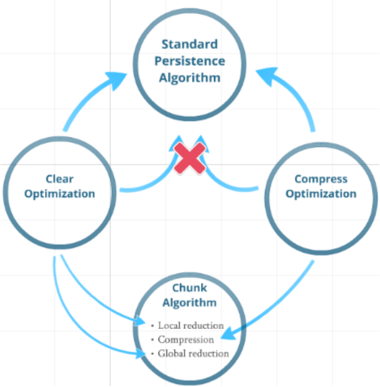
Rips filtration

Experimental results

Dataset	$n \cdot 10^{-6}$	std. [8]	twist [4]	cohom. [5]	DMT [10]	g/n	chunk (1x)	chunk (12x)
Smooth	16.6	383s	3.1s	65.8s	2.0s	0%	5.0s	0.9s
Smooth$^\perp$	16.6	432s	11.3s	20.8s	–	0%	6.3s	0.9s
Noise	16.6	336s	17.2s	15971s	13.0s	9%	28.3s	6.3s
Noise$^\perp$	16.6	1200s	29.0s	190.1s	–	9%	31.1s	5.8s
Mixed	16.6	330s	5.8s	50927s	12.3s	5%	21.6s	2.4s
Mixed$^\perp$	16.6	446s	13.0s	32.7s	–	5%	32.0s	2.9s
Torus	0.6	52s	0.3s	1.6s	–	7%	0.3s	0.1s
Torus$^\perp$	0.6	24s	0.3s	1.4s	–	7%	0.9s	0.2s
Mumford	2.4	38s	35.2s	2.8s	–	82%	14.6s	1.8s
Mumford$^\perp$	2.4	58s	0.2s	184.1s	–	82%	1.5s	0.4s

http://phat.googlecode.com

Summary



- Homological version of discrete Morse theory approach
  [Günther, R., Wagner, Hotz 2012]
- Parallelizable
- Distributed memory?
- Streaming?