CS 6170 Computational Topology: Topological Data Analysis **University of Utah**
Spring 2017 **School of Computing**

Lecture 16: March 2, 2017

Lecturer: Prof. Bei Wang <beiwang@sci.utah.edu> *Scribe: Zahra Fahimfar*

In this lecture, we are going to see a different version of standard reduction algorithm. It has reduced the computation time and gives us a better running time.

16.1 Spectral Sequences

Algorithm 1 Spectral Sequences

```
1: for r = 1 to m do                                   ▷ In this algorithm D is a boundary matrix, r is phase
2:     for j = r to m do
3:         for columns l in D^j do
4:             while low(l)∈ D_{j−r+1} and ∃ l' ∈ D_{j−r+1}
5:                     s.t. low(l')=low(l) do
6:                     add l' to l
7:             end while
8:         end for
9:     end for
10: end for
```

16.1.1 Example

In this algorithm, we split ∂ into blocks, for example if $m = 4$, we have $4 * 4$ blocks. So, filtration has 16 steps and blocks do not need to have the same size. We process each block in sequence.
first thing to notice is every block below diagonal is all zeros since this is a boundary matrix. So, we start with the block in the diagonal. For example, if $j = 3$ The columns in D^3 are as follows as you can see it in the Figure 16.1:

$$l = 9, 10, 11, 12$$

In the first phase we only process the columns in the block 3 in sequence and locally. Locally means that we reduce the block with only considering the columns in this region or blocks instead of considering all columns.

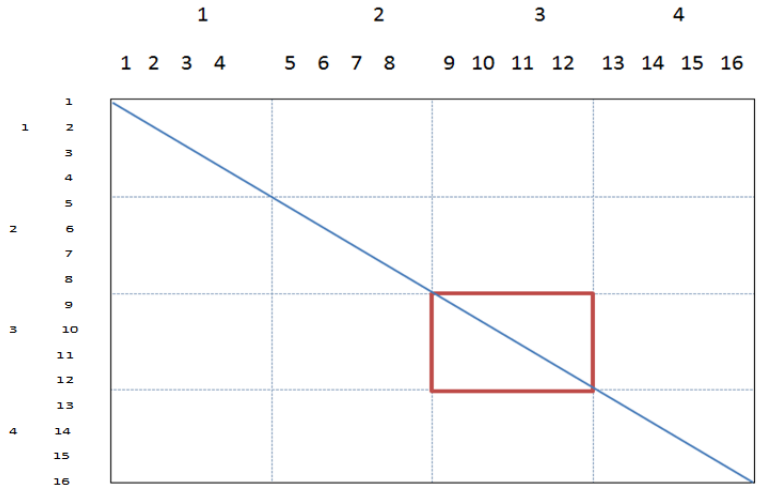
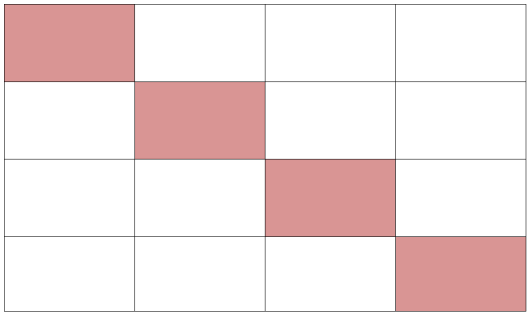


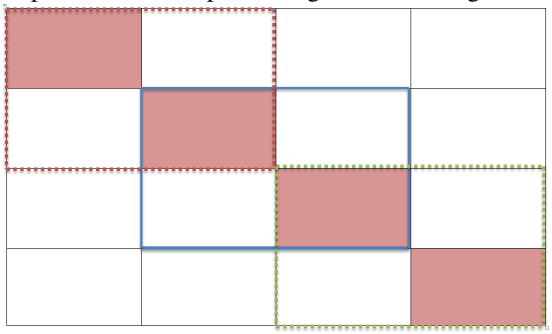
Figure 16.1:

16.1.2 Phases

In the first phase, we are going to process and reduce diagonal blocks.

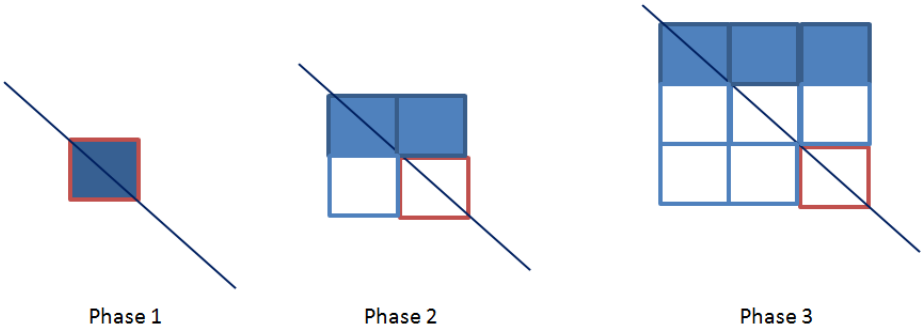


In phase 2, we still processing the same thing. we look at pivot in the blocks above and one left of the diagonal blocks.



Therefore in the last phase we reduce all columns. And what is happening here is that in the first phase we process all the diagonal. In the second phase we process the second diagonal. So instead of sweeping all matrix column wise from left to right, we are going diagonally from main diagonal.

Here is another picture to show the different phases:



We only look for columns which have pivot in the blocks highlighted in blue color in above picture.

We actually end up doing more work. In the worse case, it is $o(n^3)$ run time and takes more operation and look ups than standard algorithm. But we can do particularization. Local reduction of each block is indecent of others blocks. And that's the idea of the reduction of chunk.

16.2 Speed-ups

Algorithm 1 describes a way of reducing the boundary matrix, but it performs more operations. We now present a simple techniques which leads to a significant decrease in the number of required operations.

16.2.1 Twist or Clearing

If column j is non-zero negative column with $i = pivot(j) \Rightarrow$ i is +ve.

The key behind our optimization is the following fact: if i appears as the pivot in a reduced column of M, the index i is positive and hence there exists a sequence of left-to-right operations on Mi that turn it to zero. Instead of explicitly executing this sequence of operations, we define the clear operation by setting column Mi to zero directly. Informally speaking, a clear is a shortcut to avoid some column operations in the reduction when the result is evident.

16.2.2 Compression

Corollary 1:
If column i is negative column. Then i is not a pivot for any column j.

As a consequence, whenever a negative column with index j has been reduced, row j can be set to zero before further reducing.

Corollary 2:
Let i be negative column, for any column j s.t. ith row of column j is 1 $\Rightarrow$ setting it to 0 does not affect the pairs.

16.3 Reduction in chunks

The two optimization techniques from Section 2 both yield significant speed-ups, but they are not easily combinable. In this section, we present an algorithm which combines both optimization techniques. We call a column j local if it forms a persistence pair with another column in the same chunk or in one of the adjacent chunks. In this case, we also call the persistence pair local. Non-local columns (and pairs) are called global.

The high-level description of our new algorithm consists of following steps:

1. Partition D into blocks
2. Partially reduce each block independently (parallel)
3. Independently compress all global columns
4. Reduce submatrix of global rows and columns

Let $m \in N$. Fix $m+1$ numbers $0 = t_0 < t_1 < ... < t_{m1} < t_m = n$ and define the ith chunk of D to be the columns of D with indices $\{t_{i1} + 1, ..., t_i\}$.

Local chunk reduction: The first step of our algorithm computes the local pairs by performing two phases of the spectral sequence algorithm [Algorithm 1]. We apply left-to-right operations as usual, but in the first phase we only add columns from the same chunk, and in the second phase we only add columns from both the same chunk and its left neighbor. After phase r, for each $b \in r, ..., m$ the submatrix is reduced. Conversely, any local pair (i, j) is detected by this method after two phases. We incorporate the clearing operation for efficiency, that is, we proceed in decreasing dimension and set detected local positive columns to zero; see Algorithm 2.

Algorithm 2 Local chunk reduction ($M = \partial$)

$R \leftarrow M$;$L = [0, ..., 0]$
for $\delta = d, ..., 0$ **do** ▷ Label based on dimension
  for $r = 1, 2$ **do** ▷ Perform two phases of the spectral sequence algorithm
    for $b = r, ..., m$ **do** ▷ Loop is parallelizable
      for $j = t_{b-1} + 1, ..., t_b$with dim $\sigma_j = \delta$ **do**
        if (j is not paired) **then**
          while $R_j \neq 0$ and $L[pivot(R_j)] \neq 0$and $pivot(R_j) > t_{b-r}$ **do**
          $L[pivot(R_j)] = j$
          $R_j \leftarrow 0$ ▷ Clear column i
          end while
        end if
      end for
    end for
  end for
end for
Return(R,L)

Algorithm 3 mark active Entries(R)

```
for each unpaired column k in R do
    MARK-COLUMN(R,k)
end for
function MARK-COLUMN(R, k)
for each non-zero row index i of R_k do
    if i is unpaired then
        mark R → active
    end if
    if is is positive then
        j = L[i]
    end if
    if j ≠ k and MARK- COLUMN(R,j) then
        mark k as active
    end if
    mark k as inactive
end for
```

Global column compression: Let R be the matrix returned by Algorithm 2. Before computing the global persistence pairs, we first compress the global columns, using the ideas from Section 2.
The compression proceeds in two steps: first, every non-zero entry of a global column is classified as active or inactive (using Algorithm 3). Then, we iterate over the global columns, set all entries with inactive index to zero, and eliminate any non-zero entry with a local positive index i(see Algorithm 4).

Submatrix reduction: After having compressed all global columns, these form a matrix nested in R. To complete the computation of the persistence pairs, we simply perform standard reduction on the remaining matrix. For efficiency, we perform steps 2 and 3 alternatingly for all dimensions in decreasing order and apply the clearing optimization; this way, we avoid the compression of positive global columns. Algorithm 5 summarizes the whole method.

Algorithm 4 Compress(R,k)

```
for each non-zero entry index i of R_k do
    if i is paired then
        if i is inactive then
            R_i^k ← 0
        else
            j = [i]                          ▷ (i, j) is persistence pair
            R_i^k ⟸ 0
        end if
    end if
end for
```

Algorithm 5 Persistence in chunks

(R,L) ← Local chunk reduction(Algorithm 2) ▷ step 1: reduce local columns
Mark active entries(R) ← Algorithm 2
for $\delta = d, ..., 0$ **do**
 ▷ compress global columns
  for $j = 1, .., n$ with dim $\sigma_j = \delta$ **do** ▷ Loop is parallelizable
    if column j is not paired **then**
      Compress(R,j)
    end if
  end for
  for $j = 1, ..., n$ with dim $\sigma_j = \delta$ **do** ▷ step 3: reduce global columns
    while $R_j \neq 0$ and $L[pivot(R_j)] \neq 0$ **do**
      $R_j = R_j + R_{L[pivot(R_j)]}$
    end while
    if $R_j \neq 0$ **then**
      $L[pivot(R_j)] = j$
      $R_{pivot(R_j)} = 0$ ▷ Clear column i
      mark(i,j) as paired
    end if
  end for
end for

References

Chen C, Kerber M, editors. Persistent homology computation with a twist. Proceedings 27th European Workshop on Computational Geometry; 2011.

Chen, Chao, and Michael Kerber. "Persistent homology computation with a twist." Proceedings 27th European Workshop on Computational Geometry. Vol. 11. 2011.

Bauer, Ulrich, Michael Kerber, and Jan Reininghaus. "Distributed computation of persistent homology." 2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics, 2014.

Edelsbrunner, Herbert, and John Harer. Computational topology: an introduction. American Mathematical Soc., 2010.