Notes on Distributed Computation of Persistent Homology using the Blowup Complex

RHL DM

May 8, 2014

1 Background

Definitions and Notation. The following table outlines the basic notation and terminology.

K	the domain (cell complex)
$n = \lvert K\rvert$	complex size
$C = \{C_i\}_{i\in\triangle^p}$	cover
$N \subseteq 2^{\triangle^p}$	$\operatorname{Nrv} C$, the nerve of the cover, sub complex of $2^{\triangle^p}$
$K^\sigma = \cap_{i\in\sigma} C_i$	*i do not have a name*
$K^C = \cup_{J\subseteq[n]} K^J \times \triangle^J$	the blowup complex
$m = \lvert K^C_{>0}\rvert$	number of blowup cells
$m_\sigma = \lvert K^\sigma\rvert$	size of thing without a name.
$K^C_i = \{* \times \sigma \in K^C \mid \dim\sigma = i\}$	i-cells of the blowup
$K^C_\sigma = \{* \times \sigma \in K^C\}$	cells of the blowup indexed by $\sigma \in N$
$K^C_{<1} = K^C_0$	low dimensional blowup cells
$K^C_{>0} = K^C - K^C_0$	high dimensional blowup cells
D	graded boundary matrix of the blowup complex
D_σ	submatrix of columns of D whose columns represent the boundaries of K^C_σ
D^τ_σ	submatrix of D_σ whose rows are indexed by K^C_σ
$D_{>0}$	submatrix of D whose columns are indexed by $K^C_{>0}$
$D_{<1}$	submatrix of D whose columns are indexed by K^C_0
$D^\tau_{>0}$	submatrix of $D_{>0}$ whose rows are indexed by $\tau \in N$
D^τ_0	submatrix of D_0 whose rows are indexed by $\tau \in N$

Remark 1. *All vertices in N are of the form $\{i\}$ for $i \in [p]$.*
Let u, v be vertices in N. By the definition of ∂ for K^C the matrix $D^u_v = 0$ whenever $u \neq v$.
We write D_i and $D^i_{\geq 0}$ for $i \in [p]$ whenever $\{i\} \in N$.

2 Algorithm

- $p + 1$ processors in total.
- Processor i gets matrix D_i.
- Processor p gets matrix $D_{>0}$, with columns indexed by $K^C_{>0}$ and rows indexed by K^C.

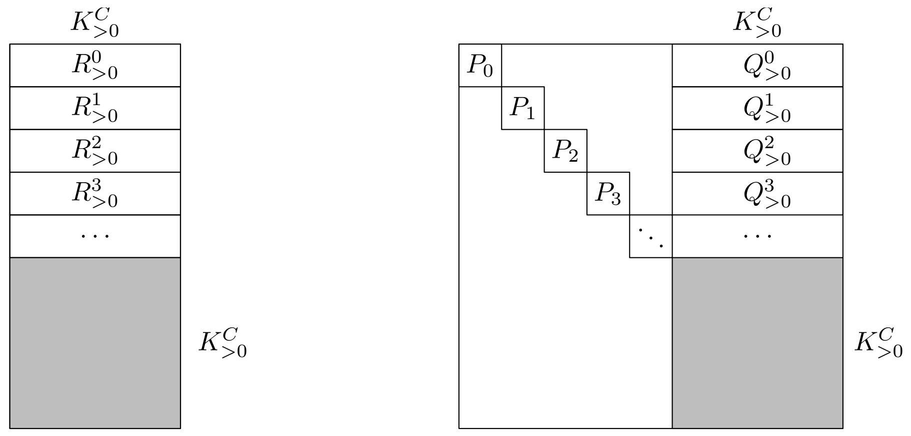

Figure 1: Structure of matrices $R_{>0}$ (left) and T (right). The columns and rows of T need to be put into the filtration order.

2.1 Version 1

1. Each processor $i \in \{0 \dots p-1\}$:
    - Reduce $D_i \to R_i$.
    - Row-reduce $R_i = S_i P_i$, where P_i is (almost) a permutation matrix, and S_i is upper-triangular. Compute S_i^{-1} while at it. See Algorithm 1.
2. Processor p:
    - Reduce $D_{>0} \to R_{>0}$, see Figure 1 on the left.
    - Send $R^i_{>0}$ to processor i. $R^i_{>0}$ is the subset of $R_{>0}$ where the rows are indexed by K_i^C.
3. Processor i:
    - Compute $Q^i_{>0} = S_i^{-1} \cdot R^i_{>0}$.
    - Send $Q^i_{>0}$ and P^i back to processor p.
4. Processor p:
    - Concatenate matrices $Q^i_{>0}$, together with the remainder of $R_{>0}$ (the part where the rows are indexed by $K^C_{>0}$), together with matrices P_i. Put all the columns and rows in the correct (filtration) order. Call the resulting matrix T, see Figure 1 on the right.
    - Reduce T using the cascade. Its lowest ones are the answer.

2.2 Version 2

We assume that each processor knows not only its local region of the domain, but also which cover sets intersect it and where. In matrix terms, it knows both matrix D_i and $D^i_{>0}$.

1. Each processor $i \in \{0 \dots p-1\}$:
    - Reduce $D_i \to R_i$.
    - Row-reduce $R_i = S_i P_i$. See Algorithm 1.
    - Compute $\bar{Q}^i_{>0} = S_i^{-1} \cdot D^i_{>0}$.

      Remark 2. *Naturally, one wouldn't explicitly compute S_i or S_i^{-1}, but rather apply the same row operations to $D^i_{>0}$ as are being performed on R_i to obtain P_i.*

- Send $\bar{Q}^i_{>0}$ and P_i to processor p.

2. Processor p:

   - Concatenate matrices $\bar{Q}^i_{>0}$, together with the remainder of $R_{>0}$ (the part where the rows are indexed by $K^C_{>0}$), together with matrices P_i. Put all the columns and rows in the correct (filtration) order. Call the resulting matrix T, see Figure 1 on the right.
   - Reduce T using the cascade. Its lowest ones are the answer.

2.3 Analysis

Theorem 3. *Lowest ones of the reduced T give the correct pairing.*

Theorem 4. *$|T| = (n - m) + nm$. T can be reduced using the cascade (Algorithm 2) in time $O(nm^2)$, while keeping its size $O(nm)$.*

Remark 5. *In both versions of the algorithm, there is an obvious optimization. When processor i sends to processor p matrix P_i, it suffices to send only those elements that fall in rows with non-zero entries in matrix $Q^i_{>0}$. It's an open question how effective this optimization is in practice.* ***Ryan:*** *This seems like it will be easy to implement, one way or the other.*

Remark 6. *As usual with the blowup complex, if dimension of K is d, we never need to construct higher than $(d + 1)$-skeleton of the blowup complex, since it captures all the homology that can possibly exist.*

3 Matrix Algorithms

[Need to change 1s to the actual coefficients.]

Algorithm 1 Row reduce: $R = SP$, $P = S^{-1}R$.

```
$P = R, S = I, S^{-1} = I$
for all rows $i$ of $P$, from the bottom up do
    if no lowest one in row $i$ then
        continue
    $j$ = the column of the lowest one in row $i$
    for all non-zero entries $P[i', j]$ in column $P[\cdot, j]$ do
        subtract row $P[i, \cdot]$ from row $P[i', \cdot]$ (really zero out the entry $P[i', j]$)
        add column $S[\cdot, i']$ to column $S[\cdot, i]$ (left to right)
        set $S^{-1}[i', i] = -1$
```

4 Cascade

Definition 7. *A matrix T is said to be an* almost-permutation matrix *if all but k of its columns have at most one non-zero entry and such entries do not collide (i.e., they appear in unique rows). We call the columns with at most one non-zero entry* ultra-sparse.

Figure 2 illustrates an almost permutation matrix. Matrix T in Section 2 is an almost-permutation matrix with $k = m$. (Concatenation of matrices P_i gives the columns with unique, non-colliding non-zero entries.) Algorithm 2 efficiently reduces an almost-permutation matrix.

Algorithm 3 recasts Algorithm 2 in the row form. The last two lines of Algorithm 3 are equivalent to first swapping columns j and j' and then subtracting the new column j from the new column j'. (Up to coefficients, as the rest of this write-up.) The salient point is that after these last two lines, we never need column $T[\cdot, j]$ again, except for the location of its lowest non-zero entry, so we can zero it out.

Theorem 8. *Algorithm 2 reduces an $n \times n$ almost-permutation matrix with k ultra-sparse columns in time $O(nk^2)$ using $O(nk)$ space.*

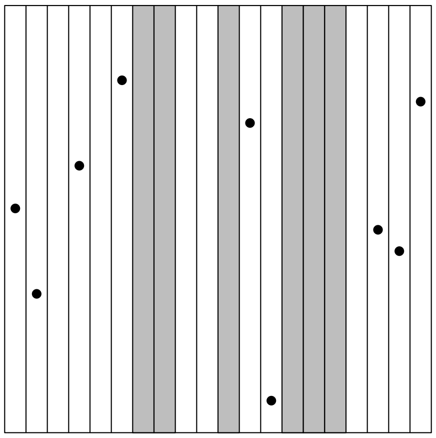

Figure 2: An almost-permutation matrix T, with $k = 9$ ultra-sparse columns. Grey columns may contain any number of non-zero elements. The white columns contain at most one non-zero element, and all such elements appear in unique rows.

Algorithm 2 Cascade algorithm for reduction of an almost-permutation matrix T.

```
for all columns $T[\cdot, j]$, from left to right do
    if $T[\cdot, j] = 0$, continue
    $i$ = row of the lowest non-zero entry in $T[\cdot, j]$
    if $T[\cdot, j]$ is an ultra-sparse non-zero column then
        if $i$ collides with a non-ultra-sparse column $T[\cdot, j']$ then
            swap columns $T[\cdot, j]$ and $T[\cdot, j']$
            subtract $T[\cdot, j']$ from $T[\cdot, j]$, i.e., zero out $T[i, j]$
        else
            Nothing to do: ultra-sparse columns cannot collide.
    if $T[i, j]$ has a collision with a column $j' < j$ then
        subtract $T[\cdot, j']$ from $T[\cdot, j]$. Equivalently, zero out $T[i, j]$
```

Algorithm 3 The row form of the cascade algorithm for reduction of an almost-permutation matrix T.

```
for all rows $T[i, \cdot]$, from bottom up do
    $J$ = the sequence of columns with the lowest non-zero entry in $T[i, \cdot]$.
    $j$ = the first entry in $J$
    if $T[\cdot, j]$ is ultra-sparse then
        for all $j' \in J, j' > j$ do
            subtract $T[\cdot, j]$ from $T[\cdot, j']$
    else
        for all $j' \in J, j' > j$ and $T[\cdot, j']$ not ultra-sparse do
            subtract $T[\cdot, j]$ from $T[\cdot, j']$
        $j'$ = the first ultra-sparse column in $J$
        for all $j'' \in J, j'' > j'$ and $T[\cdot, j'']$ ultra-sparse do
            subtract $T[\cdot, j']$ from $T[\cdot, j'']$
        subtract $T[\cdot, j]$ from $T[\cdot, j']$
        zero out all but the lowest entry of $T[\cdot, j]$
```

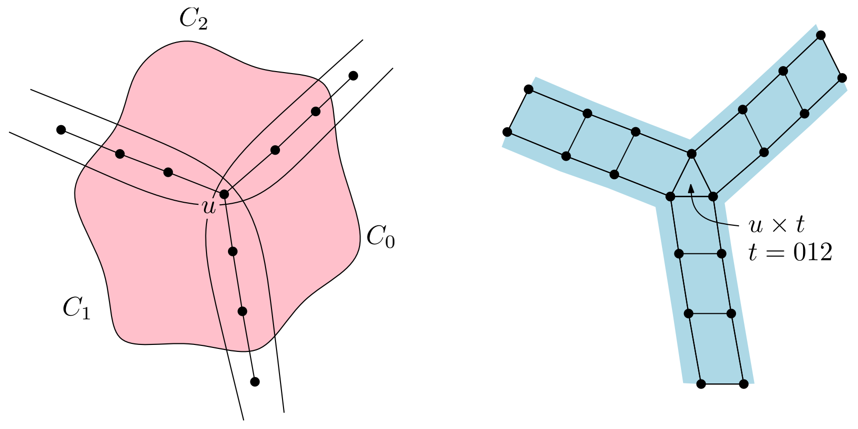

Figure 3: The 2-cycle cannot be formed without $u \times t$.

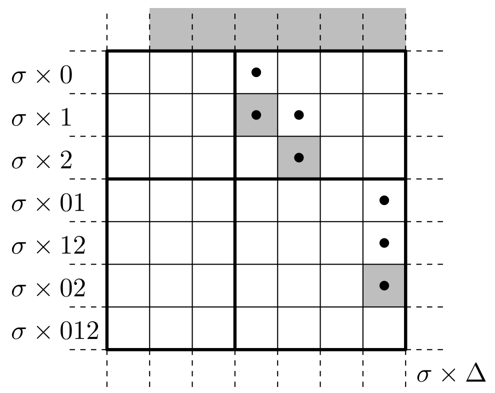

Figure 4: The structure of pairings in a blowup of a single cell σ that appears in the intersections of three cover sets C_0, C_1, C_2.

5 Pairwise Intersections Do Not Suffice

Figure 3 illustrates a triple intersection. We assume that extending the space outside the figure, the cells form a 2-cycle. This 2-cycle has to use the triangle $u \times t$ in the blowup complex.

Correct, but incomplete argument. The cells of the blowup complex $\{\sigma_i \times \Delta_j\}_{i,j}$ are ordered lexicographically, where the first order (of cells $\sigma_i \in K$) is given by the original filtration, and the second order by sorting all the faces of simplex Δ_j by dimension. Figure 4 illustrates the structure of the reduced boundary matrix for a block $\sigma \times \Delta$, where Δ is a triangle.

Since Δ is a simplex, it is collapsible. Therefore, its homology is trivial. So half of the cells in the block will kill the other half of the cells in the block. (Restricted to the block the reduction is just that of a simplex. Since a negative cell can only get paired with a positive cell, the other half of the cells are positive *in the entire blowup complex*, and therefore their columns are zero.) [It would be nice to understand directly why the column of $\sigma \times \alpha$, where α is a positive simplex in the filtration of Δ, is zero in the entire blowup complex. I.e., what exactly happens to them outside the block.]

6 Some commutative algebra for distributed memory filtrations

Given a basis for the finitely generated $k[t]$ module define the *critical grade* of a basis element σ, denoted critical-gr(σ) to be it's grade.

Suppose M is the chain $k[t]$-module of our base space and N is the $k[t]$-module for the nerve. Initially, the filtration on the nerve is skeletal. The blowup complex B is again a $k[t]$-module and it's basis is prescribed by an elementary tensor $\sigma \otimes \tau$ for σ in the basis of M and τ in the basis for N. The filtration on the blowup complex prescribes that critical-gr$(\sigma \otimes \tau) =$ critical-gr(σ). The prescribed filtration on the blowup complex also refines the filtration on the nerve. Namely,

$$\text{critical-gr}(\tau) = \min_{\substack{\sigma \in M \\ N(\sigma)=\tau}} \text{critical-gr}(\sigma)$$

However, actually using this refined grading makes the problem harder, not easier. In other words, we need not respect this refined grade.

Since we suppose that each processor knows the grade on each cell σ it has in the input, we know the grade of each cell of the blowup complex at construction. By storing this grade with each cell we may use this to aid in the distributed filtration. Comparing two cells may be achieved by first comparing grades, then comparing product cells of the same grade in any dimension preserving manner. For example, assuming that the input filtration is a total order, then comparisons within the same grade amount to ordering based on the nerve. If all the elements happen to live in the same grade, we get precisely the filtration given in Lewis & Zomorodian.