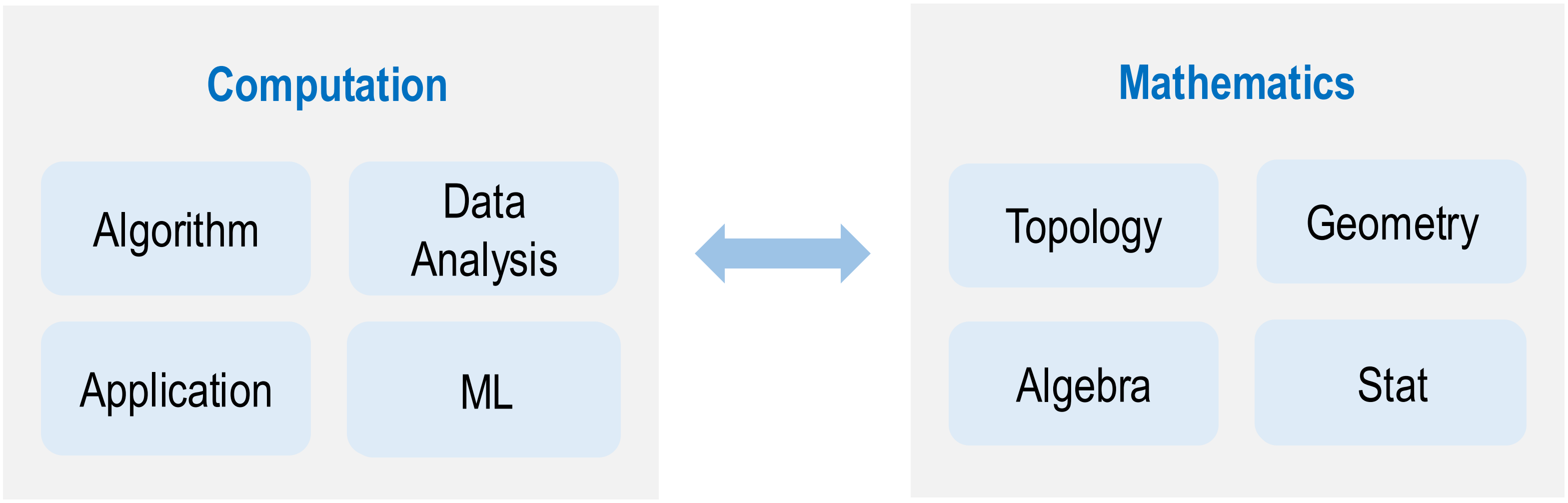
Can zigzag persistence be computed as efficiently as the standard version?

Geometry and Topology Seminar, Oregon State University

Tao Hou, CS Department
University of Oregon

Joint work with **Tamal K. Dey, Dmitriy Morozov, Salman Parsa**

Topological data analysis (TDA)



Computation

Algorithm

Data Analysis

Application

ML

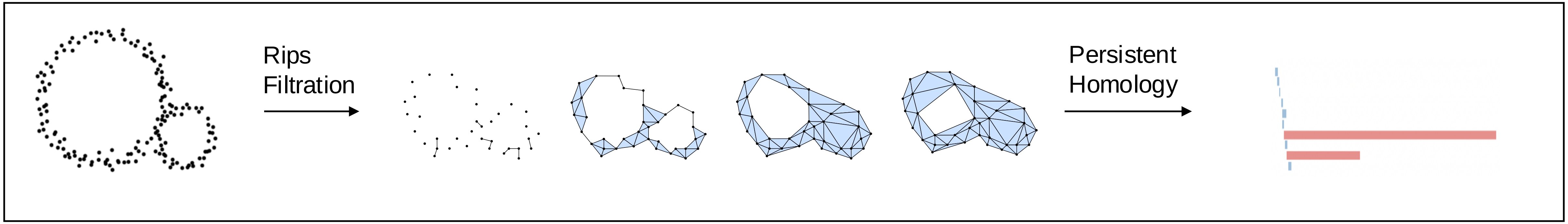
Mathematics

Topology

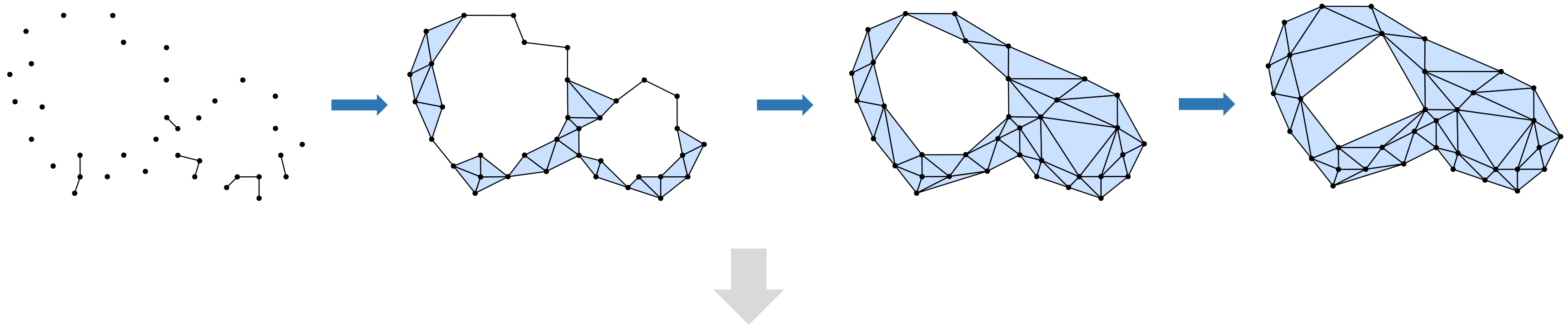
Geometry

Algebra

Stat

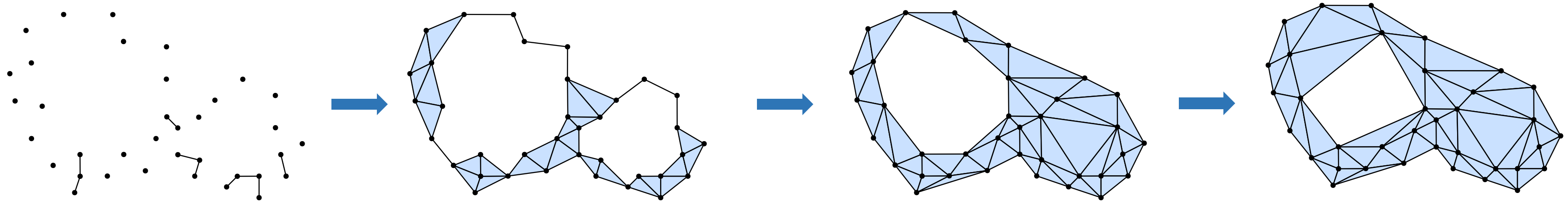


Persistent homology



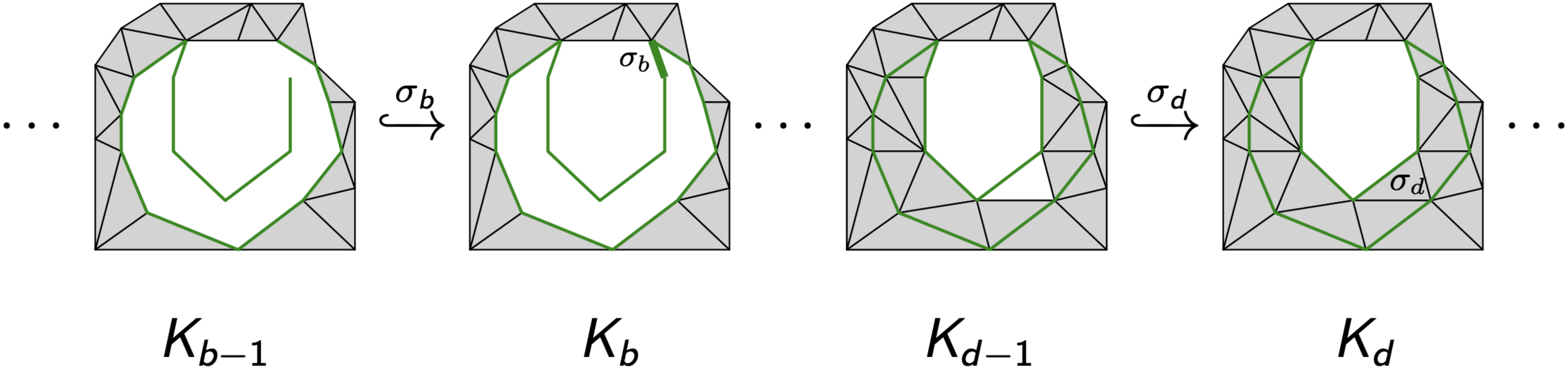
- As we add each simplex in the sequence, the homology of the complex changes, with:
  - Birth: betti number increased by 1
  - Death: betti number decreased by 1

Persistent homology



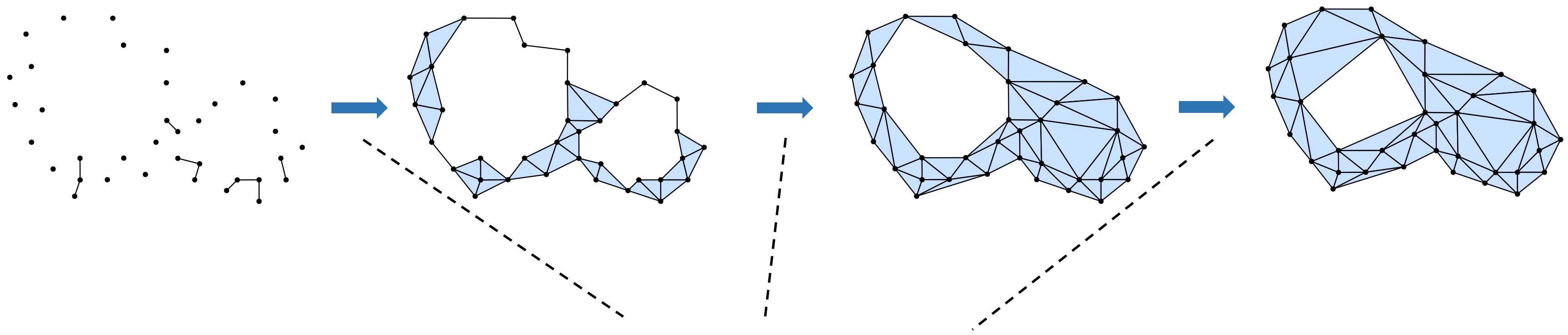
- As we add each simplex in the sequence, the homology of the complex changes, with:
  - Birth: betti number increased by 1
  - Death: betti number decreased by 1
- The birth and death points can be canonically paired, resulting in persistence barcode:

Persistent homology: example



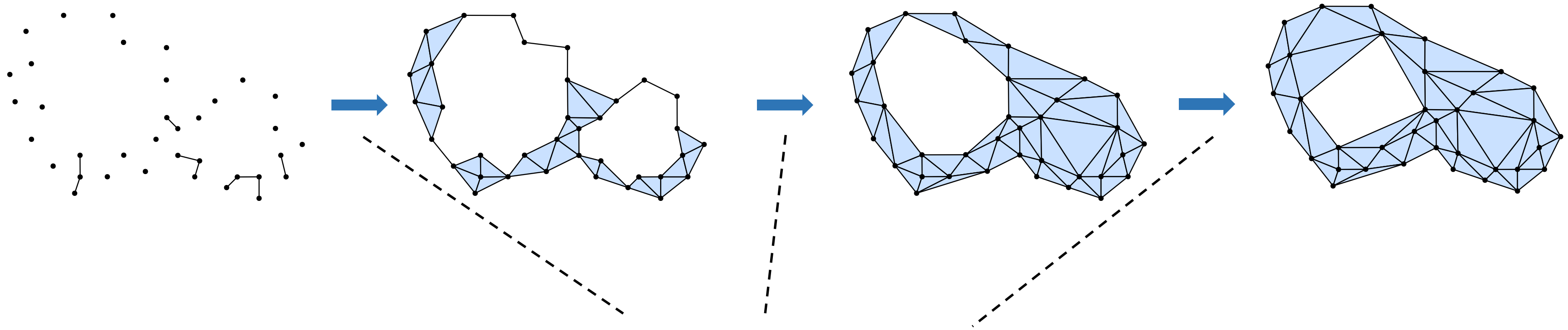
An interval: $[b, d) = [b, d-1]$

Persistent homology: Simplex-wise filtration



Expand each arrow into a sequence of additions of a single simplex

Persistent homology: Simplex-wise filtration

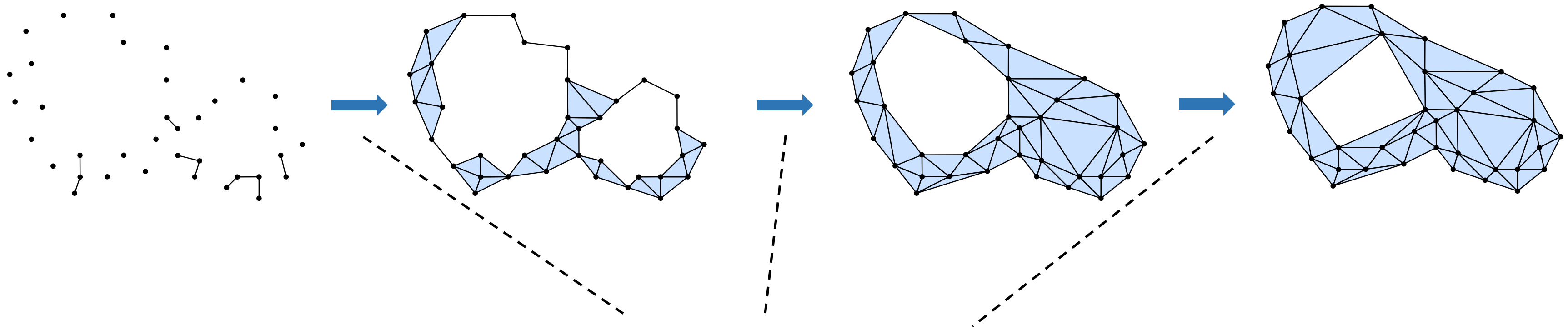


Expand each arrow into a sequence of additions of a single simplex

Simplex-wise filtration: a sequence of additions of a single simplex

$$\mathcal{F} : \varnothing = K_0 \xhookrightarrow{\sigma_0} K_1 \xhookrightarrow{\sigma_1} \cdots \xhookrightarrow{\sigma_{m-1}} K_{m-1} \xhookrightarrow{\sigma_m} K_m$$

Persistent homology: Simplex-wise filtration



Expand each arrow into a sequence of additions of a single simplex

Simplex-wise filtration: a sequence of additions of a single simplex

$$\mathcal{F} : \varnothing = K_0 \xhookrightarrow{\sigma_0} K_1 \xhookrightarrow{\sigma_1} \cdots \xhookrightarrow{\sigma_{m-1}} K_{m-1} \xhookrightarrow{\sigma_m} K_m$$

Persistent homology: Pipeline

Standard filtration:

$$\mathcal{F} : K_0 \xhookrightarrow{\sigma_0} K_1 \xhookrightarrow{\sigma_1} \cdots \xhookrightarrow{\sigma_{m-2}} K_{m-1} \xhookrightarrow{\sigma_{m-1}} K_m$$

$$\Downarrow$$

Induced module:

$$\mathsf{H}_p(\mathcal{F}) : \mathsf{H}_p(K_0) \to \mathsf{H}_p(K_1) \to \cdots \to \mathsf{H}_p(K_{m-1}) \to \mathsf{H}_p(K_m)$$

$$\Downarrow$$

Interval decomposition: **[Gabriel 72]**

$$\mathsf{H}_p(\mathcal{F}) = \bigoplus_{\alpha \in \mathcal{A}} \mathcal{I}^{[b_\alpha, d_\alpha]}$$

$$\Downarrow$$

p-th persistence barcode:

$$\mathsf{Pers}_p(\mathcal{F}) = \{[b_\alpha, d_\alpha] \,|\, \alpha \in \mathcal{A}\}$$

Persistent homology: Pipeline

Standard filtration:

$$\mathcal{F} : K_0 \xhookrightarrow{\sigma_0} K_1 \xhookrightarrow{\sigma_1} \cdots \xhookrightarrow{\sigma_{m-2}} K_{m-1} \xhookrightarrow{\sigma_{m-1}} K_m$$

$$\Downarrow$$

Induced module:

$$\mathsf{H}_p(\mathcal{F}) : \mathsf{H}_p(K_0) \to \mathsf{H}_p(K_1) \to \cdots \to \mathsf{H}_p(K_{m-1}) \to \mathsf{H}_p(K_m)$$

assumes $\mathbb{Z}_2$ as coefficients

$$\Downarrow$$

Interval decomposition: **[Gabriel 72]**

$$\mathsf{H}_p(\mathcal{F}) = \bigoplus_{\alpha \in \mathcal{A}} \mathcal{I}^{[b_\alpha, d_\alpha]}$$

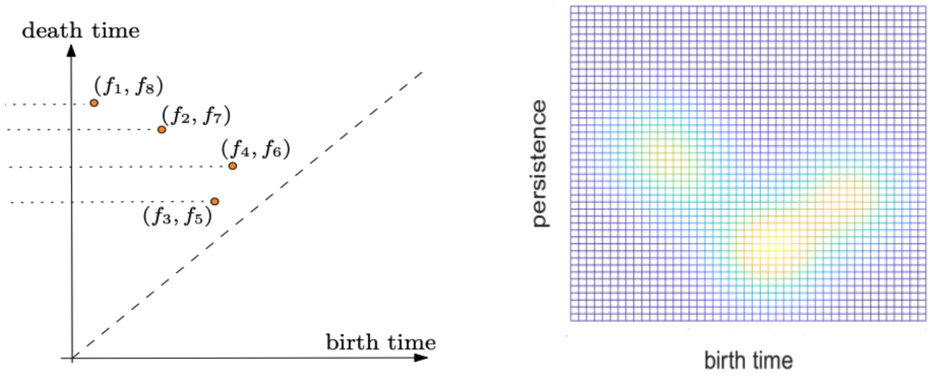
$$\Downarrow$$

p-th persistence barcode:

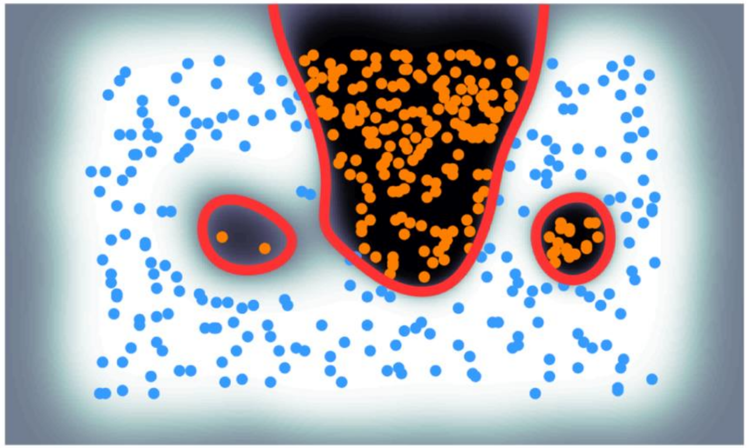
$$\mathsf{Pers}_p(\mathcal{F}) = \{[b_\alpha, d_\alpha] \mid \alpha \in \mathcal{A}\}$$

starts and ends with indices in the filtration

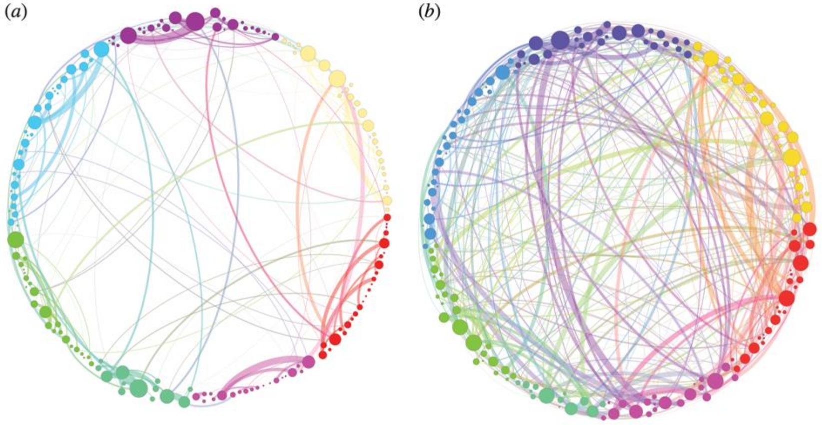
Persistent homology: Applications



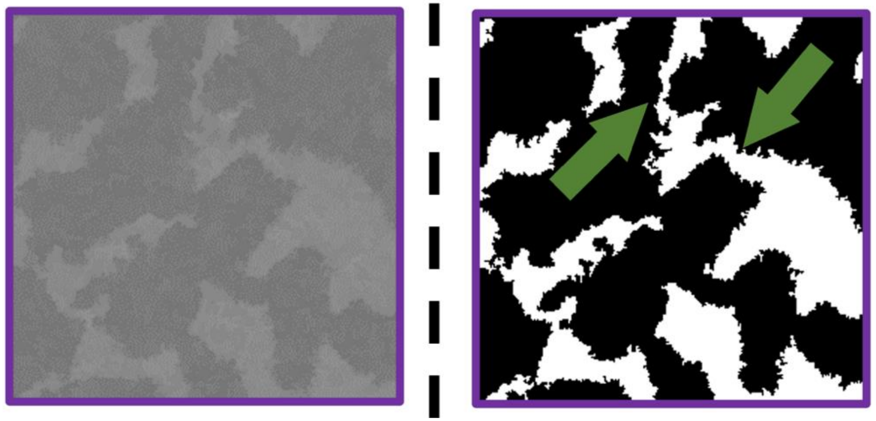
Features for ML [Zhao & Wang 19]



Topological regularizer for ML [Chen et al. 20]



Brain functional networks [Petri et al. 14]



Binarizing microstructures [Patel et al. 22]

Zigzag persistence

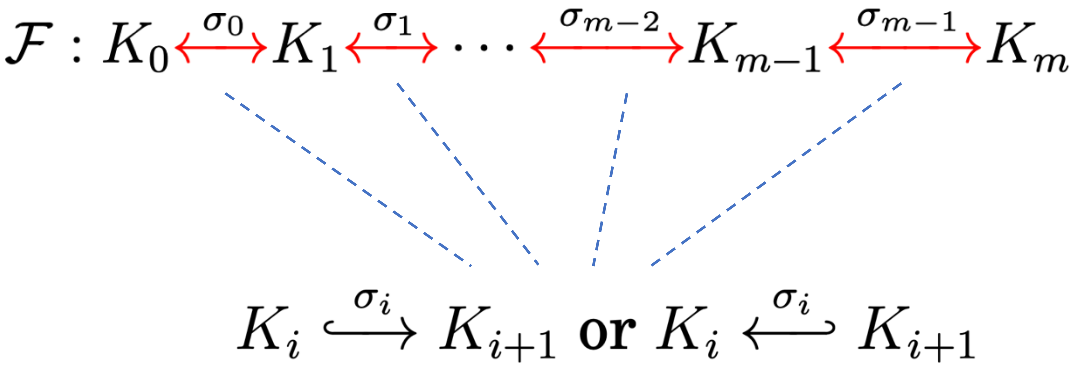
Zigzag filtration:

$$\mathcal{F} : K_0 \xleftrightarrow{\sigma_0} K_1 \xleftrightarrow{\sigma_1} \cdots \xleftrightarrow{\sigma_{m-2}} K_{m-1} \xleftrightarrow{\sigma_{m-1}} K_m$$

Gunnar Carlsson and Vin de Silva. Zigzag persistence.
Foundations of Computational Mathematics, 10(4):367–405, 2010.

Zigzag persistence

Zigzag filtration:



$$\mathcal{F} : K_0 \xleftrightarrow{\sigma_0} K_1 \xleftrightarrow{\sigma_1} \cdots \xleftrightarrow{\sigma_{m-2}} K_{m-1} \xleftrightarrow{\sigma_{m-1}} K_m$$

$$K_i \xhookrightarrow{\sigma_i} K_{i+1} \textbf{ or } K_i \xhookleftarrow{\sigma_i} K_{i+1}$$

Gunnar Carlsson and Vin de Silva. Zigzag persistence.
Foundations of Computational Mathematics, 10(4):367–405, 2010.

Zigzag persistence

Zigzag filtration:

$$\mathcal{F} : K_0 \xleftrightarrow{\sigma_0} K_1 \xleftrightarrow{\sigma_1} \cdots \xleftrightarrow{\sigma_{m-2}} K_{m-1} \xleftrightarrow{\sigma_{m-1}} K_m$$

$$\Downarrow$$

Induced module:

$$\mathsf{H}_p(\mathcal{F}) : \mathsf{H}_p(K_0) \leftrightarrow \mathsf{H}_p(K_1) \leftrightarrow \cdots \leftrightarrow \mathsf{H}_p(K_{m-1}) \leftrightarrow \mathsf{H}_p(K_m)$$

$$\Downarrow$$

Interval decomposition: **[Gabriel 72]**

$$\mathsf{H}_p(\mathcal{F}) = \bigoplus_{\alpha \in \mathcal{A}} \mathcal{I}^{[b_\alpha, d_\alpha]}$$

$$\Downarrow$$

p-th persistence barcode:

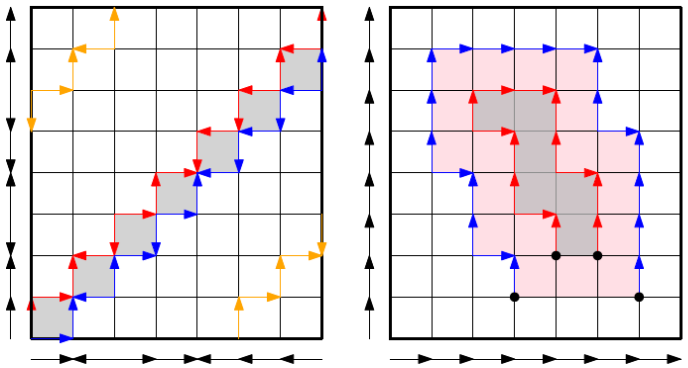
$$\mathsf{Pers}_p(\mathcal{F}) = \{[b_\alpha, d_\alpha] \mid \alpha \in \mathcal{A}\}$$

Peter Gabriel. Unzerlegbare Darstellungen I. Manuscripta Mathematica, 6(1):71–103, 1972.

Applications of Zigzag Persistence

- In time varying settings: functions, point cloud, vector field
  - G. Carlsson, V. de Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. SoCG 2009.
  - W. Kim and F. Mémoli. Spatiotemporal persistent homology for dynamic metric spaces. DCG 2020.
  - T. Dey, M. Lipinsky, M. Mrozek, R. Slechta. Tracking dynamical features via continuation and persistence. SoCG 2022.

- In multiparameter persistence



Non-Zigzag vs. Zigzag persistence

①

Bars in non-zigzag: 1 type

- closed-open

Bars in zigzag: 4 types

- closed-open
- closed-closed
- open-closed
- open-open

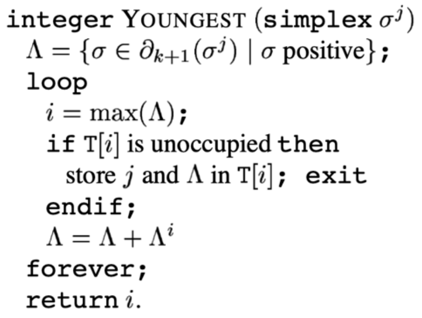
②

Simplices(σ) in zigzag: insertion($\downarrow\sigma$), deletion($\uparrow\sigma$), repeated($\downarrow\sigma$)

$$\mathcal{F} : \varnothing = K_0 \leftrightarrow \cdots \xhookrightarrow{\downarrow\sigma} \cdots \xhookleftarrow{\uparrow\sigma} \cdots \xhookrightarrow{\downarrow\sigma} \cdots \leftrightarrow K_m = \varnothing$$

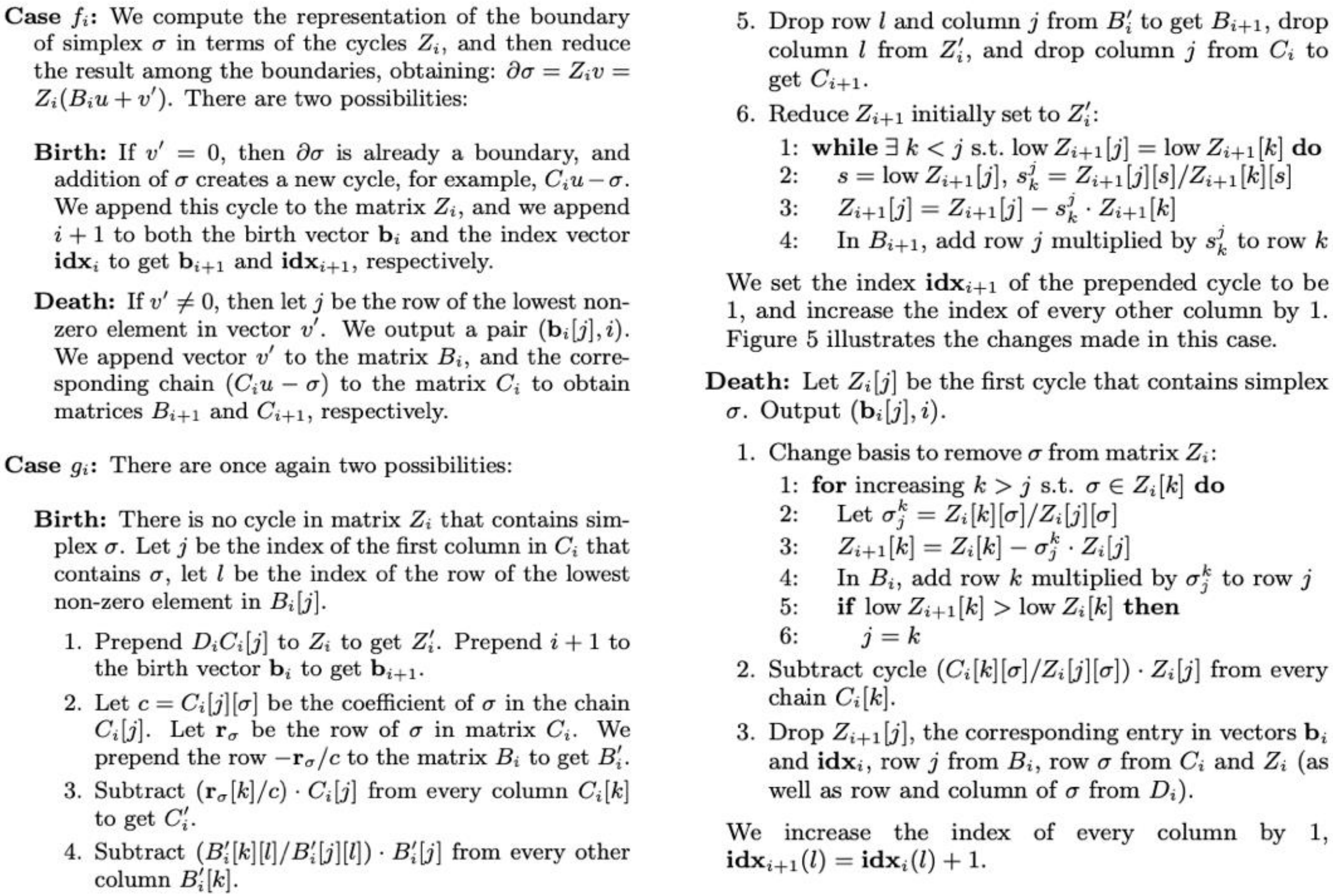
Non-Zigzag vs. Zigzag: Computing

Non-zigzag [ELZ2000]



```
integer YOUNGEST (simplex $\sigma^j$)
  $\Lambda = \{\sigma \in \partial_{k+1}(\sigma^j) \mid \sigma \text{ positive}\}$;
  loop
    $i = \max(\Lambda)$;
    if T[$i$] is unoccupied then
      store $j$ and $\Lambda$ in T[$i$]; exit
    endif;
    $\Lambda = \Lambda + \Lambda^i$
  forever;
  return $i$.
```

Zigzag [CdSM2009]



Case f_i: We compute the representation of the boundary of simplex σ in terms of the cycles Z_i, and then reduce the result among the boundaries, obtaining: $\partial\sigma = Z_i v = Z_i(B_i u + v')$. There are two possibilities:

Birth: If $v' = 0$, then $\partial\sigma$ is already a boundary, and addition of σ creates a new cycle, for example, $C_i u - \sigma$. We append this cycle to the matrix Z_i, and we append $i+1$ to both the birth vector $\mathbf{b}_i$ and the index vector $\mathbf{idx}_i$ to get $\mathbf{b}_{i+1}$ and $\mathbf{idx}_{i+1}$, respectively.

Death: If $v' \neq 0$, then let j be the row of the lowest non-zero element in vector v'. We output a pair $(\mathbf{b}_i[j], i)$. We append vector v' to the matrix B_i, and the corresponding chain $(C_i u - \sigma)$ to the matrix C_i to obtain matrices B_{i+1} and C_{i+1}, respectively.

Case g_i: There are once again two possibilities:

Birth: There is no cycle in matrix Z_i that contains simplex σ. Let j be the index of the first column in C_i that contains σ, let l be the index of the row of the lowest non-zero element in $B_i[j]$.

1. Prepend $D_i C_i[j]$ to Z_i to get Z_i'. Prepend $i+1$ to the birth vector $\mathbf{b}_i$ to get $\mathbf{b}_{i+1}$.
2. Let $c = C_i[j][\sigma]$ be the coefficient of σ in the chain $C_i[j]$. Let $\mathbf{r}_\sigma$ be the row of σ in matrix C_i. We prepend the row $-\mathbf{r}_\sigma/c$ to the matrix B_i to get B_i'.
3. Subtract $(\mathbf{r}_\sigma[k]/c) \cdot C_i[j]$ from every column $C_i[k]$ to get C_i'.
4. Subtract $(B_i'[k][l]/B_i'[j][l]) \cdot B_i'[j]$ from every other column $B_i'[k]$.
5. Drop row l and column j from B_i' to get B_{i+1}, drop column l from Z_i', and drop column j from C_i to get C_{i+1}.
6. Reduce Z_{i+1} initially set to Z_i':

```
while $\exists\, k < j$ s.t. low $Z_{i+1}[j] =$ low $Z_{i+1}[k]$ do
    $s =$ low $Z_{i+1}[j]$, $s_k^j = Z_{i+1}[j][s]/Z_{i+1}[k][s]$
    $Z_{i+1}[j] = Z_{i+1}[j] - s_k^j \cdot Z_{i+1}[k]$
    In $B_{i+1}$, add row $j$ multiplied by $s_k^j$ to row $k$
```

We set the index $\mathbf{idx}_{i+1}$ of the prepended cycle to be 1, and increase the index of every other column by 1. Figure 5 illustrates the changes made in this case.

Death: Let $Z_i[j]$ be the first cycle that contains simplex σ. Output $(\mathbf{b}_i[j], i)$.

1. Change basis to remove σ from matrix Z_i:

```
for increasing $k > j$ s.t. $\sigma \in Z_i[k]$ do
    Let $\sigma_j^k = Z_i[k][\sigma]/Z_i[j][\sigma]$
    $Z_{i+1}[k] = Z_i[k] - \sigma_j^k \cdot Z_i[j]$
    In $B_i$, add row $k$ multiplied by $\sigma_j^k$ to row $j$
    if low $Z_{i+1}[k] >$ low $Z_i[k]$ then
        $j = k$
```

2. Subtract cycle $(C_i[k][\sigma]/Z_i[j][\sigma]) \cdot Z_i[j]$ from every chain $C_i[k]$.
3. Drop $Z_{i+1}[j]$, the corresponding entry in vectors $\mathbf{b}_i$ and $\mathbf{idx}_i$, row j from B_i, row σ from C_i and Z_i (as well as row and column of σ from D_i).

We increase the index of every column by 1, $\mathbf{idx}_{i+1}(l) = \mathbf{idx}_i(l) + 1$.

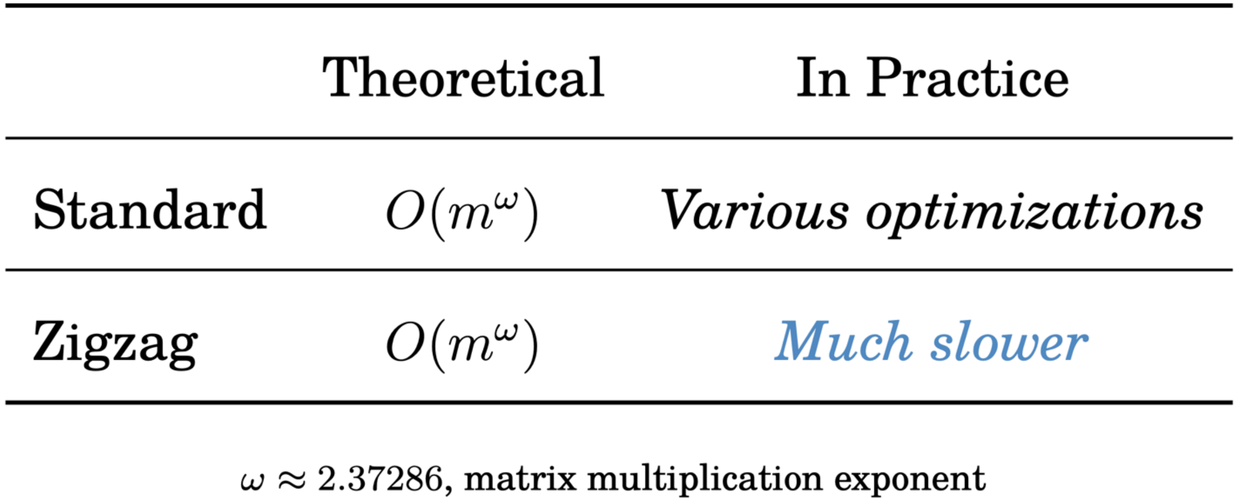
Outline

1. An algorithm for computing zigzag persistence (FastZigzag)
    - Converts to a computation of non-zigzag persistence
    - Bridges gap of efficiency for computing the two versions
2. $O(m \log m)$ algorithm for computing graph zigzag persistence
3. Algorithms for updating zigzag persistence over local changes
    - Focus on contractions and expansions
    - Match the $O(m^2)$ complexity of the non-zigzag version
4. Algorithms for updating graph persistence (over switches)
    - Non-zigzag: $O(\log m)$
    - Zigzag: $O(\sqrt{m} \log m)$
5. $O(m^2 n)$ algorithm for computing zigzag representatives

Fast computation of zigzag persistence

[DeyH]: Fast Computation of Zigzag Persistence. ESA22

Complexities of persistence computing



	Theoretical	In Practice
Standard	$O(m^\omega)$	*Various optimizations*
Zigzag	$O(m^\omega)$	*Much slower*

$\omega \approx 2.37286$, matrix multiplication exponent

Edelsbrunner, Letscher, Zomorodian. Topological persistence and simplification. FoCS 2000.
Carlsson, de Silva, Morozov. Zigzag persistent homology and real-valued functions. SoCG 2009.
Milosavljevi´c, Morozov, Skraba. Zigzag persistent homology in matrix multiplication time. SoCG 2011.
Cl´ement Maria and Steve Y. Oudot. Zigzag persistence via reflections and transpositions. SODA 2015.

Overview of FastZigzag

- Input zigzag filtration

$$\mathcal{F} : \varnothing = K_0 \xleftrightarrow{\sigma_0} K_1 \xleftrightarrow{\sigma_1} \cdots \xleftrightarrow{\sigma_{m-1}} K_m = \varnothing$$

- Convert to a non-zigzag filtration of same length (linear time)

$$\mathcal{F}' : K'_0 \xhookrightarrow{\sigma'_0} K'_1 \xhookrightarrow{\sigma'_1} \cdots \xhookrightarrow{\sigma'_{m-1}} K'_m$$

- Compute barcode for non-zigzag filtration $\mathcal{F}'$
  - Fast software [Gudhi, Phat, Dionysus etc.]

- Convert barcode of $\mathcal{F}'$ to that of $\mathcal{F}$
  - $O(1)$ conversion per bar

Overall conversion has very little cost

Conversion of Filtrations in FastZigzag

Input zigzag → Non-repetitive zigzag → Up-down → Extended persistence → Non-zigzag

All filtrations have the same length (the same number of addition/deletions)

Conversions 1,2,3,4:

- Done by a simple linear scan of the input filtration

Conversion of Filtrations in FastZigzag

Input zigzag ➡ Non-repetitive zigzag ⇨ Up-down ⇨ Extended persistence ⇨ Non-zigzag

Non-repetitive filtration: A simplex is added at most one time

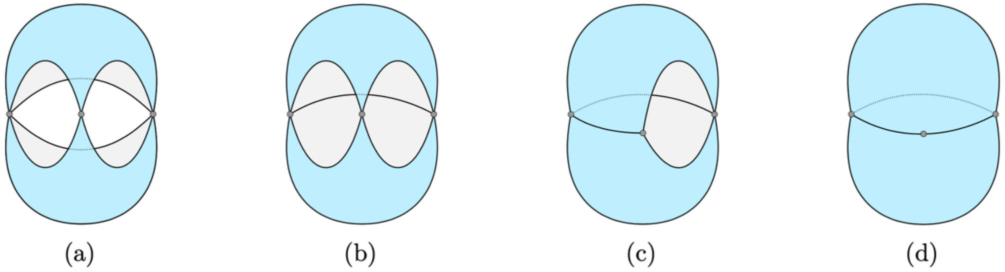
- Idea: Treat new occurrence of simplex σ as a new copy (barcodes stay the same)

$$\mathcal{F} : \varnothing = K_0 \leftrightarrow \cdots \xhookrightarrow{\sigma} \cdots \xhookleftarrow{\sigma} \cdots \xhookrightarrow{\sigma} \cdots \xhookleftarrow{\sigma} \cdots \leftrightarrow K_m = \varnothing$$

⬇

$$\hat{\mathcal{F}} : \varnothing = \hat{K}_0 \leftrightarrow \cdots \xhookrightarrow{\hat{\sigma}_1} \cdots \xhookleftarrow{\hat{\sigma}_1} \cdots \xhookrightarrow{\hat{\sigma}_2} \cdots \xhookleftarrow{\hat{\sigma}_2} \cdots \leftrightarrow \hat{K}_m = \varnothing$$

- Simplices with the same vertex set shall occur in same complex in later filtration: use Δ-complex [Hatcher02]



Two triangles sharing 0,1,2,3 edges

Conversion of Filtrations in FastZigzag

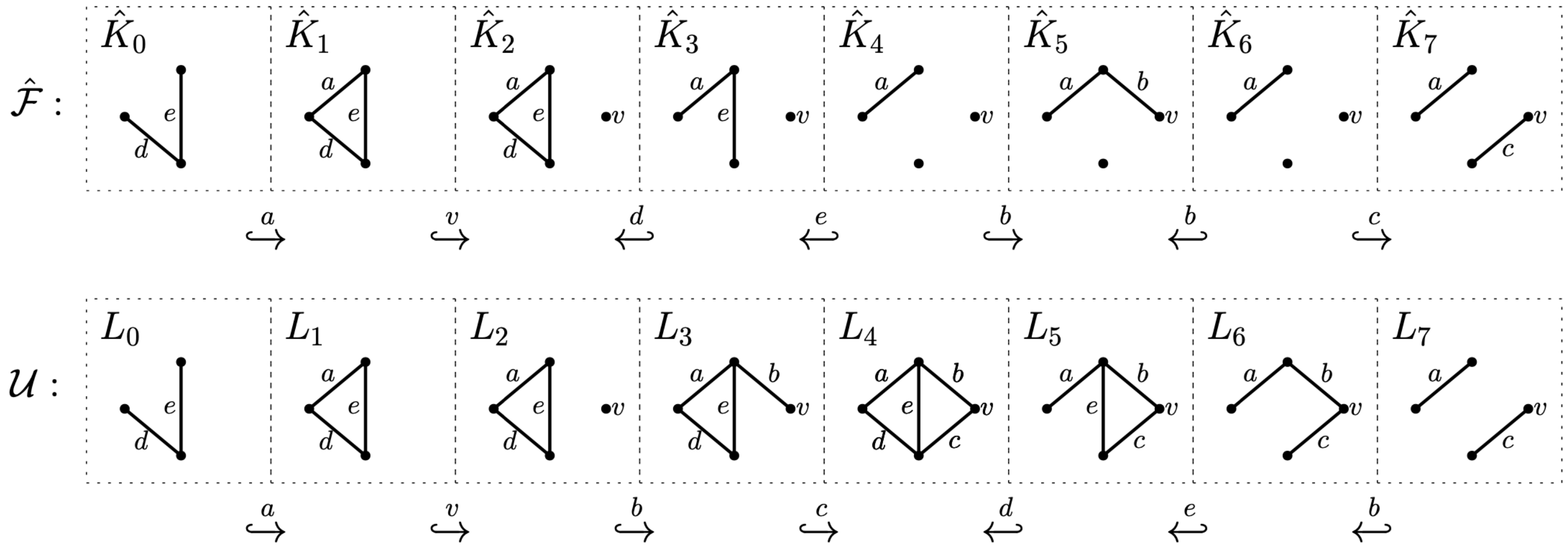
Input zigzag ⇒ **Non-repetitive zigzag** ⇒ **Up-down** ⇒ Extended persistence ⇒ Non-zigzag

$$\hat{\mathcal{F}} : \varnothing = \hat{K}_0 \xleftrightarrow{\hat{\sigma}_0} \hat{K}_1 \xleftrightarrow{\hat{\sigma}_1} \cdots \xleftrightarrow{\hat{\sigma}_{m-1}} \hat{K}_m = \varnothing$$

$$\Downarrow$$

$$\mathcal{U} : \varnothing = L_0 \xhookrightarrow{\tau_0} \cdots \xhookrightarrow{\tau_{n-1}} L_n \xhookleftarrow{\tau_n} \cdots \xhookleftarrow{\tau_{2n-1}} L_{2n} = \varnothing \quad (m = 2n)$$

List the additions in $\hat{\mathcal{F}}$ first and then the deletions in $\hat{\mathcal{F}}$, following the orders in $\hat{\mathcal{F}}$



Conversion of Filtrations in FastZigzag

$$\mathcal{U} : \varnothing = L_0 \xhookrightarrow{\tau_0} \cdots \xhookrightarrow{\tau_{n-1}} L_n \xhookleftarrow{\tau_n} \cdots \xhookleftarrow{\tau_{2n-2}} L_{2n-1} \xhookleftarrow{\tau_{2n-1}} L_{2n} = \varnothing$$

$$\Downarrow$$

$$\mathcal{E} : \varnothing = L_0 \xhookrightarrow{\tau_0} \cdots \xhookrightarrow{\tau_{n-1}} L_n = (\hat{K}, L_{2n}) \xhookrightarrow{\tau_{2n-1}} (\hat{K}, L_{2n-1}) \xhookrightarrow{\tau_{2n-2}} \cdots \xhookrightarrow{\tau_n} (\hat{K}, L_n) = (\hat{K}, \hat{K})$$

Cohen-Steiner, Edelsbrunner, Harer. Extending persistence using Poincaré and Lefschetz duality. FoCM 2009

Conversion of Filtrations in FastZigzag

Input zigzag ⇨ Non-repetitive zigzag ➡ Up-down ➡ Extended persistence ⇨ Non-zigzag

Mayer-Vietoris Diamond [CdS10]

$$\begin{tikzcd}
\mathcal{F}_1 : & & & & K_j & & & \\
& K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{j-1} \arrow[ur, hook, "\sigma"] & & K_{j+1} \arrow[ul, hook', "\tau"'] \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m \\
\mathcal{F}_2 : & & & & K'_j \arrow[ul, hook', "\tau"] \arrow[ur, hook, "\sigma"'] & & &
\end{tikzcd}$$

- $\mathcal{F}_1$ to $\mathcal{F}_2$: Outward switch
- $\mathcal{F}_2$ to $\mathcal{F}_1$: Inward switch

Gunnar Carlsson and Vin de Silva. Zigzag persistence. FoCM 2010

Conversion of Filtrations in FastZigzag

Input zigzag ⇒ Non-repetitive zigzag → Up-down → Extended persistence ⇒ Non-zigzag

Mayer-Vietoris Diamond [CdS10]

$$\begin{tikzcd}
\mathcal{F}_1 : & & & & K_j & & & \\
& K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{j-1} \arrow[ur, hook, "\sigma"] & & K_{j+1} \arrow[ul, hook', "\tau"'] \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m \\
\mathcal{F}_2 : & & & & K_j' \arrow[ul, hook', "\tau"'] \arrow[ur, hook, "\sigma"] & & &
\end{tikzcd}$$

- $\mathcal{F}_1$ to $\mathcal{F}_2$: Outward switch
- $\mathcal{F}_2$ to $\mathcal{F}_1$: Inward switch

Major takeaway: there is a bijection between the barcodes of the filtrations s.t. corresponding intervals have same creator and destroyer simplices (cells)

Conversion of Filtrations in FastZigzag

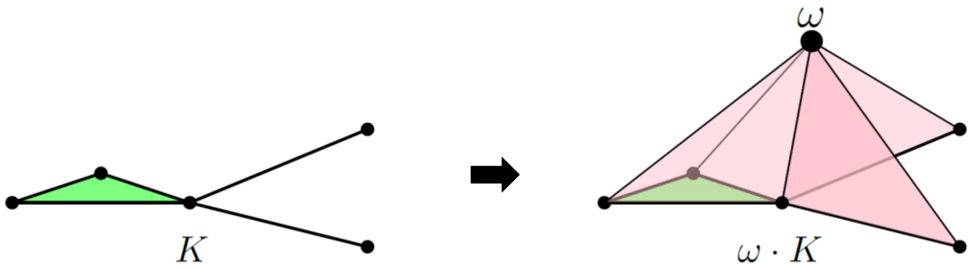
Input zigzag ⇒ Non-repetitive zigzag ⇒ Up-down ⇒ Extended persistence ➡ Non-zigzag

$$\mathcal{E} : \varnothing = L_0 \hookrightarrow \cdots \hookrightarrow L_n = (\hat{K}, L_{2n}) \hookrightarrow (\hat{K}, L_{2n-1}) \hookrightarrow \cdots \hookrightarrow (\hat{K}, L_n) = (\hat{K}, \hat{K})$$

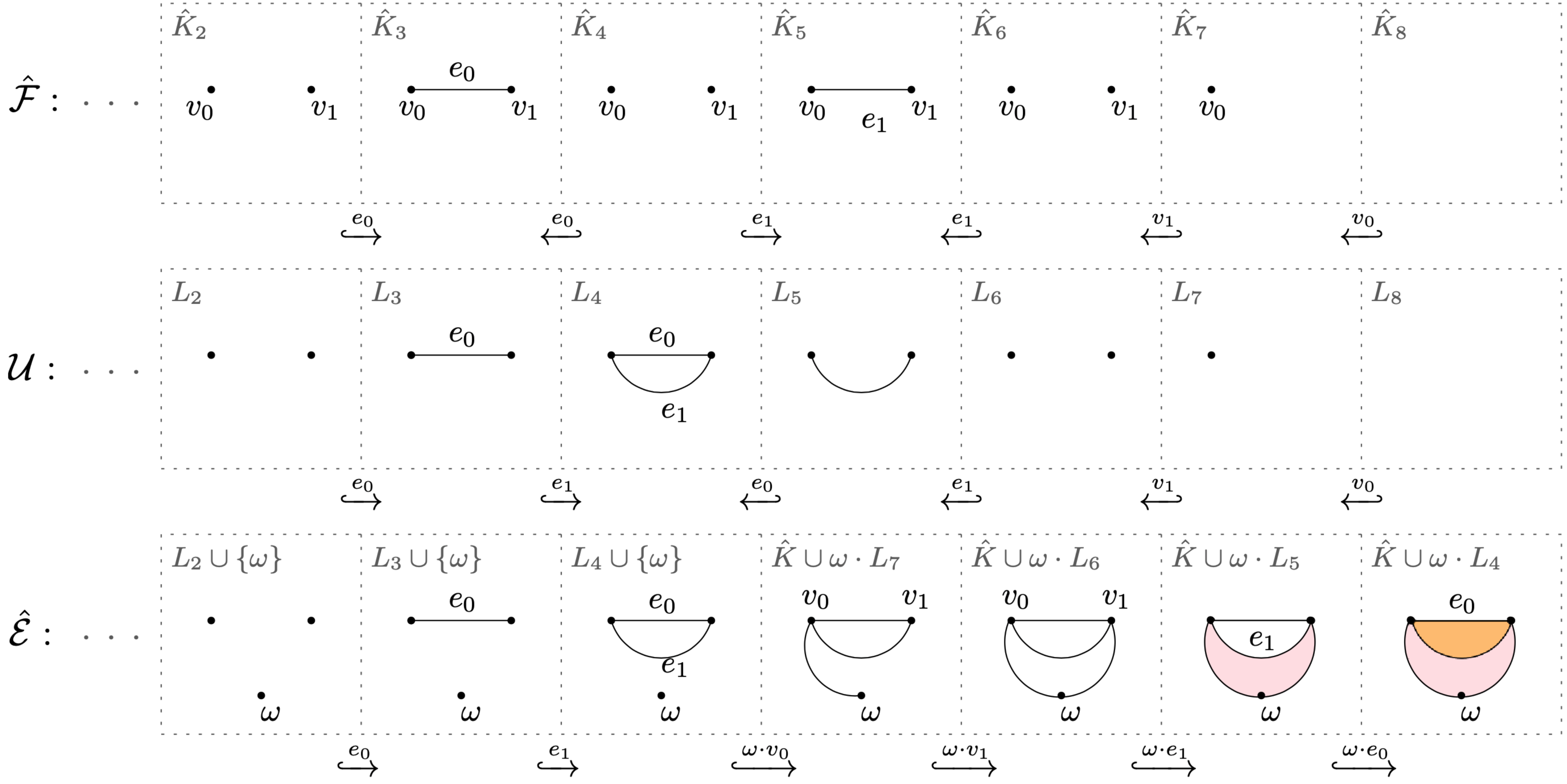
$$\Downarrow$$

$$\hat{\mathcal{E}} : L_0 \cup \{\omega\} \hookrightarrow \cdots \hookrightarrow L_n \cup \{\omega\} = \hat{K} \cup \omega \cdot L_{2n} \hookrightarrow \hat{K} \cup \omega \cdot L_{2n-1} \hookrightarrow \cdots \hookrightarrow \hat{K} \cup \omega \cdot L_n$$

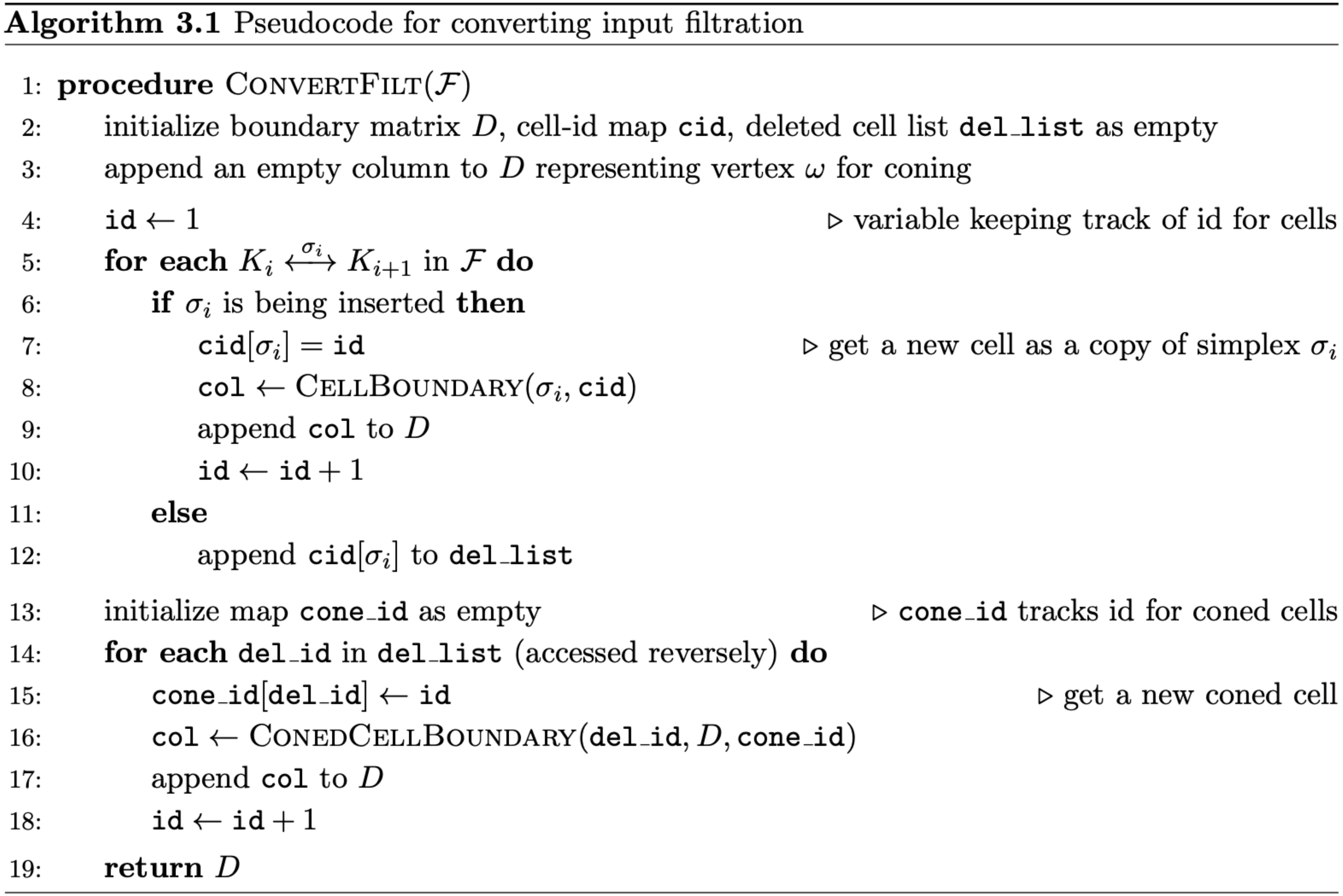
Use ‘Coning’ [CEH09]: No change in barcode



Overall Conversions



Pseudocodes for the conversion



Algorithm 3.1 Pseudocode for converting input filtration

```
procedure ConvertFilt($\mathcal{F}$)
    initialize boundary matrix $D$, cell-id map cid, deleted cell list del_list as empty
    append an empty column to $D$ representing vertex $\omega$ for coning
    id ← 1                                          ▷ variable keeping track of id for cells
    for each $K_i \xleftrightarrow{\sigma_i} K_{i+1}$ in $\mathcal{F}$ do
        if $\sigma_i$ is being inserted then
            cid[$\sigma_i$] = id                       ▷ get a new cell as a copy of simplex $\sigma_i$
            col ← CellBoundary($\sigma_i$, cid)
            append col to $D$
            id ← id + 1
        else
            append cid[$\sigma_i$] to del_list
    initialize map cone_id as empty                 ▷ cone_id tracks id for coned cells
    for each del_id in del_list (accessed reversely) do
        cone_id[del_id] ← id                        ▷ get a new coned cell
        col ← ConedCellBoundary(del_id, $D$, cone_id)
        append col to $D$
        id ← id + 1
    return $D$
```

Running time comparison

1-2: Non-repetitive random shuffles from height functions on triangular meshes

3-8: Clique complexes from random edge additions/deletions

9-11: Oscillating Rips zigzag from point clouds of 2000 – 4000 sampled from triangular meshes

No.	Length	D	Rep	MaxK	T_{DIO2}	T_{GUDHI}	T_{FZZ}	SU
1	5,260,700	5	1.0	883,350	2h02m46.0s	–	8.9s	873
2	5,254,620	4	1.0	1,570,326	19m36.6s	–	11.0s	107
3	5,539,494	5	1.3	1,671,047	3h05m00.0s	45m47.0s	3m20.8s	13.7
4	5,660,248	4	2.0	1,385,979	2h59m57.0s	29m46.7s	4m59.5s	6.0
5	5,327,422	4	3.5	760,098	43m54.8s	10m35.2s	3m32.1s	3.0
6	5,309,918	3	5.1	523,685	5h46m03.0s	1h32m37.0s	19m30.2s	4.7
7	5,357,346	3	7.3	368,830	3h37m54.0s	57m28.4s	30m25.2s	1.9
8	6,058,860	4	9.1	331,211	53m21.2s	7m19.0s	3m44.4s	2.0
9	5,135,720	3	21.9	11,859	23.8s	15.6s	8.6s	1.9
10	5,110,976	3	27.7	11,435	36.2s	39.9s	8.5s	4.3
11	5,811,310	4	44.2	7,782	38.5s	36.9s	23.9s	1.5

- All run on Intel(R), Core™, i5-9500 CPU@3.00GHz, 16GB memory, Linux OS
- Software **FZZ** using **Phat** software for non-zigzag (https://github.com/taohou01/fzz)

Running time comparison

1-2: Non-repetitive random shuffles from height functions on triangular meshes

3-8: Clique complexes from random edge additions/deletions

9-11: Oscillating Rips zigzag from point clouds of 2000 – 4000 sampled from triangular meshes

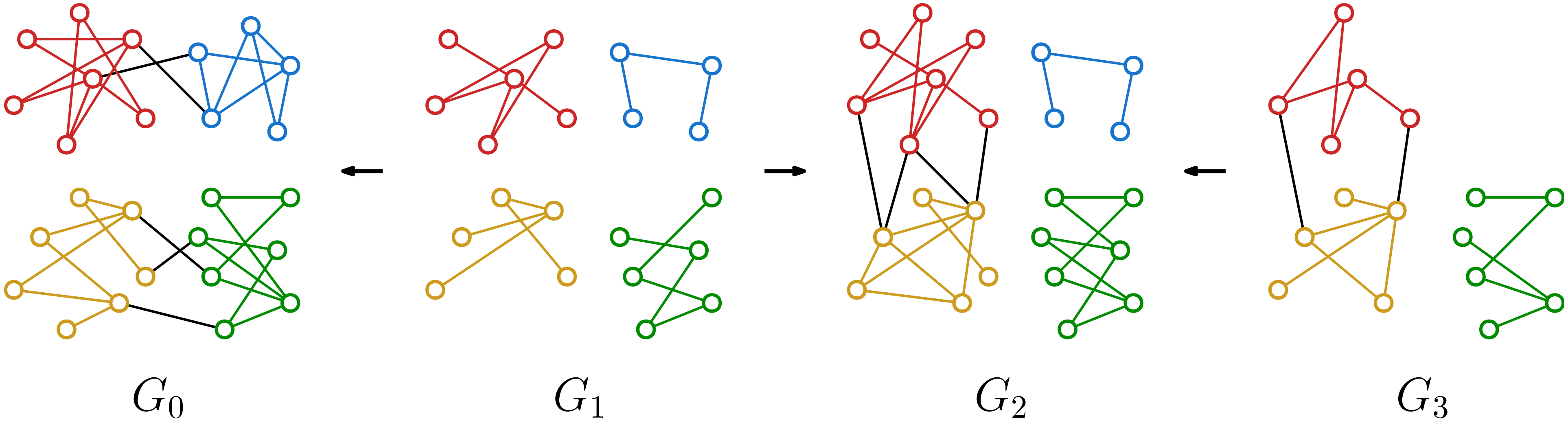
No.	Length	D	Rep	MaxK	T_{Dio2}	T_{Gudhi}	T_{FZZ}	SU
1	5,260,700	5	1.0	883,350	2h02m46.0s	–	8.9s	873
2	5,254,620	4	1.0	1,570,326	19m36.6s	–	11.0s	107
3	5,539,494	5	1.3	1,671,047	3h05m00.0s	45m47.0s	3m20.8s	13.7
4	5,660,248	4	2.0	1,385,979	2h59m57.0s	29m46.7s	4m59.5s	6.0
5	5,327,422	4	3.5	760,098	43m54.8s	10m35.2s	3m32.1s	3.0
6	5,309,918	3	5.1	523,685	5h46m03.0s	1h32m37.0s	19m30.2s	4.7
7	5,357,346	3	7.3	368,830	3h37m54.0s	57m28.4s	30m25.2s	1.9
8	6,058,860	4	9.1	331,211	53m21.2s	7m19.0s	3m44.4s	2.0
9	5,135,720	3	21.9	11,859	23.8s	15.6s	8.6s	1.9
10	5,110,976	3	27.7	11,435	36.2s	39.9s	8.5s	4.3
11	5,811,310	4	44.2	7,782	38.5s	36.9s	23.9s	1.5

- All run on Intel(R), Core™, i5-9500 CPU@3.00GHz, 16GB memory, Linux OS
- Software **FZZ** using **Phat** software for non-zigzag (https://github.com/taohou01/fzz)

$O(m \log m)$ computation of graph zigzag persistence

[Dey-H-Parsa]. Revisiting Graph Persistence for Updates and Efficiency. WADS 2023

An application of graph zigzag persistence: Dynamic networks



Petter Holme and Jari Saramaki. **Temporal networks**. Physics Reports, 519(3):97–125, 2012.

Complexities of persistence computing

	*	Graphs
Standard	$O(m^{\omega})$	$O(m\,\alpha(m))$
Zigzag	$O(m^{\omega})$	$O(m \log^4 n)$

m: length of filtration
$\omega \approx 2.37286$: matrix multiplication exponent
$\alpha(m)$: inverse Ackermann function

Input for graph zigzag:

$$\mathcal{F} : \varnothing = G_0 \xleftrightarrow{\sigma_0} G_1 \xleftrightarrow{\sigma_1} \cdots \xleftrightarrow{\sigma_{m-1}} G_m; G = \bigcup_{i=0}^{m} G_i$$

n: size of G

Nikola Milosavljevi´c, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. 2011.
Tamal K. Dey and Tao Hou. Computing Zigzag Persistence on Graphs in Near-Linear Time. 2021

Computation

Utilize the conversion in FastZigzag to convert the input zigzag into an up-down filtration $\mathcal{U}$, with the following barcode mapping:

Compute them separately

	$\mathcal{U}$		$\mathcal{F}$
①	$\mathsf{Pers}_0^{\mathrm{co}}(\mathcal{U})$	$\leftrightarrow$	$\mathsf{Pers}_0^{\mathrm{co}}(\mathcal{F})$
	$\mathsf{Pers}_0^{\mathrm{oc}}(\mathcal{U})$	$\leftrightarrow$	$\mathsf{Pers}_0^{\mathrm{oc}}(\mathcal{F})$
②	$\mathsf{Pers}_0^{\mathrm{cc}}(\mathcal{U})$	$\leftrightarrow$	$\mathsf{Pers}_0^{\mathrm{cc}}(\mathcal{F})$
③	$\mathsf{Pers}_1^{\mathrm{cc}}(\mathcal{U})$	$\leftrightarrow$	$\mathsf{Pers}_0^{\mathrm{oo}}(\mathcal{F}) \cup \mathsf{Pers}_1^{\mathrm{cc}}(\mathcal{F})$

Computation

1. Pers$_0^{\mathrm{CO}}(\mathcal{U})$, Pers$_0^{\mathrm{OC}}(\mathcal{U})$: run the persistence pairing for 0-dimensional standard persistence with Union-Find on the *ascending* and *descending* parts of $\mathcal{U}$ in $O(m\,\alpha(m))$ time

$$\mathcal{U} : \underbrace{\varnothing = \hat{G}_0 \xhookrightarrow{\hat{\sigma}_0} \hat{G}_1 \xhookrightarrow{\hat{\sigma}_1} \cdots \xhookrightarrow{\hat{\sigma}_{n-1}} \hat{G}_n}_{\text{ascending part}} \underbrace{\xhookleftarrow{\hat{\sigma}_n} \hat{G}_{n+1} \xhookleftarrow{\hat{\sigma}_{n+1}} \cdots \xhookleftarrow{\hat{\sigma}_{m-1}} \hat{G}_m = \varnothing}_{\text{descending part}}$$

Computation

2. $\mathrm{Pers}_0^{\mathrm{CC}}(\mathcal{U})$:

- Identify each connected component C of $\hat{G}_n$
- Pair:
  - first vertex in the ascending part coming from C
  - first vertex in the descending part coming from C
- Can be done in linear time

$$\mathcal{U} : \underbrace{\varnothing = \hat{G}_0 \xhookrightarrow{\hat{\sigma}_0} \hat{G}_1 \xhookrightarrow{\hat{\sigma}_1} \cdots \xhookrightarrow{\hat{\sigma}_{n-1}} \hat{G}_n}_{\text{ascending part}} \underbrace{\xhookleftarrow{\hat{\sigma}_n} \hat{G}_{n+1} \xhookleftarrow{\hat{\sigma}_{n+1}} \cdots \xhookleftarrow{\hat{\sigma}_{m-1}} \hat{G}_m = \varnothing}_{\text{descending part}}$$

Computation

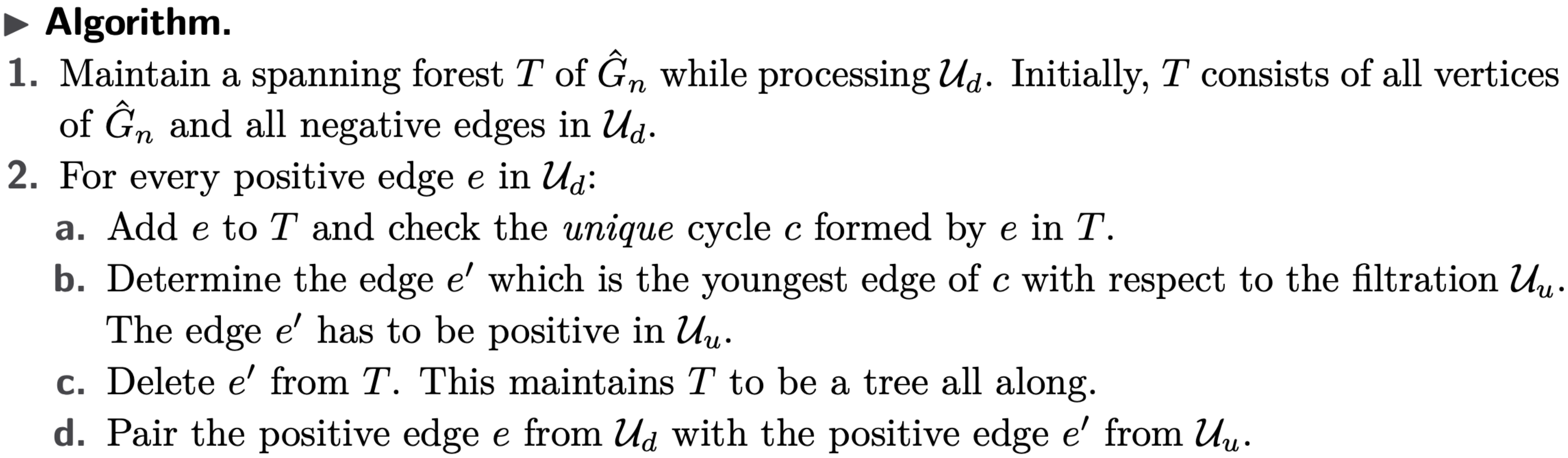
3. $\mathrm{Pers}_1^{\mathrm{CC}}(\mathcal{U})$: from the edge-edge pairs; the first edge is a positive edge from the ascending part $\mathcal{U}_u$, the second edge is a positive edge from the descending part $\mathcal{U}_d$.

Positive edge: connect to the same connected component

Negative edge: connect to the different connected components

Computation

3. $\mathrm{Pers}_1^{\mathrm{CC}}(\mathcal{U})$: from the edge-edge pairs; the first edge is a positive edge from the ascending part $\mathcal{U}_u$, the second edge is a positive edge from the descending part $\mathcal{U}_d$.



▶ **Algorithm.**

1. Maintain a spanning forest T of $\hat{G}_n$ while processing $\mathcal{U}_d$. Initially, T consists of all vertices of $\hat{G}_n$ and all negative edges in $\mathcal{U}_d$.
2. For every positive edge e in $\mathcal{U}_d$:
    a. Add e to T and check the *unique* cycle c formed by e in T.
    b. Determine the edge e' which is the youngest edge of c with respect to the filtration $\mathcal{U}_u$. The edge e' has to be positive in $\mathcal{U}_u$.
    c. Delete e' from T. This maintains T to be a tree all along.
    d. Pair the positive edge e from $\mathcal{U}_d$ with the positive edge e' from $\mathcal{U}_u$.

Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, and Chao Chen. Link prediction with persistent homology: An interactive view. 2021.

Positive edge: connect to the same connected component

Negative edge: connect to the different connected components

Computation

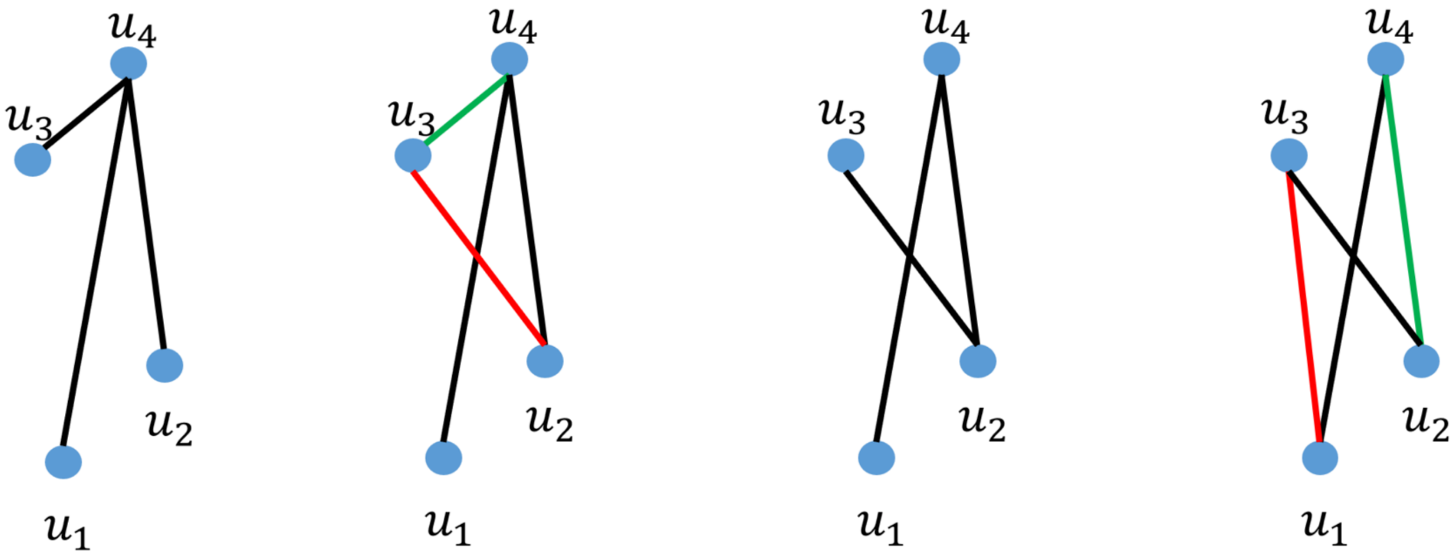


Figure from [Yan et al. 2021]

Computation

3. $\text{Pers}_1^{\text{CC}}(\mathcal{U})$:

- The algorithm in [Yan et al. 2021] runs in $O(m^2)$ time using a direct implementation for the trees
- We propose to use the Link-Cut tree [Sleator, Tarjan, 1981] so that finding the edge-edge pairs runs in $O(m\log m)$ time.
- Since the conversions between input zigzag and up-down are linear time, the overall complexity is $O(m\log m)$.

Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. 1981.

Updating zigzag persistence

[DeyH]: Computing Zigzag Vineyard Efficiently Including Expansions and Contractions. SoCG24

Background

Consider local changes on filtration and update the barcode accordingly

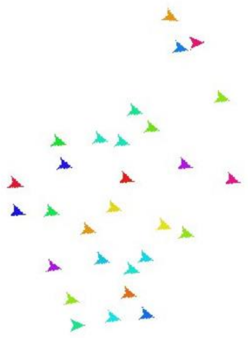
- Produces ***vineyard*** (a stack of barcodes)

Background

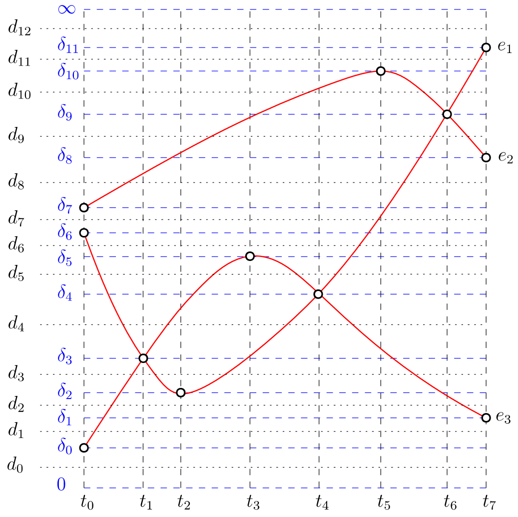
Consider local changes on filtration and update the barcode accordingly

- Produces ***vineyard*** (a stack of barcodes)

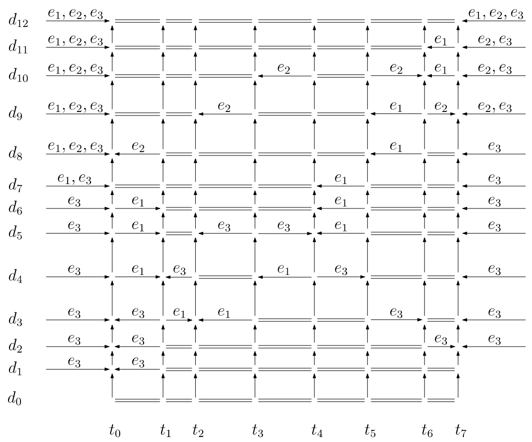
Example: Dynamic point cloud



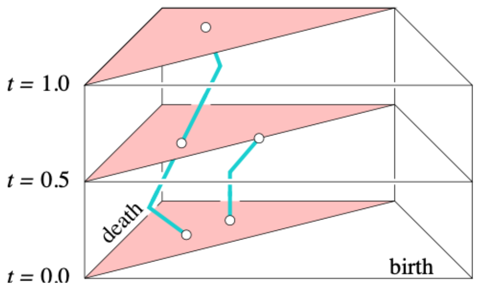
Point cloud moving with time



Distance-time curves of all pairs



Zigzag filtrations changing over distance



Vineyard
(figure from *Computational topology: An Introduction*)

Background

Consider local changes on filtration and update the barcode accordingly

- Produces ***vineyard*** (a stack of barcodes)

Operations in standard persistence, computed in O(m) time [CEM06]

Switch (transposition)

$$\begin{tikzcd}
\mathcal{F} : \varnothing = K_0 \arrow[r, hook] & \cdots \arrow[r, hook] & K_{i-1} \arrow[r, hook, "\sigma"] & K_i \arrow[r, hook, "\tau"] & K_{i+1} \arrow[r, hook] & \cdots \arrow[r, hook] & K_m \arrow[d, dashed, bend left] \\
\mathcal{F}' : \varnothing = K_0 \arrow[r, hook] & \cdots \arrow[r, hook] & K_{i-1} \arrow[r, hook, "\tau"] & K_i' \arrow[r, hook, "\sigma"] & K_{i+1} \arrow[r, hook] & \cdots \arrow[r, hook] & K_m
\end{tikzcd}$$

Cohen-Steiner, Edelsbrunner, and Morozov. Vines and vineyards by updating persistence in linear time. 2006

Background

Consider local changes on filtration and update the barcode accordingly

- Produces ***vineyard*** (a stack of barcodes)

Operations we consider

Forward switch

$$\begin{tikzcd} \mathcal{F} : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-1} \xhookrightarrow{\sigma} K_i \xhookrightarrow{\tau} K_{i+1} \leftrightarrow \cdots \leftrightarrow K_m \arrow[d, dashed, bend left] \\ \mathcal{F}' : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-1} \xhookrightarrow{\tau} K_i' \xhookrightarrow{\sigma} K_{i+1} \leftrightarrow \cdots \leftrightarrow K_m \end{tikzcd}$$

Backward switch

$$\begin{tikzcd} \mathcal{F} : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-1} \xhookleftarrow{\sigma} K_i \xhookleftarrow{\tau} K_{i+1} \leftrightarrow \cdots \leftrightarrow K_m \arrow[d, dashed, bend left] \\ \mathcal{F}' : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-1} \xhookleftarrow{\tau} K_i' \xhookleftarrow{\sigma} K_{i+1} \leftrightarrow \cdots \leftrightarrow K_m \end{tikzcd}$$

Outward/inward switch

$$\begin{tikzcd} \mathcal{F} : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-1} \xhookrightarrow{\sigma} K_i \xhookleftarrow{\tau} K_{i+1} \leftrightarrow \cdots \leftrightarrow K_m \arrow[d, dashed, bend left, <->] \\ \mathcal{F}' : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-1} \xhookleftarrow{\tau} K_i' \xhookrightarrow{\sigma} K_{i+1} \leftrightarrow \cdots \leftrightarrow K_m \end{tikzcd}$$

Keep filtration size

Inward contraction/expansion

$$\begin{tikzcd} \mathcal{F} : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_{i-1} \xhookrightarrow{\sigma} K_i \xhookleftarrow{\sigma} K_{i+1} \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m \arrow[d, dashed, bend left, <->] \\ \mathcal{F}' : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_i' \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m \end{tikzcd}$$

Outward contraction/expansion

$$\begin{tikzcd} \mathcal{F} : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_{i-1} \xhookleftarrow{\sigma} K_i \xhookrightarrow{\sigma} K_{i+1} \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m \arrow[d, dashed, bend left, <->] \\ \mathcal{F}' : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_i' \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m \end{tikzcd}$$

Increase/decrease filtration size

Easy updates

For following switch operations (do not change input length):

- Barcodes can be easily updated in O(m) time
- Using the conversion in FastZigzag

Forward switch

$$\begin{tikzcd}[column sep=small]
\mathcal{F} : K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{i-1} \arrow[r, hook, "\sigma"] & K_i \arrow[r, hook, "\tau"] & K_{i+1} \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m \arrow[d, dashed, bend left] \\
\mathcal{F}' : K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{i-1} \arrow[r, hook, "\tau"] & K_i' \arrow[r, hook, "\sigma"] & K_{i+1} \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m
\end{tikzcd}$$

Backward switch

$$\begin{tikzcd}[column sep=small]
\mathcal{F} : K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{i-1} & K_i \arrow[l, hook', "\sigma"'] & K_{i+1} \arrow[l, hook', "\tau"'] \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m \arrow[d, dashed, bend left] \\
\mathcal{F}' : K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{i-1} & K_i' \arrow[l, hook', "\tau"'] & K_{i+1} \arrow[l, hook', "\sigma"'] \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m
\end{tikzcd}$$

Outward/inward switch

$$\begin{tikzcd}[column sep=small]
\mathcal{F} : K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{i-1} \arrow[r, hook, "\sigma"] & K_i & K_{i+1} \arrow[l, hook', "\tau"'] \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m \arrow[d, dashed, bend left, leftrightarrow] \\
\mathcal{F}' : K_0 \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_{i-1} & K_i' \arrow[l, hook', "\tau"'] \arrow[r, hook, "\sigma"] & K_{i+1} \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & K_m
\end{tikzcd}$$

Cohen-Steiner, Edelsbrunner, and Morozov. Vines and vineyards by updating persistence in linear time. 2006

Difficulties with (outward) contraction/expansion

Difficulties lie in (outward) contraction/expansion (input length changes)

Inward contraction/expansion

$$\begin{tikzcd}
\mathcal{F} : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_{i-1} \xhookrightarrow{\sigma} K_i \xhookleftarrow{\sigma} K_{i+1} \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m \arrow[d, dashed, <->, bend left] \\
\mathcal{F}' : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_i' \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m
\end{tikzcd}$$

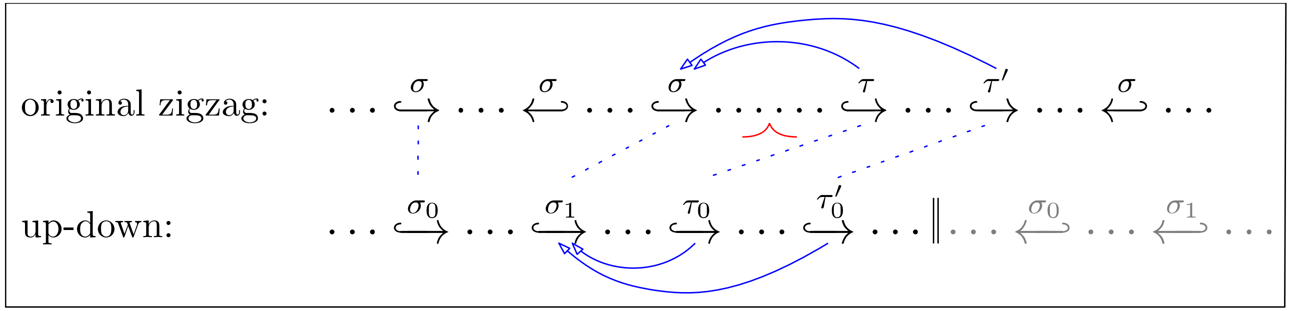
Outward contraction/expansion

$$\begin{tikzcd}
\mathcal{F} : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_{i-1} \xhookleftarrow{\sigma} K_i \xhookrightarrow{\sigma} K_{i+1} \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m \arrow[d, dashed, <->, bend left] \\
\mathcal{F}' : K_0 \leftrightarrow \cdots \leftrightarrow K_{i-2} \leftrightarrow K_i' \leftrightarrow K_{i+2} \leftrightarrow \cdots \leftrightarrow K_m
\end{tikzcd}$$

Difficulties with (outward) contraction/expansion

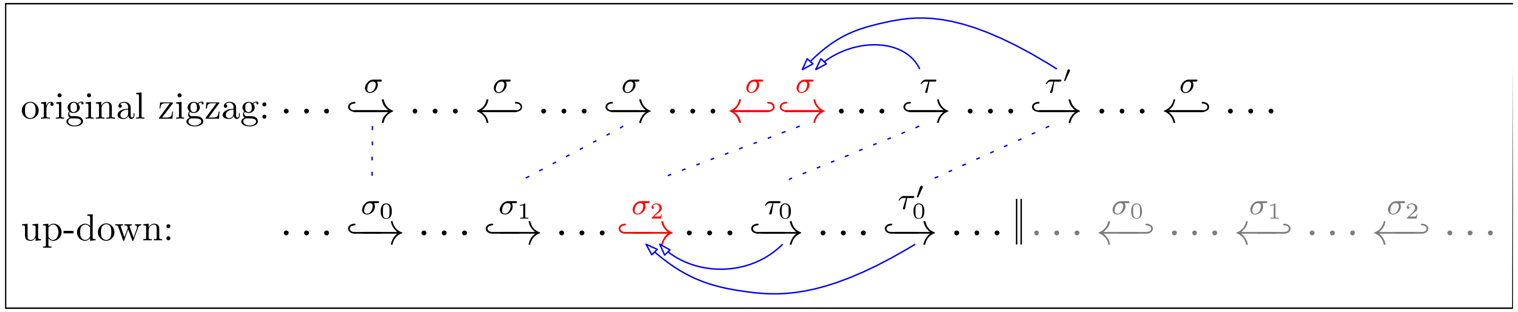
- If we convert the zigzag filtrations into up-down/non-zigzag filtrations, there are some **adjacency change** on the cells:
  - *Before and after the operation, boundary faces of certain $(p+1)$-cells change into other p-cells (which come in earlier/later in the up-down/non-zigzag filtration)*
- Straightforward approach takes O(m^3) time

Before outward expansion



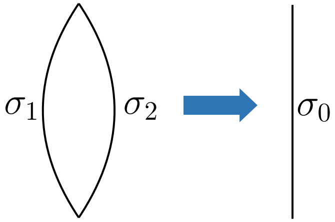
⇓

After outward expansion



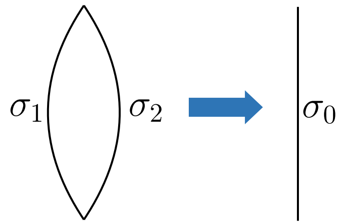
Idea of the computation for outward contraction

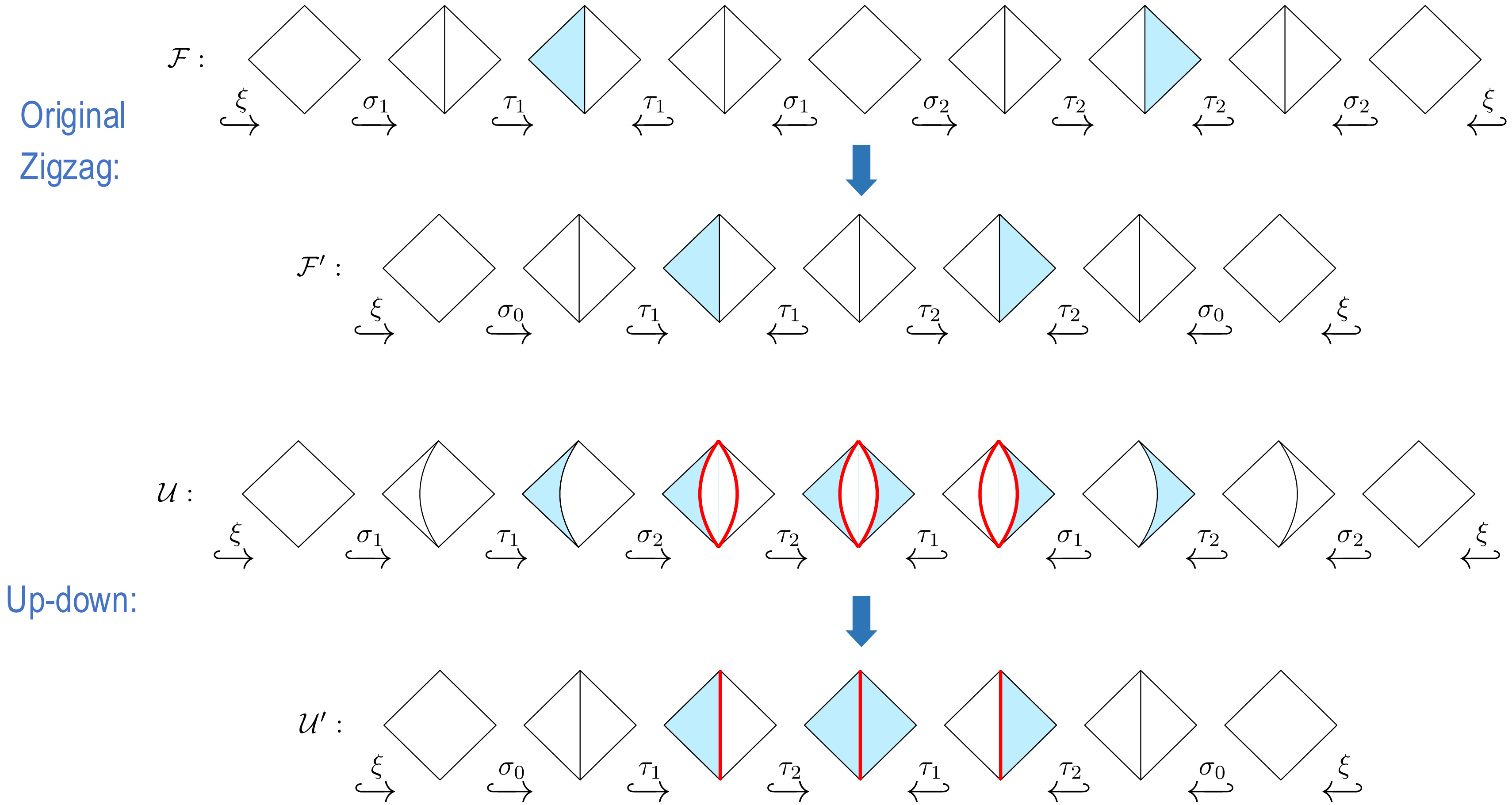
- Convert input zigzag filtration into up-down filtration
- [*Observation*] The boundary change of cells in the up-down filtration in the contraction:
  - Two p-cells σ_1, σ_2 are identified as the same p-cell σ_0



Idea of the computation for outward contraction

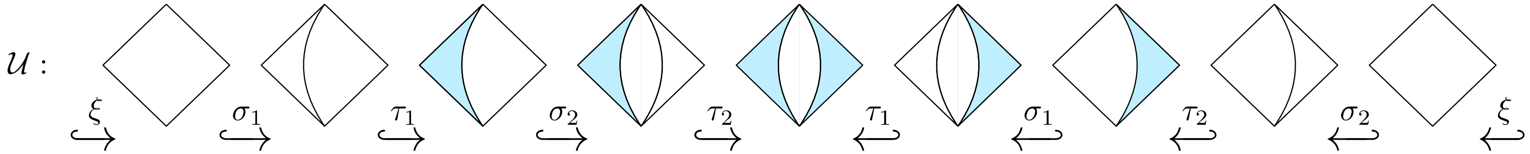
- Convert input zigzag filtration into up-down filtration
- [*Observation*] The boundary change of cells in the up-down filtration in the contraction:
    - ○ Two p-cells σ_1, σ_2 are identified as the same p-cell σ_0





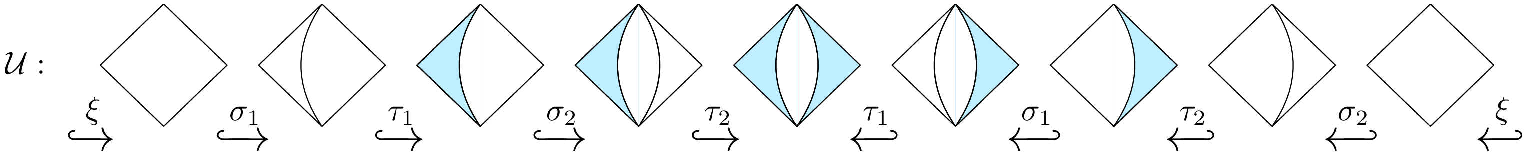
Solution for outward contraction

Given:

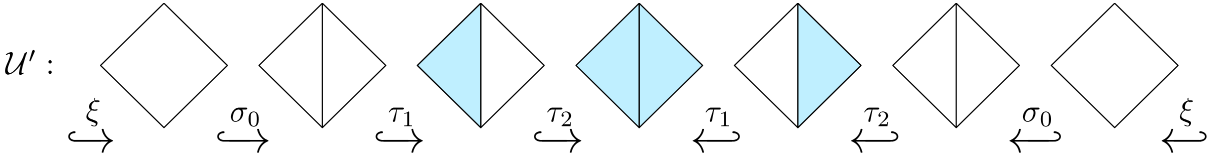


Solution for outward contraction

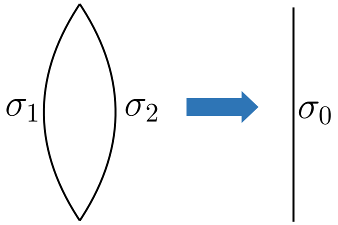
Given:



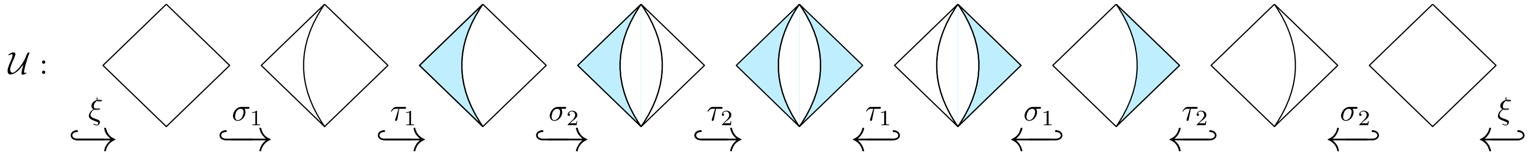
Compute:



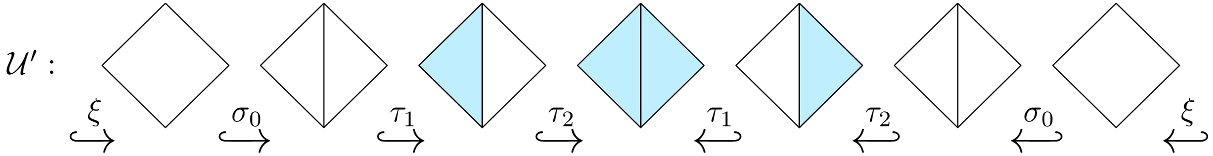
Solution for outward contraction



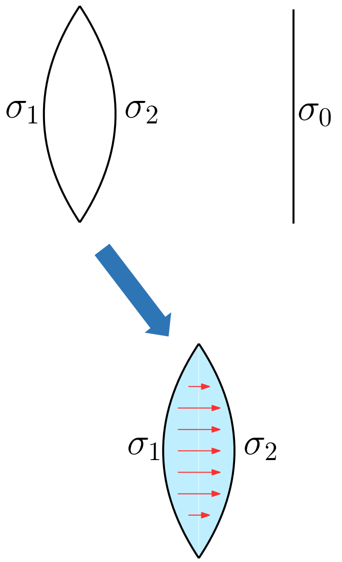
Given:



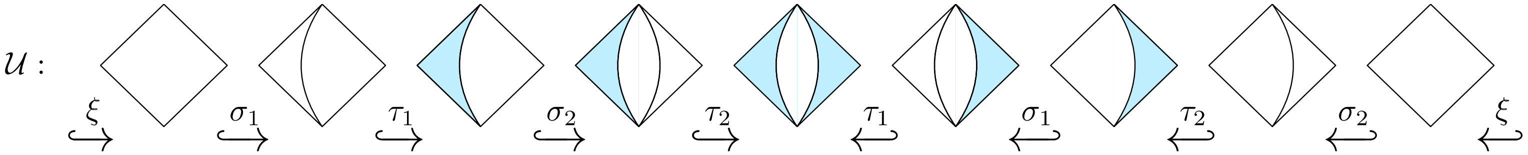
Compute:



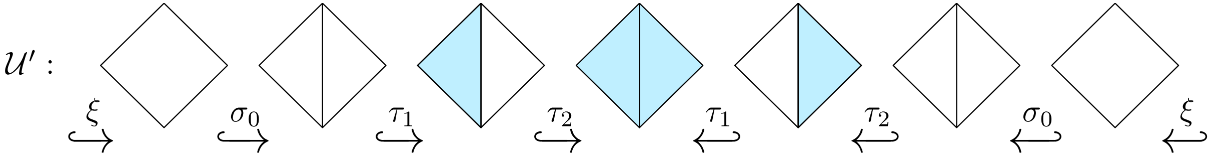
Solution for outward contraction



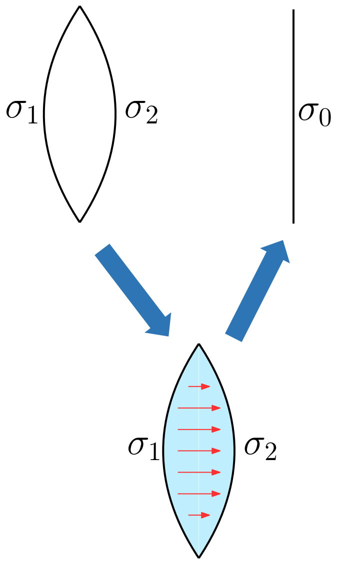
Given:



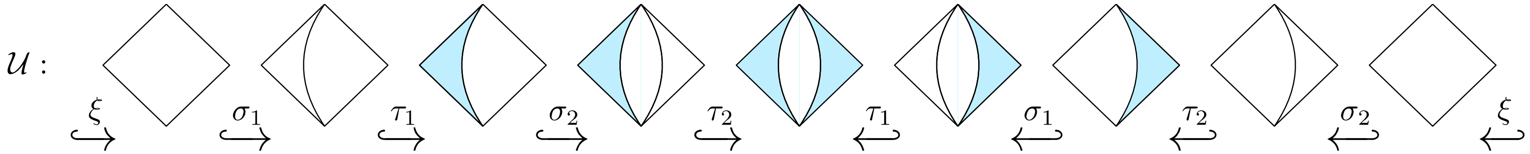
Compute:



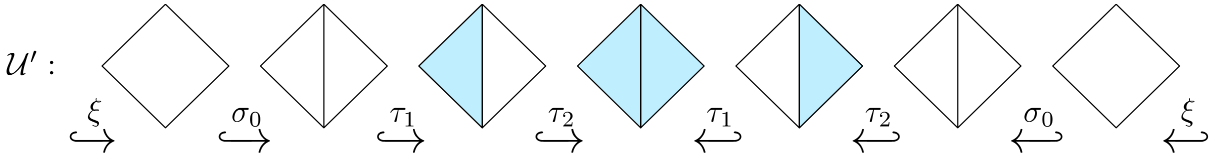
Solution for outward contraction



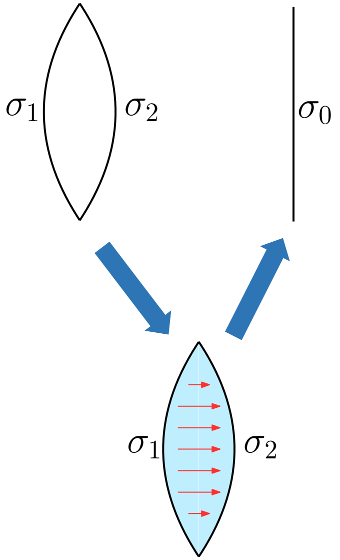
Given:

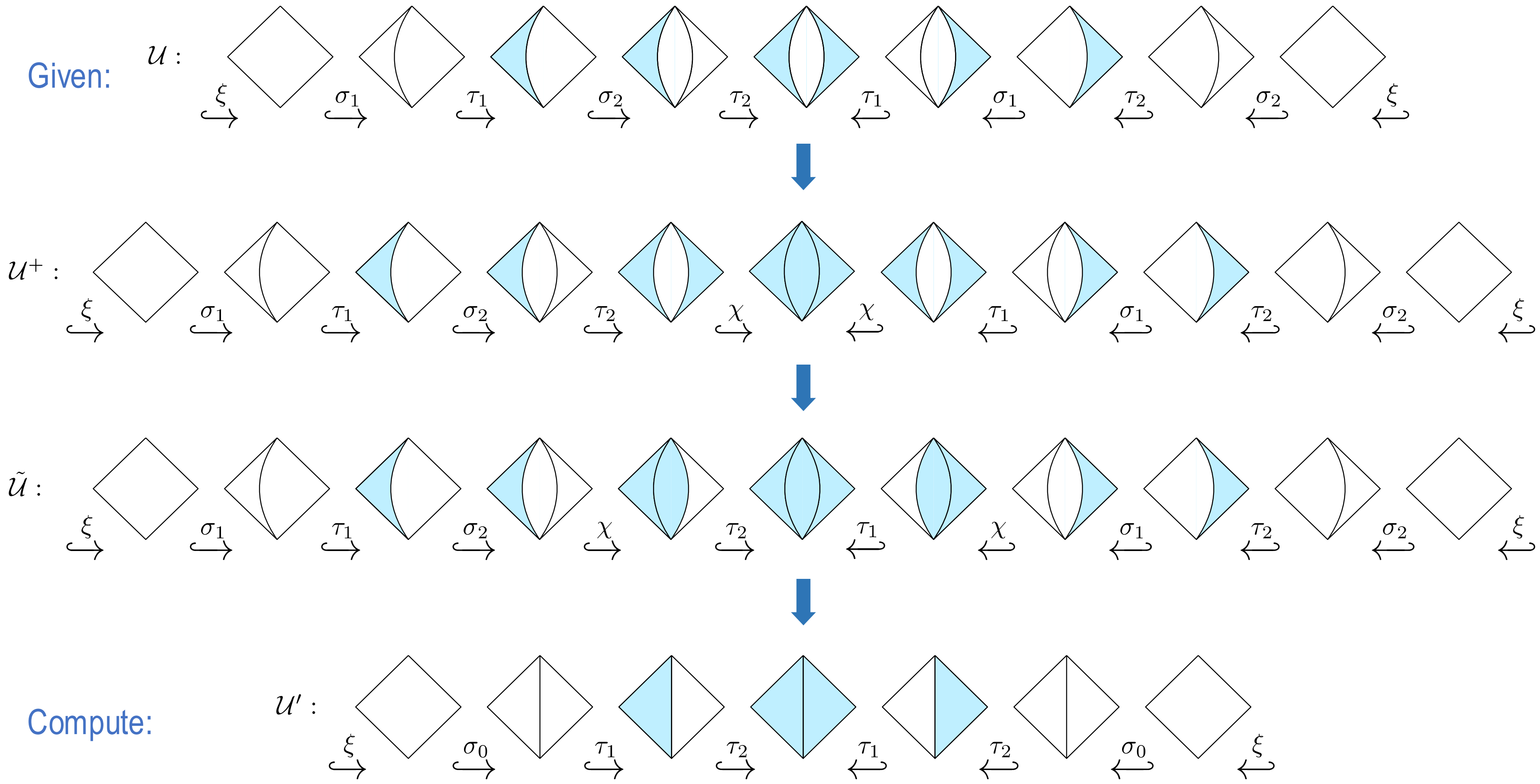


Compute:



Solution for outward contraction



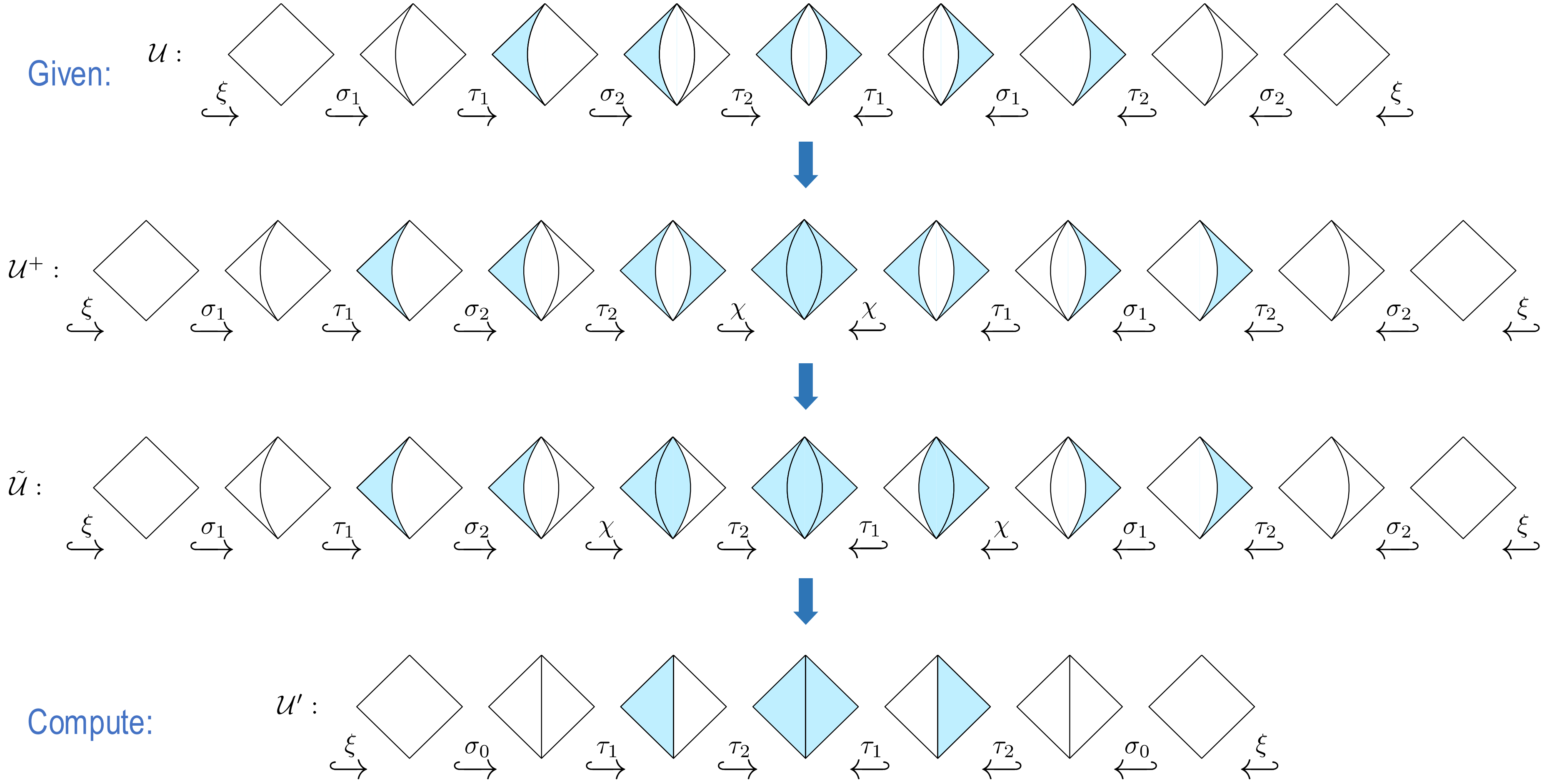

Given:

Compute:

Solution for outward contraction

$U \Rightarrow U^+$:

- Attaching χ
- O(m^2)



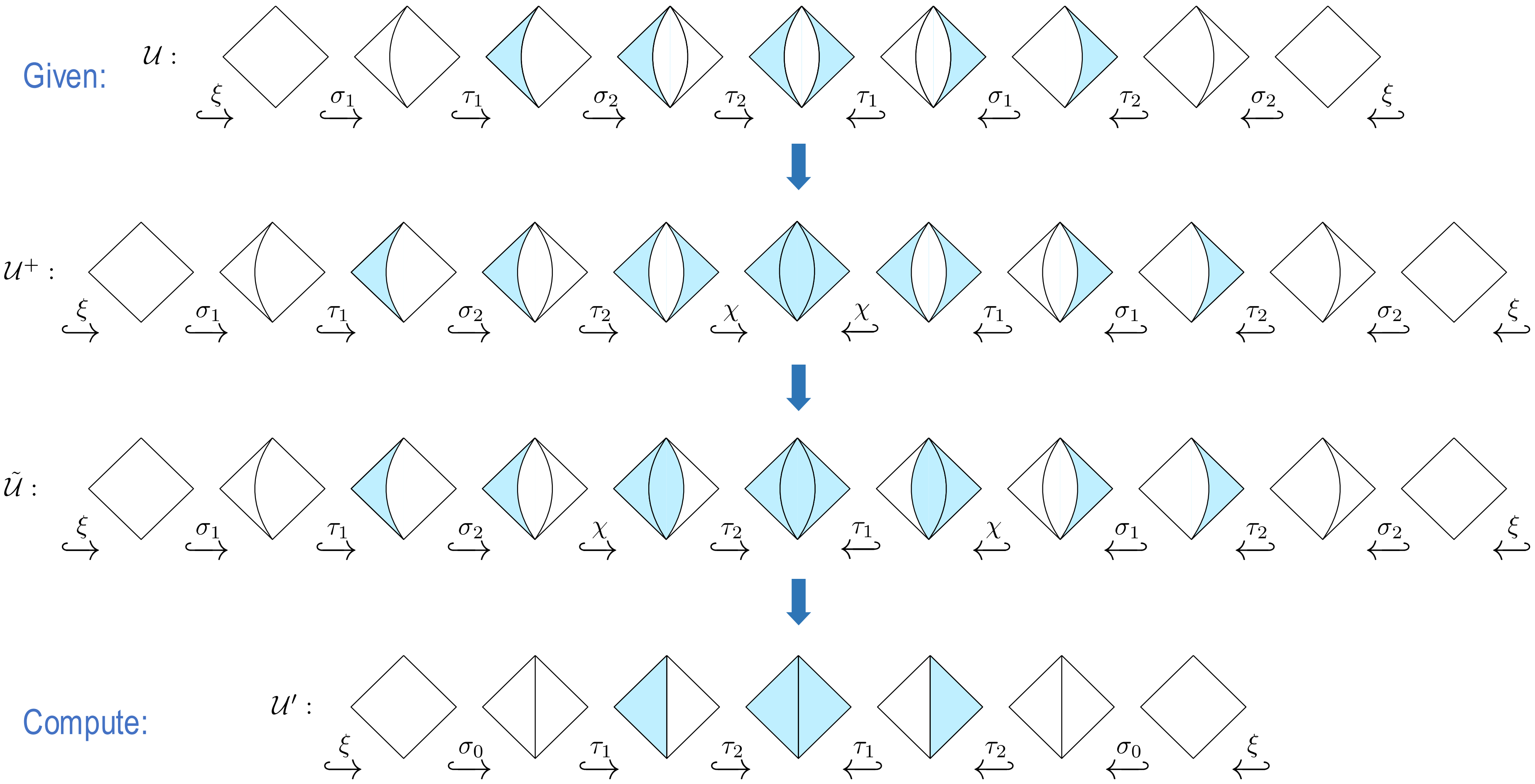
Solution for outward contraction

$U \Rightarrow U^{+}$:

- Attaching χ
- $O(m^2)$

$U^{+} \Rightarrow \widetilde{U}$:

- Perform switches
- $O(m^2)$


Given:

Compute:

Solution for outward contraction

$U \Rightarrow U^{+}$:

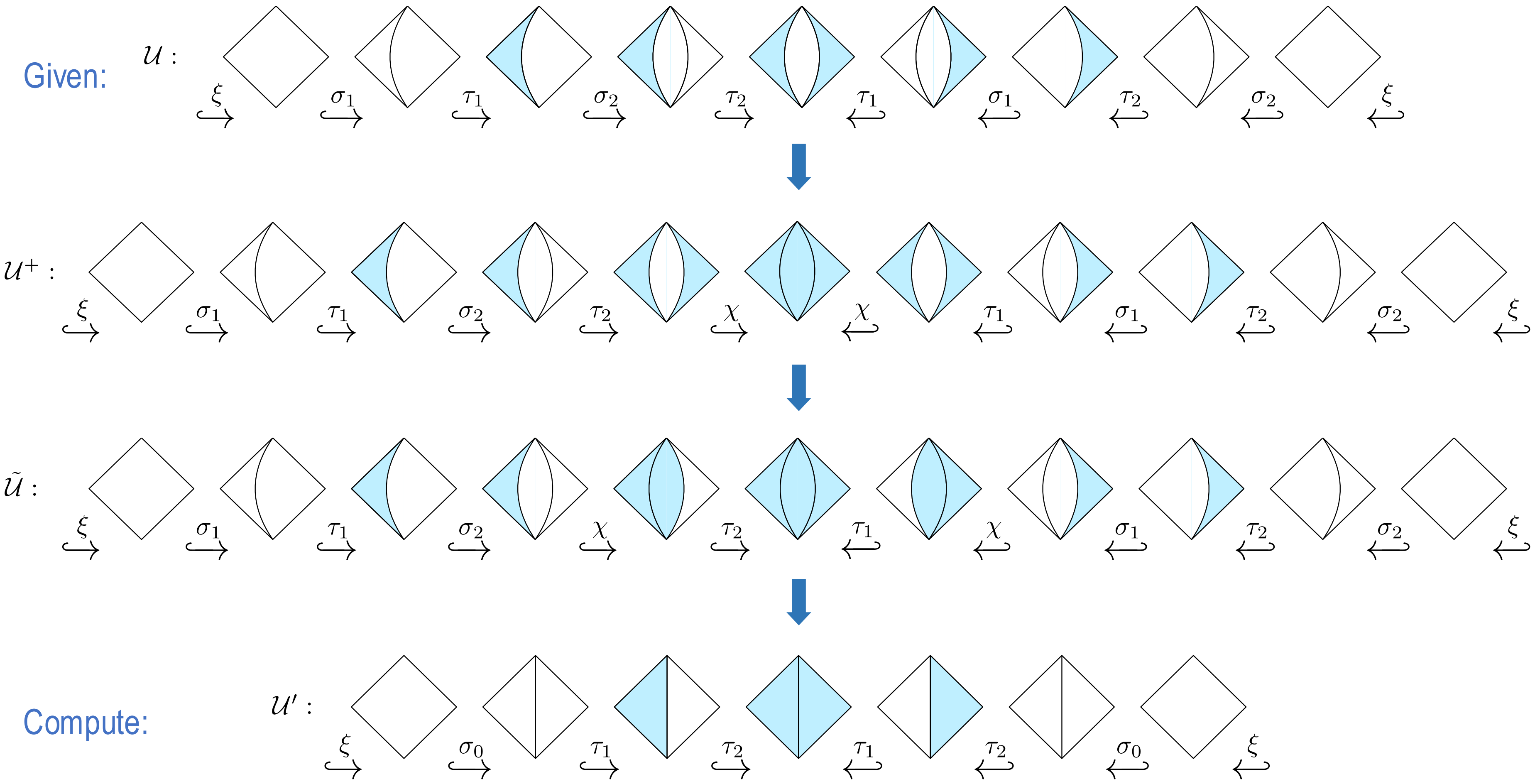
- Attaching χ
- O(m^2)

$U^{+} \Rightarrow \widetilde{U}$:

- Perform switches
- O(m^2)

$\widetilde{U} \Rightarrow U'$:

- “Almost” the same
- O(m)



Solution for outward contraction

$\widetilde{U} \Rightarrow U'$, formally:

Proposition. *Given* $\mathsf{Pers}_*(\tilde{\mathcal{U}})$*, one only needs to do the following to obtain* $\mathsf{Pers}_*(\mathcal{U}')$***: Ignoring the pairs*** $(\searrow\sigma_2, \searrow\xi)$ ***and*** $(\nwarrow\xi, \nwarrow\sigma_1)$ ***in*** $\mathsf{Pers}_*(\tilde{\mathcal{U}})$***, for each remaining pair*** $(\nwsearrow\eta, \nwsearrow\gamma) \in \mathsf{Pers}_*(\tilde{\mathcal{U}})$***, produce a corresponding pair*** $(\theta(\nwsearrow\eta), \theta(\nwsearrow\gamma)) \in \mathsf{Pers}_*(\mathcal{U}')$.

Solution for outward contraction

$\widetilde{U} \Rightarrow U'$, formally:

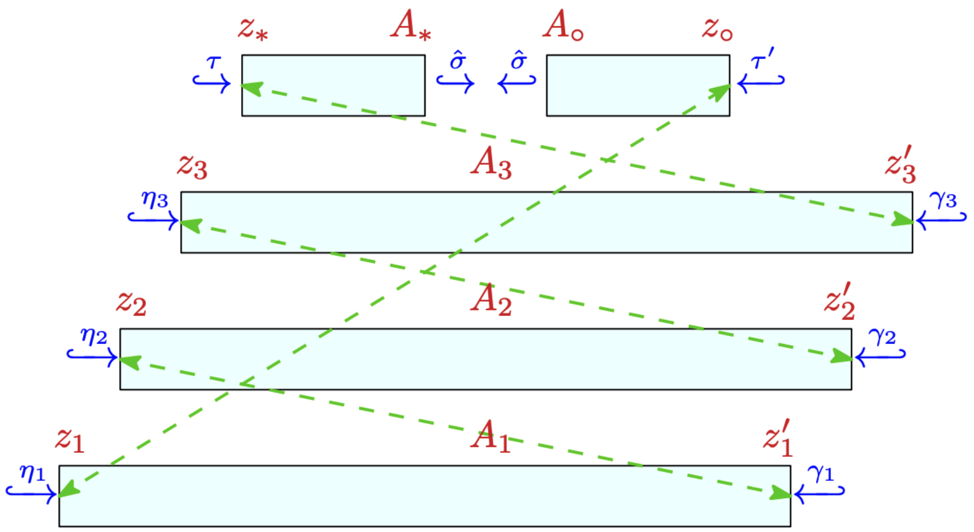
Proposition. *Given* $\mathsf{Pers}_*(\tilde{\mathcal{U}})$*, one only needs to do the following to obtain* $\mathsf{Pers}_*(\mathcal{U}')$*: Ignoring the pairs* $(\searrow\sigma_2, \searrow\xi)$ *and* $(\nwarrow\xi, \nwarrow\sigma_1)$ *in* $\mathsf{Pers}_*(\tilde{\mathcal{U}})$*, for each remaining pair* $(\nwsearrow\eta, \nwsearrow\gamma) \in \mathsf{Pers}_*(\tilde{\mathcal{U}})$*, produce a corresponding pair* $(\theta(\nwsearrow\eta), \theta(\nwsearrow\gamma)) \in \mathsf{Pers}_*(\mathcal{U}')$.

Conclusion:

Theorem. *The barcodes for an outward contraction on zigzag filtrations can be updated in* $O(m^2)$ *time, matching the complexity for a contraction on the standard filtrations.*

Inward contraction

- While inward contraction is easy by converting to non-zigzag, it becomes non-trivial when converting to up-down
- Algorithm idea:
    - After some preprocessing, we are left with certain intervals which are not 'settled' (contains the cell being removed)
    - These intervals follow a fixed pattern, and we utilized an 'alternative relinking' to produce intervals for the new filtration



Updating zigzag persistence on graphs over switches

[Dey-H-Parsa]. Revisiting Graph Persistence for Updates and Efficiency. WADS 2023

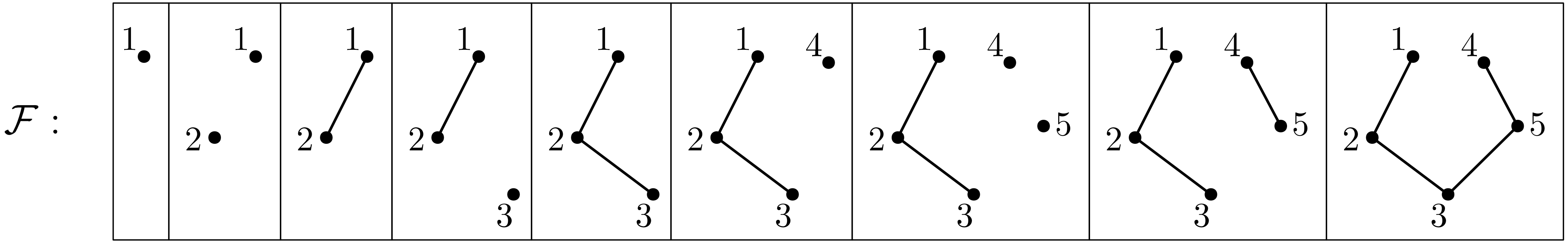
Update for non-zigzag graph filtrations

- Propose $O(\log m)$ algorithms for updating non-zigzag filtrations on graphs over switches

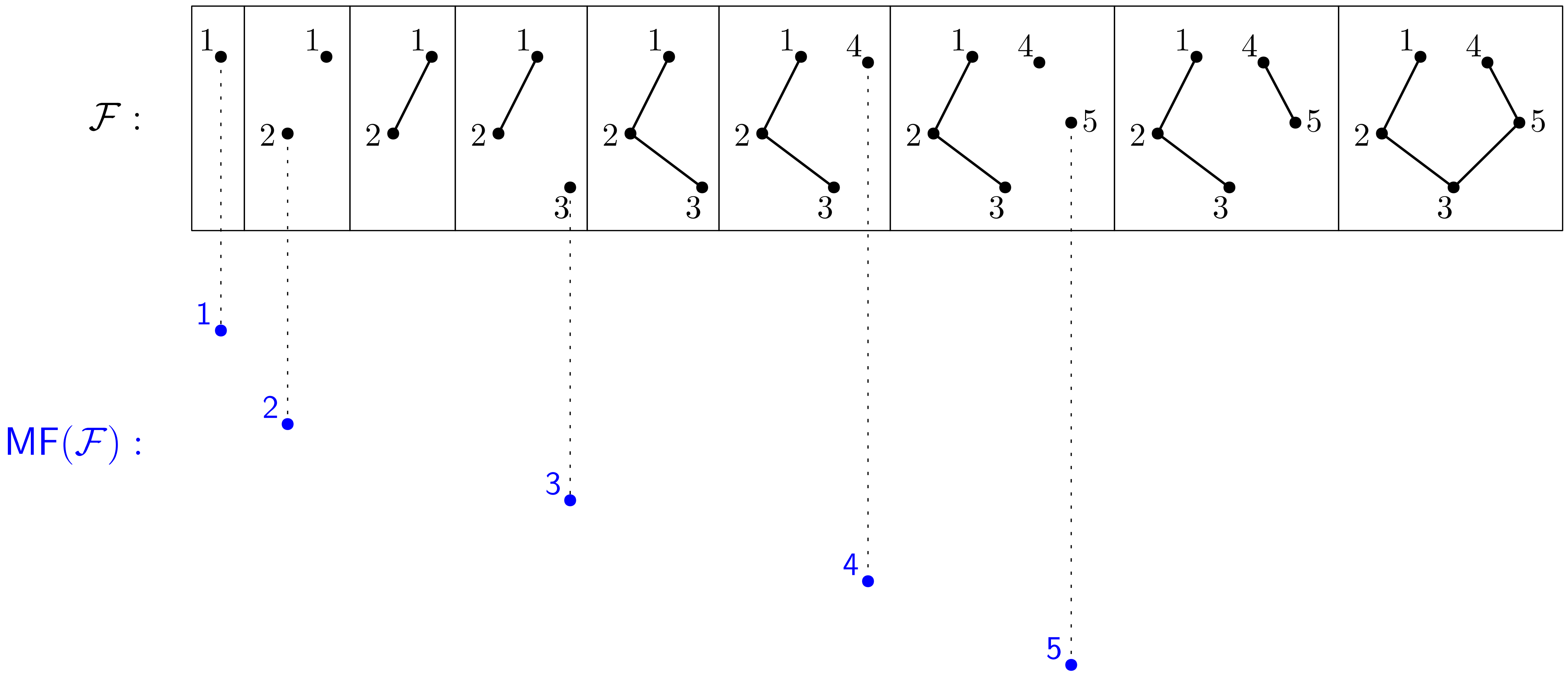
$$\begin{tikzcd}
\mathcal{F} : \varnothing = G_0 \arrow[r, hook] & \cdots \arrow[r, hook] & G_{i-1} \arrow[r, hook, "\sigma"] & G_i \arrow[r, hook, "\tau"] & G_{i+1} \arrow[r, hook] & \cdots \arrow[r, hook] & G_m \arrow[d, dashed, bend left] \\
\mathcal{F}' : \varnothing = G_0 \arrow[r, hook] & \cdots \arrow[r, hook] & G_{i-1} \arrow[r, hook, "\tau"] & G_i' \arrow[r, hook, "\sigma"] & G_{i+1} \arrow[r, hook] & \cdots \arrow[r, hook] & G_m
\end{tikzcd}$$

- Maintain merge forest (trees) encoding all info in the pers module
- Case analysis: Perform the update in difference cases
- Use two dynamic trees data structure (DFT tree, Link-Cut tree) to achieve the complexity

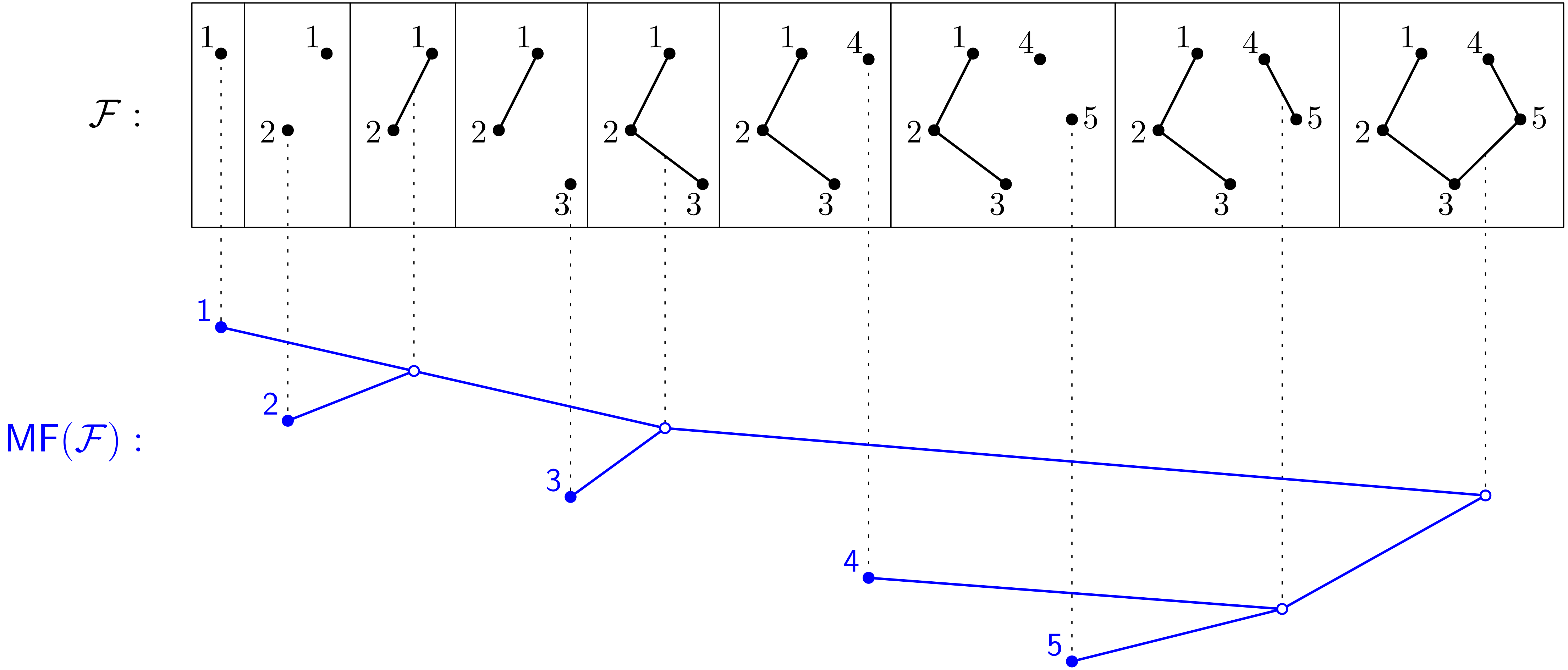
Merge forest (tree)



Merge forest (tree)

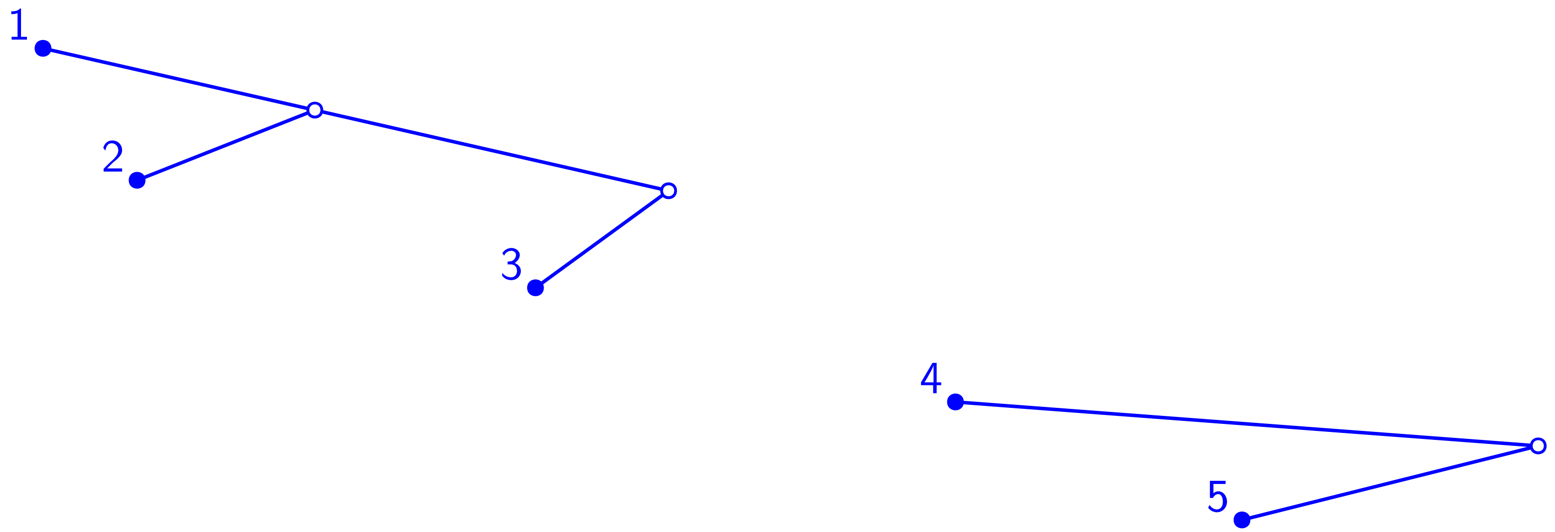


Merge forest (tree)



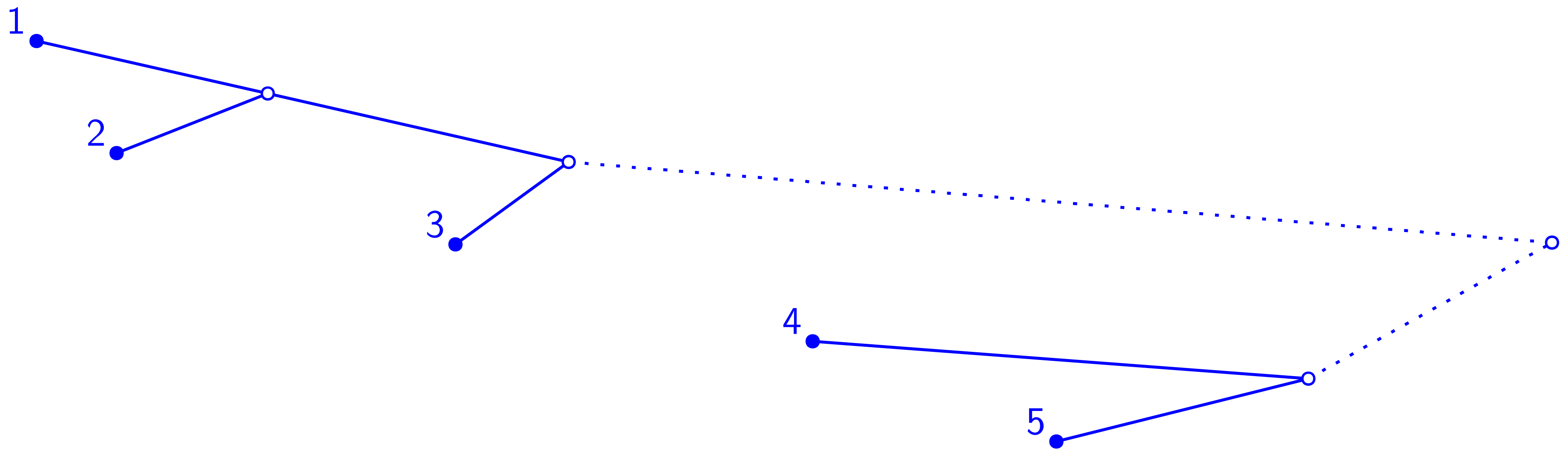
Merge forest (tree)

MF($\mathcal{F}$) :



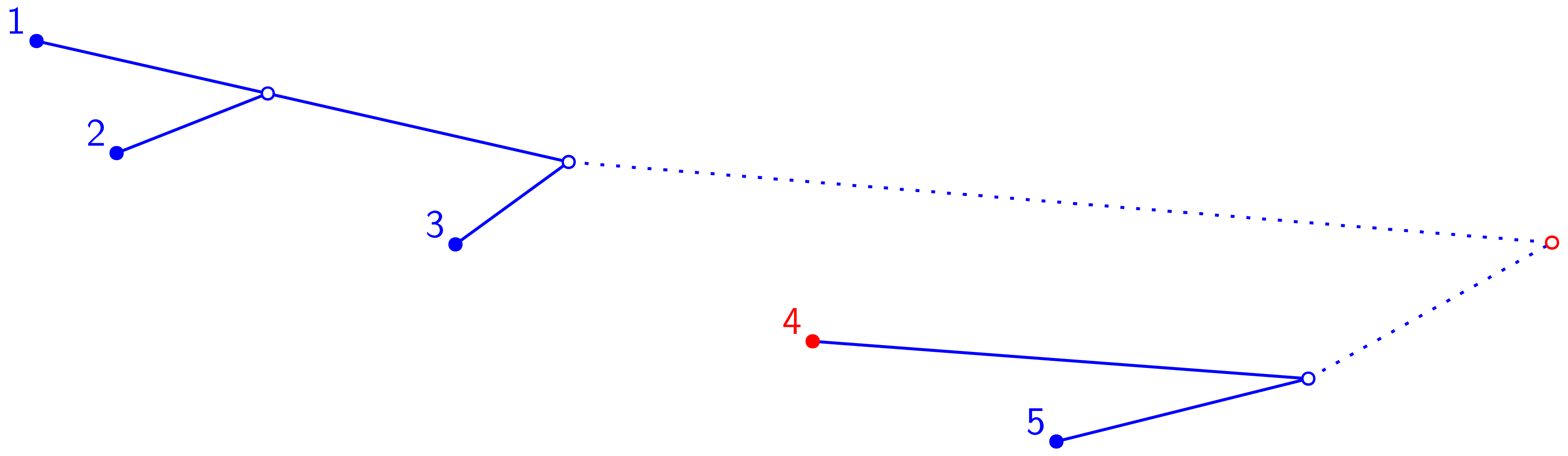
Merge forest (tree)

$\mathsf{MF}(\mathcal{F})$:



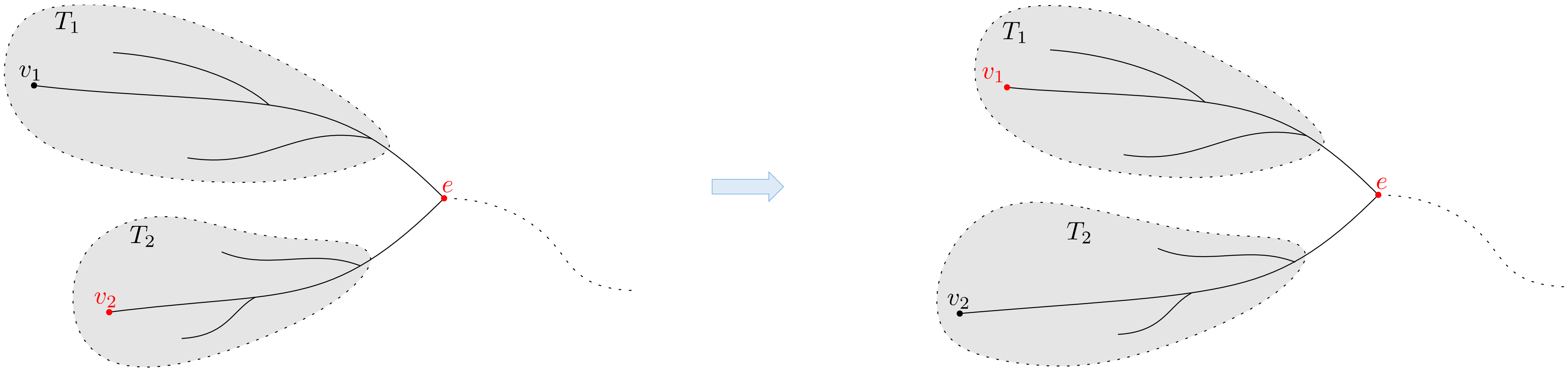
Merge forest (tree)

$\mathsf{MF}(\mathcal{F})$:



1. Switch two vertices v_1, v_2

- The only situation where the pairing changes:
  - v_1, v_2 are in the same tree in the merge forest
  - v_1, v_2 are both unpaired when e is added in $\mathcal{F}$, where e is the edge corresponding to the *nearest common ancestor* of v_1, v_2 in the merge forest
- In above case, we switch the paired edges of v_1, v_2



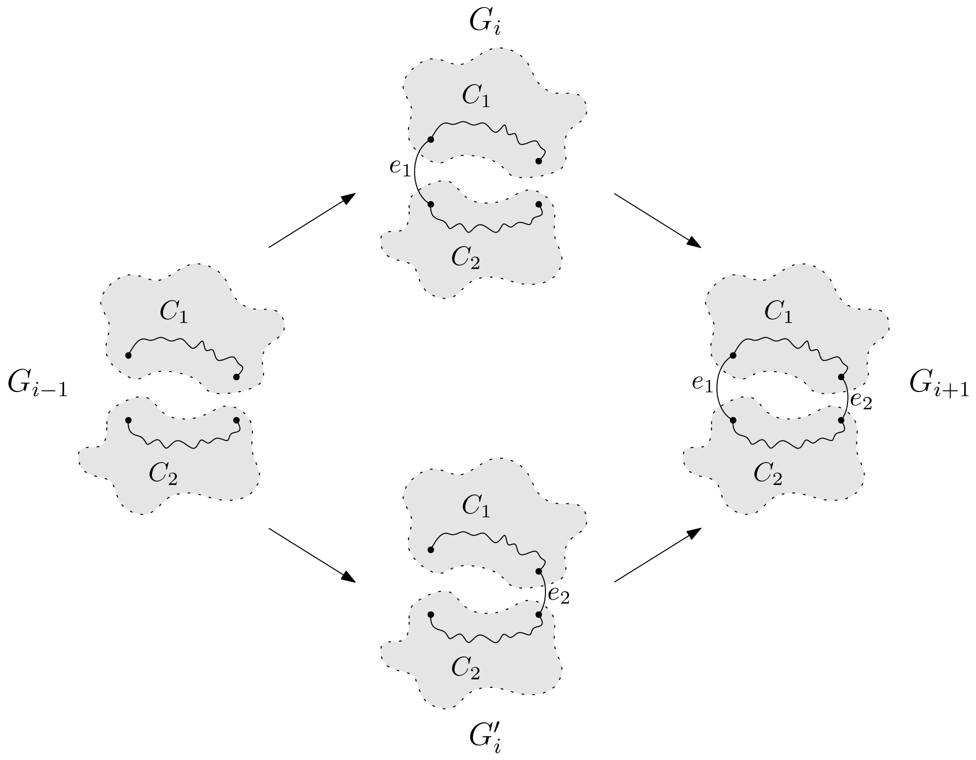
More definitions

Two types of edges in the graph filtration:

- Negative edge: connect two different connected components
- Positive edge: connect the same connected component

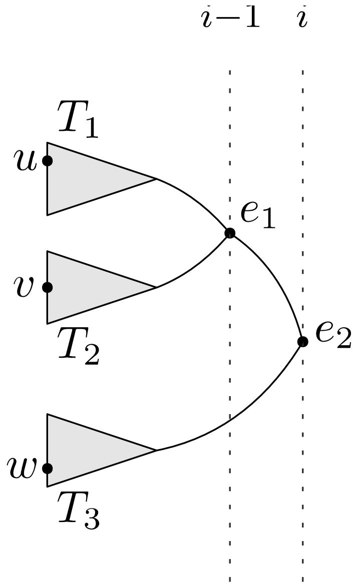
2. Switch a negative edge e_1 and a positive edge e_2

- If e_1 is in a 1-cycle after is e_2 added:
  - This is the case where e_1, e_2 connect to the same two connected components
  - After the switch, e_1 becomes positive and e_2 becomes negative
  - We pair e_2 with the vertex that e_1 previously pairs with
  - The node in the merge forest corresponding to e_1 should now correspond to e_2



3. Switch two negative edges e_1, e_2

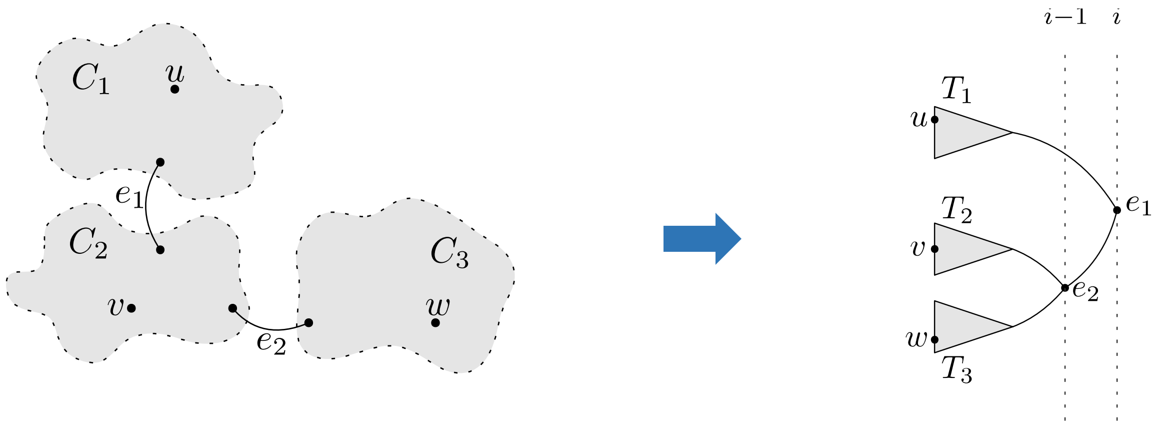
- Only need to make changes when the corresponding node of e_1 is a child of the corresponding node of e_2 in the merge forest
- Let u, v, w be the lowest leaves in T_1, T_2, T_3.
- WLOG, assume v is lower than u.



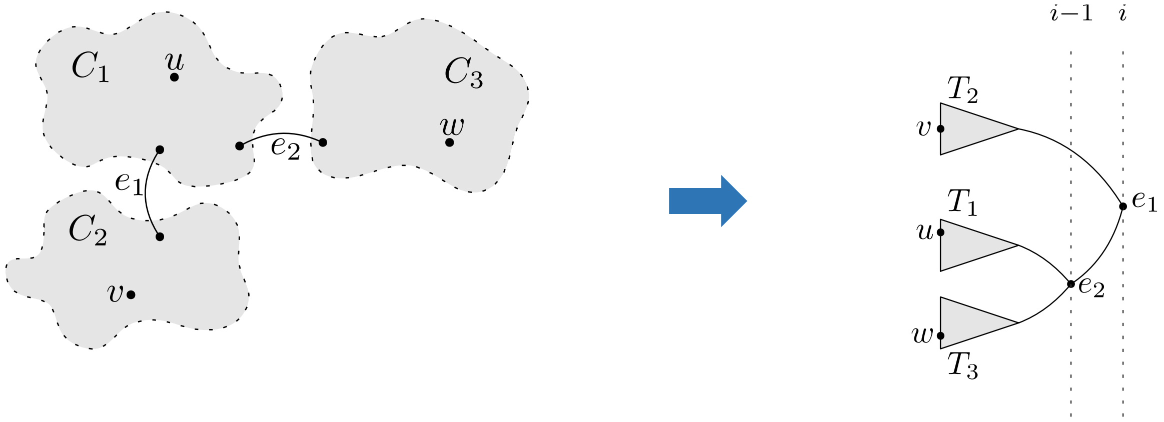
3. Switch two negative edges e_1, e_2

- Based on the structure of the merge forest, there are two connecting configurations for C_1, C_2, C_3 in G_{i-1} (C_1, C_2, C_3 are the connected components containing u, v, w respectively)

(a)



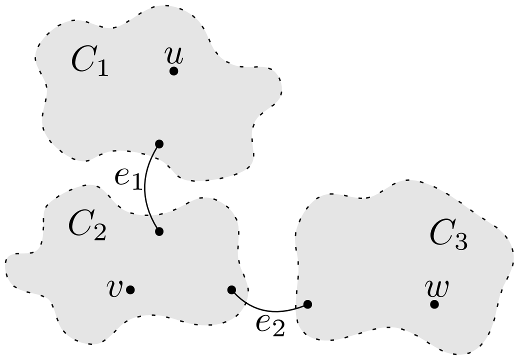
(b)



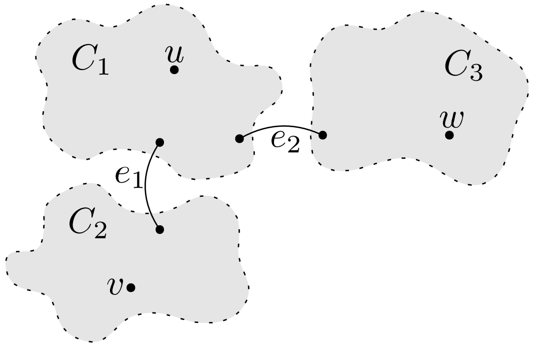
3. Switch two negative edges e_1, e_2

- Based on the structure of the merge forest, there are two connecting configurations for C_1, C_2, C_3 in G_{i-1} (C_1, C_2, C_3 are the connected components containing u, v, w respectively)

(a)



(b)



If w is lower than u, then swap the paired vertices of e_1, e_2

- Provable by case analysis

DFT-Tree

Data structures implementing the merge forests [Farina, Laura, 2015]:

$O(\log m)$

- **ROOT**(v): Returns the root of the tree containing node v.
- **CUT**(v): Deletes the edge connecting node v to its parent.
- **LINK**(u, v): Makes the root of the tree containing node v be a child of node u.
- **NCA**(u, v): Returns the nearest common ancestor of two nodes u, v in the same tree.
- **CHANGE-VAL**(v, x): Assigns the value associated to a leaf v to be x.
- **SUBTREE-MIN**(v): Returns the leaf with the minimum associated value in the subtree rooted at v.

Returns the lowest leaf for a subtree

Gabriele Farina and Luigi Laura. Dynamic subtrees queries revisited: The depth first tour tree. 2015.

Detecting if $e_1 = (u, v)$ is in a cycle in G_{i+1}

- Check if u, v are connected in G'_i

$$\mathcal{F}' : \varnothing = G_0 \hookrightarrow \cdots \hookrightarrow G_{i-1} \hookrightarrow G'_i \xhookrightarrow{e_1} G_{i+1} \hookrightarrow \cdots \hookrightarrow G_m$$

- Check the first time u, v are connected in the filtration $\mathcal{F}'$
- Based on an idea in [DH21], do following:
  - Let edges in $G \coloneqq G_m$ be weighted by their indices in $\mathcal{F}'$
  - The first time u, v are connected = 1 + the bottleneck weight of the path in the MSF of G (bottleneck weight: max weight of edges)
- Maintain the MSF over the switch by the Link-Cut tree [ST81]:
  - Everything can be in $O(\log m)$ time
  - This is possible because we are only doing switches (switching the weights for edges whose weights are consecutive)

- Tamal K. Dey and Tao Hou. Computing zigzag persistence on graphs in near-linear time. 2021.
- Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. 1981.

Update for zigzag graph filtrations

- Four switch operations:

Forward switch

$$\mathcal{F} : G_0 \leftrightarrow \cdots \leftrightarrow G_{i-1} \xhookrightarrow{\sigma} G_i \xhookrightarrow{\tau} G_{i+1} \leftrightarrow \cdots \leftrightarrow G_m$$
$$\mathcal{F}' : G_0 \leftrightarrow \cdots \leftrightarrow G_{i-1} \xhookrightarrow{\tau} G_i' \xhookrightarrow{\sigma} G_{i+1} \leftrightarrow \cdots \leftrightarrow G_m$$

Backward switch

$$\mathcal{F} : G_0 \leftrightarrow \cdots \leftrightarrow G_{i-1} \xhookleftarrow{\sigma} G_i \xhookleftarrow{\tau} G_{i+1} \leftrightarrow \cdots \leftrightarrow G_m$$
$$\mathcal{F}' : G_0 \leftrightarrow \cdots \leftrightarrow G_{i-1} \xhookleftarrow{\tau} G_i' \xhookleftarrow{\sigma} G_{i+1} \leftrightarrow \cdots \leftrightarrow G_m$$

Outward switch

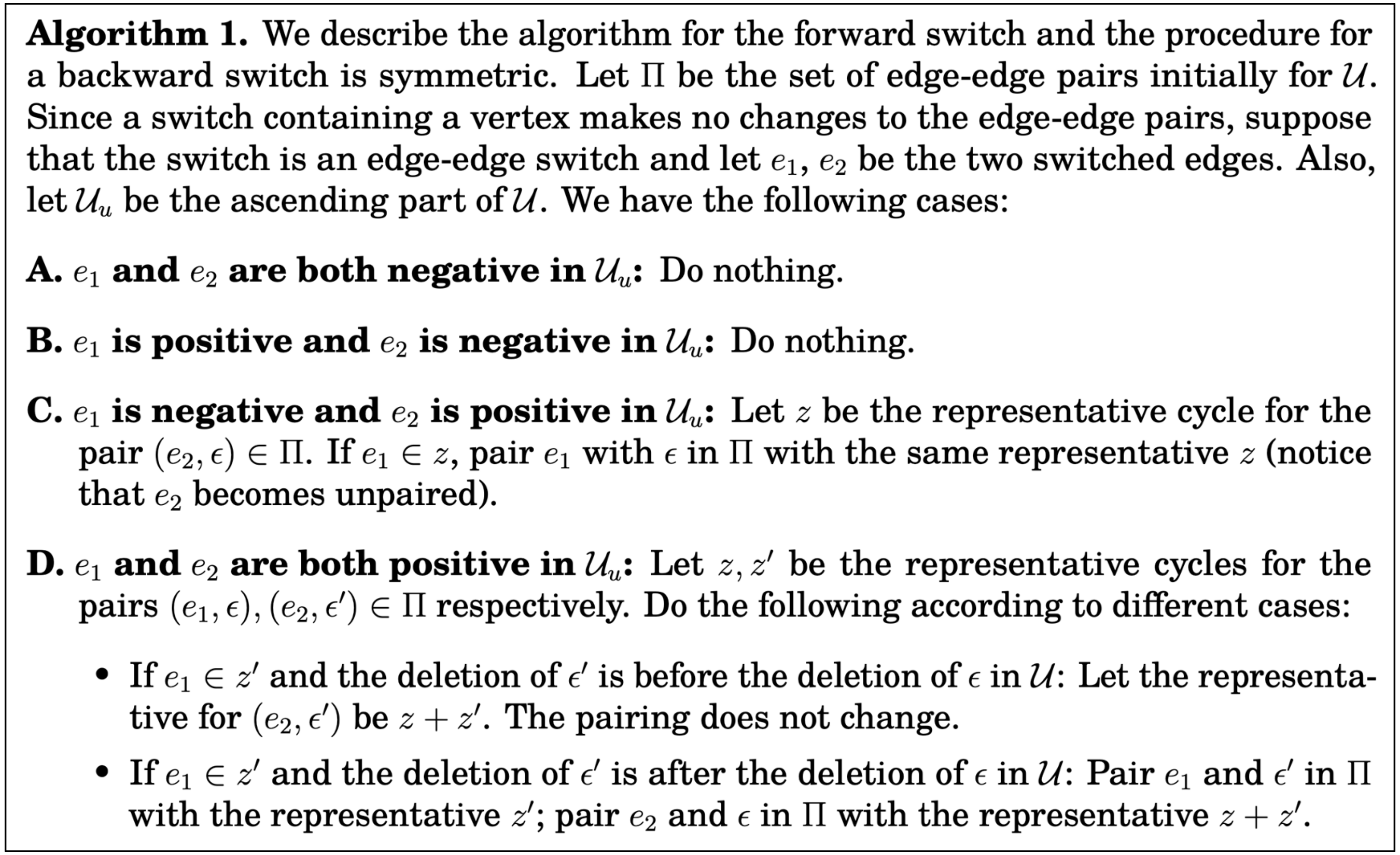
$$\mathcal{F} : G_0 \leftrightarrow \cdots \leftrightarrow G_{i-1} \xhookrightarrow{\sigma} G_i \xhookleftarrow{\tau} G_{i+1} \leftrightarrow \cdots \leftrightarrow G_m$$
$$\mathcal{F}' : G_0 \leftrightarrow \cdots \leftrightarrow G_{i-1} \xhookleftarrow{\tau} G_i' \xhookrightarrow{\sigma} G_{i+1} \leftrightarrow \cdots \leftrightarrow G_m$$

Inward switch

- Strategy: Convert the zigzag filtrations to up-down filtrations as previous
- Immediately, inward and outward switches take $O(1)$ time
- Forward and backward switches: For intervals other than those from the edge-edge pairs, the update reduces to the standard persistence case, hence $O(\log m)$ time

$O(m)$ algorithm for updating edge-edge pairs

- Based on a direct maintenance of representative cycles for pairs



Algorithm 1. We describe the algorithm for the forward switch and the procedure for a backward switch is symmetric. Let Π be the set of edge-edge pairs initially for $\mathcal{U}$. Since a switch containing a vertex makes no changes to the edge-edge pairs, suppose that the switch is an edge-edge switch and let e_1, e_2 be the two switched edges. Also, let $\mathcal{U}_u$ be the ascending part of $\mathcal{U}$. We have the following cases:

A. e_1 and e_2 are both negative in $\mathcal{U}_u$: Do nothing.

B. e_1 is positive and e_2 is negative in $\mathcal{U}_u$: Do nothing.

C. e_1 is negative and e_2 is positive in $\mathcal{U}_u$: Let z be the representative cycle for the pair $(e_2, \epsilon) \in \Pi$. If $e_1 \in z$, pair e_1 with ϵ in Π with the same representative z (notice that e_2 becomes unpaired).

D. e_1 and e_2 are both positive in $\mathcal{U}_u$: Let z, z' be the representative cycles for the pairs $(e_1, \epsilon), (e_2, \epsilon') \in \Pi$ respectively. Do the following according to different cases:

- If $e_1 \in z'$ and the deletion of ϵ' is before the deletion of ϵ in $\mathcal{U}$: Let the representative for (e_2, ϵ') be $z + z'$. The pairing does not change.
- If $e_1 \in z'$ and the deletion of ϵ' is after the deletion of ϵ in $\mathcal{U}$: Pair e_1 and ϵ' in Π with the representative z'; pair e_2 and ϵ in Π with the representative $z + z'$.

$O(\sqrt{m}\log m)$ algorithm: ideas

- Eliminate the explicit maintenance of representative cycles by observing:
  - We only need to check the connectivity of two vertices in the intersection of two graphs in the up-down
  - One graph is from the ascending part, the other is from the descending part
- Maintain the MSF's for $\sqrt{m}$ graphs in the ascending part where the edges are weighted by indices in the descending part.
- Each MSF is a Link-Cut tree

Computing zigzag representatives in O(m^2n) time

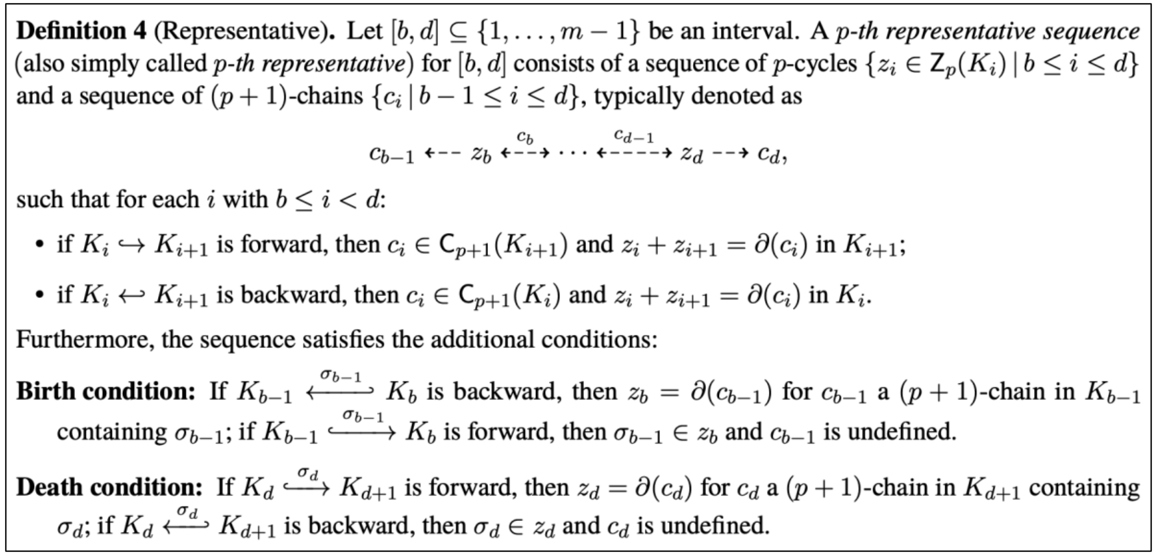
[Dey-H-Morozov] A fast Algorithm for computing zigzag representatives. SODA25 (to appear)

Computing representatives for persistence

- Standard persistence:
    - $O(m^3)$ or $O(m^\omega)$ time by simply using the original persistence algorithm

Computing representatives for persistence

- Standard persistence:
  - $O(m^3)$ or $O(m^\omega)$ time by simply using the original persistence algorithm
- Zigzag persistence:



Definition 4 (Representative)**.** Let $[b, d] \subseteq \{1, \ldots, m-1\}$ be an interval. A *p-th representative sequence* (also simply called *p-th representative*) for $[b, d]$ consists of a sequence of p-cycles $\{z_i \in \mathsf{Z}_p(K_i) \mid b \leq i \leq d\}$ and a sequence of $(p+1)$-chains $\{c_i \mid b-1 \leq i \leq d\}$, typically denoted as

$$c_{b-1} \dashleftarrow z_b \overset{c_b}{\dashleftarrow\dashrightarrow} \cdots \overset{c_{d-1}}{\dashleftarrow\dashrightarrow} z_d \dashrightarrow c_d,$$

such that for each i with $b \leq i < d$:

- if $K_i \hookrightarrow K_{i+1}$ is forward, then $c_i \in \mathsf{C}_{p+1}(K_{i+1})$ and $z_i + z_{i+1} = \partial(c_i)$ in K_{i+1};
- if $K_i \hookleftarrow K_{i+1}$ is backward, then $c_i \in \mathsf{C}_{p+1}(K_i)$ and $z_i + z_{i+1} = \partial(c_i)$ in K_i.

Furthermore, the sequence satisfies the additional conditions:

Birth condition: If $K_{b-1} \xhookleftarrow{\sigma_{b-1}} K_b$ is backward, then $z_b = \partial(c_{b-1})$ for c_{b-1} a $(p+1)$-chain in K_{b-1} containing σ_{b-1}; if $K_{b-1} \xhookrightarrow{\sigma_{b-1}} K_b$ is forward, then $\sigma_{b-1} \in z_b$ and c_{b-1} is undefined.

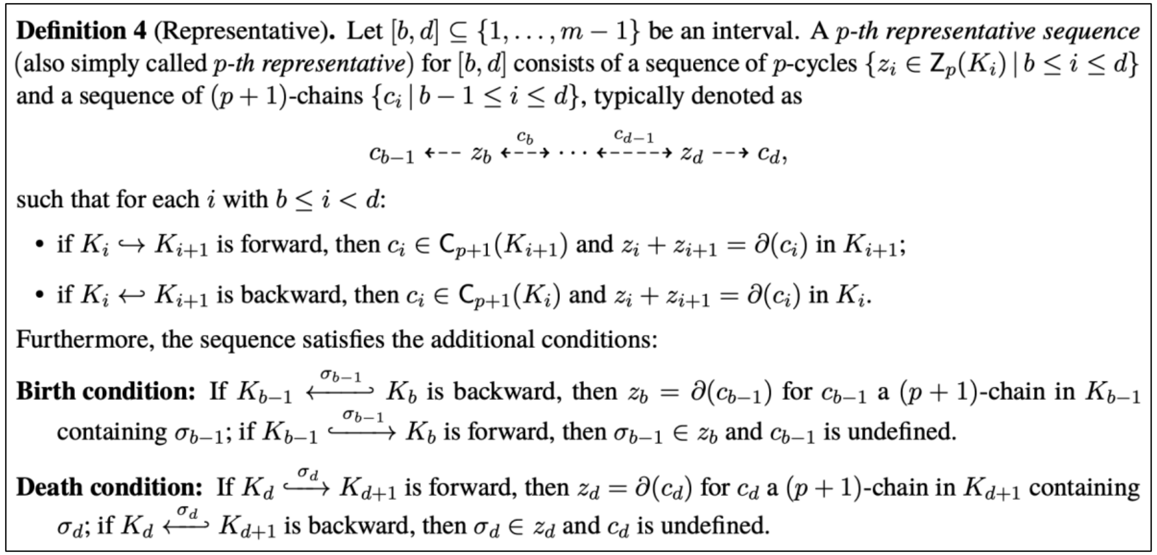
Death condition: If $K_d \xhookrightarrow{\sigma_d} K_{d+1}$ is forward, then $z_d = \partial(c_d)$ for c_d a $(p+1)$-chain in K_{d+1} containing σ_d; if $K_d \xhookleftarrow{\sigma_d} K_{d+1}$ is backward, then $\sigma_d \in z_d$ and c_d is undefined.

  - $O(m^2 n^2)$ time by directly adapt the algorithm in [MO15]

Maria and Oudot. Zigzag persistence via reflections and transpositions. 2015

Computing representatives for persistence

- Standard persistence:
    - $O(m^3)$ or $O(m^\omega)$ time by simply using the original persistence algorithm
- Zigzag persistence:



Definition 4 (Representative)**.** Let $[b,d] \subseteq \{1,\dots,m-1\}$ be an interval. A *p-th representative sequence* (also simply called *p-th representative*) for $[b,d]$ consists of a sequence of p-cycles $\{z_i \in \mathsf{Z}_p(K_i) \mid b \le i \le d\}$ and a sequence of $(p+1)$-chains $\{c_i \mid b-1 \le i \le d\}$, typically denoted as

$$c_{b-1} \dashleftarrow z_b \overset{c_b}{\dashleftarrow\!\dashrightarrow} \cdots \overset{c_{d-1}}{\dashleftarrow\!\dashrightarrow} z_d \dashrightarrow c_d,$$

such that for each i with $b \le i < d$:

- if $K_i \hookrightarrow K_{i+1}$ is forward, then $c_i \in \mathsf{C}_{p+1}(K_{i+1})$ and $z_i + z_{i+1} = \partial(c_i)$ in K_{i+1};
- if $K_i \hookleftarrow K_{i+1}$ is backward, then $c_i \in \mathsf{C}_{p+1}(K_i)$ and $z_i + z_{i+1} = \partial(c_i)$ in K_i.

Furthermore, the sequence satisfies the additional conditions:

Birth condition: If $K_{b-1} \xhookleftarrow{\sigma_{b-1}} K_b$ is backward, then $z_b = \partial(c_{b-1})$ for c_{b-1} a $(p+1)$-chain in K_{b-1} containing σ_{b-1}; if $K_{b-1} \xhookrightarrow{\sigma_{b-1}} K_b$ is forward, then $\sigma_{b-1} \in z_b$ and c_{b-1} is undefined.

Death condition: If $K_d \xhookrightarrow{\sigma_d} K_{d+1}$ is forward, then $z_d = \partial(c_d)$ for c_d a $(p+1)$-chain in K_{d+1} containing σ_d; if $K_d \xhookleftarrow{\sigma_d} K_{d+1}$ is backward, then $\sigma_d \in z_d$ and c_d is undefined.

-
    - $O(m^2n^2)$ time by directly adapt the algorithm in [MO15]
- Find a way to compress the representatives to achieve the $O(m^2n)$ complexity

Maria and Oudot. Zigzag persistence via reflections and transpositions. 2015

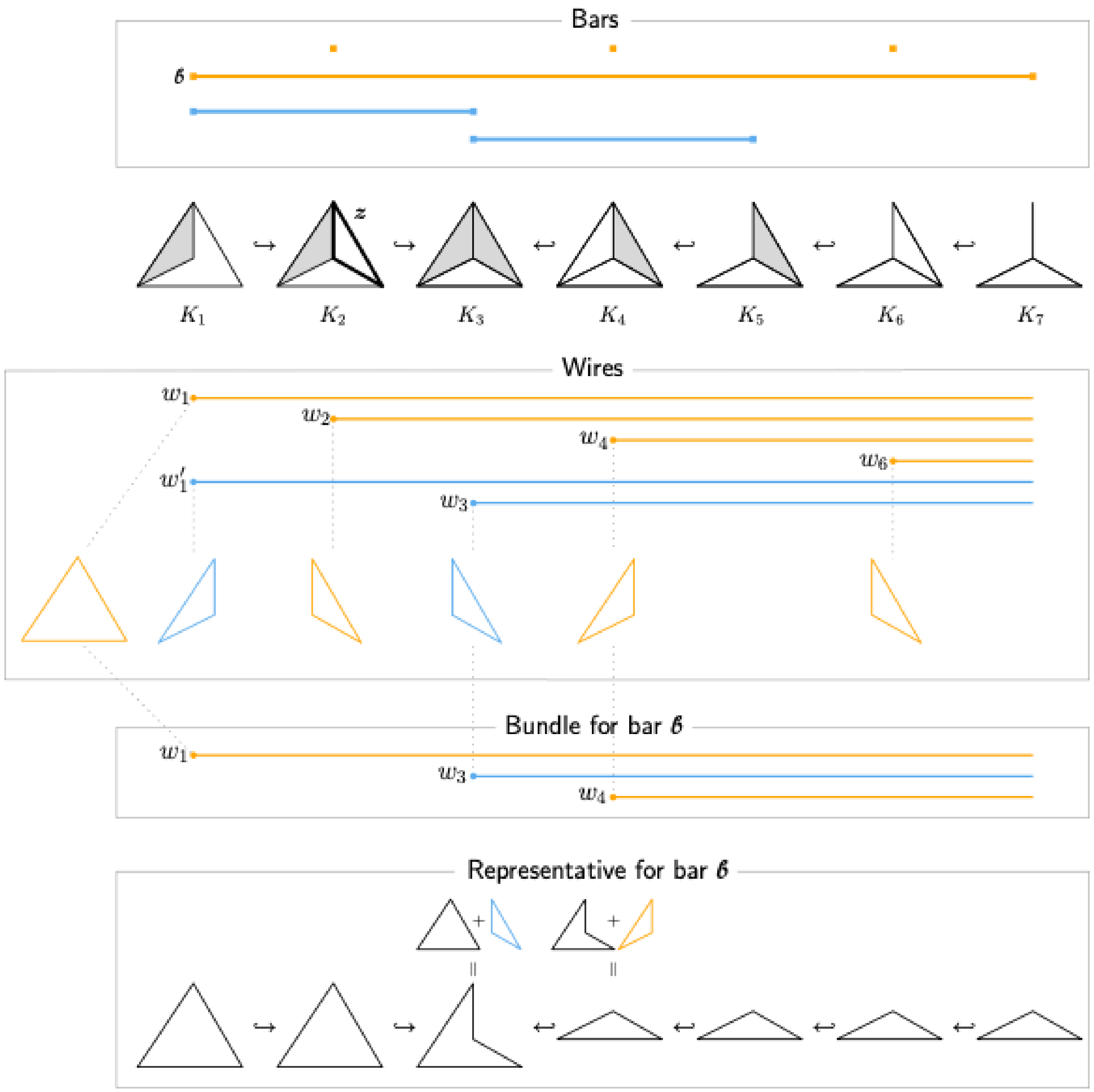
Key to bringing down the complexity

- How to store a representative for an interval in memory:
    - The straightforward method takes $O(mn)$ space, so that summing two representatives takes $O(mn)$ time, and hence the $O(m^2n^2)$ complexity

Key to bringing down the complexity

- How to store a representative for an interval in memory:
    - The straightforward method takes $O(mn)$ space, so that summing two representatives takes $O(mn)$ time, and hence the $O(m^2n^2)$ complexity
- We find a *compressed* way to store a representative
    - The compressed method takes $O(m)$ space, so that summing two representatives takes $O(m)$ time, and hence the $O(m^2n)$ complexity
    - This is by storing a representatives as a set of *wires*, each a cycle born at a certain time and extending indefinitely

An example for storing a rep. as wires



To Answer the Question in the Title

Can zigzag persistence be computed as efficiently as the standard version?

Problems		**Wall-clock time**	**Complexity**
Compute persistence	General	Yes!	
	Graph	Not far away	$O(m\,\alpha(m))$ vs $O(m \log m)$
Update	General	?	Yes
	Graph	No	$O(\log m)$ vs $O(\sqrt{m} \log m)$
Compute representatives		Still a gap	$O(m^{\omega})$ vs $O(m^2 n)$

Thank you!