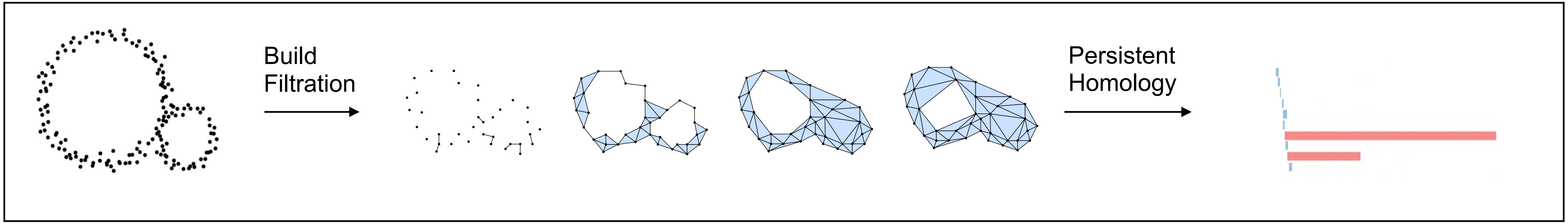
Revisiting Graph Persistence for Updates and Efficiency

Tamal K. Dey, Purdue U.
Tao Hou, DePaul U.
Salman Parsa, DePaul U.

WADS 2023

Summary

- Our focus: Persistent homology on **graphs** (i.e., graph persistence)
    - *Persistent homology is a major tool in TDA*
- We consider **both standard and zigzag** persistence
    - *Zigzag persistence is an extension of the standard version by incorporating* deletions
- We propose efficient algorithms for graph persistence with a special focus on **update**
    - *Update means the* transposition *operation proposed in the* vineyard *paper [CEM06]*



Existing results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^\omega)$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^\omega)$ / $O(mn^2)$	$O(m\log^4 n)$

Existing results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^{\omega})$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^{\omega})$ / $O(mn^2)$	$O(m \log^4 n)$

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

Existing results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^\omega)$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^\omega)$ / $O(mn^2)$	$O(m\log^4 n)$

Tamal K. Dey and Tao Hou. Updating barcodes and representatives for zigzag persistence. 2022.

Existing results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^{\omega})$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^{\omega})$ / $O(mn^2)$	$O(m\log^4 n)$

Nikola Milosavljevi ́ c, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. 2011.

Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. 2000.

Clement Maria and Steve Y. Oudot. Zigzag persistence via reflections and transpositions. 2014.

Tamal K. Dey and Tao Hou. Fast computation of zigzag persistence. 2022

... (and **many many more**!)

Existing results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^\omega)$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^\omega)$ / $O(mn^2)$	$O(m \log^4 n)$

By using Union-Find; $\alpha(m)$: Inverse Ackermann function

Existing results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^\omega)$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^\omega)$ / $O(mn^2)$	$O(m \log^4 n)$

Tamal K. Dey and Tao Hou. Computing zigzag persistence on graphs in near-linear time. 2021

New results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^{\omega})$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^{\omega})$ / $O(mn^2)$	$O(m\log^4 n)$

New results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m) \Longrightarrow O(\log m)$
	Zigzag	$O(m)$	$O(m)$
Comp. from Scratch	Standard	$O(m^\omega)$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^\omega)$ / $O(mn^2)$	$O(m \log^4 n)$

New results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m) \Longrightarrow O(\log m)$
	Zigzag	$O(m)$	$O(m) \Longrightarrow O(\sqrt{m}\log m)$
Comp. from Scratch	Standard	$O(m^\omega)$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^\omega)$ / $O(mn^2)$	$O(m\log^4 n)$

New results

m : length of the filtration, or number of additions/deletions

		General	Graph
Update	Standard	$O(m)$	$O(m) \Longrightarrow O(\log m)$
	Zigzag	$O(m)$	$O(m) \Longrightarrow O(\sqrt{m}\log m)$
Comp. from Scratch	Standard	$O(m^\omega)$ / $O(m^3)$	$O(m\,\alpha(m))$
	Zigzag	$O(m^\omega)$ / $O(mn^2)$	$O(m\log^4 n) \Longrightarrow O(m\log m)$

Assume $n \in \Omega(m^\epsilon)$ for arbitrarily small $\epsilon > 0$

New results

Detailed complexity for update on zigzag persistence of graphs

- Closed-closed intervals in dim 0: $O(1)$
- Closed-open and open-closed intervals: $O(\log m)$
- Open-open in dim 0 and closed-closed in dim 1: $O(\sqrt{m}\log m)$

Updating standard persistence on graphs in $O(\log m)$ time

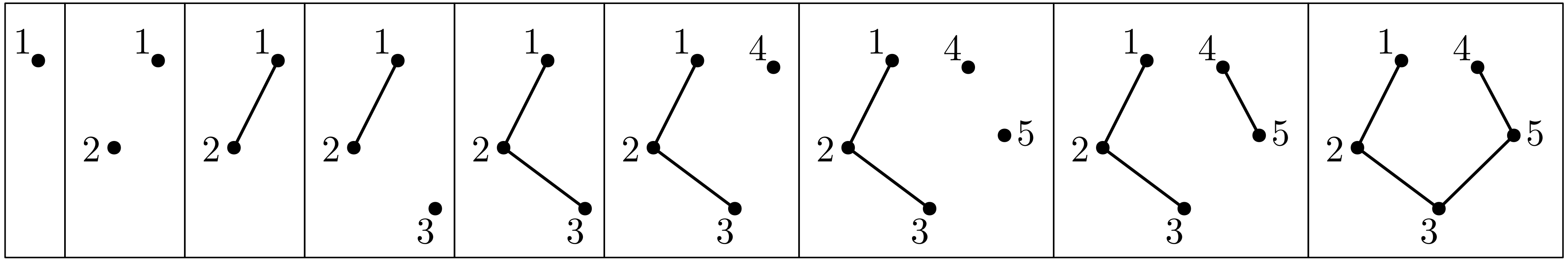
Transposition (switch) operation

$$\mathcal{F} : \varnothing = G_0 \hookrightarrow \cdots \hookrightarrow G_{i-1} \xhookrightarrow{\sigma} G_i \xhookrightarrow{\tau} G_{i+1} \hookrightarrow \cdots \hookrightarrow G_m$$

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

Merge forest (tree)

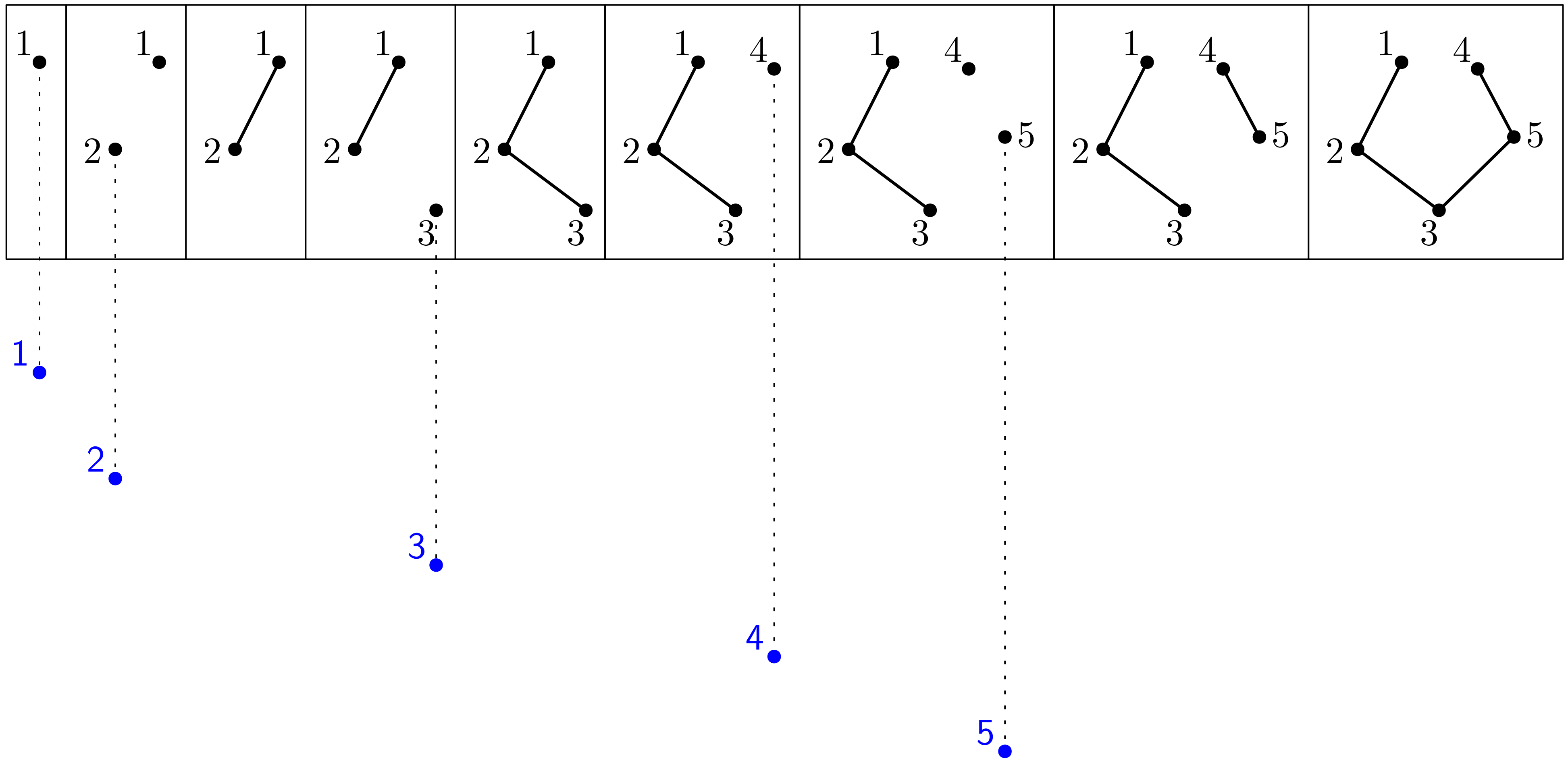
$\mathcal{F}$:



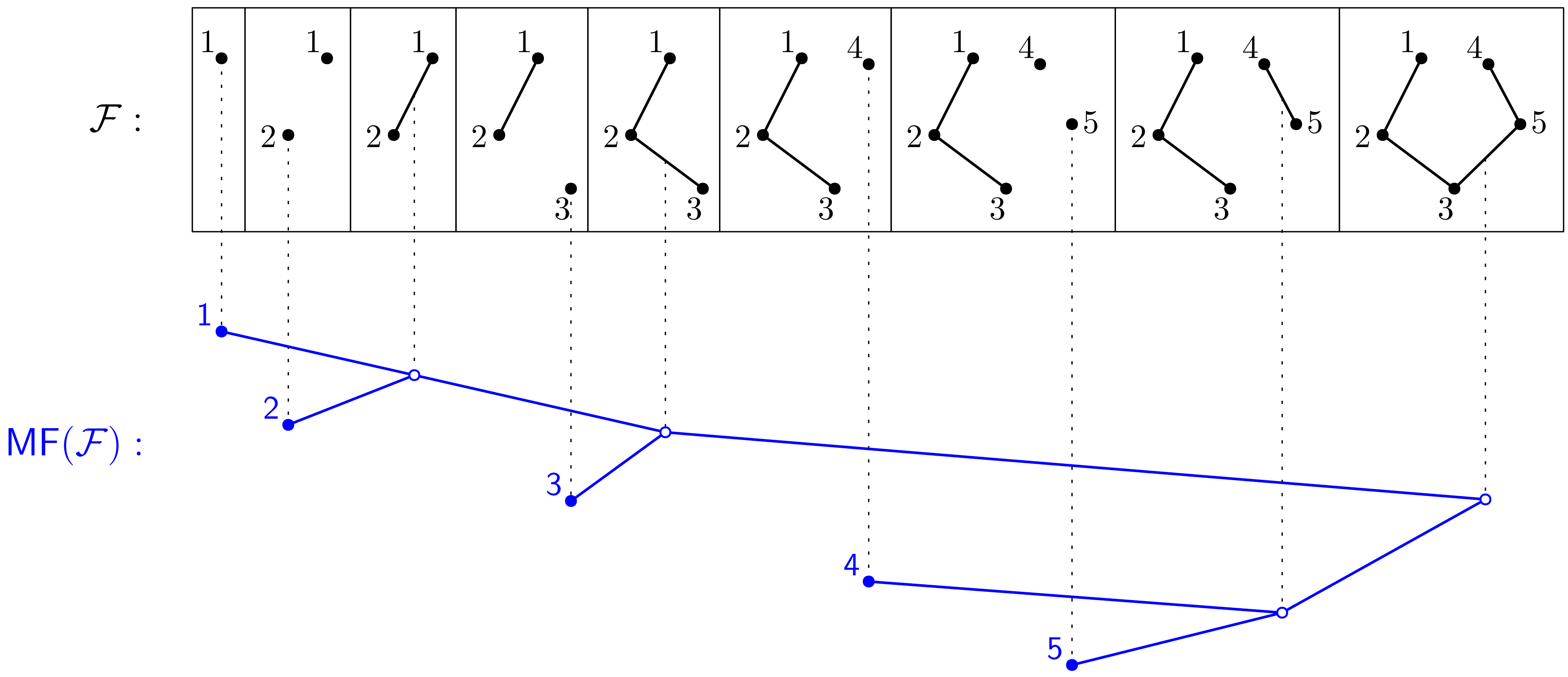
Merge forest (tree)

$\mathcal{F}$:

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

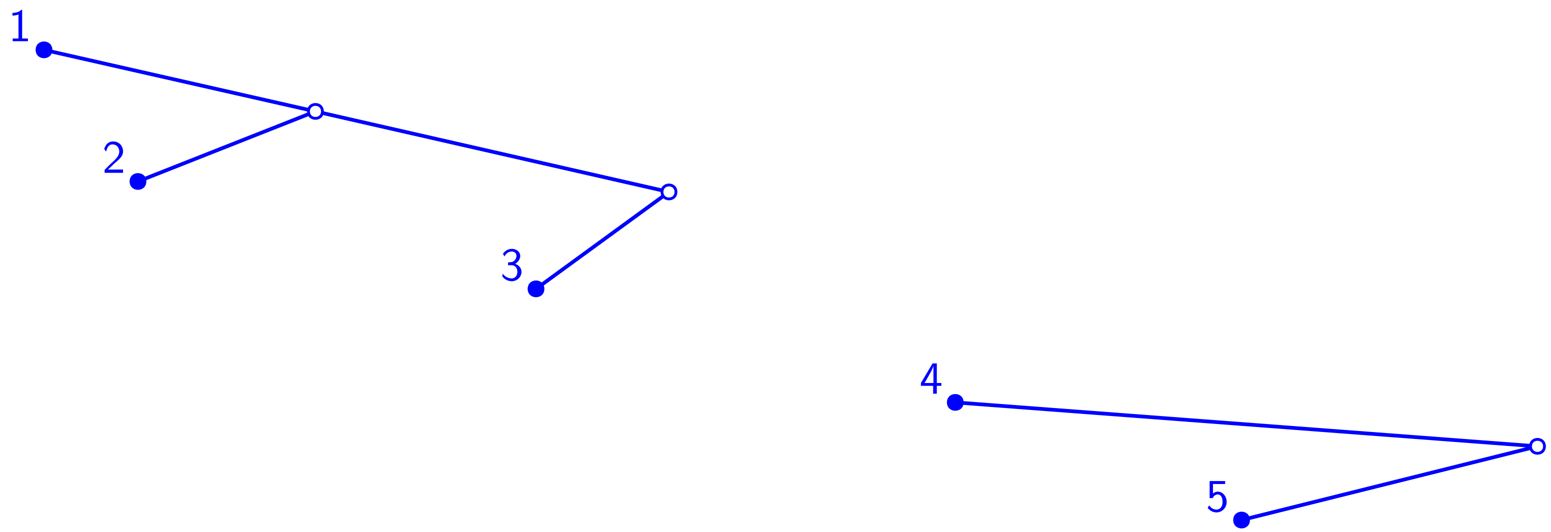


Merge forest (tree)



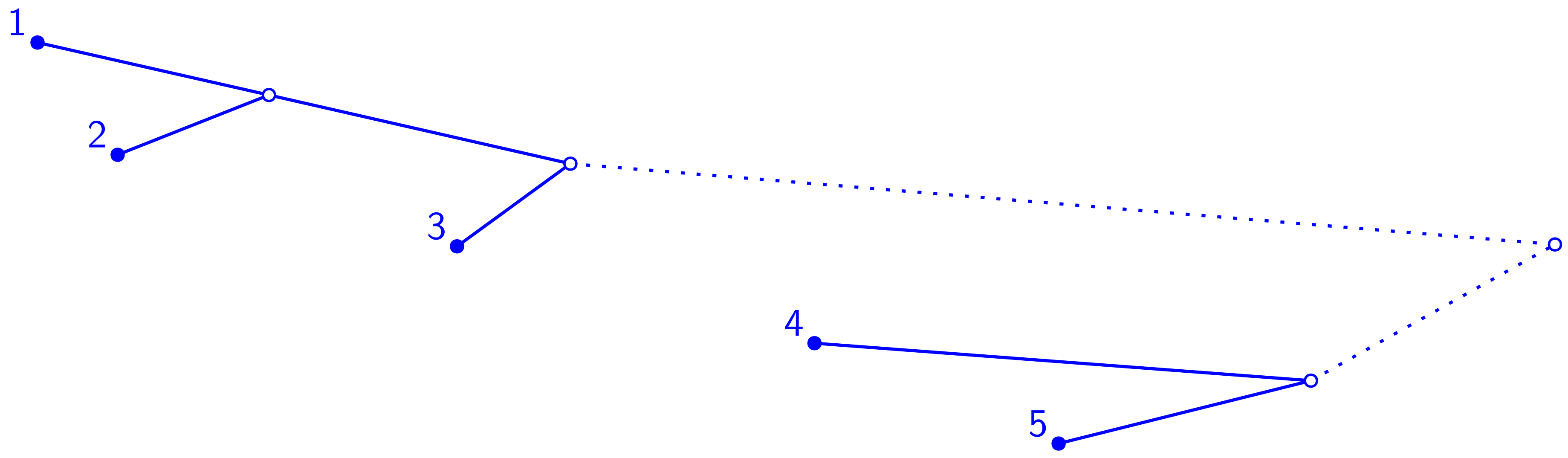
Merge forest (tree)

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



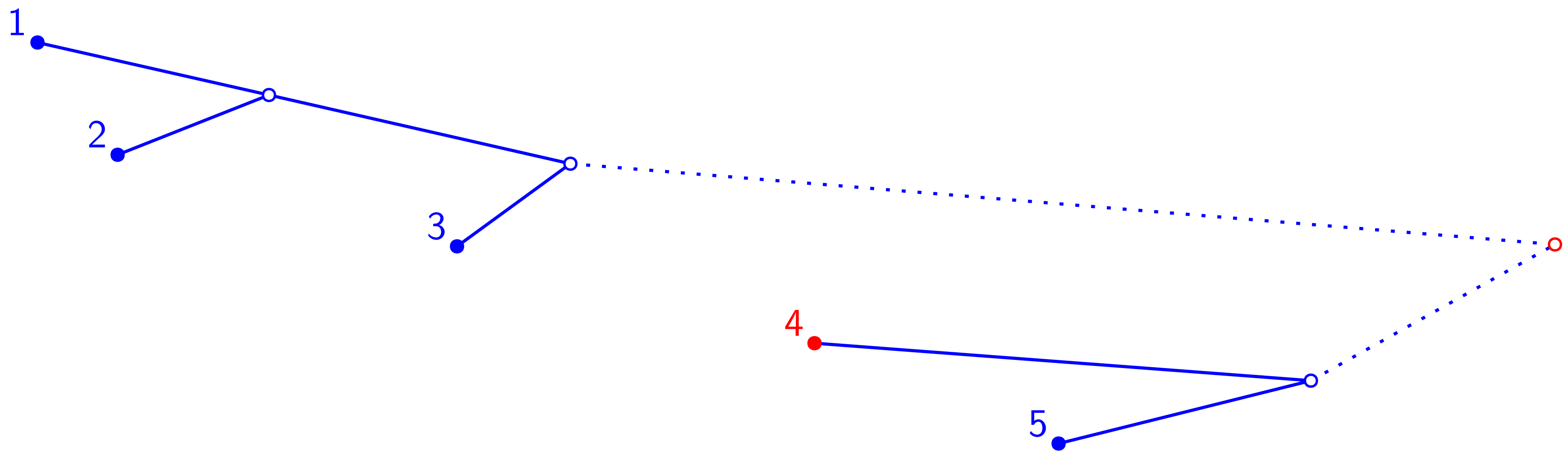
Merge forest (tree)

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



Merge forest (tree)

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



More definitions

Two types of edges in the graph filtration:

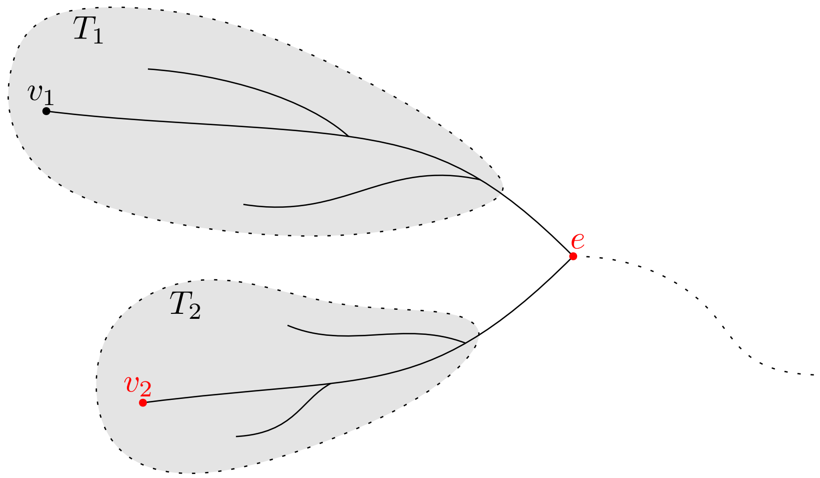
- Negative edge: connect two different connected components
- Positive edge: connect the same connected component

Focus on certain cases

- There are different cases for the update, and the data structures for some cases do not change.
- We will focus on those cases where the merge forest of the paring do change.

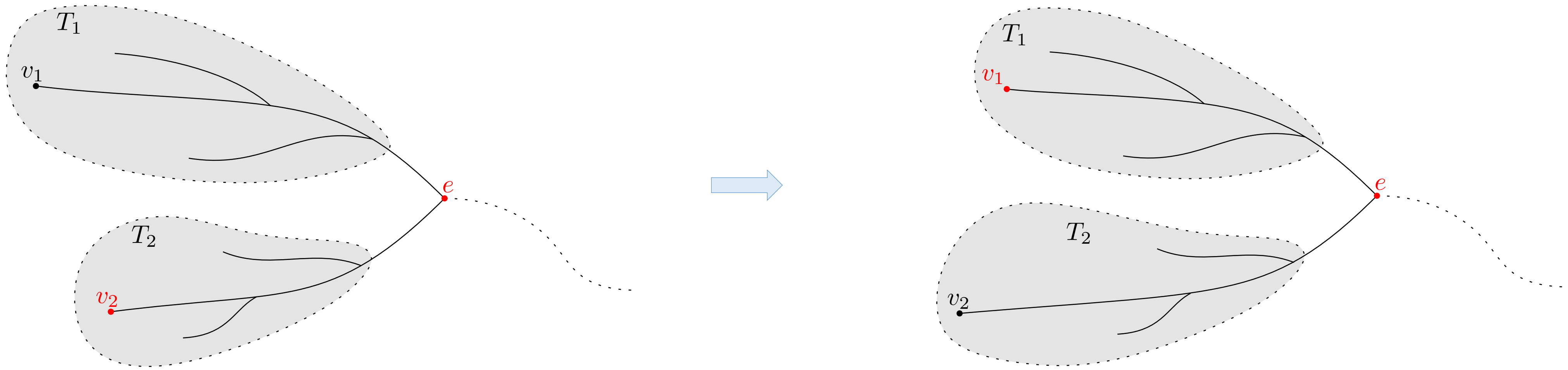
1. Switch two vertices v_1, v_2

- 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



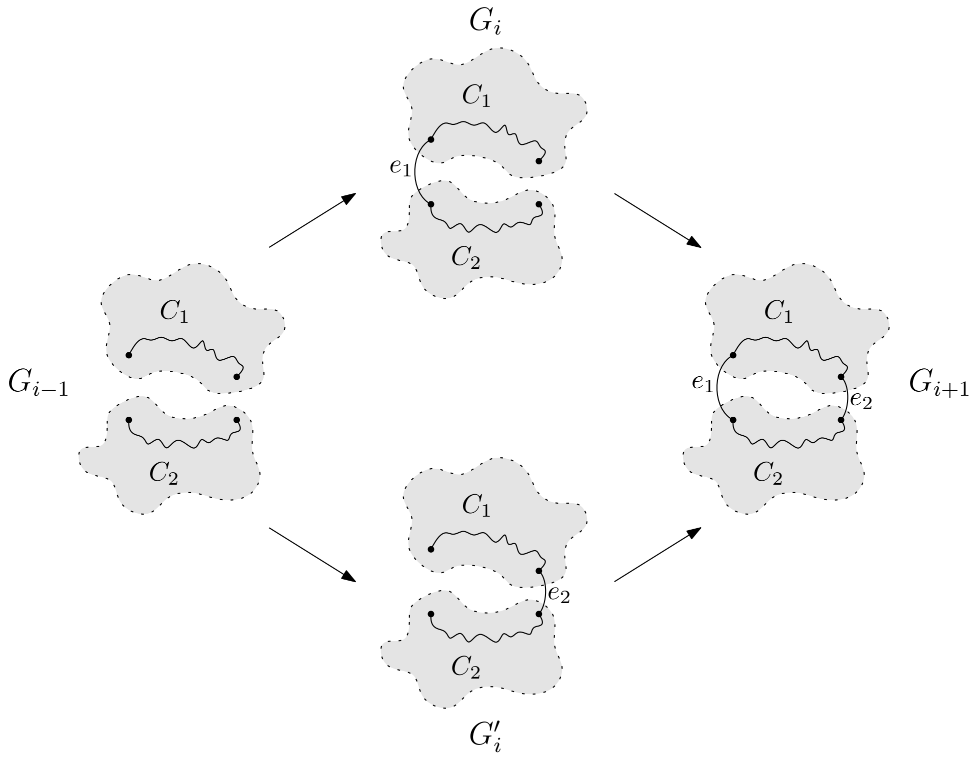
1. Switch two vertices v_1, v_2

- 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



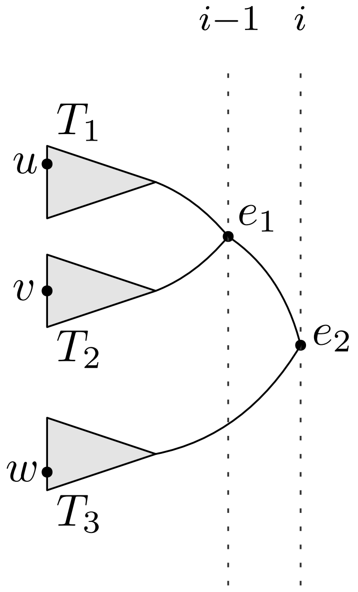
2. Switch a negative edge e_1 and a positive edge e_2

- 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



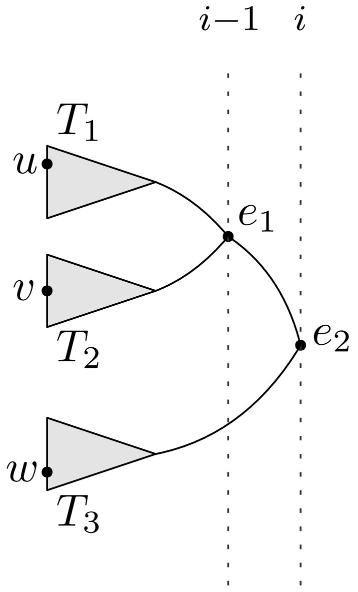
3. Switch two negative edges e_1, e_2

- 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



3. Switch two negative edges e_1, e_2

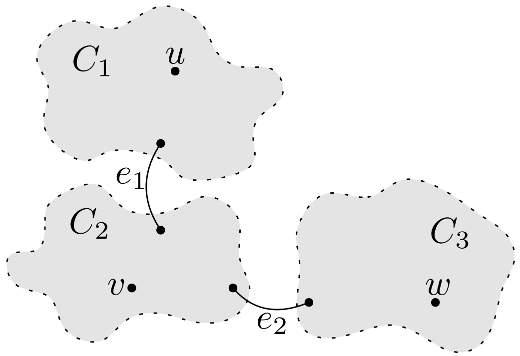
- Only need to make changes when the corresponding node of e_1 is a child of the corresponding node of e_2 in the merge forest
- Let u, v, w be the lowest leaves in T_1, T_2, T_3.
- WLOG, assume v is lower than u.



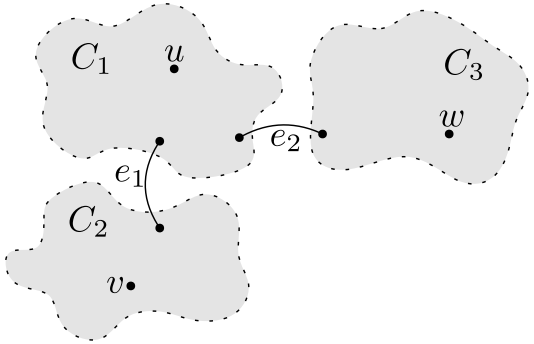
3. Switch two negative edges e_1, e_2

- 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)

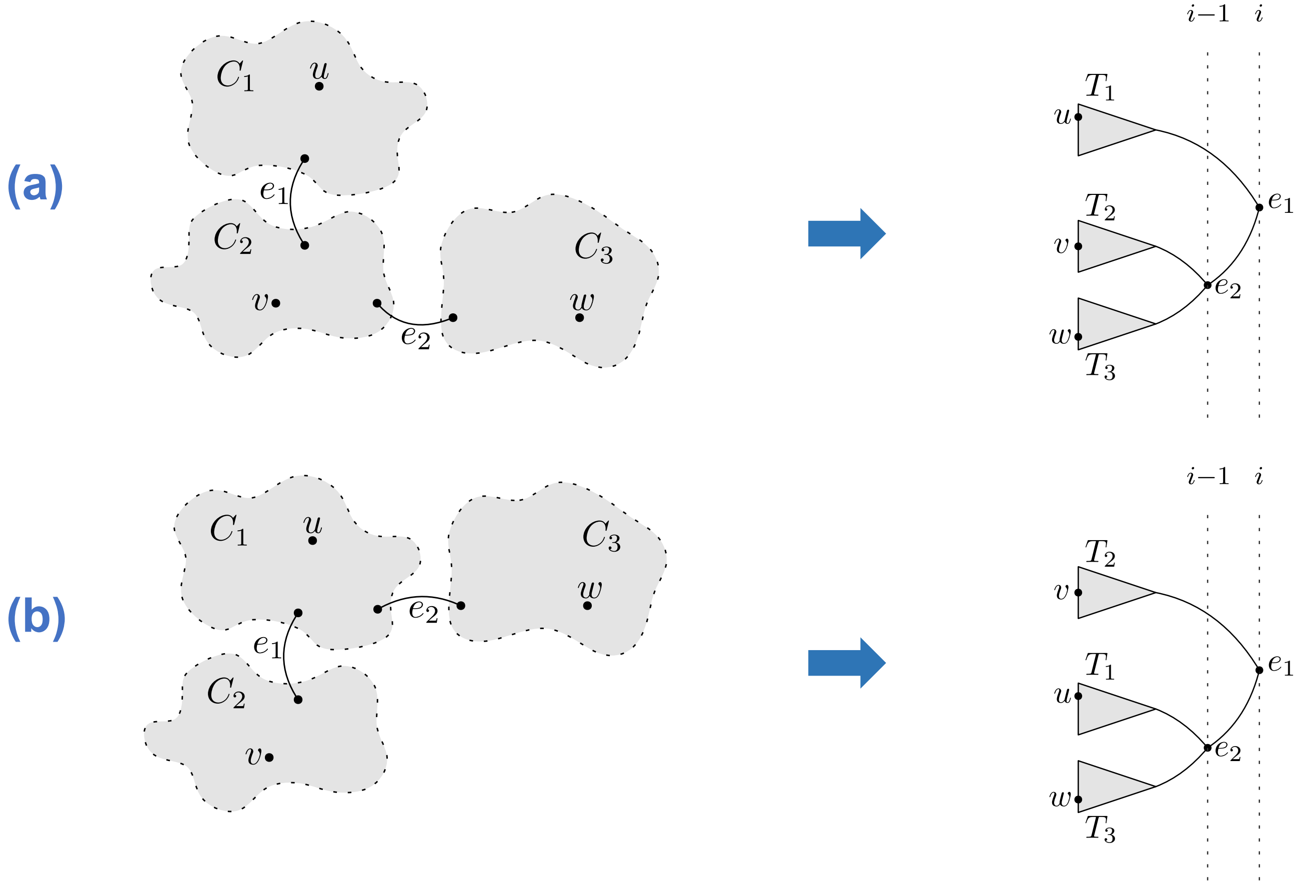


(b)



3. Switch two negative edges e_1, e_2

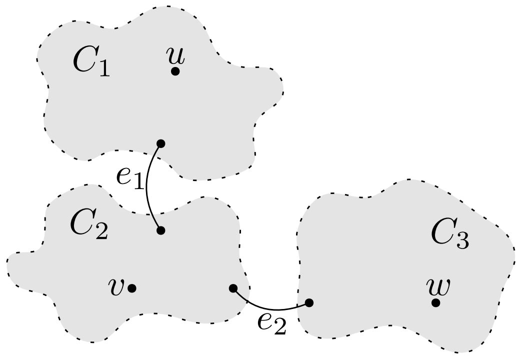
- Based on the structure of the merge forest, there are two connecting configurations for C_1, C_2, C_3 in G_{i-1} (C_1, C_2, C_3 are the connected components containing u, v, w respectively)



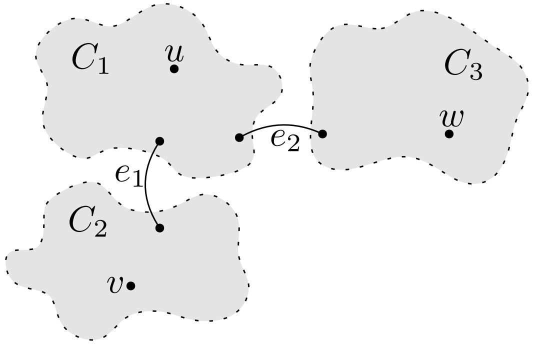
3. Switch two negative edges e_1, e_2

- 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]:

- $\textsc{root}(v)$: Returns the root of the tree containing node v.
- $\textsc{cut}(v)$: Deletes the edge connecting node v to its parent.
- $\textsc{link}(u, v)$: Makes the root of the tree containing node v be a child of node u.
- $\textsc{nca}(u, v)$: Returns the nearest common ancestor of two nodes u, v in the same tree.
- $\textsc{change-val}(v, x)$: Assigns the value associated to a leaf v to be x.
- $\textsc{subtree-min}(v)$: Returns the leaf with the minimum associated value in the subtree rooted at v.

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

DFT-Tree

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

$O(\log m)$

- $\textsc{root}(v)$: Returns the root of the tree containing node v.
- $\textsc{cut}(v)$: Deletes the edge connecting node v to its parent.
- $\textsc{link}(u, v)$: Makes the root of the tree containing node v be a child of node u.
- $\textsc{nca}(u, v)$: Returns the nearest common ancestor of two nodes u, v in the same tree.
- $\textsc{change-val}(v, x)$: Assigns the value associated to a leaf v to be x.
- $\textsc{subtree-min}(v)$: 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}

- 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.

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

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

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

- 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.

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 $\mathcal{F}'$

- 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.

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)

- 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.

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.

Updating zigzag persistence on graphs

Four witches on zigzag filtrations

- Switch two consecutive simplex-wise inclusions (additions or deletions)
- Four 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

Converting to up-down filtrations

- Our strategy: Convert the zigzag filtrations to up-down filtrations as in [DH22]
  - The first half is only additions, and the second is only deletions
  - Barcodes of the two filtrations can be easily converted for constant time per bar

Tamal K. Dey and Tao Hou. Fast Computation of Zigzag Persistence. 2022.

Converting to up-down filtrations

- Our strategy: Convert the zigzag filtrations to up-down filtrations as in [DH22]
  - The first half is only additions, and the second is only deletions
  - Barcodes of the two filtrations can be easily converted for constant time per bar
- Immediately, inward and outward switches take $O(1)$ time

Tamal K. Dey and Tao Hou. Fast Computation of Zigzag Persistence. 2022.

Converting to up-down filtrations

- Our strategy: Convert the zigzag filtrations to up-down filtrations as in [DH22]
  - The first half is only additions, and the second is only deletions
  - Barcodes of the two filtrations can be easily converted for constant time per bar
- 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

Tamal K. Dey and Tao Hou. Fast Computation of Zigzag Persistence. 2022.

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

- Based on a direct maintenance of representative cycles for the pairs
- Similar to the vineyard algorithm [CEM06]

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: the update only need to check the connectivity of two vertices in the intersection of two graphs in the up-down filtration, where one graph is from the ascending part and the other is from the descending part.

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

- Eliminate the explicit maintenance of representative cycles by observing: the update only need to check the connectivity of two vertices in the intersection of two graphs in the up-down filtration, where one graph is from the ascending part and 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 persistence on graphs in $O(m \log m)$ time

Ideas

- Converting to up-down filtration as done previously in $O(m)$ time
- Utilize the pairing algorithm proposed in [YMGTC21]
- Use the Link-Cut tree to perform the pairing

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

Thank you!