Better Decremental and Fully Dynamic Sensitivity Oracles for Subgraph Connectivity

Yaowei Long

University of Michigan

Yunfan Wang

Tsinghua University, IIIS

Problem Definition

- Graph $G = (V, E)$.
- Divide V into activated vertices V_{on} and inactivated V_{off}.

Problem Definition

- Graph $G = (V, E)$.
- Divide V into activated vertices V_{on} and inactivated V_{off}.
- **A single update** -- Change the status of up to d vertices.

Problem Definition

- Graph $G = (V, E)$.
- Divide V into activated vertices V_{on} and inactivated V_{off}.
- **A single update** -- Change the status of up to d vertices.
- Task: Given $D \subseteq V$ with size up to d, make all vertices in D flipped.
- We need to maintain a data structure that can quickly determine the connectivity between any two vertices in the graph induced by the activated vertices.

Problem Definition

- Graph $G = (V, E)$.
- Divide V into activated vertices V_{on} and inactivated V_{off}.
- **A single update** -- Change the status of up to d vertices.
- Task: Given $D \subseteq V$ with size up to d, make all vertices in D flipped.
- We need to maintain a data structure that can quickly determine the connectivity between any two vertices in the graph induced by the activated vertices.

Sensitivity Oracle Model

- There are three phases: Preprocessing, Update, Query
- **Preprocessing:** In this phase, we are given G, V_{on}, V_{off}, and d.

Sensitivity Oracle Model

- There are three phases: Preprocessing, Update, Query
- **Preprocessing:** In this phase, we are given G, V_{on}, V_{off}, and d.
- **Update:** In this phase, we are given $D \subseteq V$.

Sensitivity Oracle Model

- There are three phases: Preprocessing, Update, Query
- **Preprocessing:** In this phase, we are given G, V_{on}, V_{off}, and d.
- **Update:** In this phase, we are given $D \subseteq V$.
- **Query:** In this phase, we will receive subsequent query. Each query will give a pair of vertices u, v in the subgraph induced by the new activated vertices, and ask the connectivity of u and v.

Sensitivity Oracle Model

- There are three phases: Preprocessing, Update, Query
- **Preprocessing:** In this phase, we are given G, V_{on}, V_{off}, and d.
- **Update:** In this phase, we are given $D \subseteq V$.
- **Query:** In this phase, we will receive subsequent query. Each query will give a pair of vertices u, v in the subgraph induced by the new activated vertices, and ask the connectivity of u and v.
- We aim to construct a sensitivity oracles for subgraph connectivity with S space, t_p preprocessing time, t_u update time, and t_q query time upper bounds.

Sensitivity Oracle Model

- There are three phases: Preprocessing, Update, Query
- **Preprocessing:** In this phase, we are given G, V_{on}, V_{off}, and d.
- **Update:** In this phase, we are given $D \subseteq V$.
- **Query:** In this phase, we will receive subsequent query. Each query will give a pair of vertices u, v in the subgraph induced by the new activated vertices, and ask the connectivity of u and v.
- We aim to construct a sensitivity oracles for subgraph connectivity with S space, t_p preprocessing time, t_u update time, and t_q query time upper bounds.
- **Hardness:** We require that t_u and t_q can only rely on d.

Sensitivity Oracle Model

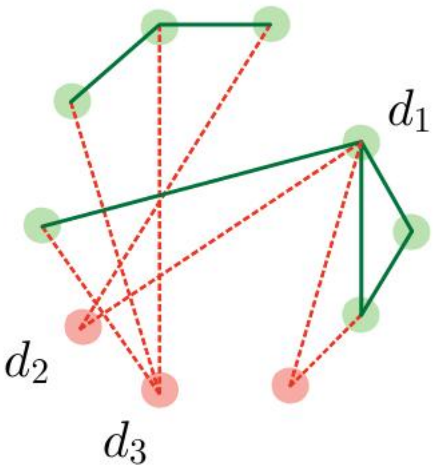
- There are three phases: Preprocessing, Update, Query
- **Preprocessing:** In this phase, we are given G, V_{on}, V_{off}, and d.
- **Update:** In this phase, we are given $D \subseteq V$.
- **Query:** In this phase, we will receive subsequent query. Each query will give a pair of vertices u, v in the subgraph induced by the new activated vertices, and ask the connectivity of u and v.
- We aim to construct a sensitivity oracles for subgraph connectivity with S space, t_p preprocessing time, t_u update time, and t_q query time upper bounds.
- **Hardness:** We require that t_u and t_q can only rely on d.

Decremental v.s. Fully Dynamic

- **Decremental Setting (Vertex-Failure)**
- There are no inactivated vertices initially, i.e., $V_{off} = \emptyset$.

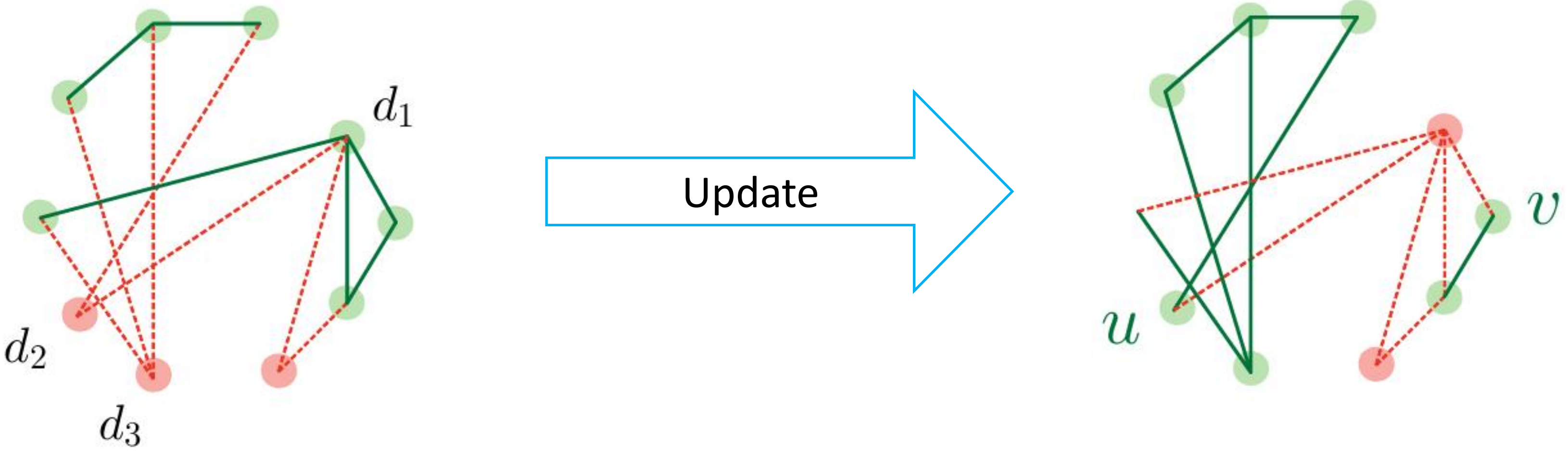
- **Fully Dynamic Setting**
- No constraints

Example



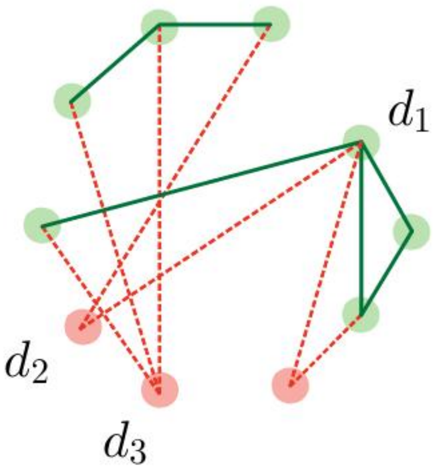
Red Vertices: V_{off}
Green Vertices: V_{on}

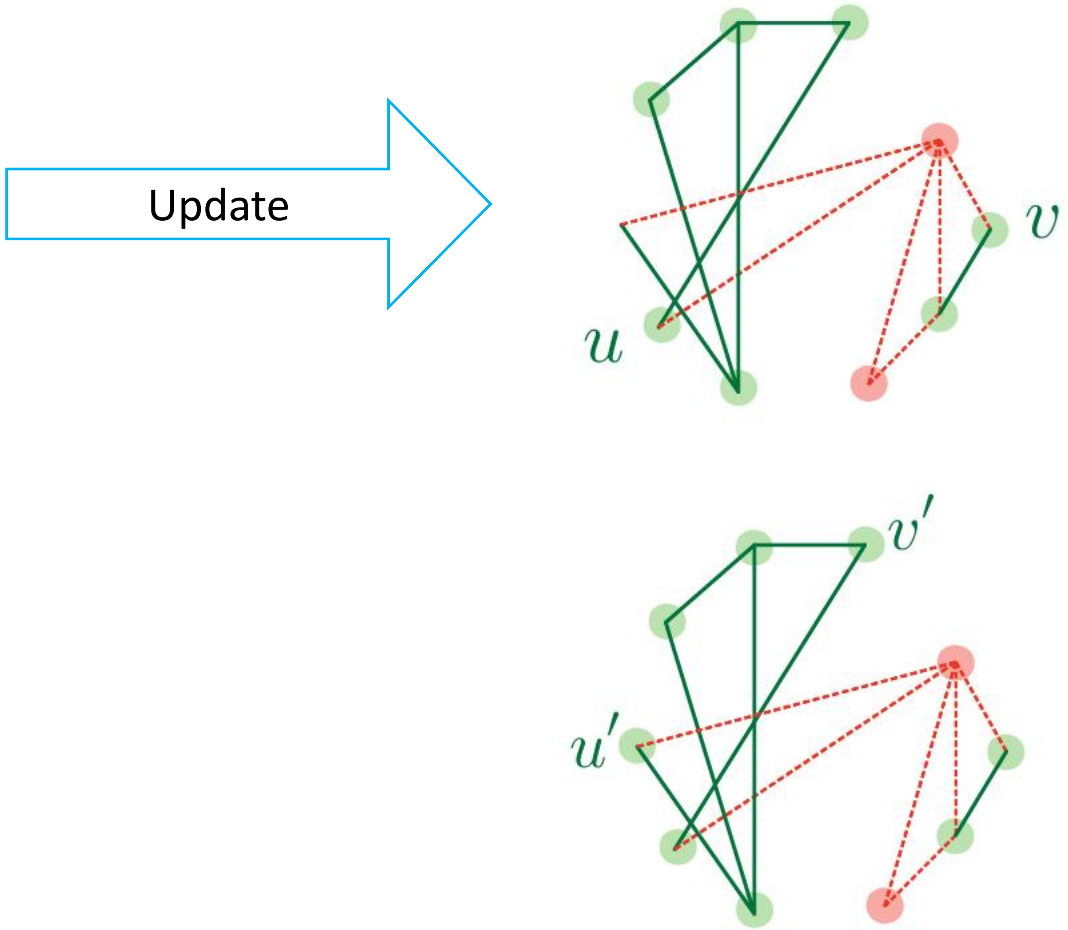
Example



Red Vertices: V_{off}
Green Vertices: V_{on}

Example





Red Vertices: V_{off}
Green Vertices: V_{on}

History (Decremental)

	Det./ Rand.	Space	Preprocessing	Update	Query
Block trees, SQRT trees, and [KTDBC91] only when $d_\star \leq 3$	Det.	$O(n)$	$\widetilde{O}(m)$	$O(1)$	$O(1)$
Duan & Pettie [DP10] for $c \geq 1$	Det.	linear in preprocessing time	$\widetilde{O}(md_\star^{1-\frac{2}{c}} n^{\frac{1}{c}-\frac{1}{c\log(2d_\star)}})$	$\widetilde{O}(d^{2c+4})$	$O(d)$
Duan & Pettie [DP20]	Det.	$O(md_\star \log n)$	$O(mn \log n)$	$O(d^3 \log^3 n)$	$O(d)$
	Rand.	$O(m \log^6 n)$	$O(mn \log n)$	$\bar{O}(d^2 \log^3 n)$ w.h.p.	$O(d)$
Brand & Saranurak [vdBS19]	Rand.	$O(n^2)$	$O(n^\omega)$	$O(d^\omega)$	$O(d^2)$
Pilipczuk et al. [PSS+22]	Det.	$m2^{2^{O(d_\star)}}$	$mn^2 2^{2^{O(d_\star)}}$	$2^{2^{O(d_\star)}}$	$2^{2^{O(d_\star)}}$
	Det.	$n^2\mathrm{poly}(d_\star)$	$\mathrm{poly}(n)2^{O(d_\star \log d_\star)}$	$\mathrm{poly}(d_\star)$	$\mathrm{poly}(d_\star)$
Long & Saranurak [LS22]	Det.	$O(m \log^3 n)$	$O(mn \log n)$	$\bar{O}(d^2 \log^3 n \log^4 d)$	$O(d)$
	Det.	$O(m \log^* n)$	$\widehat{O}(m) + \widetilde{O}(d_\star m)$	$\widehat{O}(d^2)$	$O(d)$
Kosinas [Kos23]	Det.	$O(d_\star m \log n)$	$O(d_\star m \log n)$	$O(d^4 \log n)$	$O(d)$
This paper	Det.	$O(m \log^3 n)$	$\widehat{O}(m) + O(d_\star m \log^3 n)$	$O(d^2(\log^7 n + \log^5 n \log^4 d))$	$O(d)$

History (Fully Dynamic)

	Det./ Rand.	Space	Preprocessing	Update	Query
Henzinger & Neumann [HN16]	Det.	$\widetilde{O}(n_{\text{off}}^2 m)$	$\widehat{O}(n_{\text{off}}^2 m) + \widetilde{O}(d_\star n_{\text{off}}^2 m)$	$\widehat{O}(d^4)$	$O(d^2)$
Hu, Kosinas & Polak [HKP23]	Det.	$\widetilde{O}((n_{\text{off}} + d_\star)m)$	$\widetilde{O}((n_{\text{off}} + d_\star)m)$	$\widetilde{O}(d^4)$	$O(d)$
This paper	Det.	$O(\min\{(n_{\text{off}} + d_\star)m \log^2 n, n^2\})$	$\widehat{O}(m)+$ $O(\min\{(n_{\text{off}} + d_\star)m, n^\omega\} \log^2 n)$	$O(d^2 \log^7 n)$	$O(d)$

Technique Overview

- **Low Degree Hierarchy,** which can reduced to $O(\log n)$ calls to the low-degree Steiner forest decomposition.[DP20, LS22]
- We say a forest $F \subseteq E(G)$, is a spanning forest of U in G if F spans the whole U (may also span vertices not in U).

Technique Overview

- **Low Degree Hierarchy,** which can reduced to $O(\log n)$ calls to the low-degree Steiner forest decomposition.[DP20, LS22]
- We say a forest $F \subseteq E(G)$, is a spanning forest of U in G if F spans the whole U (may also span vertices not in U).
- **Key Lemma** Let $U \subseteq V(G)$, there is an almost-linear-time algorithm that computes a separator $X \subseteq V(G)$ of size $|X| \leq |U|/2$, and a low-degree Steiner forest of U/X in G/X with maximum degree Δ.

Technique Overview

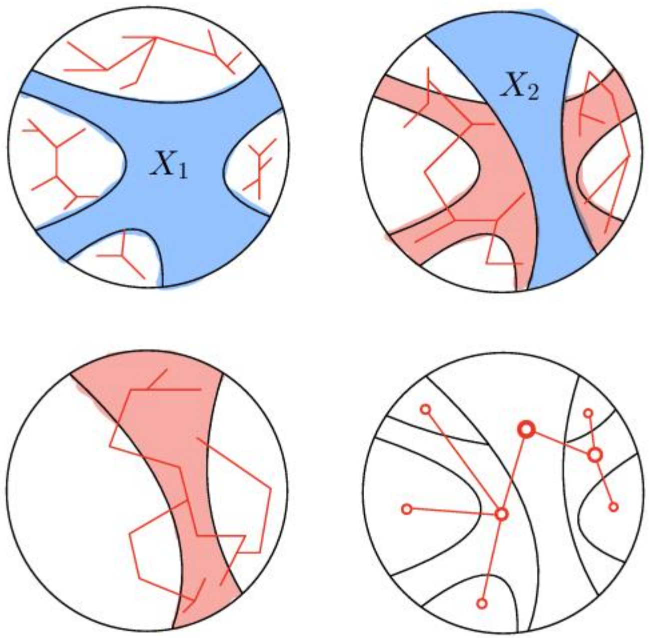
- **Low Degree Hierarchy,** which can reduced to $O(\log n)$ calls to the low-degree Steiner forest decomposition.[DP20, LS22]
- We say a forest $F \subseteq E(G)$, is a spanning forest of U in G if F spans the whole U (may also span vertices not in U).
- **Key Lemma** Let $U \subseteq V(G)$, there is an almost-linear-time algorithm that computes a separator $X \subseteq V(G)$ of size $|X| \leq |U|/2$, and a low-degree Steiner forest of U/X in G/X with maximum degree Δ.
- We improve Δ from $n^{o(1)}$ [LS22] to $O(\log^2 n)$.

Technique Overview

- **Low Degree Hierarchy,** which can reduced to $O(\log n)$ calls to the low-degree Steiner forest decomposition.[DP20, LS22]
- We say a forest $F \subseteq E(G)$, is a spanning forest of U in G if F spans the whole U (may also span vertices not in U).
- **Key Lemma** Let $U \subseteq V(G)$, there is an almost-linear-time algorithm that computes a separator $X \subseteq V(G)$ of size $|X| \leq |U|/2$, and a low-degree Steiner forest of U/X in G/X with maximum degree Δ.
- We improve Δ from $n^{o(1)}$ [LS22] to $O(\log^2 n)$.

Low Degree Hierarchy

- Low Degree Hierarchy: a laminar set C of components.
- Components form a tree hierarchy.
- $X_i = \mathrm{Decomp}(G, X_{i-1})$



Low Degree Hierarchy

- How do we get low degree hierarchy?
- Idea: Do Vertex Expander Decomposition![LS22]

Low Degree Hierarchy

- How do we get low degree hierarchy?
- Idea: Do Vertex Expander Decomposition![LS22]
- Cut Matching Game

Low Degree Hierarchy

- How do we get low degree hierarchy?
- Idea: Do Vertex Expander Decomposition![LS22]
- **Cut Matching Game**
- 1. Find a balanced sparse vertex cut OR
- 2. Cetify the graph is an expander.

Low Degree Hierarchy

- How do we get low degree hierarchy?
- Idea: Do Vertex Expander Decomposition![LS22]
- **Cut Matching Game**
- 1. Find a balanced sparse vertex cut OR
- 2. Cetify the graph is an expander.
- Can we do better?

Low Degree Hierarchy

- How do we get low degree hierarchy?
- Idea: Do Vertex Expander Decomposition![LS22]
- **Cut Matching Game**
- 1. Find a balanced sparse vertex cut OR
- 2. Cetify the graph is an expander.
- Can we do better?
- **Weak version of cut matching game**

Low Degree Hierarchy

- How do we get low degree hierarchy?
- Idea: Do Vertex Expander Decomposition![LS22]
- **Cut Matching Game**
- 1. Find a balanced sparse vertex cut OR
- 2. Cetify the graph is an expander.
- Can we do better?
- **Weak version of cut matching game**
- 1. Find a balanced sparse vertex cut OR
- 2. A low-degree Steiner tree covering a large fraction of terminals.

Low Degree Hierarchy

- How do we get low degree hierarchy?
- Idea: Do Vertex Expander Decomposition![LS22]
- **Cut Matching Game**
- 1. Find a balanced sparse vertex cut OR
- 2. Cetify the graph is an expander.
- Can we do better?
- **Weak version of cut matching game**
- 1. Find a balanced sparse vertex cut OR
- 2. A low-degree Steiner tree covering a large fraction of terminals.

Cut Matching Game

- Let G be a undirected graph with a terminal set U. Given a parameter φ and a partition (A, B) of U, there is a deterministic algorithm that computes either
- 1. A vertex cut (L, S, R) with $|R \cap U| \geq |L \cap U| \geq \min\{|A|, |B|\}/3$ and $S \leq \varphi \cdot |L \cap U|$
- 2. A matching M between A and B with size $|M| \geq \min\{|A|, |B|\}/3$ s.t. there is an embedding of M into G with congestion $O(1/\varphi)$

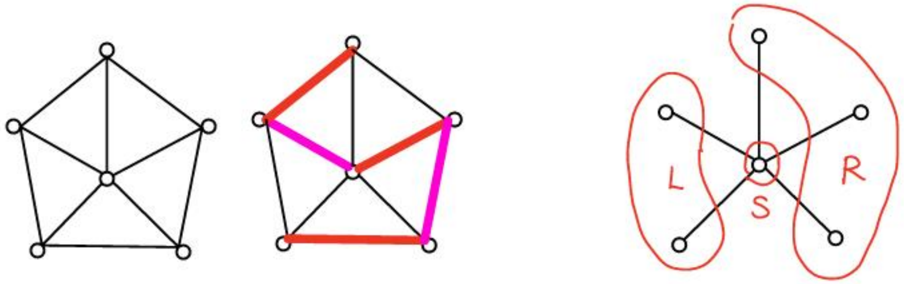


Figure Left: Low Degree Steiner Tree Right: Balanced Vertex Cut

Low Degree Hierarchy

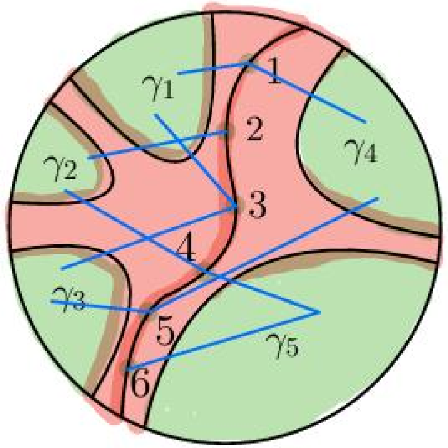
- Why we need low degree hierarchy?
- Reason: It allows us to consider semi-bipartite graph. We can assume that $G[V_{on}] = (L, R, E)$, where there is no edge between R.

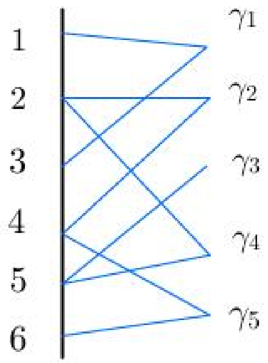
Low Degree Hierarchy

- Why we need low degree hierarchy?
- Reason: It allows us to consider semi-bipartite graph. We can assume that $G[V_{on}] = (L, R, E)$, where there is no edge between R.
- Moreover, L is spanned by a path τ, and τ is given to us.

Low Degree Hierarchy

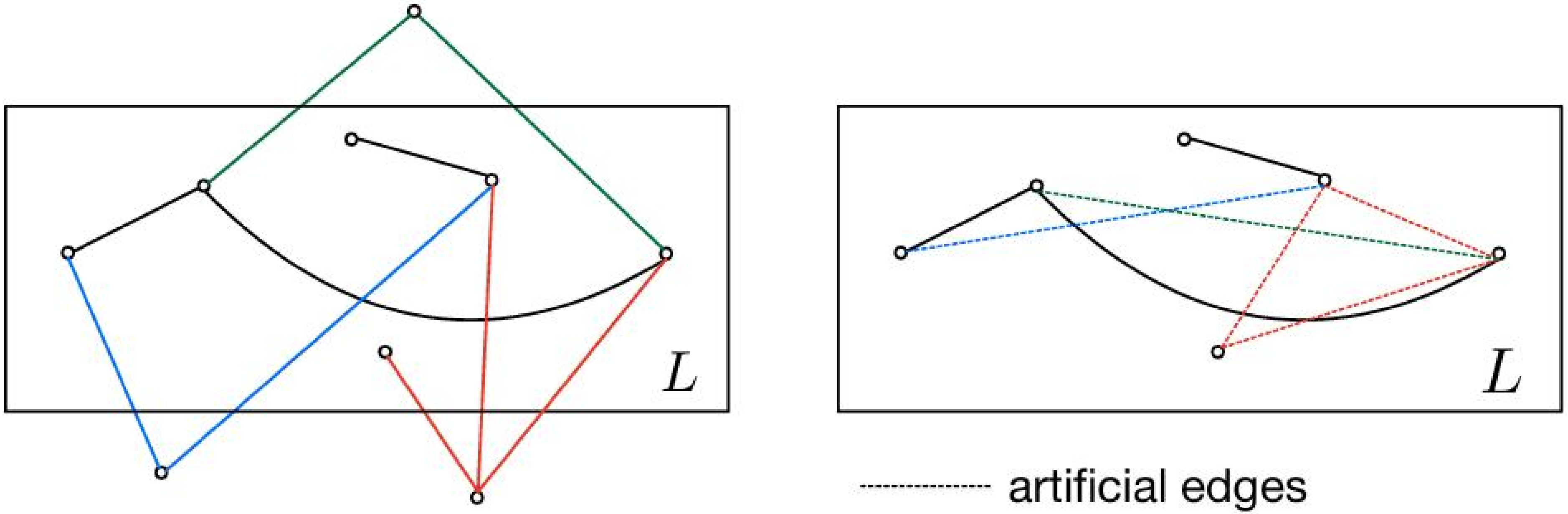
- Why we need low degree hierarchy?
- Reason: It allows us to consider semi-bipartite graph. We can assume that $G[V_{on}] = (L, R, E)$, where there is no edge between R.
- Moreover, L is spanned by a path τ, and τ is given to us.





Abstract Graph

- High level idea: maintain all vertices in L and ignore the vertices in R



Abstract Graph

- Deleting Vertices will divide L into at most $\mathrm{d} + 1$ intervals. For the new activated vertices, we simply add them into the abstract graph.
- There are $\mathrm{O}(\mathrm{d})$ vertices in the abstract graph after an update.

Abstract Graph

- Deleting Vertices will divide L into at most $\mathrm{d}+1$ intervals. For the new activated vertices, we simply add them into the abstract graph.
- There are $\mathrm{O}(\mathrm{d})$ vertices in the abstract graph after an update.
- For any two vertices u and v in the abstrach graph, we can determine whether they are still connected by computing their edge weights in the abstract graph and subtracting the weights of all affected artificial edges.

Abstract Graph

- Deleting Vertices will divide L into at most $\mathrm{d}+1$ intervals. For the new activated vertices, we simply add them into the abstract graph.
- There are $\mathrm{O}(\mathrm{d})$ vertices in the abstract graph after an update.
- For any two vertices u and v in the abstrach graph, we can determine whether they are still connected by computing their edge weights in the abstract graph and subtracting the weights of all affected artificial edges.
- Through preprocessing, we can obtain the edge weights in the abstract graph in $\mathrm{O}(1)$ time and the the weights of all affected artificial edges in $\mathrm{O}(\mathrm{d})$ time.

Abstract Graph

- Deleting Vertices will divide L into at most $\mathrm{d}+1$ intervals. For the new activated vertices, we simply add them into the abstract graph.
- There are $\mathrm{O}(\mathrm{d})$ vertices in the abstract graph after an update.
- For any two vertices u and v in the abstrach graph, we can determine whether they are still connected by computing their edge weights in the abstract graph and subtracting the weights of all affected artificial edges.
- Through preprocessing, we can obtain the edge weights in the abstract graph in $\mathrm{O}(1)$ time and the the weights of all affected artificial edges in $\mathrm{O}(\mathrm{d})$ time.

Borůvka-based algorithm

- For any two vertices, we can detect whether they are connected in $\tilde{O}(d)$ time. A naive algorithm can compute all pairs connectivity in $\tilde{O}(d^3)$ time. To achieve $\tilde{O}(d^2)$, we need to consider a Borůvka-based algorithm.
- Idea: "Hook and Track", Each phase we find a neighbor of all the sets, and contract them with their neighbors.

Borůvka-based algorithm

- For any two vertices, we can detect whether they are connected in $\tilde{O}(d)$ time. A naive algorithm can compute all pairs connectivity in $\tilde{O}(d^3)$ time. To achieve $\tilde{O}(d^2)$, we need to consider a Borůvka-based algorithm.
- Idea: **"Hook and Track"**, Each phase we find a neighbor of all the sets, and contract them with their neighbors.

Conditional Lower Bound

1. If $t_u + t_p = f(d) \cdot n^{o(1)}$, then $S = \widehat{\Omega}(n^2)$.
2. If $t_u + t_p = f(d) \cdot n^{o(1)}$, then $t_u = \widehat{\Omega}((n_{\text{off}} + d)m)$. [HKP23]
3. If $t_u + t_p = f(d) \cdot n^{o(1)}$, then $t_u = \widehat{\Omega}(n^{\omega_{\text{bool}}})$.
4. If $t_p = \text{poly}(n)$, then $t_u + t_q = \widehat{\Omega}(d^2)$. [LS22]
5. If $t_p = \text{poly}(n)$ and $t_u = \text{poly}(dn^{o(1)})$, then $t_q = \widehat{\Omega}(d)$. [HKNS15]

- The $f(d)$ above can be an arbitrary growing function, and ω_{bool} is the exponent of Boolean matrix multiplication.

Acknowledgement

- We thank Thatchaphol Saranurak for helpful discussions.

Thank you!