Orienting Fully Dynamic Graphs with Worst-Case Time Bounds

Krzysztof Potępa

Theoretical Computer Science

May 6, 2021

Sources

- Orienting Fully Dynamic Graphs with Worst-Case Time Bounds (2013)
  Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, Shay Solomon

Low out-degree orientations

c-orientation

Orientation of graph edges such that out-degree of every vertex is at most c.

We want c to be small.

Low out-degree orientations

c-orientation

Orientation of graph edges such that out-degree of every vertex is at most c.

We want c to be small.

Motivation

Low out-degree orientations enable efficient algorithms for many graph problems:

- adjacency queries in $O(\log \log c)$ using linear memory
- shortest-path queries
- maximal matchings (under inclusion)
- and many more

Arboricity

$$\alpha(G) = \max_{U \subseteq V(G)} \left\lceil \frac{|E(U)|}{|U| - 1} \right\rceil$$

Arboricity

Arboricity

$$\alpha(G) = \max_{U \subseteq V(G)} \left\lceil \frac{|E(U)|}{|U| - 1} \right\rceil$$

Nash-Williams Theorem

Graph $G = (V, E)$ has arboricity $\alpha(G)$ iff $\alpha(G)$ is the smallest number of sets $E_1, ..., E_{\alpha(G)}$ that E can be partitioned into, such that each subgraph (V, E_i) is a forest.

Arboricity

Arboricity

$$\alpha(G) = \max_{U \subseteq V(G)} \left\lceil \frac{|E(U)|}{|U| - 1} \right\rceil$$

Nash-Williams Theorem

Graph $G = (V, E)$ has arboricity $\alpha(G)$ iff $\alpha(G)$ is the smallest number of sets $E_1, ..., E_{\alpha(G)}$ that E can be partitioned into, such that each subgraph (V, E_i) is a forest.

Classes of graphs with α bounded by constant

- planar graphs
- excluded-minor families

Arboricity vs low out-degree orientations

Let $P(G)$ = smallest c such that G has c-orientation.

Arboricity vs low out-degree orientations

Let $P(G)$ = smallest c such that G has c-orientation.

Lemma

$\alpha(G) - 1 \leq P(G) \leq \alpha(G)$

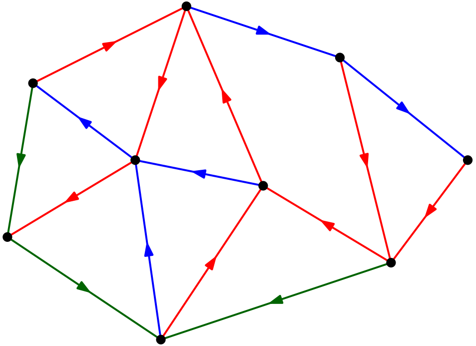
Arboricity vs low out-degree orientations

Let $P(G)$ = smallest c such that G has c-orientation.

Lemma

$\alpha(G) - 1 \leq P(G) \leq \alpha(G)$

- $P(G) \leq \alpha(G)$: split G into $\alpha(G)$ forests and orient each separately



Arboricity vs low out-degree orientations

Let $P(G)$ = smallest c such that G has c-orientation.

Lemma

$\alpha(G) - 1 \leq P(G) \leq \alpha(G)$

- $P(G) \leq \alpha(G)$: split G into $\alpha(G)$ forests and orient each separately
- $\alpha(G) - 1 \leq P(G)$
  Let $U \subseteq V$ be s.t. $\left\lceil \frac{|E(U)|}{|U|-1} \right\rceil = \alpha(G)$, hence $\frac{|E(U)|}{|U|-1} > \alpha(G) - 1$.

Arboricity vs low out-degree orientations

Let $P(G) =$ smallest c such that G has c-orientation.

Lemma

$\alpha(G) - 1 \leq P(G) \leq \alpha(G)$

- $P(G) \leq \alpha(G)$: split G into $\alpha(G)$ forests and orient each separately
- $\alpha(G) - 1 \leq P(G)$
  Let $U \subseteq V$ be s.t. $\left\lceil \frac{|E(U)|}{|U|-1} \right\rceil = \alpha(G)$, hence $\frac{|E(U)|}{|U|-1} > \alpha(G) - 1$.

$$P(G) \geq \frac{|E(U)|}{|U|}$$

Arboricity vs low out-degree orientations

Let $P(G)$ = smallest c such that G has c-orientation.

Lemma

$\alpha(G) - 1 \leq P(G) \leq \alpha(G)$

- $P(G) \leq \alpha(G)$: split G into $\alpha(G)$ forests and orient each separately
- $\alpha(G) - 1 \leq P(G)$
  Let $U \subseteq V$ be s.t. $\left\lceil \frac{|E(U)|}{|U|-1} \right\rceil = \alpha(G)$, hence $\frac{|E(U)|}{|U|-1} > \alpha(G) - 1$.

$$P(G) \geq \frac{|E(U)|}{|U|} > \frac{|U|-1}{|U|} \cdot (\alpha(G) - 1)$$

Arboricity vs low out-degree orientations

Let $P(G)$ = smallest c such that G has c-orientation.

Lemma

$\alpha(G) - 1 \leq P(G) \leq \alpha(G)$

- $P(G) \leq \alpha(G)$: split G into $\alpha(G)$ forests and orient each separately
- $\alpha(G) - 1 \leq P(G)$
  Let $U \subseteq V$ be s.t. $\left\lceil \frac{|E(U)|}{|U|-1} \right\rceil = \alpha(G)$, hence $\frac{|E(U)|}{|U|-1} > \alpha(G) - 1$.

$$P(G) \geq \frac{|E(U)|}{|U|} > \frac{|U| - 1}{|U|} \cdot (\alpha(G) - 1)$$
$$|U| \cdot P(G) > (|U| - 1) \cdot (\alpha(G) - 1)$$
$$|U| \cdot (P(G) - (\alpha(G) - 2)) > 1$$

Orienting fully dynamic graphs

Problem

Maintain Δ-orientation of graph G under operations:

- insert edge
- remove edge

Graph G has arboricity bounded by α at any time.

Orienting fully dynamic graphs

Problem

Maintain Δ-orientation of graph G under operations:

- insert edge
- remove edge

Graph G has arboricity bounded by α at any time.

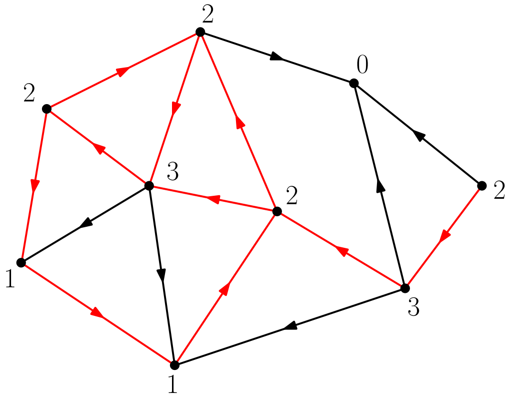
In the following, we assume that α is constant.

	Bound on Δ	Edge insertion	Edge removal
Brodal, Fagerberg (1999)	4α	amortized $O(1)$	amortized $O(\log n)$
Kowalik (2007)	4α	amortized $O(\log n)$	worst-case $O(1)$
	$O(\log n)$	amortized $O(1)$	worst-case $O(1)$
Kopelowitz et al. (2013)	$O(\log n)$	worst-case $O(\log n)$	worst-case $O(\log n)$

Invariants for bounding the out-degrees

Valid edge

Edge $u \to v$ is **valid** iff $d_{out}(u) \le d_{out}(v) + 1$, else it is **violated**.



Invariants for bounding the out-degrees

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

Invariant

For each vertex w, at least $\min(d_{out}(w), \gamma)$ outgoing edges are valid.

Invariants for bounding the out-degrees

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

Invariant

For each vertex w, at least $\min(d_{out}(w), \gamma)$ outgoing edges are valid.

Theorem

If the invariant holds, then $\Delta \leq \gamma + \lceil \log_\beta n \rceil$.

Invariants for bounding the out-degrees

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

Invariant

For each vertex w, at least $\min(d_{out}(w), \gamma)$ outgoing edges are valid.

Theorem

If the invariant holds, then $\Delta \leq \gamma + \lceil \log_\beta n \rceil$.

- Suppose we have vertex s with $d_{out}(s) > \gamma + \lceil \log_\beta n \rceil$.

Invariants for bounding the out-degrees

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

Invariant

For each vertex w, at least $\min(d_{out}(w), \gamma)$ outgoing edges are valid.

Theorem

If the invariant holds, then $\Delta \leq \gamma + \lceil \log_\beta n \rceil$.

- Suppose we have vertex s with $d_{out}(s) > \gamma + \lceil \log_\beta n \rceil$.
- Let V_i = vertices reachable from s using at most i valid edges

Invariants for bounding the out-degrees

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

Invariant

For each vertex w, at least $\min(d_{out}(w), \gamma)$ outgoing edges are valid.

Theorem

If the invariant holds, then $\Delta \leq \gamma + \lceil \log_\beta n \rceil$.

- Suppose we have vertex s with $d_{out}(s) > \gamma + \lceil \log_\beta n \rceil$.
- Let V_i = vertices reachable from s using at most i valid edges
- For every $i \in \{1, ..., \lceil \log_\beta n \rceil\}$ and $w \in V_i$ we have:

$$d_{out}(w) \geq d_{out}(s) - i > \gamma + \lceil \log_\beta n \rceil - i \geq \gamma$$

Invariants for bounding the out-degrees

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, \dots, \lceil \log_\beta n \rceil\}$.

Invariants for bounding the out-degrees

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, ..., \lceil \log_\beta n \rceil\}$.

- For $i = 1$ we have: $|V_1| = 1 + |N_{out}(s)| \geq \gamma + 1 > \gamma \geq \beta$

Invariants for bounding the out-degrees

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, ..., \lceil \log_\beta n \rceil\}$.

- For $i = 1$ we have: $|V_1| = 1 + |N_{out}(s)| \geq \gamma + 1 > \gamma \geq \beta$
- Suppose now that $|V_{i-1}| > \beta^{i-1}$.

Invariants for bounding the out-degrees

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, ..., \lceil \log_\beta n \rceil\}$.

- For $i = 1$ we have: $|V_1| = 1 + |N_{out}(s)| \geq \gamma + 1 > \gamma \geq \beta$
- Suppose now that $|V_{i-1}| > \beta^{i-1}$.

$$|E(V_i)| \geq \gamma |V_{i-1}|$$

Invariants for bounding the out-degrees

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, ..., \lceil \log_\beta n \rceil\}$.

- For $i = 1$ we have: $|V_1| = 1 + |N_{out}(s)| \geq \gamma + 1 > \gamma \geq \beta$
- Suppose now that $|V_{i-1}| > \beta^{i-1}$.

$$|E(V_i)| \geq \gamma |V_{i-1}| \qquad \alpha \geq \frac{|E(V_i)|}{|V_i| - 1}$$

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, ..., \lceil \log_\beta n \rceil\}$.

- For $i = 1$ we have: $|V_1| = 1 + |N_{out}(s)| \geq \gamma + 1 > \gamma \geq \beta$
- Suppose now that $|V_{i-1}| > \beta^{i-1}$.

$$|E(V_i)| \geq \gamma |V_{i-1}| \qquad \alpha \geq \frac{|E(V_i)|}{|V_i| - 1}$$

$$|V_i| - 1 \geq \frac{\gamma |V_{i-1}|}{\alpha}$$

Invariants for bounding the out-degrees

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, ..., \lceil \log_\beta n \rceil\}$.

- For $i = 1$ we have: $|V_1| = 1 + |N_{out}(s)| \geq \gamma + 1 > \gamma \geq \beta$
- Suppose now that $|V_{i-1}| > \beta^{i-1}$.

$$|E(V_i)| \geq \gamma |V_{i-1}| \qquad \alpha \geq \frac{|E(V_i)|}{|V_i| - 1}$$

$$|V_i| - 1 \geq \frac{\gamma |V_{i-1}|}{\alpha} \geq \beta |V_{i-1}| > \beta^i$$

Invariants for bounding the out-degrees

We prove by induction on i that $|V_i| > \beta^i$ for all $i \in \{1, \ldots, \lceil \log_\beta n \rceil\}$.

- For $i = 1$ we have: $|V_1| = 1 + |N_{out}(s)| \geq \gamma + 1 > \gamma \geq \beta$
- Suppose now that $|V_{i-1}| > \beta^{i-1}$.

$$|E(V_i)| \geq \gamma |V_{i-1}| \qquad \alpha \geq \frac{|E(V_i)|}{|V_i| - 1}$$

$$|V_i| - 1 \geq \frac{\gamma |V_{i-1}|}{\alpha} \geq \beta |V_{i-1}| > \beta^i$$

We have $\left| V_{\lceil \log_\beta n \rceil} \right| > \beta^{\lceil \log_\beta n \rceil} \geq n$, contradiction.

Simple algorithm

Invariant

For each vertex w, at least $\min(d_{out}(w), \gamma)$ outgoing edges are valid.

Theorem

If the invariant holds, then $\Delta \leq \gamma + \lceil \log_\beta n \rceil$.

Simple algorithm

Strong invariant

For each vertex w, **all** outgoing edges are valid.

Theorem

If the strong invariant holds, then $\Delta \leq \inf_{\beta>1} \left\{\beta \cdot \alpha(G) + \lceil \log_\beta n \rceil \right\}$.

Simple algorithm

Strong invariant

For each vertex w, **all** outgoing edges are valid.

Theorem

If the strong invariant holds, then $\Delta \leq \inf_{\beta>1} \left\{\beta \cdot \alpha(G) + \lceil \log_\beta n \rceil \right\}$.

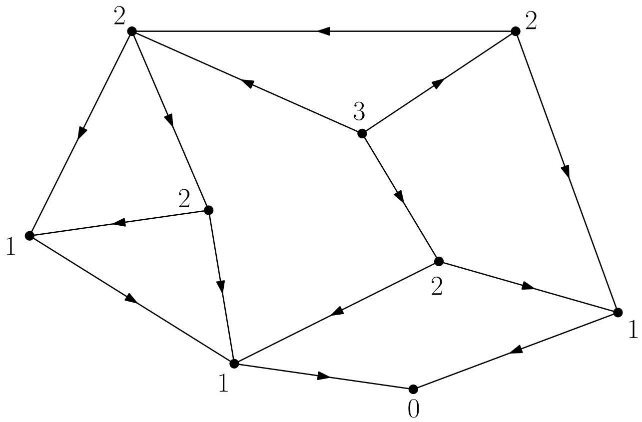
Algorithm 1

We maintain the strong invariant for dynamic graph G and support:

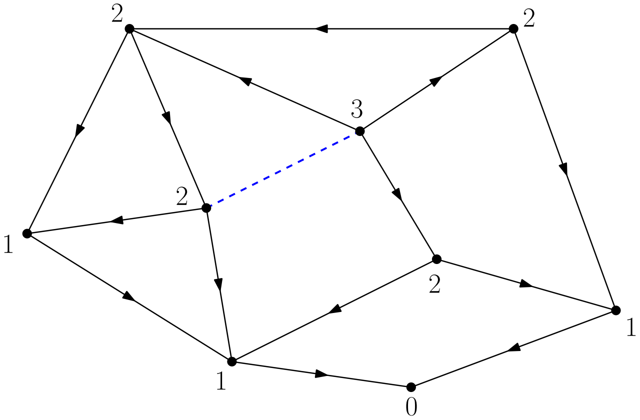
- edge insertion in worst-case $O(\Delta^2)$ time
- edge removal in worst-case $O(\Delta)$ time

Both operations reorient at most $\Delta + 1$ edges.

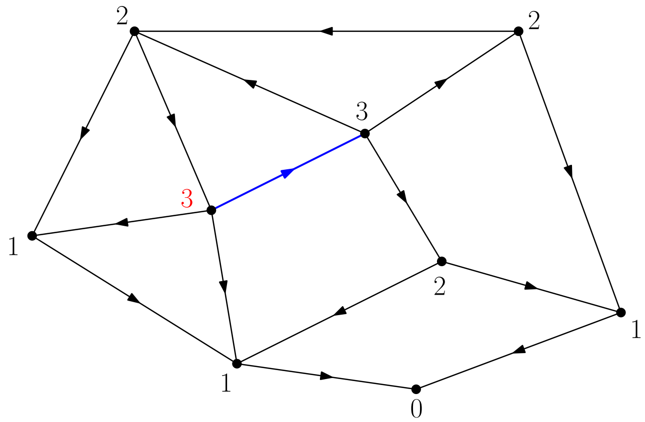
Edge insertion



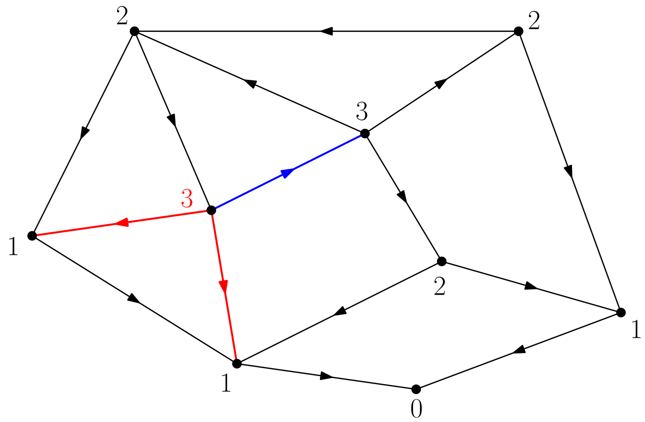
Edge insertion



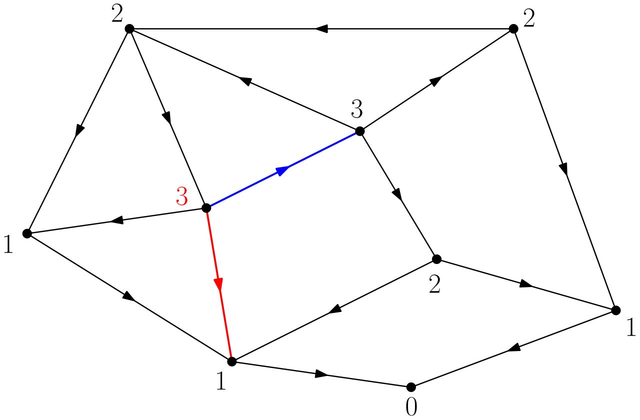
Edge insertion



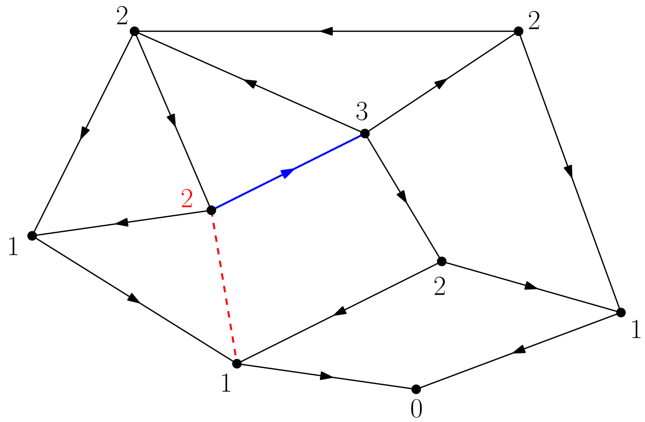
Edge insertion



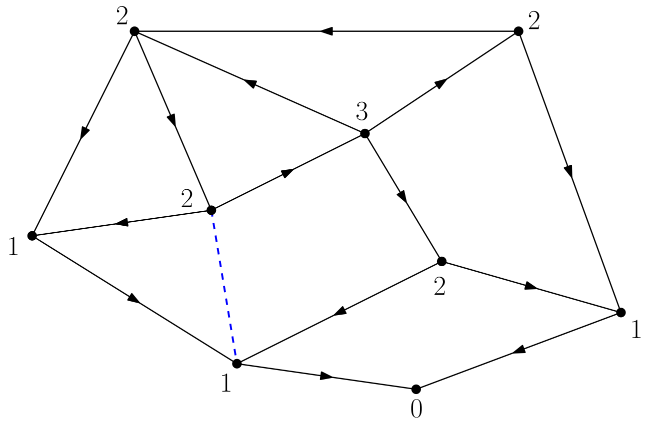
Edge insertion



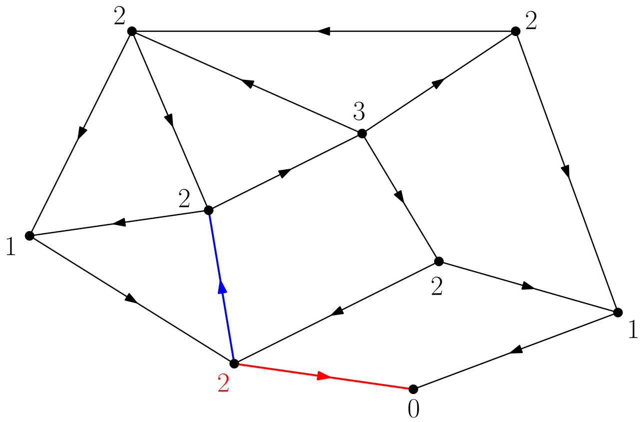
Edge insertion



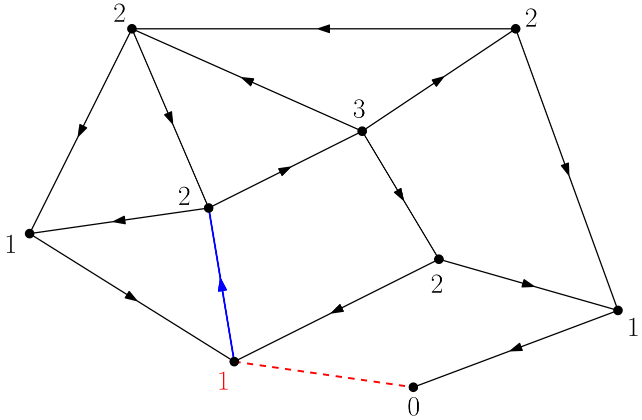
Edge insertion



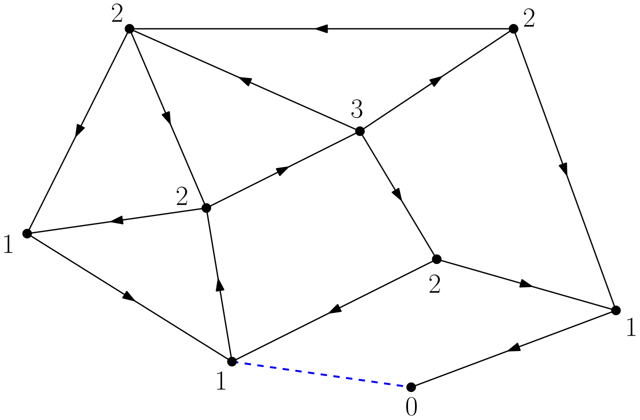
Edge insertion



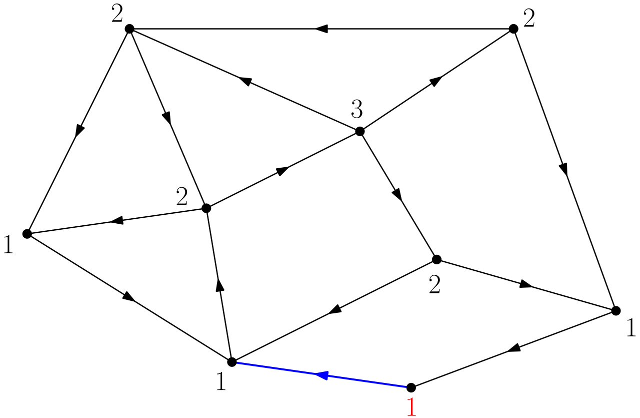
Edge insertion



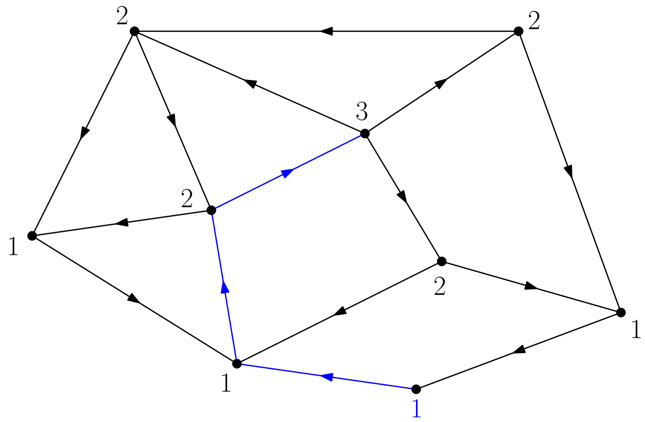
Edge insertion



Edge insertion



Edge insertion



Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ among Δ out-edges of u
3. If such edge exists, remove $u \to v'$ and insert $v' \to u$ recursively

Edge insertion

Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ among Δ out-edges of u
3. If such edge exists, remove $u \to v'$ and insert $v' \to u$ recursively

Number of recursive calls

- $d_{out}(u) = d_{out}(v') + 1$ (degree "decreases" by 1 in each recursion)
- Δ recursive calls excluding the initial one

Edge insertion

Insert $u \rightarrow v$ such that $d_{out}(u) \leq d_{out}(v)$

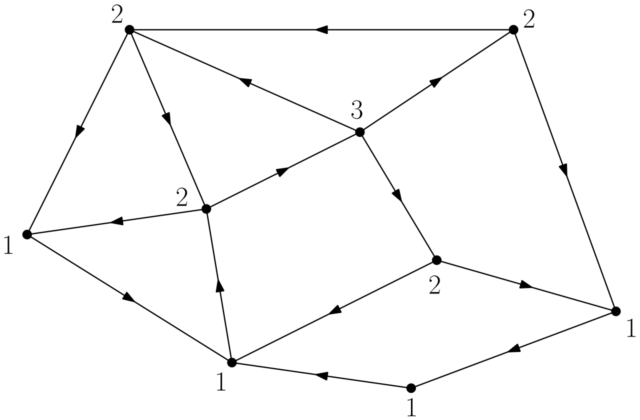
1. Add edge $u \rightarrow v$ to graph
2. Find violated edge $u \rightarrow v'$ among Δ out-edges of u
3. If such edge exists, remove $u \rightarrow v'$ and insert $v' \rightarrow u$ recursively

Number of recursive calls

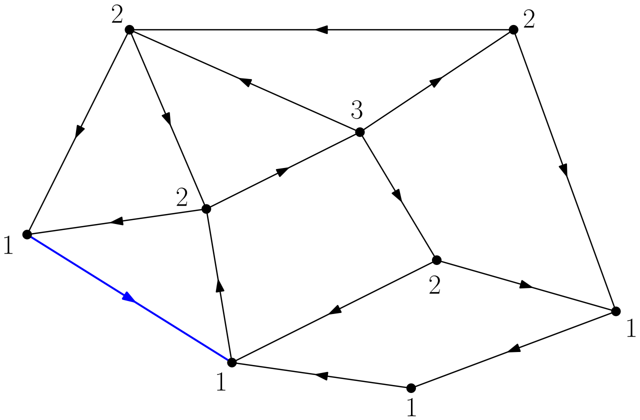
- $d_{out}(u) = d_{out}(v') + 1$ (degree "decreases" by 1 in each recursion)
- Δ recursive calls excluding the initial one

Edge insertion time: $O(\Delta^2)$

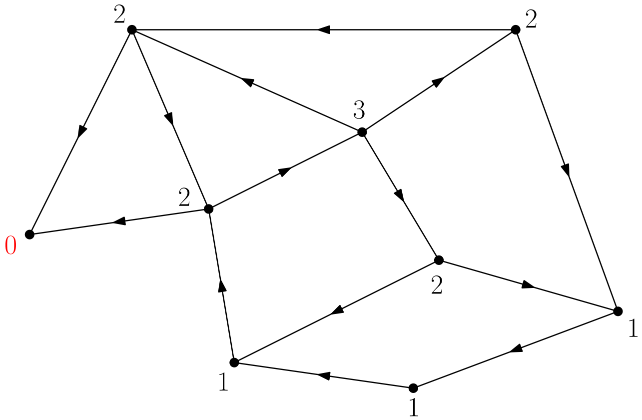
Edge removal



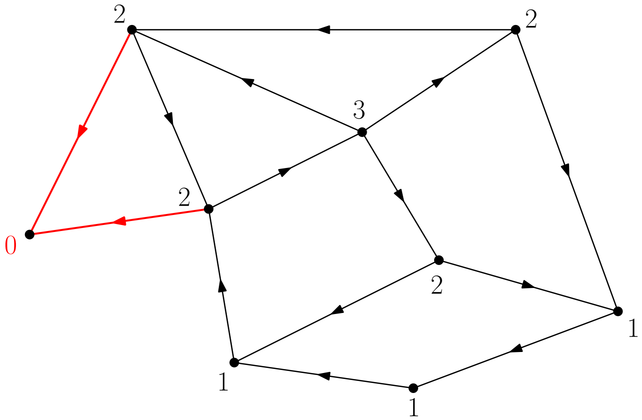
Edge removal



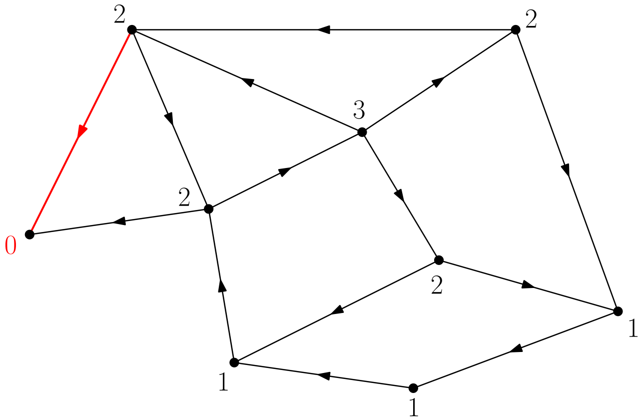
Edge removal



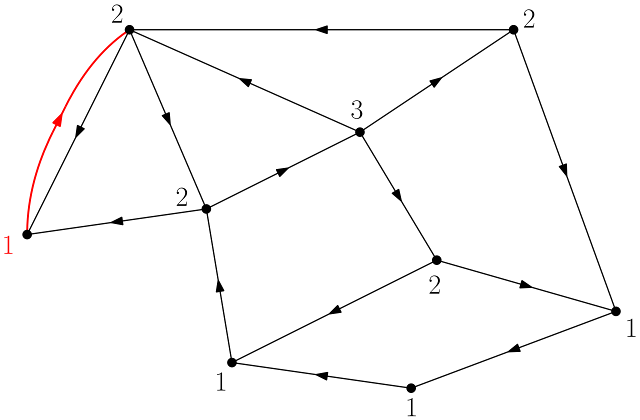
Edge removal



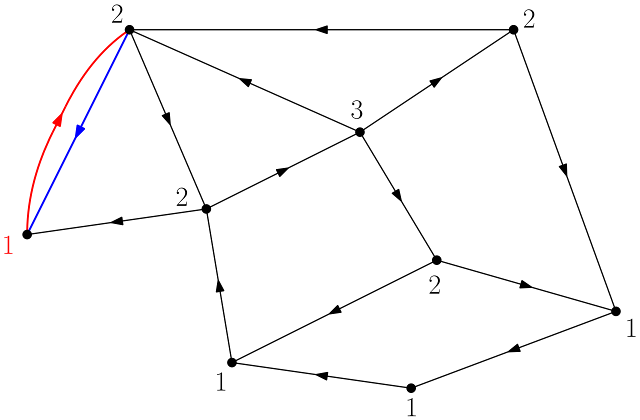
Edge removal



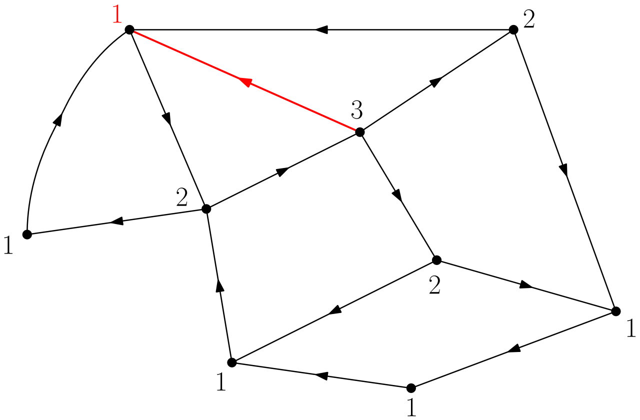
Edge removal



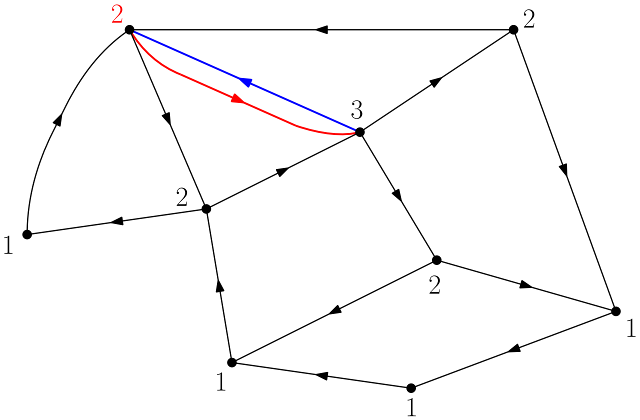
Edge removal



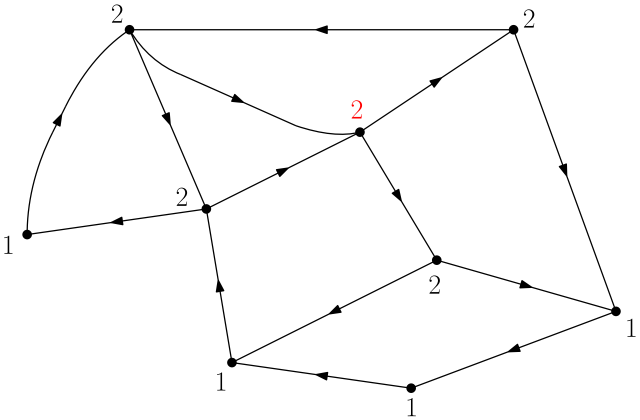
Edge removal



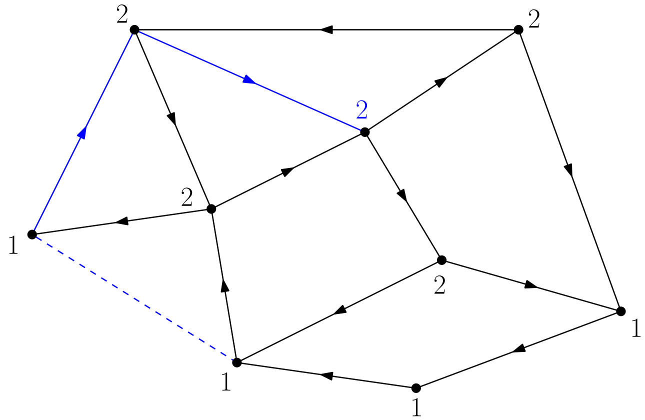
Edge removal



Edge removal



Edge removal



Remove $u \to v$

1. Remove $u \to v$ from graph
2. Find violated edge $v' \to u$ among in-edges of u (how?)
3. If such edge exists, add $u \to v'$ and remove $v' \to u$ recursively

Edge removal

Remove $u \to v$

1. Remove $u \to v$ from graph
2. Find violated edge $v' \to u$ among in-edges of u (how?)
3. If such edge exists, add $u \to v'$ and remove $v' \to u$ recursively

Number of recursive calls

- $d_{out}(v') = d_{out}(u) + 1$ (degree "increases" by 1 in each recursion)
- Δ recursive calls excluding the initial one

Edge removal

Remove $u \to v$

1. Remove $u \to v$ from graph
2. Find violated edge $v' \to u$ among in-edges of u (how?)
3. If such edge exists, add $u \to v'$ and remove $v' \to u$ recursively

Number of recursive calls

- $d_{out}(v') = d_{out}(u) + 1$ (degree "increases" by 1 in each recursion)
- Δ recursive calls excluding the initial one

How to find the violated edge quickly?

Finding violated incoming edge

Data structure

Let k_0 be a parameter.
Maintain set of elements X, each with associated integer key, under operations:

- get element with maximum key in $O(1)$
- insert element with key $0 \leq k \leq k_0$ in $O(1)$
- remove element in $O(1)$
- increment/decrement key of given element in $O(1)$
- increment/decrement parameter k_0 in $O(k_0)$

Data structure uses $O(n + k_0)$ memory, where n is the number of elements.

Finding violated incoming edge

Data structure

Let k_0 be a parameter.
Maintain set of elements X, each with associated integer key, under operations:

- get element with maximum key in $O(1)$
- insert element with key $0 \leq k \leq k_0$ in $O(1)$
- remove element in $O(1)$
- increment/decrement key of given element in $O(1)$
- increment/decrement parameter k_0 in $O(k_0)$

Data structure uses $O(n + k_0)$ memory, where n is the number of elements.

Incoming edges

For each vertex w, maintain data structure H_w over all incoming edges.
Key of edge $u \to w$ is $d_{out}(w)$. Parameter k_0 is $d_{out}(w) + 1$.

Simple algorithm

Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ by iterating over Δ out-edges of u
3. If such edge exists, remove $u \to v'$ and insert $v' \to u$ recursively

Remove $u \to v$

1. Remove $u \to v$ from graph
2. Find violated edge $v' \to u$ using data structure H_u
3. If such edge exists, add $u \to v'$ and remove $v' \to u$ recursively

Simple algorithm

Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ by iterating over Δ out-edges of u
3. If such edge exists, remove $u \to v'$ and insert $v' \to u$ recursively

Remove $u \to v$

1. Remove $u \to v$ from graph
2. Find violated edge $v' \to u$ using data structure H_u
3. If such edge exists, add $u \to v'$ and remove $v' \to u$ recursively

Edge insertion time: $O(\Delta^2)$
Edge removal time: $O(\Delta)$

Improving runtime

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

Improving runtime

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

i-valid edge

Edge $u \rightarrow v$ is ***i*-valid** iff $d_{out}(u) \leq d_{out}(v) + i$, else it is ***i*-violated**.

Improving runtime

Let $n = |V(G)|$. Let $\beta > 1$ be an arbitrary parameter and let $\gamma = \beta \cdot \alpha$.

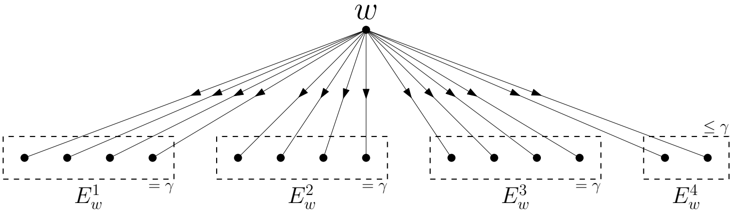
i-valid edge

Edge $u \rightarrow v$ is ***i*-valid** iff $d_{out}(u) \leq d_{out}(v) + i$, else it is ***i*-violated**.

Spectrum-validity for vertex w

Vertex w is spectrum-valid if its set of outgoing edges E_w can be partitioned into $q = \lceil |E_w|/\gamma \rceil$ sets $E_w^1, \ldots, E_w^q$ such that:

- $|E_w^i| = \gamma$ for each $i \in \{1, \ldots, q-1\}$
- all edges in E_w^i are i-valid



Improving runtime

Intermediate invariant

Every vertex is spectrum-valid.

Improving runtime

Intermediate invariant

Every vertex is spectrum-valid.

Algorithm 2

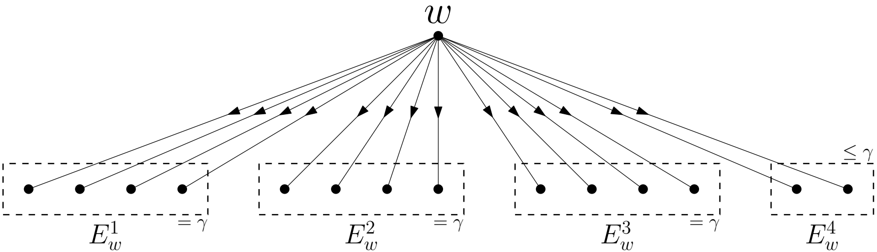
We maintain the intermediate invariant for dynamic graph G and support:

- edge insertion in worst-case $O(\gamma\Delta)$ time
- edge removal in worst-case $O(\Delta)$ time

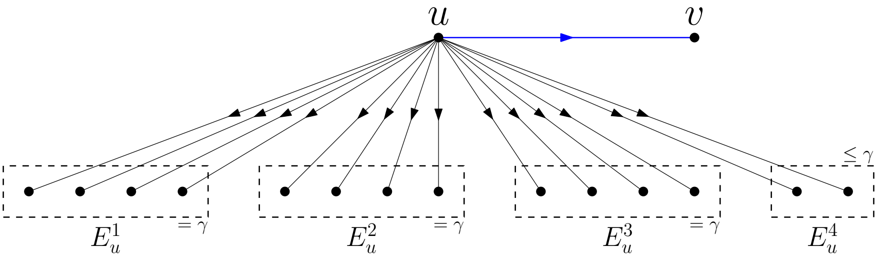
Both operations reorient at most $\Delta + 1$ edges.

Improving runtime

For each vertex w, we keep list L_w of outgoing vertices such that the first γ vertices are 1-valid, the next γ vertices 2-valid and so on.



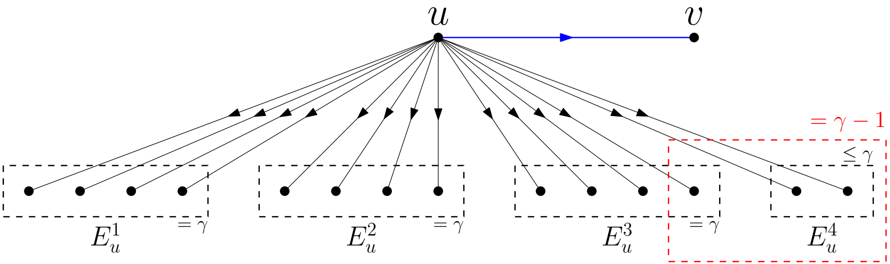
Edge insertion



Insert $u \rightarrow v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \rightarrow v$ to graph

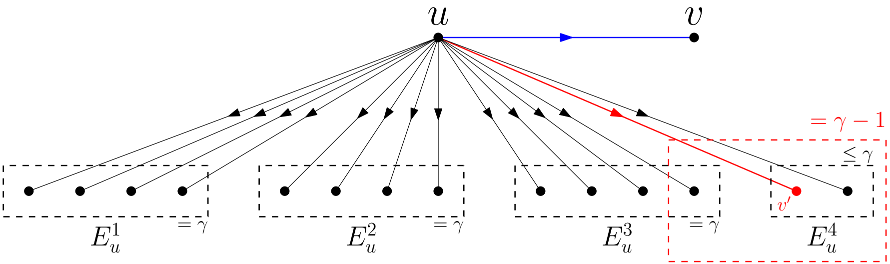
Edge insertion



Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph

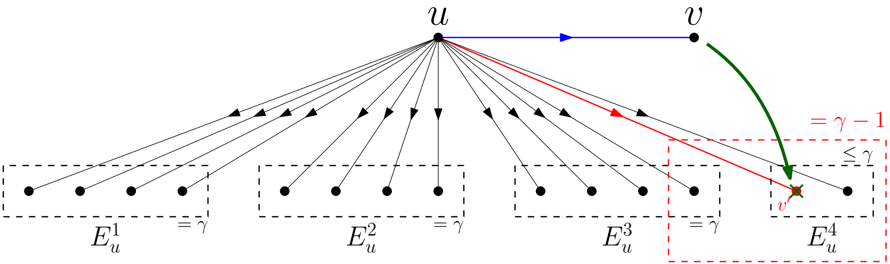
Edge insertion



Insert $u \rightarrow v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \rightarrow v$ to graph
2. Find violated edge $u \rightarrow v'$ among last $\gamma - 1$ edges of L_w

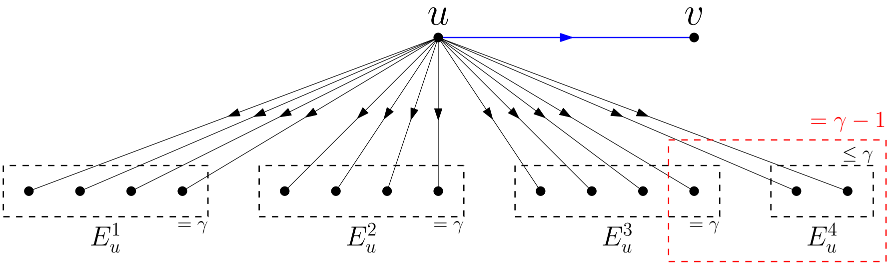
Edge insertion



Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ among last $\gamma - 1$ edges of L_w
3. If such edge exists, replace $u \to v'$ with $u \to v$ in L_w, and remove $u \to v'$ and insert $v' \to u$ recursively

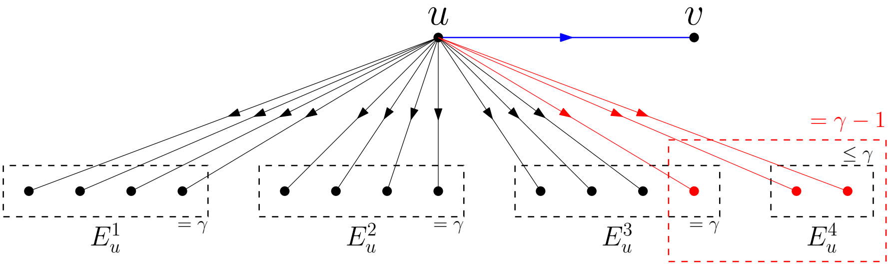
Edge insertion



Insert $u \rightarrow v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \rightarrow v$ to graph
2. Find violated edge $u \rightarrow v'$ among last $\gamma - 1$ edges of L_w
3. If such edge exists, replace $u \rightarrow v'$ with $u \rightarrow v$ in L_w, and remove $u \rightarrow v'$ and insert $v' \rightarrow u$ recursively

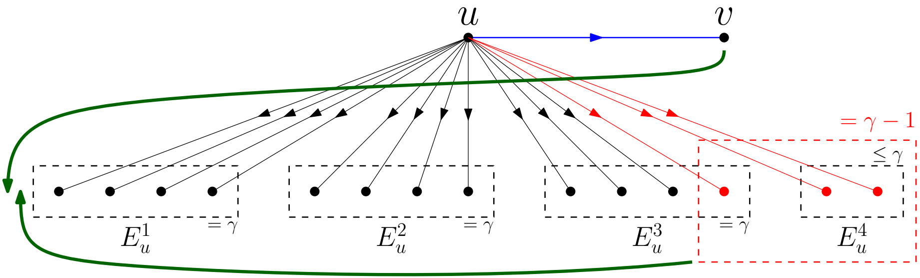
Edge insertion



Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ among last $\gamma - 1$ edges of L_w
3. If such edge exists, replace $u \to v'$ with $u \to v$ in L_w, and remove $u \to v'$ and insert $v' \to u$ recursively

Edge insertion



Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ among last $\gamma - 1$ edges of L_w
3. If such edge exists, replace $u \to v'$ with $u \to v$ in L_w, and remove $u \to v'$ and insert $v' \to u$ recursively
4. Else move last $\gamma - 1$ edges of L_w to the front and add $u \to v$ to the front

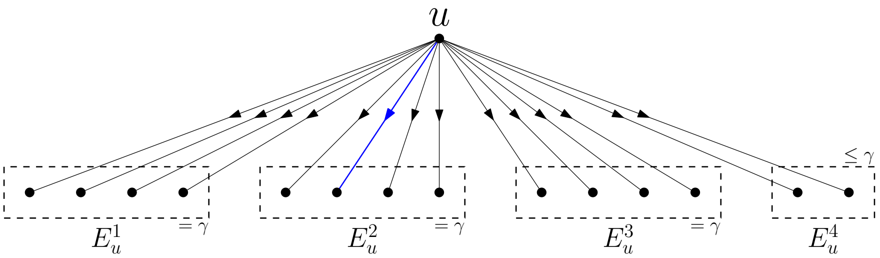
Edge insertion

Insert $u \to v$ such that $d_{out}(u) \leq d_{out}(v)$

1. Add edge $u \to v$ to graph
2. Find violated edge $u \to v'$ among last $\gamma - 1$ edges of L_w
3. If such edge exists, replace $u \to v'$ with $u \to v$ in L_w, and remove $u \to v'$ and insert $v' \to u$ recursively
4. Else move last $\gamma - 1$ edges of L_w to the front and add $u \to v$ to the front

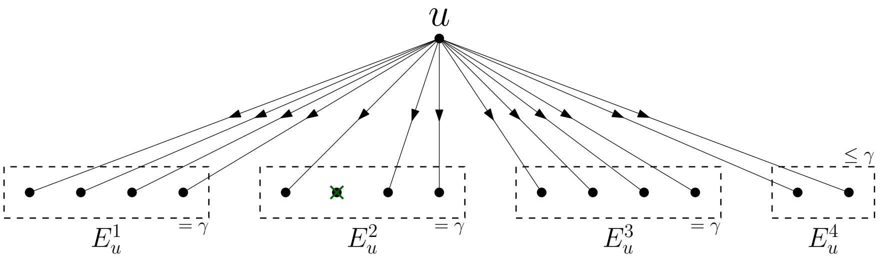
Edge insertion time: $O(\gamma\Delta)$

Edge removal



Remove $u \to v$

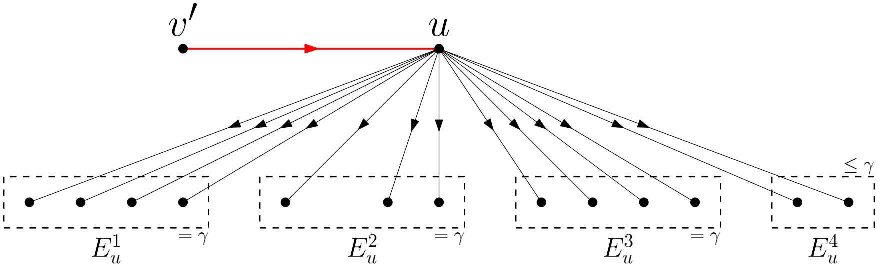
Edge removal



Remove $u \to v$

1. Remove $u \to v$ from graph and the list L_w

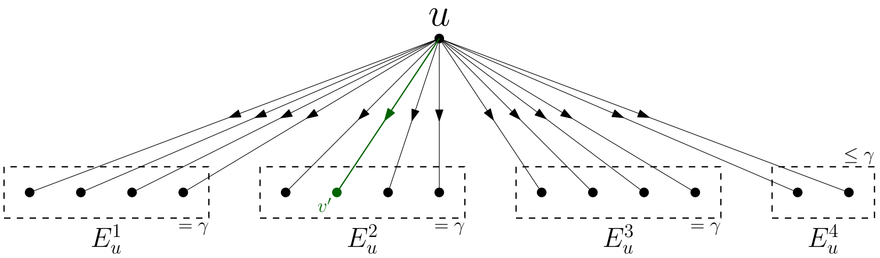
Edge removal



Remove $u \rightarrow v$

1. Remove $u \rightarrow v$ from graph and the list L_w
2. Find violated edge $v' \rightarrow u$ using data structure H_u

Edge removal



Remove $u \rightarrow v$

1. Remove $u \rightarrow v$ from graph and the list L_w
2. Find violated edge $v' \rightarrow u$ using data structure H_u
3. If such edge exists, add $u \rightarrow v'$ in place of $u \rightarrow v$ in L_w, add $u \rightarrow v'$ to graph and remove $v' \rightarrow u$ recursively

Edge removal

Remove $u \to v$

1. Remove $u \to v$ from graph and the list L_w
2. Find violated edge $v' \to u$ using data structure H_u
3. If such edge exists, add $u \to v'$ in place of $u \to v$ in L_w, add $u \to v'$ to graph and remove $v' \to u$ recursively

Edge removal time: $O(\Delta)$

Summary

Final algorithm

We maintain Δ-orientation of graph G with arboricity bounded by α, where:

- $\Delta \leq \inf_{\beta > 1} \left\{ \beta\alpha + \left\lceil \log_\beta n \right\rceil \right\}$
- edge insertion works in worst-case $O(\beta\alpha\Delta)$ time
- edge removal works in worst-case $O(\Delta)$ time

Both operations reorient at most $\Delta + 1$ edges.

Summary

Final algorithm

We maintain Δ-orientation of graph G with arboricity bounded by α, where:

- $\Delta \leq \inf_{\beta>1} \left\{\beta\alpha + \left\lceil \log_\beta n \right\rceil \right\}$
- edge insertion works in worst-case $O(\beta\alpha\Delta)$ time
- edge removal works in worst-case $O(\Delta)$ time

Both operations reorient at most $\Delta + 1$ edges.

For constant arboricity, if we set $\beta = 2$, then bounds translate to $O(\log n)$.