Computing and Testing Small Connectivity in Near-Linear Time and Queries via Fast Local Cut Algorithms

Sebastian Forster
University of Salzburg

Joint work with Danupon Nanongkai, Thatchaphol Saranurak, Liu Yang, and Sorrachai Yingchareonthawornchai

Workshop: Recent Trends in Theoretical Computer Science
Edge and Vertex Connectivity

**Edge connectivity $\lambda$ / vertex connectivity $\kappa$**

Minimum number of edges/vertices to remove in order to make the graph not strongly connected
Edge and Vertex Connectivity

**Edge connectivity** $\lambda$/**vertex connectivity** $\kappa$

Minimum number of edges/vertices to remove in order to make the graph not strongly connected

Edge cut:
Edge and Vertex Connectivity

**Edge connectivity $\lambda$/vertex connectivity $\kappa$**

Minimum number of edges/vertices to remove in order to make the graph not strongly connected

**Edge cut:**

**Vertex cut:**
Edge and Vertex Connectivity

Edge connectivity $\lambda$ / vertex connectivity $\kappa$

Minimum number of edges/vertices to remove in order to make the graph not strongly connected

**Motivation:**
- Fundamental graph-theoretic notion
- Applications: Reliability analysis, community detection
State of the Art and Results

Vertex connectivity in directed graphs:

<table>
<thead>
<tr>
<th>Running time</th>
<th>Deterministic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{O}(n^{2.373} + nk^{2.373})$</td>
<td>no</td>
<td>[Cheriyan/Reif ’92]</td>
</tr>
<tr>
<td>$\tilde{O}(mn)$</td>
<td>no</td>
<td>[Henzinger et al. ’96]</td>
</tr>
<tr>
<td>$O(mn + \kappa mn^{3/4})$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$O(mn + \kappa^{5/2}m)$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{4/3})$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{2/3}n)$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^2 m)$</td>
<td>no</td>
<td>Our result</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^{3/2} m^{1/2}n + \kappa^3 n)$</td>
<td>no</td>
<td>Our result</td>
</tr>
</tbody>
</table>

Undirected graphs:

$m \rightarrow n \kappa$ [Nagamochi/Ibaraki ‘92]

State of the art for edge connectivity in directed graphs:

$\tilde{O}(\lambda m)$ [Gabow ‘95]

Improvements also for finding $k$-edge connected subgraphs [Chechik et al. ’17]
State of the Art and Results

Vertex connectivity in directed graphs:

<table>
<thead>
<tr>
<th>Running time</th>
<th>Deterministic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{O}(n^{2.373} + nk^{2.373})$</td>
<td>no</td>
<td>[Cheriyan/Reif ’92]</td>
</tr>
<tr>
<td>$\tilde{O}(mn)$</td>
<td>no</td>
<td>[Henzinger et al. ’96]</td>
</tr>
<tr>
<td>$O(mn + \kappa mn^{3/4})$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$O(mn + \kappa^{5/2}m)$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{4/3})$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{2/3})$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^2 m)$</td>
<td>no</td>
<td>Our result</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^{3/2}m^{1/2}n + \kappa^3n)$</td>
<td>no</td>
<td>Our result</td>
</tr>
</tbody>
</table>

Undirected graphs: $m \rightarrow nk$ [Nagamochi/Ibaraki ’92]
### State of the Art and Results

**Vertex connectivity in directed graphs:**

<table>
<thead>
<tr>
<th>Running time</th>
<th>Deterministic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{O}(n^{2.373} + nk^{2.373})$</td>
<td>no</td>
<td>[Cheriyan/Reif ’92]</td>
</tr>
<tr>
<td>$\tilde{O}(mn)$</td>
<td>no</td>
<td>[Henzinger et al. ’96]</td>
</tr>
<tr>
<td>$O(mn + \kappa mn^{3/4})$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$O(mn + \kappa^{5/2} m)$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{4/3})$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{2/3} n)$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^2 m)$</td>
<td>no</td>
<td>Our result</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^{3/2} m^{1/2} n + \kappa^3 n)$</td>
<td>no</td>
<td>Our result</td>
</tr>
</tbody>
</table>

**Undirected graphs:** $m \rightarrow nk$ [Nagamochi/Ibaraki ’92]

**State of the art for edge connectivity in directed graphs:** $\tilde{O}(\lambda m)$ [Gabow ’95]
## State of the Art and Results

### Vertex connectivity in directed graphs:

<table>
<thead>
<tr>
<th>Running time</th>
<th>Deterministic</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{O}(n^{2.373} + nk^{2.373})$</td>
<td>no</td>
<td>[Cheriyan/Reif ’92]</td>
</tr>
<tr>
<td>$\tilde{O}(mn)$</td>
<td>no</td>
<td>[Henzinger et al. ’96]</td>
</tr>
<tr>
<td>$O(mn + \kappa mn^{3/4})$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$O(mn + \kappa^{5/2}m)$</td>
<td>yes</td>
<td>[Gabow ’00]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{4/3})$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa m^{2/3}n)$</td>
<td>no</td>
<td>[Nanongkai et al. ’19]</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^2 m)$</td>
<td>no</td>
<td>Our result</td>
</tr>
<tr>
<td>$\tilde{O}(\kappa^{3/2} m^{1/2}n + \kappa n)$</td>
<td>no</td>
<td>Our result</td>
</tr>
</tbody>
</table>

### Undirected graphs: $m \rightarrow n\kappa$ [Nagamochi/Ibaraki ’92]

### State of the art for **edge connectivity** in directed graphs: $\tilde{O}(\lambda m)$ [Gabow ’95]

### Improvements also for finding $k$-edge connected subgraphs [Chechik et al. ’17]
Algorithm needs to distinguish between graphs that are $k$-connected and graphs that are ε-far from being $k$-connected (cannot be made $k$-connected by changing an ε-fraction of the edges). Want to minimize the number of edge queries to the graph.
Algorithm needs to distinguish between graphs that are $k$-connected and graphs that are $\epsilon$-far from being $k$-connected (cannot be made $k$-connected by changing an $\epsilon$-fraction of the edges). Want to minimize the number of edge queries to the graph.

**Graphs of bounded degree $d$:**

<table>
<thead>
<tr>
<th>Problem</th>
<th>State of the art</th>
<th>Our result</th>
</tr>
</thead>
<tbody>
<tr>
<td>undirected $k$-edge conn.</td>
<td>$\tilde{O}\left(\frac{k^3}{\epsilon^{3-\frac{2}{k}}d^{2-\frac{2}{k}}k}\right)$ [Goldreich/Ron ’02]</td>
<td>$\tilde{O}\left(\frac{k}{\epsilon}\right)$</td>
</tr>
<tr>
<td>directed $k$-edge conn.</td>
<td>$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Yoshida/Ito ’10]</td>
<td>$\tilde{O}\left(\frac{k}{\epsilon}\right)$</td>
</tr>
<tr>
<td>undirected $k$-vertex conn.</td>
<td>$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Yoshida/Ito ’12]</td>
<td>$\tilde{O}\left(\frac{k}{\epsilon}\right)$</td>
</tr>
<tr>
<td>directed $k$-vertex conn.</td>
<td>$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Orenstein/Ron ’11]</td>
<td>$\tilde{O}\left(\frac{k}{\epsilon}\right)$</td>
</tr>
</tbody>
</table>
Property Testing Results

Algorithm needs to distinguish between graphs that are \( k \)-connected and graphs that are \( \epsilon \)-far from being \( k \)-connected (cannot be made \( k \)-connected by changing an \( \epsilon \)-fraction of the edges). Want to minimize the number of edge queries to the graph.

Graphs of bounded degree \( d \):

<table>
<thead>
<tr>
<th>Problem</th>
<th>State of the art</th>
<th>Our result</th>
</tr>
</thead>
<tbody>
<tr>
<td>undirected ( k )-edge conn.</td>
<td>( \tilde{O} \left( \frac{k^3}{\epsilon^3 \frac{2}{k} d^2 \frac{2}{k}} \right) ) [Goldreich/Ron ’02]</td>
<td>( \tilde{O} \left( \frac{k}{\epsilon} \right) )</td>
</tr>
<tr>
<td>directed ( k )-edge conn.</td>
<td>( \tilde{O} \left( \left( \frac{ck}{\epsilon d} \right)^k d \right) ) [Yoshida/Ito ’10]</td>
<td>( \tilde{O} \left( \frac{k}{\epsilon} \right) )</td>
</tr>
<tr>
<td>undirected ( k )-vertex conn.</td>
<td>( \tilde{O} \left( \left( \frac{ck}{\epsilon d} \right)^k d \right) ) [Yoshida/Ito ’12]</td>
<td>( \tilde{O} \left( \frac{k}{\epsilon} \right) )</td>
</tr>
<tr>
<td>directed ( k )-vertex conn.</td>
<td>( \tilde{O} \left( \left( \frac{ck}{\epsilon d} \right)^k d \right) ) [Orenstein/Ron ’11]</td>
<td>( \tilde{O} \left( \frac{k}{\epsilon} \right) )</td>
</tr>
</tbody>
</table>

Similar improvements for graphs of unbounded degree (w.r.t. avg. degree)
Local Cut Problem

**Idea:** Detect smaller side of partition in time proportional to its volume (= number of interior + outgoing edges)
Local Cut Problem

**Idea:** Detect smaller side of partition in time proportional to its volume (= number of interior + outgoing edges)

A *k*-out component $U \subseteq V$ has at most $k$ edges going from $U$ to $V \setminus U$. 
Local Cut Problem

**Idea:** Detect smaller side of partition in time proportional to its volume (= number of interior + outgoing edges)

A \textit{k-out component} $U \subseteq V$ has at most $k$ edges going from $U$ to $V \setminus U$.

**Lemma**

There is a local procedure that, given a seed vertex $s$, a target cut size $k$ and a target volume $\Delta$ runs in time $O(k^2 \Delta)$, and returns as follows:

1. If $s$ is contained in an $\ell$-out component of volume $\leq \Delta$ for $\ell \leq k$, then it returns an $\ell$-out component of volume $\leq 3k\Delta$ with probability at least $\frac{1}{2}$

2. Otherwise, it might return a $k$-out-component or $\perp$
Local Cut Problem

Idea: Detect smaller side of partition in time proportional to its volume (= number of interior + outgoing edges)

A $k$-out component $U \subseteq V$ has at most $k$ edges going from $U$ to $V \setminus U$.

Lemma

There is a local procedure that, given a seed vertex $s$, a target cut size $k$ and a target volume $\Delta$ runs in time $O(k^2 \Delta)$, and returns as follows:

1. If $s$ is contained in an $\ell$-out component of volume $\leq \Delta$ for $\ell \leq k$, then it returns an $\ell$-out component of volume $\leq 3k\Delta$ with probability at least $\frac{1}{2}$

2. Otherwise, it might return a $k$-out-component or $\perp$

Core problem! Plugging in almost immediately implies our results!
Local Cut Problem

Idea: Detect smaller side of partition in time proportional to its volume (= number of interior + outgoing edges)

A $k$-out component $U \subseteq V$ has at most $k$ edges going from $U$ to $V \setminus U$.

Lemma

There is a local procedure that, given a seed vertex $s$, a target cut size $k$ and a target volume $\Delta$ runs in time $O(k^2 \Delta)$, and returns as follows:

1. If $s$ is contained in an $\ell$-out component of volume $\leq \Delta$ for $\ell \leq k$, then it returns an $\ell$-out component of volume $\leq 3k\Delta$ with probability at least $\frac{1}{2}$

2. Otherwise, it might return a $k$-out-component or $\perp$

Core problem! Plugging in almost immediately implies our results!

Prior work:

- “Local” version of Karger’s algorithm [Goldreich/Ron ’02]
- Exponential time [Orenstein/Ron ’11] [Chechik et al. ’17]
- Local flow techniques [Nanongkai/Saranurak/Yingchareonthawornchai ’19]
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return $\bot$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return $\bot$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

Algorithm:

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return $\perp$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return ⊥
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return $\bot$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return $\perp$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

Algorithm:

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return $\bot$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return $\bot$
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return ⊥
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex \( s \), target cut size \( \leq k \), target volume \( \leq \Delta \)

**Algorithm:**

- Repeat \( k + 1 \) times:
  - Perform depth-first-search from \( s \) processing up to \( 2k\Delta \) many edges
  - If DFS processes less than \( 2k\Delta \) edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let \( t \) be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from \( s \) to \( t \) in DFS tree

- Return \( \bot \)
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

**Algorithm:**
- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return ⊥
Randomization of Augmenting-Path Idea [Chechik et al. ’17]

Seed vertex $s$, target cut size $\leq k$, target volume $\leq \Delta$

Algorithm:

- Repeat $k + 1$ times:
  - Perform depth-first-search from $s$ processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let $t$ be tail of sampled edge (ignoring reversal of edge)
  - Reverse edges on path from $s$ to $t$ in DFS tree

- Return ⊥
Claim 1 [Chechik et al. ’17]

Let $U \subseteq V$ contain $s$, let $t \in V$, and reverse the edges on a path from $s$ to $t$.

- If $t \notin U$, then the number of edges leaving $U$ is reduced by one.
- Otherwise, the number of edges leaving $U$ stays the same.
Claim 1 [Chechik et al. ’17]

Let $U \subseteq V$ contain $s$, let $t \in V$, and reverse the edges on a path from $s$ to $t$.

- If $t \notin U$, then the number of edges leaving $U$ is reduced by one.
- Otherwise, the number of edges leaving $U$ stays the same.

Case 1: $t \notin U$

Case 2: $t \in U$
Claim 1 [Chechik et al. ’17]

Let $U \subseteq V$ contain $s$, let $t \in V$, and reverse the edges on a path from $s$ to $t$.

- If $t \notin U$, then the number of edges leaving $U$ is reduced by one.
- Otherwise, the number of edges leaving $U$ stays the same.

Case 1: $t \notin U$

Odd number of crossings

Case 2: $t \in U$

Even number of crossings
Claim 1 [Chechik et al. ’17]

Let $U \subseteq V$ contain $s$, let $t \in V$, and reverse the edges on a path from $s$ to $t$.
- If $t \notin U$, then the number of edges leaving $U$ is reduced by one.
- Otherwise, the number of edges leaving $U$ stays the same.

Case 1: $t \notin U$

Odd number of crossings

Case 2: $t \in U$

Even number of crossings
Claim 2
If the procedure returns a set of vertices $U$ in iteration $\ell + 1$, then $U$ is an $\ell$-out-component with $\text{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$. 

Idea:
For component found by DFS, number of out-edges reduces by at most one in each iteration.
Analysis II

Claim 2
If the procedure returns a set of vertices $U$ in iteration $\ell + 1$, then $U$ is an $\ell$-out-component with $\text{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

Idea: For component found by DFS, number of out-edges reduces by at most one in each iteration
Claim 2

If the procedure returns a set of vertices $U$ in iteration $\ell + 1$, then $U$ is an $\ell$-out-component with $\text{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

Idea: For component found by DFS, number of out-edges reduces by at most one in each iteration.

Claim 3

If there is an $\ell$-out-component $C$ of volume $\leq \Delta$ containing $s$ for $\ell \leq k$, then the procedure returns an $\ell$-out-component with probability $\geq \frac{1}{2}$. 
**Claim 2**

If the procedure returns a set of vertices $U$ in iteration $\ell + 1$, then $U$ is an $\ell$-out-component with $\text{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

**Idea:** For component found by DFS, number of out-edges reduces by at most one in each iteration.

**Claim 3**

If there is an $\ell$-out-component $C$ of volume $\leq \Delta$ containing $s$ for $\ell \leq k$, then the procedure returns an $\ell$-out-component with probability $\geq \frac{1}{2}$.

**Proof**

- Algorithm succeeds if in first $k$ iterations always tail of sampled edge outside of component $C$ (known to exist)
Claim 2
If the procedure returns a set of vertices $U$ in iteration $\ell + 1$, then $U$ is an $\ell$-out-component with $\text{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

Idea: For component found by DFS, number of out-edges reduces by at most one in each iteration

Claim 3
If there is an $\ell$-out-component $C$ of volume $\leq \Delta$ containing $s$ for $\ell \leq k$, then the procedure returns an $\ell$-out-component with probability $\geq \frac{1}{2}$.

Proof
- Algorithm succeeds if in first $k$ iterations always tail of sampled edge outside of component $C$ (known to exist)
- $\text{vol}(C) \leq \Delta$ and DFS processes $= 2k\Delta$ many edges
Analysis II

Claim 2
If the procedure returns a set of vertices $U$ in iteration $\ell + 1$, then $U$ is an $\ell$-out-component with $\text{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

Idea: For component found by DFS, number of out-edges reduces by at most one in each iteration

Claim 3
If there is an $\ell$-out-component $C$ of volume $\leq \Delta$ containing $s$ for $\ell \leq k$, then the procedure returns an $\ell$-out-component with probability $\geq \frac{1}{2}$.

Proof
- Algorithm succeeds if in first $k$ iterations always tail of sampled edge outside of component $C$ (known to exist)
- $\text{vol}(C) \leq \Delta$ and DFS processes $= 2k\Delta$ many edges
- Tail of sampled edge will lie inside of $C$ with probability $\leq \frac{1}{2k}$
Analysis II

Claim 2
If the procedure returns a set of vertices $U$ in iteration $\ell + 1$, then $U$ is an $\ell$-out-component with $\text{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

Idea: For component found by DFS, number of out-edges reduces by at most one in each iteration

Claim 3
If there is an $\ell$-out-component $C$ of volume $\leq \Delta$ containing $s$ for $\ell \leq k$, then the procedure returns an $\ell$-out-component with probability $\geq \frac{1}{2}$.

Proof
- Algorithm succeeds if in first $k$ iterations always tail of sampled edge outside of component $C$ (known to exist)
- $\text{vol}(C) \leq \Delta$ and DFS processes $= 2k\Delta$ many edges
- Tail of sampled edge will lie inside of $C$ with probability $\leq \frac{1}{2k}$
- By Union Bound: algorithms fails with probability $\leq \frac{1}{2}$
Conclusion

Extensions:

1. Extension to vertex connectivity
   Standard reduction (directed!) with some minor adjustments

Summary:
Significant progress for fundamental graph problems
Local procedure was pivotal to be/t_ter time/query complexities
Exponential improvement:
from $O\left(2^{O(k)\Delta}\right)$ [Chechik et al. '17] to $O(k^2\Delta)$
at the cost of randomization
Conclusion

Extensions:

1. Extension to vertex connectivity
   Standard reduction (directed!) with some minor adjustments

2. Approximation version
   Sampling only outside of component in a fraction of cases

Summary:

Significant progress for fundamental graph problems
Local procedure was pivotal to be/t_ter time/query complexities
Exponential improvement: from $O(2^{O(k)} \Delta)$ to $O(k^2 \Delta)$ at the cost of randomization
Conclusion

Extensions:

1. Extension to vertex connectivity
   Standard reduction (directed!) with some minor adjustments

2. Approximation version
   Sampling only outside of component in a fraction of cases

3. Can save a factor of $k$ in query complexity
   (Useful for property testing)
Conclusion

Extensions:

1. Extension to vertex connectivity
   Standard reduction (directed!) with some minor adjustments

2. Approximation version
   Sampling only outside of component in a fraction of cases

3. Can save a factor of $k$ in query complexity
   (Useful for property testing)

Summary:

- Significant progress for fundamental graph problems
Conclusion

Extensions:

1. Extension to vertex connectivity
   Standard reduction (directed!) with some minor adjustments

2. Approximation version
   Sampling only outside of component in a fraction of cases

3. Can save a factor of $k$ in query complexity
   (Useful for property testing)

Summary:

- Significant progress for fundamental graph problems
- Local procedure was pivotal to better time/query complexities
Conclusion

Extensions:

1. Extension to vertex connectivity
   Standard reduction (directed!) with some minor adjustments

2. Approximation version
   Sampling only outside of component in a fraction of cases

3. Can save a factor of $k$ in query complexity
   (Useful for property testing)

Summary:

- Significant progress for fundamental graph problems
- Local procedure was pivotal to better time/query complexities

*Exponential improvement:* from $O(2^{O(k)} \Delta)$ [Chechik et al. ’17] to $O(k^2 \Delta)$ at the cost of randomization
Thank you!