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 λ/vertex connectivity κ

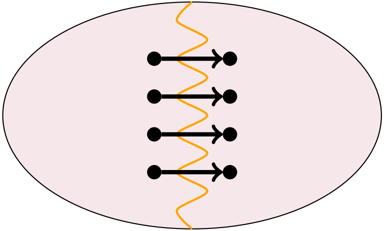
Minimum number of edges/vertices to remove in order to make the graph not strongly connected

Edge and Vertex Connectivity

Edge connectivity λ/vertex connectivity κ

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

Edge cut:

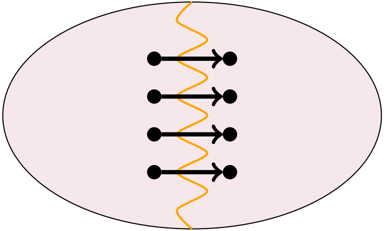


Edge and Vertex Connectivity

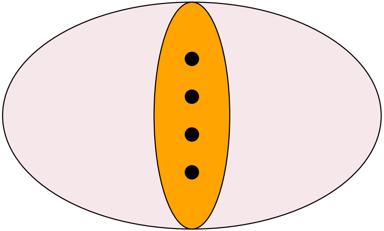
Edge connectivity λ/vertex connectivity κ

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

Edge cut:



Vertex cut:

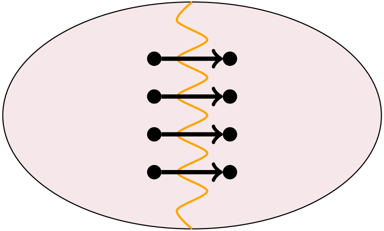


Edge and Vertex Connectivity

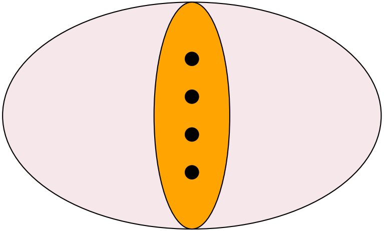
Edge connectivity λ/vertex connectivity κ

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

Edge cut:



Vertex cut:



Motivation:

- Fundamental graph-theoretic notion
- Applications: Reliability analysis, community detection

State of the Art and Results

Vertex connectivity in directed graphs:

Running time	Deterministic	Reference
$\tilde{O}(n^{2.373} + n\kappa^{2.373})$	no	[Cheriyan/Reif '92]
$\tilde{O}(mn)$	no	[Henzinger et al. '96]
$O(mn + \kappa mn^{3/4})$	yes	[Gabow '00]
$O(mn + \kappa^{5/2}m)$	yes	[Gabow '00]
$\tilde{O}(\kappa m^{4/3})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa m^{2/3}n)$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa^2 m)$	no	Our result
$\tilde{O}(\kappa^{3/2}m^{1/2}n + \kappa^3 n)$	no	Our result

State of the Art and Results

Vertex connectivity in directed graphs:

Running time	Deterministic	Reference
$\tilde{O}(n^{2.373} + n\kappa^{2.373})$	no	[Cheriyan/Reif '92]
$\tilde{O}(mn)$	no	[Henzinger et al. '96]
$O(mn + \kappa mn^{3/4})$	yes	[Gabow '00]
$O(mn + \kappa^{5/2}m)$	yes	[Gabow '00]
$\tilde{O}(\kappa m^{4/3})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa m^{2/3}n)$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa^2 m)$	no	Our result
$\tilde{O}(\kappa^{3/2}m^{1/2}n + \kappa^3 n)$	no	Our result

Undirected graphs: $m \to n\kappa$ [Nagamochi/Ibaraki '92]

State of the Art and Results

Vertex connectivity in directed graphs:

Running time	Deterministic	Reference
$\tilde{O}(n^{2.373} + n\kappa^{2.373})$	no	[Cheriyan/Reif '92]
$\tilde{O}(mn)$	no	[Henzinger et al. '96]
$O(mn + \kappa mn^{3/4})$	yes	[Gabow '00]
$O(mn + \kappa^{5/2}m)$	yes	[Gabow '00]
$\tilde{O}(\kappa m^{4/3})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa m^{2/3}n)$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa^2 m)$	no	Our result
$\tilde{O}(\kappa^{3/2}m^{1/2}n + \kappa^3 n)$	no	Our result

Undirected graphs: $m \to n\kappa$ [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:

Running time	Deterministic	Reference
$\tilde{O}(n^{2.373} + n\kappa^{2.373})$	no	[Cheriyan/Reif '92]
$\tilde{O}(mn)$	no	[Henzinger et al. '96]
$O(mn + \kappa mn^{3/4})$	yes	[Gabow '00]
$O(mn + \kappa^{5/2}m)$	yes	[Gabow '00]
$\tilde{O}(\kappa m^{4/3})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa m^{2/3}n)$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa^2 m)$	no	Our result
$\tilde{O}(\kappa^{3/2}m^{1/2}n + \kappa^3 n)$	no	Our result

Undirected graphs: $m \to n\kappa$ [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]

Property Testing Results

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.

Property Testing Results

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.

Graphs of bounded degree d:

Problem	State of the art	Our result
undirected k-edge conn.	$\tilde{O}\left(\frac{k^3}{\epsilon^{3-\frac{2}{k}} d^{2-\frac{2}{k}}}\right)$ [Goldreich/Ron '02]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$
directed k-edge conn.	$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Yoshida/Ito '10]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$
undirected k-vertex conn.	$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Yoshida/Ito '12]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$
directed k-vertex conn.	$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Orenstein/Ron '11]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$

Property Testing Results

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.

Graphs of bounded degree d:

Problem	State of the art	Our result
undirected k-edge conn.	$\tilde{O}\left(\frac{k^3}{\epsilon^{3-\frac{2}{k}} d^{2-\frac{2}{k}}}\right)$ [Goldreich/Ron '02]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$
directed k-edge conn.	$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Yoshida/Ito '10]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$
undirected k-vertex conn.	$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Yoshida/Ito '12]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$
directed k-vertex conn.	$\tilde{O}\left(\left(\frac{ck}{\epsilon d}\right)^k d\right)$ [Orenstein/Ron '11]	$\tilde{O}\left(\frac{k}{\epsilon}\right)$

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 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 Δ runs in time $O(k^2\Delta)$, and returns as follows:

1. *If s is contained in an ℓ-out component of volume $\leq \Delta$ for $\ell \leq k$, then it returns an ℓ-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 Δ runs in time $O(k^2\Delta)$, and returns as follows:

1. *If s is contained in an ℓ-out component of volume $\leq \Delta$ for $\ell \leq k$, then it returns an ℓ-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 Δ runs in time $O(k^2\Delta)$, and returns as follows:

1. *If s is contained in an ℓ-out component of volume $\leq \Delta$ for $\ell \leq k$, then it returns an ℓ-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 $\perp$

Randomization of Augmenting-Path Idea [Chechik et al. '17]

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

Algorithm:

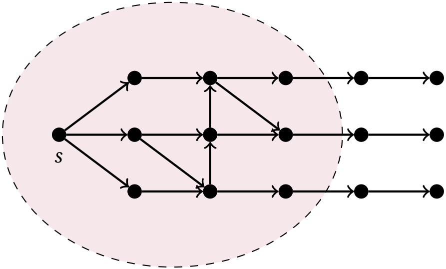
- Repeat $k + 1$ times:
  - Perform depth-first-search from s processing up to $2k\Delta$ many edges
  - If DFS processes less than $2k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let t be tail of sampled edge (ignoring reversal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

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 $\perp$

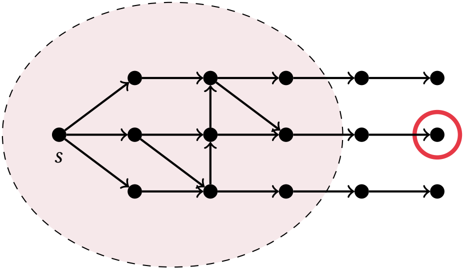


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$

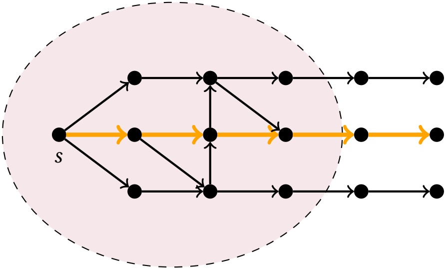


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$

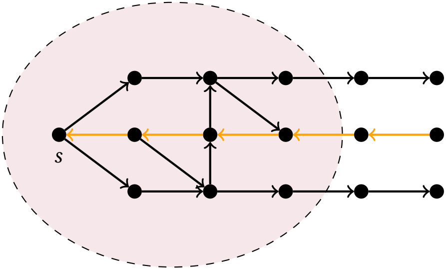


Randomization of Augmenting-Path Idea [Chechik et al. ’17]

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 $\perp$

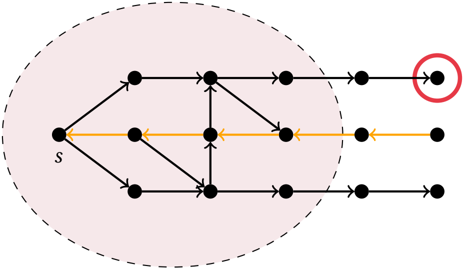


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$

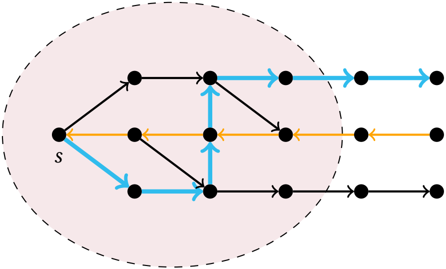


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$

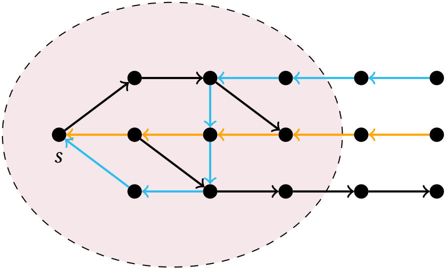


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$

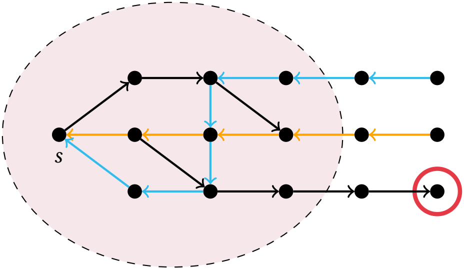


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$

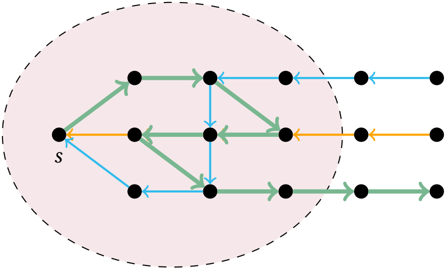


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$

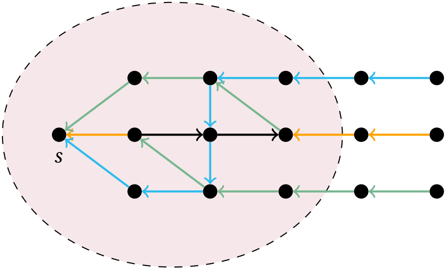


Randomization of Augmenting-Path Idea [Chechik et al. '17]

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 $\perp$



Analysis I

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.

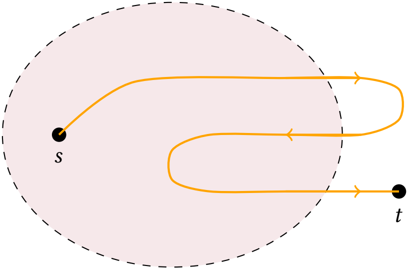
Analysis I

Claim 1 [Chechik et al. '17]

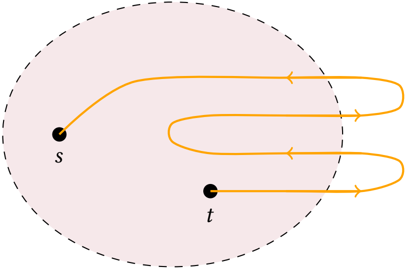
Let $U \subseteq V$ contain s, let $t \in V$, and reverse the edges on a path from s to t.

- 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: $t \in U$



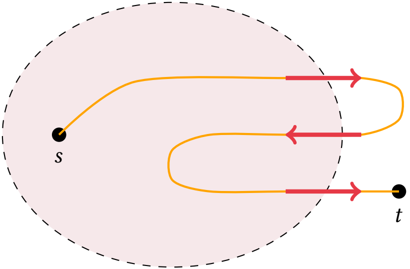
Analysis I

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.

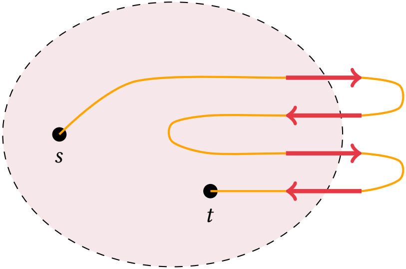
- If $t \notin U$, then the number of edges leaving U is reduced by one.
- Otherwise, the number of edges leaving U stays the same.

Case 1: $t \notin U$



Odd number of crossings

Case 2: $t \in U$



Even number of crossings

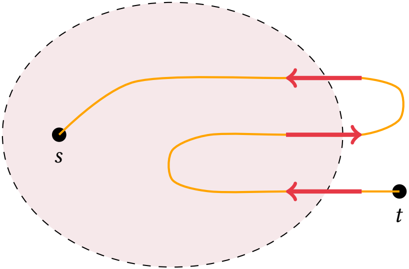
Analysis I

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.

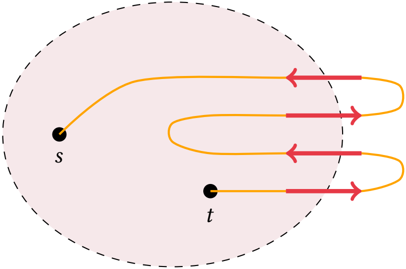
- If $t \notin U$, then the number of edges leaving U is reduced by one.
- Otherwise, the number of edges leaving U stays the same.

Case 1: $t \notin U$



Odd number of crossings

Case 2: $t \in U$



Even number of crossings

Analysis II

Claim 2

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

Analysis II

Claim 2

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

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 ℓ-out-component with $\mathrm{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

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

Claim 3

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

Analysis II

Claim 2

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

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

Claim 3

If there is an ℓ-out-component C of volume $\leq \Delta$ containing s for $\ell \leq k$, then the procedure returns an ℓ-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)

Analysis II

Claim 2

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

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

Claim 3

If there is an ℓ-out-component C of volume $\leq \Delta$ containing s for $\ell \leq k$, then the procedure returns an ℓ-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)
- $\mathrm{vol}(C) \leq \Delta$ 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 ℓ-out-component with $\mathrm{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

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

Claim 3

If there is an ℓ-out-component C of volume $\leq \Delta$ containing s for $\ell \leq k$, then the procedure returns an ℓ-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)
- $\mathrm{vol}(C) \leq \Delta$ 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 ℓ-out-component with $\mathrm{vol}(U) \leq 2k\Delta + \ell \leq 3k\Delta$.

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

Claim 3

If there is an ℓ-out-component C of volume $\leq \Delta$ containing s for $\ell \leq k$, then the procedure returns an ℓ-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)
- $\mathrm{vol}(C) \leq \Delta$ 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

Conclusion

Extensions:

1. Extension to vertex connectivity
   Standard reduction (directed!) with some minor adjustments
2. Appproximation version
   Sampling only outside of component in a fraction of cases

Conclusion

Extensions:

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