Computing and Testing Small Connectivity in Near-Linear Time and Queries via Fast Local Cut Algorithms [SODA '20]

Reading Group Algorithms

Sebastian Forster

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

Universität Salzburg

18.11.2019

$G = (V, E)$

Definitions

Definition

A (directed) graph G is called (strongly) connected if for every pair of vertices $s, t \in V$ there is a path from s to t in G.

Definitions

Definition

A (directed) graph G is called (strongly) connected if for every pair of vertices $s, t \in V$ there is a path from s to t in G.

Definition

An edge cut F is a subset of edges $F \subseteq E$ that disconnects the graph, i.e., the graph $G' = (V, E \setminus F)$ is not (strongly) connected.

Definitions

Definition

A (directed) graph G is called (strongly) connected if for every pair of vertices $s, t \in V$ there is a path from s to t in G.

Definition

An edge cut F is a subset of edges $F \subseteq E$ that disconnects the graph, i.e., the graph $G' = (V, E \setminus F)$ is not (strongly) connected.

Definition

A vertex cut U is a subset of vertices $U \subseteq V$ that disconnects the graph, i.e., the graph $G' = (V \setminus U, E \setminus (V \times U \cup U \times V))$ is not (strongly) connected.

Cuts and Partitions

Observation

For every edge cut F, there is an induced partition (L, R) such that $L \cap R = \emptyset$, $L \cup R = V$, and there F is the set of edges from L to R.

Cuts and Partitions

Observation

For every edge cut F, there is an induced partition (L, R) such that $L \cap R = \emptyset$, $L \cup R = V$, and there F is the set of edges from L to R.

Observation

For every vertex cut U, there is a partition (L, U, R) such that L, M, R are pairwise disjoint, $L \cup U \cup R = V$, and there are no edges from L to R.

Cuts and Partitions

Observation

For every edge cut F, there is an induced partition (L, R) such that $L \cap R = \emptyset$, $L \cup R = V$, and there F is the set of edges from L to R.

Observation

For every vertex cut U, there is a partition (L, U, R) such that L, M, R are pairwise disjoint, $L \cup U \cup R = V$, and there are no edges from L to R.

Definition

The edge connectivity λ of a graph is the size of its smallest edge cut and the vertex connectivity κ is the size of its smallest vertex cut.

Cuts and Partitions

Observation

For every edge cut F, there is an induced partition (L, R) such that $L \cap R = \emptyset$, $L \cup R = V$, and there F is the set of edges from L to R.

Observation

For every vertex cut U, there is a partition (L, U, R) such that L, M, R are pairwise disjoint, $L \cup U \cup R = V$, and there are no edges from L to R.

Definition

The edge connectivity λ of a graph is the size of its smallest edge cut and the vertex connectivity κ is the size of its smallest vertex cut.

Attention: Common definitions disagree on corner cases

Cuts and Partitions

Observation

For every edge cut F, there is an induced partition (L, R) such that $L \cap R = \emptyset$, $L \cup R = V$, and there F is the set of edges from L to R.

Observation

For every vertex cut U, there is a partition (L, U, R) such that L, M, R are pairwise disjoint, $L \cup U \cup R = V$, and there are no edges from L to R.

Definition

The edge connectivity λ of a graph is the size of its smallest edge cut and the vertex connectivity κ is the size of its smallest vertex cut.

Attention: Common definitions disagree on corner cases

Motivation for computing higher connectivity:

- Reliability analysis
- Community detection

State of the Art

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]
$O(mn + \kappa m \cdot \min\{n^{3/4}, \kappa^{3/2}\})$	yes	[Gabow '00]
$\tilde{O}(\kappa \cdot \min\{m^{4/3}, m^{2/3}n\})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa \cdot \min\{\kappa m, \kappa^{1/2}m^{1/2}n + \kappa^2 n\})$	no	**Our result**

State of the Art

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]
$O(mn + \kappa m \cdot \min\{n^{3/4}, \kappa^{3/2}\})$	yes	[Gabow '00]
$\tilde{O}(\kappa \cdot \min\{m^{4/3}, m^{2/3}n\})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa \cdot \min\{\kappa m, \kappa^{1/2}m^{1/2}n + \kappa^2 n\})$	no	**Our result**

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

State of the Art

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]
$O(mn + \kappa m \cdot \min\{n^{3/4}, \kappa^{3/2}\})$	yes	[Gabow '00]
$\tilde{O}(\kappa \cdot \min\{m^{4/3}, m^{2/3}n\})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa \cdot \min\{\kappa m, \kappa^{1/2}m^{1/2}n + \kappa^2 n\})$	no	**Our result**

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

Plan for today:

Theorem

There is an algorithm to compute the edge connectivity λ of a directed graph in time $O(\lambda^2 m \log n)$ with success probability $1/2$.

State of the Art

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]
$O(mn + \kappa m \cdot \min\{n^{3/4}, \kappa^{3/2}\})$	yes	[Gabow '00]
$\tilde{O}(\kappa \cdot \min\{m^{4/3}, m^{2/3}n\})$	no	[Nanongkai et al. '19]
$\tilde{O}(\kappa \cdot \min\{\kappa m, \kappa^{1/2}m^{1/2}n + \kappa^2 n\})$	no	**Our result**

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

Plan for today:

Theorem

There is an algorithm to compute the edge connectivity λ of a directed graph in time $O(\lambda^2 m \log n)$ with success probability $1/2$.

- Covers main technique, extension to vertex connectivity is a technicality
- In general: $O(\lambda^2 m \log n \log \frac{1}{p})$ with success probability p
- State of the art for directed edge connectivity: $O(\lambda m \log n)$ [Gabow '91]

Review of Naive Algorithm

Definition

An s-t edge cut is a cut with induced partition (L, R) such that $s \in L$ and $t \in R$.

Review of Naive Algorithm

Definition

An s-t edge cut is a cut with induced partition (L, R) such that $s \in L$ and $t \in R$.

Observation

The edge connectivity λ is the minimum size of any s-t edge cut among all pairs of vertices s and t.

Review of Naive Algorithm

Definition

An s-t edge cut is a cut with induced partition (L, R) such that $s \in L$ and $t \in R$.

Observation

The edge connectivity λ is the minimum size of any s-t edge cut among all pairs of vertices s and t.

Algorithm:

- For every pair of vertices s and t compute the minimum s-t cut
- Return minimum-size cut among all returned cuts

Review of Naive Algorithm

Definition

An s-t edge cut is a cut with induced partition (L, R) such that $s \in L$ and $t \in R$.

Observation

The edge connectivity λ is the minimum size of any s-t edge cut among all pairs of vertices s and t.

Algorithm:

- For every pair of vertices s and t compute the minimum s-t cut
- Return minimum-size cut among all returned cuts

By "max s-t flow = min s-t cut", the minimum s-t cut can be computed with the Ford-Fulkerson algorithm in time $O(mn)$.

Review of Naive Algorithm

Definition

An s-t edge cut is a cut with induced partition (L, R) such that $s \in L$ and $t \in R$.

Observation

The edge connectivity λ is the minimum size of any s-t edge cut among all pairs of vertices s and t.

Algorithm:

- For every pair of vertices s and t compute the minimum s-t cut
- Return minimum-size cut among all returned cuts

By "max s-t flow = min s-t cut", the minimum s-t cut can be computed with the Ford-Fulkerson algorithm in time $O(mn)$.

Running time of algorithm above: $O(n^3 m)$

Naive Algorithm – Doubling Approach

Ford-Fulkerson algorithm with parameters s, t, k

The algorithm runs in time $O(km)$ and if $k \geq \lambda$, then the algorithm returns the minimum s-t cut; otherwise it returns $\perp$.

Naive Algorithm – Doubling Approach

Ford-Fulkerson algorithm with parameters s, t, k

The algorithm runs in time $O(km)$ and if $k \geq \lambda$, then the algorithm returns the minimum s-t cut; otherwise it returns $\perp$.

Algorithm:

- For $i = 1$ to $r = \lceil \log n \rceil$
  - Set $k_i = 2^i$
  - For every pair of vertices s and t: run the Ford-Fulkerson algorithm with parameters s, t, and k_i
  - If one of the Ford-Fulkerson instances returns a cut, then return the minimum-size cut among all returned cuts

Naive Algorithm – Doubling Approach

Ford-Fulkerson algorithm with parameters s, t, k

The algorithm runs in time $O(km)$ and if $k \geq \lambda$, then the algorithm returns the minimum s-t cut; otherwise it returns $\perp$.

Algorithm:

- For $i = 1$ to $r = \lceil \log n \rceil$
  - Set $k_i = 2^i$
  - For every pair of vertices s and t: run the Ford-Fulkerson algorithm with parameters s, t, and k_i
  - If one of the Ford-Fulkerson instances returns a cut, then return the minimum-size cut among all returned cuts

Running time: $\sum_{i=1}^{r} O(n^2 k_i m) = O(\lambda n^2 m)$

Naive Algorithm – Doubling Approach

Ford-Fulkerson algorithm with parameters s, t, k

The algorithm runs in time $O(km)$ and if $k \geq \lambda$, then the algorithm returns the minimum s-t cut; otherwise it returns $\perp$.

Algorithm:

- For $i = 1$ to $r = \lceil \log n \rceil$
  - Set $k_i = 2^i$
  - For every pair of vertices s and t: run the Ford-Fulkerson algorithm with parameters s, t, and k_i
  - If one of the Ford-Fulkerson instances returns a cut, then return the minimum-size cut among all returned cuts

Running time: $\sum_{i=1}^{r} O(n^2 k_i m) = O(\lambda n^2 m)$

Observation

It suffices to design an algorithm that returns a global minimum cut if parameter $k \geq \lambda$.

Sampling Approach

Idea: The problem is easy if the partition induced by the minimum cut is somewhat *balanced*.

Sampling Approach

Idea: The problem is easy if the partition induced by the minimum cut is somewhat *balanced*.

Definition

The *volume* $\mathrm{vol}(U)$ of a set of vertices U is the sum of the outgoing edges of vertices in U.

Sampling Approach

Idea: The problem is easy if the partition induced by the minimum cut is somewhat *balanced*.

Definition

The *volume* $\mathrm{vol}(U)$ of a set of vertices U is the sum of the outgoing edges of vertices in U.

Volume = interior edges + leaving edges

Sampling Approach

Idea: The problem is easy if the partition induced by the minimum cut is somewhat *balanced*.

Definition

The *volume* $\text{vol}(U)$ of a set of vertices U is the sum of the outgoing edges of vertices in U.

Volume = interior edges + leaving edges

Definition

An edge cut F is balanced if for its induced partition (L, R) both $\text{vol}(L) \geq \frac{m}{14k}$ and $\text{vol}(R) \geq \frac{m}{14k}$.

Sampling Approach

Idea: The problem is easy if the partition induced by the minimum cut is somewhat *balanced*.

Definition

The *volume* $\mathrm{vol}(U)$ of a set of vertices U is the sum of the outgoing edges of vertices in U.

Volume = interior edges + leaving edges

Definition

An edge cut F is balanced if for its induced partition (L, R) both $\mathrm{vol}(L) \geq \frac{m}{14k}$ and $\mathrm{vol}(R) \geq \frac{m}{14k}$.

Lemma

For any edge (u, v) chosen from E uniformly at random, the tail u is contained in L with probability $\frac{\mathrm{vol}(L)}{m} \geq \frac{1}{14k}$ (same with R).

Case 1: Minimum Cut is Balanced [Nanongkai et al. '19]

Algorithm:

- Repeat $28k$ times:
  - Sample two edges e and f uniformly at random
  - Let s be the tail of e and let t be the tail of f
  - Run Ford-Fulkerson algorithm with parameters s, t, and k
- Return minimum-size cut among all returned cuts

Case 1: Minimum Cut is Balanced [Nanongkai et al. '19]

Algorithm:

- Repeat $28k$ times:
  - Sample two edges e and f uniformly at random
  - Let s be the tail of e and let t be the tail of f
  - Run Ford-Fulkerson algorithm with parameters s, t, and k
- Return minimum-size cut among all returned cuts

Lemma

If $k \geq \lambda$ and the minimum cut is balanced, then the algorithm above runs in time $O(k^2 m)$ and finds a cut of size λ with probability at least $\frac{1}{2}$.

Case 2: Minimum cut is not Balanced

Assumption: $\text{vol}(L) < \frac{m}{14k}$ or $\text{vol}(R) < \frac{m}{14k}$

Case 2: Minimum cut is not Balanced

Assumption: $\mathrm{vol}(L) < \frac{m}{14k}$ or $\mathrm{vol}(R) < \frac{m}{14k}$

Idea: Detect smaller side of partition time proportional to its volume

Case 2: Minimum cut is not Balanced

Assumption: $\text{vol}(L) < \frac{m}{14k}$ or $\text{vol}(R) < \frac{m}{14k}$

Idea: Detect smaller side of partition time proportional to its volume

Definition

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

Case 2: Minimum cut is not Balanced

Assumption: $\mathrm{vol}(L) < \frac{m}{14k}$ or $\mathrm{vol}(R) < \frac{m}{14k}$

Idea: Detect smaller side of partition time proportional to its volume

Definition

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 $\leq 7k\Delta$ with probability at least $\frac{5}{6}$ (and $\perp$ with probability at most $\frac{1}{6}$).*
2. *Otherwise, it might return a k-out-component or $\perp$*

Case 2: Minimum cut is not Balanced

Assumption: $\mathrm{vol}(L) < \frac{m}{14k}$ or $\mathrm{vol}(R) < \frac{m}{14k}$

Idea: Detect smaller side of partition time proportional to its volume

Definition

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 $\leq 7k\Delta$ with probability at least $\frac{5}{6}$ (and $\perp$ with probability at most $\frac{1}{6}$).*
2. *Otherwise, it might return a k-out-component or $\perp$*

Note: $k^2\Delta$ may be much smaller than m. **Sublinear running time!**

Case 2: Minimum cut is not Balanced (ctd.)

Assumption: $\text{vol}(L) < \frac{m}{14k}$ or $\text{vol}(R) < \frac{m}{14k}$

Case 2: Minimum cut is not Balanced (ctd.)

Assumption: $\text{vol}(L) < \frac{m}{14k}$ or $\text{vol}(R) < \frac{m}{14k}$

Algorithm:

- For $i = 1$ to $r = \lfloor \log \frac{m}{7k} \rfloor$
  - Repeat $\lceil \frac{m}{2^{i-1}} \rceil$ times
    - ★ Sample an edge e uniformly at random and let s be its tail
    - ★ Try to find a k-out-component using the local procedure with parameters s, k and $\Delta_i = 2^i - 1$
    - ★ Try to find a k-in-component using the local procedure on the reverse graph with parameters s, k and $\Delta_i = 2^i - 1$
- Return the minimum-size cut among all found cuts

Case 2: Minimum cut is not Balanced (ctd.)

Assumption: $\text{vol}(L) < \frac{m}{14k}$ or $\text{vol}(R) < \frac{m}{14k}$

Algorithm:

- For $i = 1$ to $r = \lfloor \log \frac{m}{7k} \rfloor$
  - Repeat $\lceil \frac{m}{2^{i-1}} \rceil$ times
    - Sample an edge e uniformly at random and let s be its tail
    - Try to find a k-out-component using the local procedure with parameters s, k and $\Delta_i = 2^i - 1$
    - Try to find a k-in-component using the local procedure on the reverse graph with parameters s, k and $\Delta_i = 2^i - 1$
- Return the minimum-size cut among all found cuts

Running time: $\sum_{i=1}^{r} \frac{m}{2^{i-1}} \cdot O(k^2 2^i) = O(k^2 m \log n)$

Case 2: Minimum cut is not Balanced (ctd.)

Assumption: $\mathrm{vol}(L) < \frac{m}{14k}$ or $\mathrm{vol}(R) < \frac{m}{14k}$

Algorithm:

- For $i = 1$ to $r = \lfloor \log \frac{m}{7k} \rfloor$
  - Repeat $\lceil \frac{m}{2^{i-1}} \rceil$ times
    - Sample an edge e uniformly at random and let s be its tail
    - Try to find a k-out-component using the local procedure with parameters s, k and $\Delta_i = 2^i - 1$
    - Try to find a k-in-component using the local procedure on the reverse graph with parameters s, k and $\Delta_i = 2^i - 1$
- Return the minimum-size cut among all found cuts

Running time: $\sum_{i=1}^{r} \frac{m}{2^{i-1}} \cdot O(k^2 2^i) = O(k^2 m \log n)$

Lemma

If the minimum cut is not balanced, then the algorithm above returns a proper λ-out-component $L' \subset V$ or a proper λ-out-component $R' \subset V$ (inducing a minimum cut) with probability at least $\frac{1}{2}$.

Case 2: Minimum cut is not Balanced (ctd.)

Assumption: $\mathrm{vol}(L) < \frac{m}{14k}$ or $\mathrm{vol}(R) < \frac{m}{14k}$

Algorithm:

- For $i = 1$ to $r = \lfloor \log \frac{m}{7k} \rfloor$
  - Repeat $\lceil \frac{m}{2^{i-1}} \rceil$ times
    - Sample an edge e uniformly at random and let s be its tail
    - Try to find a k-out-component using the local procedure with parameters s, k and $\Delta_i = 2^i - 1$
    - Try to find a k-in-component using the local procedure on the reverse graph with parameters s, k and $\Delta_i = 2^i - 1$
- Return the minimum-size cut among all found cuts

Running time: $\sum_{i=1}^{r} \frac{m}{2^{i-1}} \cdot O(k^2 2^i) = O(k^2 m \log n)$

Lemma

If the minimum cut is not balanced, then the algorithm above returns a proper λ-out-component $L' \subset V$ or a proper λ-out-component $R' \subset V$ (inducing a minimum cut) with probability at least $\frac{1}{2}$.

Note: Parameter choice ensures that $\mathrm{vol}(L') < m$ or $\mathrm{vol}(R') < m$

Local Procedure

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

Local Procedure

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

Algorithm: (with sampling idea of [Nanongkai et al. '19])

- Repeat $k + 1$ times:
  - Perform a depth-first-search from s processing up to $6k\Delta$ many edges
  - If DFS processes less than $6k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let t be the tail of the sampled edge (ignoring reversal of edge)
  - Reverse the edges on the DFS path from s to t
- Return $\perp$

Local Procedure

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

Algorithm: (with sampling idea of [Nanongkai et al. '19])

- Repeat $k + 1$ times:
  - Perform a depth-first-search from s processing up to $6k\Delta$ many edges
  - If DFS processes less than $6k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let t be the tail of the sampled edge (ignoring reversal of edge)
  - Reverse the edges on the DFS path from s to t
- Return $\perp$

Running time: $O(k^2\Delta)$

Local Procedure

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

Algorithm: (with sampling idea of [Nanongkai et al. '19])

- Repeat $k + 1$ times:
  - Perform a depth-first-search from s processing up to $6k\Delta$ many edges
  - If DFS processes less than $6k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let t be the tail of the sampled edge (ignoring reversal of edge)
  - Reverse the edges on the DFS path from s to t
- Return $\perp$

Running time: $O(k^2\Delta)$

Claim 1

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

- If $t \in V \setminus U$, then the number of edges from U to $V \setminus U$ is reduced by one by the reversing the edges.
- Otherwise, the number of edges from U to $V \setminus U$ stays the same.

Local Procedure

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

Algorithm: (with sampling idea of [Nanongkai et al. '19])

- Repeat $k + 1$ times:
  - Perform a depth-first-search from s processing up to $6k\Delta$ many edges
  - If DFS processes less than $6k\Delta$ edges, return set of visited vertices
  - Sample one of the edges processed in the DFS uniformly at random
  - Let t be the tail of the sampled edge (ignoring reversal of edge)
  - Reverse the edges on the DFS path from s to t
- Return $\perp$

Running time: $O(k^2\Delta)$

Claim 1

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

- If $t \in V \setminus U$, then the number of edges from U to $V \setminus U$ is reduced by one by the reversing the edges.
- Otherwise, the number of edges from U to $V \setminus U$ stays the same.

Idea: Odd or even number of crossings

Correctness Proof

Claim 2

If the procedure returns a set of vertices U in iteration $\ell + 1$, then U is an ℓ-out-component with $\mathrm{vol}(U) \leq 6k\Delta + \ell \leq 7k\Delta$.

Correctness Proof

Claim 2

If the procedure returns a set of vertices U in iteration $\ell + 1$, then U is an ℓ-out-component with $\mathrm{vol}(U) \leq 6k\Delta + \ell \leq 7k\Delta$.

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

Correctness Proof

Claim 2

If the procedure returns a set of vertices U in iteration $\ell + 1$, then U is an ℓ-out-component with $\mathrm{vol}(U) \leq 6k\Delta + \ell \leq 7k\Delta$.

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 of volume $\leq \Delta$ containing s for $\ell \leq k$, then the procedure returns an ℓ-out-component with probability $\geq \frac{5}{6}$.

Correctness Proof

Claim 2

If the procedure returns a set of vertices U in iteration $\ell + 1$, then U is an ℓ-out-component with $\mathrm{vol}(U) \leq 6k\Delta + \ell \leq 7k\Delta$.

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 of volume $\leq \Delta$ containing s for $\ell \leq k$, then the procedure returns an ℓ-out-component with probability $\geq \frac{5}{6}$.

Idea: Each sampled t will lie inside of component with probability $\leq \frac{1}{6k}$

Questions?

Summary

- Significant progress for a fundamental graph problem
- Local procedure was pivotal to faster algorithm
  Exponential improvement over $O(2^{O(k)}\Delta)$ by [Chechik et al. '17]

Summary

- Significant progress for a fundamental graph problem
- Local procedure was pivotal to faster algorithm
  Exponential improvement over $O(2^{O(k)}\Delta)$ by [Chechik et al. '17]
- Local procedure has further implications to property testing algorithms
- Local computation algorithms are a current trend in algorithm design

Thesis Opportunities

Theory:

- Distributed algorithms
- Dynamic algorithms
- Local computation algorithms

Thesis Opportunities

Theory:

- Distributed algorithms
- Dynamic algorithms
- Local computation algorithms

Algorithm Engineering:

- Experimental analysis of cut sparsification algorithms
- Practical algorithm for computing the vertex connectivity

Thank you!