Quantum Speedup for Graph Sparsification, Cut Approximation and Laplacian Solving

Simon Apers[1] Ronald de Wolf[2]

[1] Inria, France and CWI, the Netherlands
[2] QuSoft, CWI and University of Amsterdam, the Netherlands

Simons Institute, April 2020

(arXiv:1911.07306)

Graphs

graphs are nice

graphs are nice

- all over computer science, discrete math, biology, ...

graphs are nice

- all over computer science, discrete math, biology, . . .
- describe relations, networks, groups, . . .

graphs are nice

- all over computer science, discrete math, biology, . . .
- describe relations, networks, groups, . . .

sparse graphs are nicer

graphs are nice

- all over computer science, discrete math, biology, ...
- describe relations, networks, groups, ...

sparse graphs are nicer

- less space to store

graphs are nice

- all over computer science, discrete math, biology, . . .
- describe relations, networks, groups, . . .

sparse graphs are nicer

- less space to store
- less time to process

graphs are nice

- all over computer science, discrete math, biology, ...
- describe relations, networks, groups, ...

sparse graphs are nicer

- less space to store
- less time to process
- example: expanders are more interesting than complete graphs

graphs are nice

- all over computer science, discrete math, biology, . . .
- describe relations, networks, groups, . . .

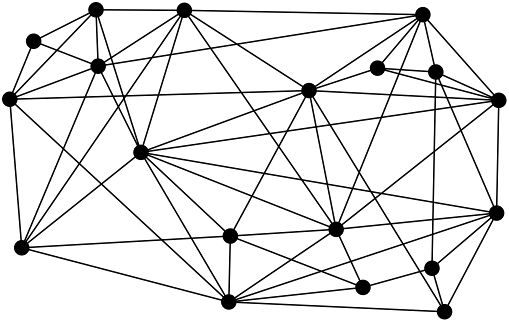
sparse graphs are nicer

- less space to store
- less time to process
- example: expanders are more interesting than complete graphs

can we **compress** general graphs to sparse graphs ?

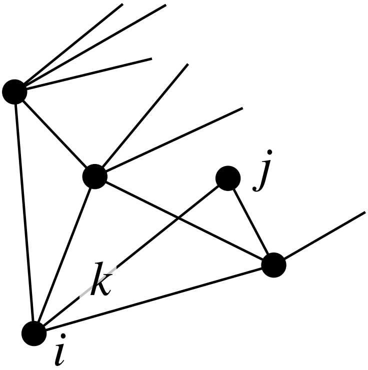
Graph Sparsification

undirected, weighted graph $G = (V, E, w)$
n nodes and m edges, $m \leq \binom{n}{2}$



undirected, weighted graph $G = (V, E, w)$

n nodes and m edges, $m \leq \binom{n}{2}$



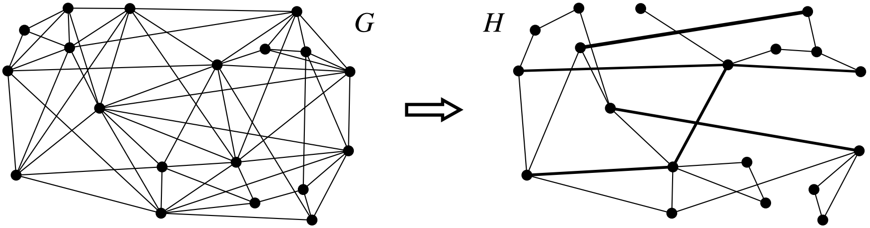
adjacency-list access

query (i, k) returns k-th neighbor j of node i

Graph Sparsification

"graph sparsification"

= reduce number of edges, while preserving interesting quantities



Graph Sparsification

what are "interesting quantities"?

Graph Sparsification

what are “interesting quantities”?

extremal cuts, eigenvalues, random walk properties, . . .

Graph Sparsification

what are "interesting quantities"?

extremal cuts, eigenvalues, random walk properties, ...

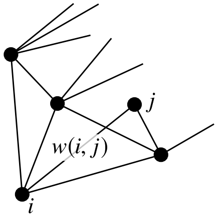
$\rightarrow$ typically captured by **graph Laplacian** L_G

Graph Sparsification

what are "interesting quantities"?

extremal cuts, eigenvalues, random walk properties, ...

$\rightarrow$ typically captured by **graph Laplacian** L_G



$$L_G = D - A$$

with

$$(D)_{ii} = \sum_j w(i,j) \text{ and } (A)_{ij} = w(i,j)$$

Graph Laplacian

equivalently,

Graph Laplacian

equivalently,

$$L_G = \sum_{(i,j)\in E} w(i,j)\, L_{(i,j)}$$

Graph Laplacian

equivalently,

$$L_G = \sum_{(i,j)\in E} w(i,j)\; L_{(i,j)}$$

with

$$L_{(i,j)} = (e_i - e_j)\,(e_i - e_j)^T = \begin{bmatrix} 0 & & \dots & & 0 \\ \vdots & & \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}_{(i,j)} & & \vdots \\ 0 & & \dots & & 0 \end{bmatrix}$$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x$$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x$$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j)\; (x(i) - x(j))^2$$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

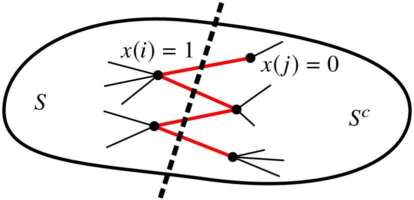
$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j)\; (x(i) - x(j))^2$$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j)\; (x(i) - x(j))^2$$

e.g., if x_S **indicator vector** on $S \subseteq V$:

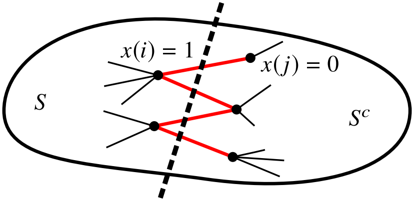


Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j)\; (x(i) - x(j))^2$$

e.g., if x_S **indicator vector** on $S \subseteq V$:



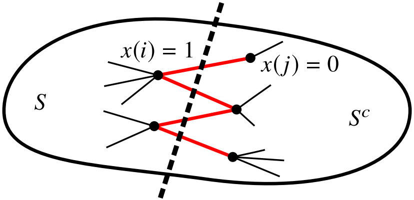
$x_S^T L_G x_S$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j)\; (x(i) - x(j))^2$$

e.g., if x_S **indicator vector** on $S \subseteq V$:



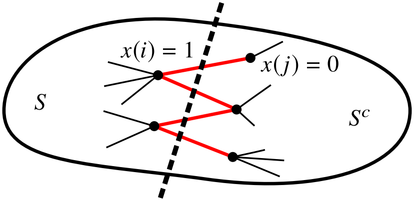
$$x_S^T L_G x_S = \sum_{(i,j)} w(i,j)(x_S(i) - x_S(j))^2$$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j)\; (x(i) - x(j))^2$$

e.g., if x_S **indicator vector** on $S \subseteq V$:



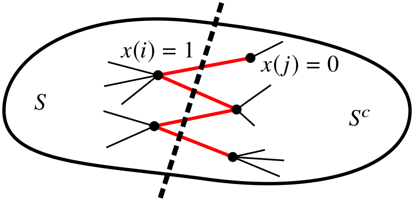
$$x_S^T L_G x_S = \sum_{(i,j)} w(i,j)(x_S(i) - x_S(j))^2 = \sum_{i \in S, j \in S^c} w(i,j)$$

Graph Laplacian

mainly interested in **quadratic forms in L_G**

$$x^T L_G x = \sum_{(i,j)} w(i,j)\; x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j)\; (x(i) - x(j))^2$$

e.g., if x_S **indicator vector** on $S \subseteq V$:



$$x_S^T L_G x_S = \sum_{(i,j)} w(i,j)(x_S(i) - x_S(j))^2 = \sum_{i \in S, j \in S^c} w(i,j) = \mathrm{cut}_G(S)$$

Graph Laplacian

as it turns out,
quadratic forms

$x^T L_G x$ and $x^T L_G^+ x$ for $x \in \mathbb{R}^n$

describe cut values, eigenvalues,
effective resistances, hitting times, ...

Graph Laplacian

as it turns out,
quadratic forms

$x^T L_G x$ and $x^T L_G^+ x$ for $x \in \mathbb{R}^n$

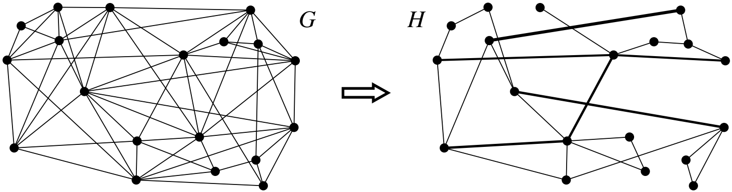
describe cut values, eigenvalues,
effective resistances, hitting times, ...

$\rightarrow$ interested in preserving quadratic forms!

Spectral Sparsification

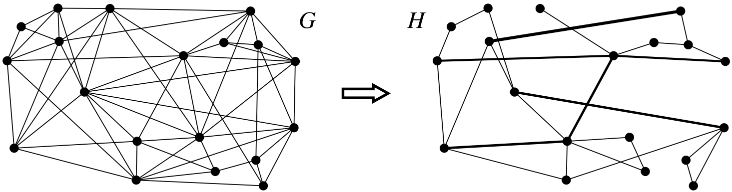
Spectral Sparsification

= approximately **preserve all quadratic forms**



Spectral Sparsification

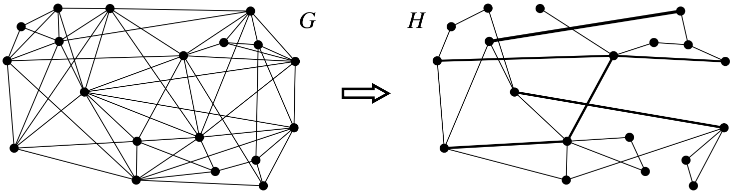
= approximately **preserve all quadratic forms**



definition: H is **ϵ-spectral sparsifier** of G

Spectral Sparsification

= approximately **preserve all quadratic forms**

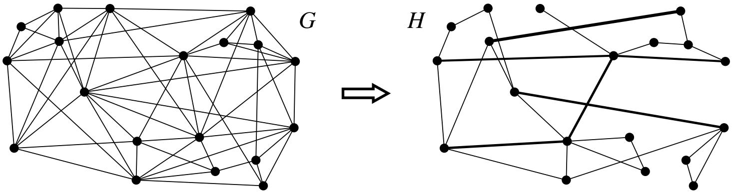


definition: H is **ϵ-spectral sparsifier** of G iff

$$x^T L_H x = (1 \pm \epsilon)\, x^T L_G x \quad \text{for all } x \in \mathbb{R}^n$$

Spectral Sparsification

= approximately **preserve all quadratic forms**



definition: H is **ϵ-spectral sparsifier** of G iff

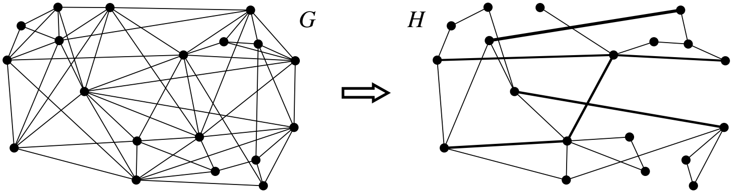
$$x^T L_H x = (1 \pm \epsilon)\, x^T L_G x \quad \text{for all } x \in \mathbb{R}^n$$

equivalently:

$$x^T L_H^+ x = (1 \pm O(\epsilon))\, x^T L_G^+ x$$

Spectral Sparsification

= approximately **preserve all quadratic forms**



definition: H is **ϵ-spectral sparsifier** of G iff

$$x^T L_H x = (1 \pm \epsilon)\, x^T L_G x \quad \text{for all } x \in \mathbb{R}^n$$

equivalently:

$$x^T L_H^+ x = (1 \pm O(\epsilon))\, x^T L_G^+ x$$

equivalently:

$$(1-\epsilon)\, L_G \preceq L_H \preceq (1+\epsilon)\, L_G$$

Spectral Sparsification

how sparse can we go ?

how sparse can we go ?

Karger '94, Benczúr-Karger '96,
Spielman-Teng '04, Batson-Spielman-Srivastava '08:

Theorem

how sparse can we go ?

Karger '94, Benczúr-Karger '96,
Spielman-Teng '04, Batson-Spielman-Srivastava '08:

Theorem

- *every graph has ϵ-spectral sparsifier H with a number of edges*

$$\widetilde{O}(n/\epsilon^2)$$

Spectral Sparsification

how sparse can we go ?

Karger '94, Benczúr-Karger '96,
Spielman-Teng '04, Batson-Spielman-Srivastava '08:

Theorem

- *every graph has ϵ-spectral sparsifier H with a number of edges*

$$\widetilde{O}(n/\epsilon^2)$$

- *H can be found in time $\widetilde{O}(m)$*

how sparse can we go ?

Karger '94, Benczúr-Karger '96,
Spielman-Teng '04, Batson-Spielman-Srivastava '08:

Theorem

- *every graph has ϵ-spectral sparsifier H with a number of edges*

$$\widetilde{O}(n/\epsilon^2)$$

- *H can be found in time $\widetilde{O}(m)$*

(only relevant when $\epsilon \gg \sqrt{n/m}$)

Applications

important building stone of many

$\widetilde{O}(m)$ cut approximation algorithms

Applications

important building stone of many

$\widetilde{O}(m)$ cut approximation algorithms

- max cut (Arora-Kale ’07)
- min cut (Karger ’00)
- min st-cut (Peng ’16)
- sparsest cut (Sherman ’09)
- …

Applications

crucial component of Spielman-Teng breakthrough Laplacian solver:

Applications

crucial component of Spielman-Teng breakthrough Laplacian solver:

Theorem (Spielman-Teng '04)

Let G be a graph with m edges. The Laplacian system $L_G x = b$ can be approximately solved in time $\widetilde{O}(m)$.

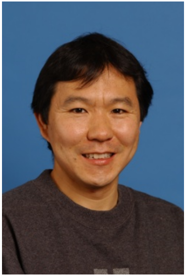
Applications

crucial component of Spielman-Teng breakthrough Laplacian solver:

Theorem (Spielman-Teng '04)

Let G be a graph with m edges. The Laplacian system $L_G x = b$ can be approximately solved in time $\widetilde{O}(m)$.

= Gödel prize 2015



Applications

crucial component of Spielman-Teng breakthrough Laplacian solver:

Theorem (Spielman-Teng '04)

Let G be a graph with m edges. The Laplacian system $L_G x = b$ can be approximately solved in time $\widetilde{O}(m)$.

$\widetilde{O}(m)$ approximation algorithms for

- electrical flows and max flows
- spectral clustering
- random walk properties
- learning from data on graphs
- ...

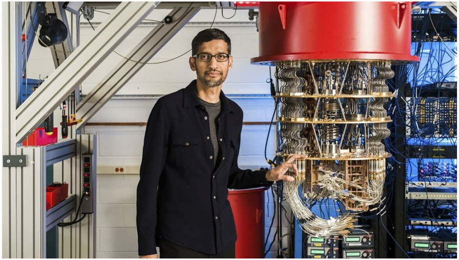
Our Contribution

classically, $\widetilde{O}(m)$ runtime is optimal for most graph algorithms

Our Contribution

classically, $\widetilde{O}(m)$ runtime is optimal for most graph algorithms

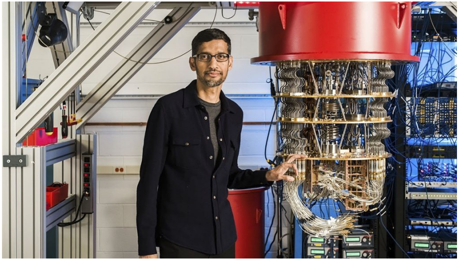
can we do better using a quantum computer?



Our Contribution

classically, $\widetilde{O}(m)$ runtime is optimal for most graph algorithms

can we do better using a quantum computer?



(disclaimer: not with this one we won't)

Our Contribution

this work:

Our Contribution

this work:

1. **quantum algorithm** to find ϵ-spectral sparsifier H in time

$$\widetilde{O}(\sqrt{mn}/\epsilon)$$

Our Contribution

this work:

1. **quantum algorithm** to find ϵ-spectral sparsifier H in time

$$\widetilde{O}(\sqrt{mn}/\epsilon)$$

2. matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ **lower bound**

Our Contribution

this work:

1. **quantum algorithm** to find ϵ-spectral sparsifier H in time

$$\widetilde{O}(\sqrt{mn}/\epsilon)$$

2. matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ **lower bound**

3. **applications:** quantum speedup for
   - max cut, min cut, min st-cut, sparsest cut, ...
   - Laplacian solving, approximating resistances and random walk properties, spectral clustering, ...

this work:

1. **quantum algorithm** to find ϵ-spectral sparsifier H in time

$$\widetilde{O}(\sqrt{mn}/\epsilon)$$

2. matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound
3. applications: quantum speedup for
   - max cut, min cut, min st-cut, sparsest cut, ...
   - Laplacian solving, approximating resistances and random walk properties, spectral clustering, ...

Classical Sparsification Algorithm

Classical Sparsification Algorithm

Sparsification by edge sampling:

1. associate probabilities $\{p_e\}$ to every edge
2. keep every edge e with probability p_e, rescale its weight by $1/p_e$

Classical Sparsification Algorithm

Sparsification by edge sampling:

1. associate probabilities $\{p_e\}$ to every edge
2. keep every edge e with probability p_e, rescale its weight by $1/p_e$

ensures that

$$\mathbb{E}(w_e^H) = w_e^G$$

Classical Sparsification Algorithm

Sparsification by edge sampling:

1. associate probabilities $\{p_e\}$ to every edge
2. keep every edge e with probability p_e, rescale its weight by $1/p_e$

ensures that

$$\mathbb{E}(w_e^H) = w_e^G$$

and hence

$$\mathbb{E}(L_H) = \mathbb{E}\left(\sum w_e L_e\right) = L_G$$

Classical Sparsification Algorithm

Sparsification by edge sampling:

1. associate probabilities $\{p_e\}$ to every edge
2. keep every edge e with probability p_e, rescale its weight by $1/p_e$

ensures that

$$\mathbb{E}(w_e^H) = w_e^G$$

and hence

$$\mathbb{E}(L_H) = \mathbb{E}\left(\sum w_e L_e\right) = L_G$$

how to ensure **concentration?**

Classical Sparsification Algorithm

Sparsification by edge sampling:

1. associate probabilities $\{p_e\}$ to every edge
2. keep every edge e with probability p_e, rescale its weight by $1/p_e$

ensures that

$$\mathbb{E}(w_e^H) = w_e^G$$

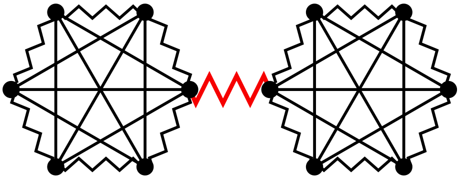
and hence

$$\mathbb{E}(L_H) = \mathbb{E}\left(\sum w_e L_e\right) = L_G$$

how to ensure **concentration?**

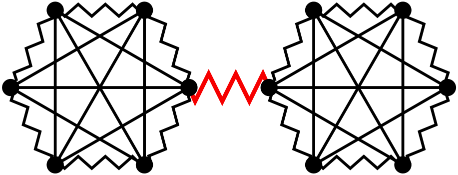
[Spielman-Srivastava '08]:
give high p_e to edges with high **effective resistance!**

Classical Sparsification Algorithm



effective resistance $R_{(i,j)}$

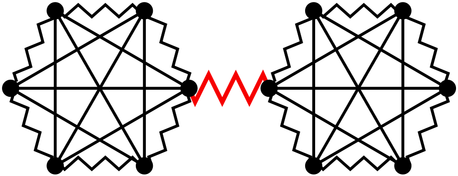
Classical Sparsification Algorithm



effective resistance $R_{(i,j)}$

= resistance between i, j
after replacing all edges with resistors

Classical Sparsification Algorithm

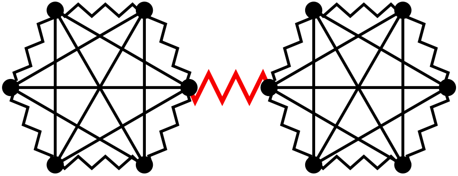


effective resistance $R_{(i,j)}$

= resistance between i, j
after replacing all edges with resistors

$\overset{\text{(Ohm's law)}}{=}$ voltage difference required between i, j
when sending unit current from i to j

Classical Sparsification Algorithm



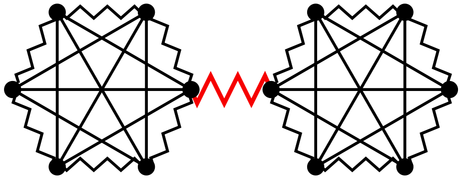
effective resistance $R_{(i,j)}$

= resistance between i, j
after replacing all edges with resistors

$\overset{\text{(Ohm's law)}}{=}$ voltage difference required between i, j
when sending unit current from i to j

$\rightarrow$ small if many short and parallel paths from i to j !

Classical Sparsification Algorithm



effective resistance $R_{(i,j)}$

red edge: $R_e = 1$

black edges: $R_e \in O(1/n)$

? how to identify high-resistance edges ?

? how to identify high-resistance edges ?

[Koutis-Xu '14]:
a **graph spanner** must contain all high-resistance edges

? how to identify high-resistance edges ?

[Koutis-Xu '14]:

a **graph spanner** must contain all high-resistance edges

=

- subgraph F of G with $\widetilde{O}(n)$ edges

? how to identify high-resistance edges ?

[Koutis-Xu '14]:
a **graph spanner** must contain all high-resistance edges

=

- subgraph F of G with $\widetilde{O}(n)$ edges
- all distances stretched by factor $\leq \log n$: for all i,j

$$d_G(i,j) \leq d_F(i,j) \leq \log(n)\, d_G(i,j)$$

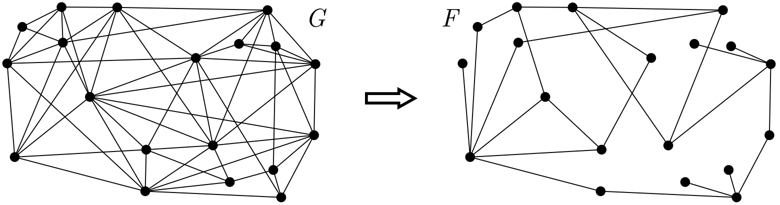
? how to identify high-resistance edges ?

[Koutis-Xu '14]:
a **graph spanner** must contain all high-resistance edges

=

- subgraph F of G with $\widetilde{O}(n)$ edges
- all distances stretched by factor $\leq \log n$: for all i, j

$$d_G(i,j) \leq d_F(i,j) \leq \log(n)\, d_G(i,j)$$



? how to identify high-resistance edges ?

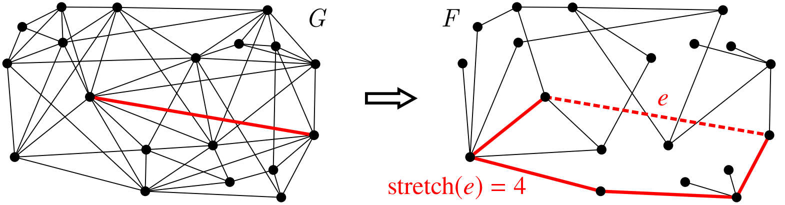
[Koutis-Xu '14]:

a **graph spanner** must contain all high-resistance edges

=

- subgraph F of G with $\widetilde{O}(n)$ edges
- all distances stretched by factor $\leq \log n$: for all i, j

$$d_G(i,j) \leq d_F(i,j) \leq \log(n)\; d_G(i,j)$$



[Koutis-Xu '14]:
a graph spanner must contain all high-resistance edges!

proof idea for $R_e = 1$*:*

[Koutis-Xu '14]:
a graph spanner must contain all high-resistance edges!

proof idea for $R_e = 1$:

- if $R_e = 1$, there are no alternative paths between endpoints

[Koutis-Xu '14]:
a graph spanner must contain all high-resistance edges!

proof idea for $R_e = 1$:

- if $R_e = 1$, there are no alternative paths between endpoints
- hence, e must be present in spanner

Classical Sparsification Algorithm

Iterative sparsification:

1. construct $\widetilde{O}(1/\epsilon^2)$ spanners and keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

Classical Sparsification Algorithm

Iterative sparsification:

1. construct $\widetilde{O}(1/\epsilon^2)$ spanners and keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

(i.e., we set $p_e = 1$ for spanner edges and $p_e = 1/2$ for other edges)

Classical Sparsification Algorithm

Iterative sparsification:

1. construct $\widetilde{O}(1/\epsilon^2)$ spanners and keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

(i.e., we set $p_e = 1$ for spanner edges and $p_e = 1/2$ for other edges)

Theorem (Spielman-Srivastava '08, Koutis-Xu '14)

W.h.p. output is ϵ-spectral sparsifier with $m/2 + \widetilde{O}(n/\epsilon^2)$ edges

Classical Sparsification Algorithm

Iterative sparsification:

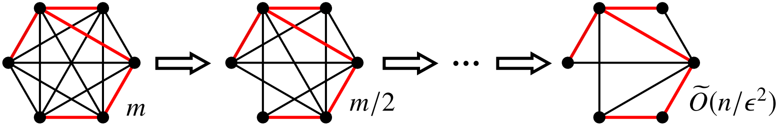
1. construct $\widetilde{O}(1/\epsilon^2)$ spanners and keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

(i.e., we set $p_e = 1$ for spanner edges and $p_e = 1/2$ for other edges)

Theorem (Spielman-Srivastava '08, Koutis-Xu '14)

W.h.p. output is ϵ-spectral sparsifier with $m/2 + \widetilde{O}(n/\epsilon^2)$ edges

$\rightarrow$ *repeat $O(\log n)$ times:* ϵ-spectral sparsifier with $\widetilde{O}(n/\epsilon^2)$ edges



Quantum Sparsification Algorithm

Quantum Sparsification Algorithm

= quantum spanner algorithm

+ k-independent oracle

+ a magic trick

Quantum Spanner Algorithm

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

- greedy spanner algorithm:

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

- greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

- greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$
  2. iterate over every edge $(i,j) \in E \setminus E_F$:
     if $\delta_F(i,j) > \log n$, add (i,j) to F

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

- greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$
  2. iterate over every edge $(i,j) \in E \backslash E_F$:
     if $\delta_F(i,j) > \log n$, add (i,j) to F
- **quantum** greedy spanner algorithm:

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

- greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$
  2. iterate over every edge $(i,j) \in E \setminus E_F$:
     if $\delta_F(i,j) > \log n$, add (i,j) to F
- **quantum** greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

- greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$
  2. iterate over every edge $(i,j) \in E \backslash E_F$:
     if $\delta_F(i,j) > \log n$, add (i,j) to F
- **quantum** greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$
  2. until no more edges are found, do:
     Grover search for edge (i,j) such that $\delta_F(i,j) > \log n$. add (i,j) to F

Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with **query complexity**

$$\widetilde{O}(\sqrt{mn})$$

- greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$
  2. iterate over every edge $(i,j) \in E \setminus E_F$:
     if $\delta_F(i,j) > \log n$, add (i,j) to F
- **quantum** greedy spanner algorithm:
  1. set $F = (V, E_F = \emptyset)$
  2. until no more edges are found, do:
     Grover search for edge (i,j) such that $\delta_F(i,j) > \log n$. add (i,j) to F

$\rightarrow$ can prove: $\widetilde{O}(n)$ edges are found using $\widetilde{O}(\sqrt{mn})$ queries

Quantum Spanner Algorithm

Theorem ("less easy")

There is a quantum spanner algorithm with **time complexity**

$$\widetilde{O}(\sqrt{mn})$$

Quantum Spanner Algorithm

Theorem ("less easy")

There is a quantum spanner algorithm with **time complexity**

$$\widetilde{O}(\sqrt{mn})$$

= *(roughly)*

[Thorup-Zwick '01]

classical construction of a spanner by growing small **shortest-path trees** (SPTs)

Quantum Spanner Algorithm

Theorem ("less easy")

There is a quantum spanner algorithm with **time complexity**

$$\widetilde{O}(\sqrt{mn})$$

= *(roughly)*

[Thorup-Zwick '01]

classical construction of a spanner by growing small **shortest-path trees** (SPTs)

+

[Dürr-Heiligman-Høyer-Mhalla '04]

quantum speedup for constructing SPTs

Quantum Sparsification Algorithm

Iterative sparsification:

1. use quantum algorithm to construct $\widetilde{O}(1/\epsilon^2)$ spanners, keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

Quantum Sparsification Algorithm

Iterative sparsification:

1. use quantum algorithm to construct $\widetilde{O}(1/\epsilon^2)$ spanners, keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

$\rightarrow$ after 1 iteration: "intermediate" graph with $\approx m/2$ edges

Quantum Sparsification Algorithm

Iterative sparsification:

1. use quantum algorithm to construct $\widetilde{O}(1/\epsilon^2)$ spanners, keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

$\rightarrow$ after 1 iteration: “intermediate” graph with $\approx m/2$ edges

? how to keep track in time $o(m)$?

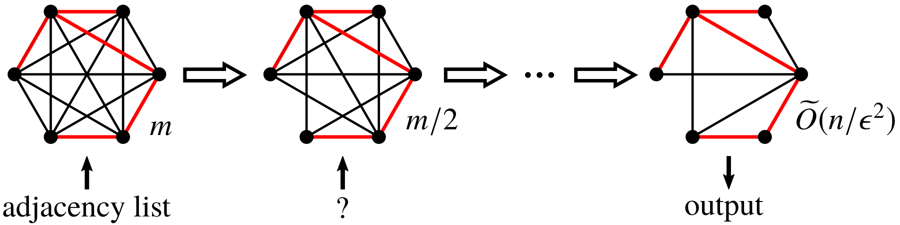
Quantum Sparsification Algorithm

Iterative sparsification:

1. use quantum algorithm to construct $\widetilde{O}(1/\epsilon^2)$ spanners, keep these edges
2. keep any remaining edge with probability $1/2$, and double its weight

$\rightarrow$ after 1 iteration: "intermediate" graph with $\approx m/2$ edges

? how to keep track in time $o(m)$?



Query Access to Random String

- maintain (offline) random string $x \in \{0,1\}^{\binom{n}{2}}$

1	0	0	1	1	0	1	1	1	0	1	0	0

edge (i,j) discarded   edge (i',j') kept

Query Access to Random String

- maintain (offline) random string $x \in \{0,1\}^{\binom{n}{2}}$

1	0	0	1	1	0	1	1	1	0	1	0	0

edge (i,j) discarded   edge (i',j') kept

(oblivious to the graph!)

Query Access to Random String

maintain (offline) random string $x \in \{0,1\}^{\binom{n}{2}}$

1	0	0	1	1	0	1	1	1	0	1	0	0

edge (i,j) discarded   edge (i',j') kept

(oblivious to the graph!)

query $(i,k) \longrightarrow (j, {\color{red}x(i,j)})$

Query Access to Random String

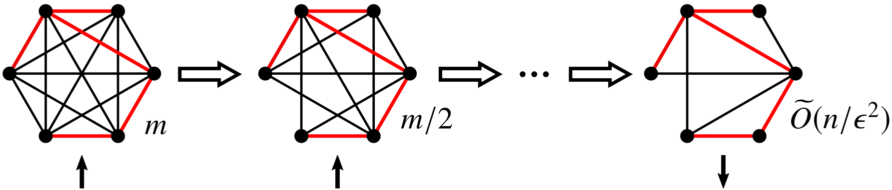
maintain (offline) random string $x \in \{0,1\}^{\binom{n}{2}}$

1	0	0	1	1	0	1	1	1	0	1	0	0

edge (i,j) discarded   edge (i',j') kept

(oblivious to the graph!)

query $(i,k) \longrightarrow (j, x(i,j))$



adjacency list   adj. list + random string   output

Query Access to Random String

problem:

time $\Omega(n^2)$ to generate random $x \in \{0,1\}^{\binom{n}{2}}$

Query Access to Random String

problem:

time $\Omega(n^2)$ to generate random $x \in \{0,1\}^{\binom{n}{2}}$

- classical solution: “lazy sampling” (generate bits on demand)

Query Access to Random String

problem:

time $\Omega(n^2)$ to generate random $x \in \{0,1\}^{\binom{n}{2}}$

- classical solution: “lazy sampling” (generate bits on demand)
- quantum this is not possible: can address all bits in superposition

Rid of Random String

luckily, we can outsmart this quantum demon:

Rid of Random String

luckily, we can outsmart this quantum demon:

Fact

*$k/2$-query quantum algorithm cannot distinguish uniformly random string from k-wise independent string **

= easy consequence of *polynomial method*

[Beals-Buhrman-Cleve-Mosca-de Wolf ’98]

Rid of Random String

luckily, we can outsmart this quantum demon:

Fact

*$k/2$-query quantum algorithm cannot distinguish uniformly random string from k-wise independent string **

= easy consequence of *polynomial method*

[Beals-Buhrman-Cleve-Mosca-de Wolf '98]

* k-wise independent string $x \in \{0,1\}^{\binom{n}{2}}$
behaves uniformly random on every subset of k bits

Rid of Random String

aim for quantum algorithm making $\sim \sqrt{mn}$ queries,
so suffices to use k-wise independent $\binom{n}{2}$-bit string with $k \sim \sqrt{mn}$

Rid of Random String

aim for quantum algorithm making $\sim \sqrt{mn}$ queries,
so suffices to use k-wise independent $\binom{n}{2}$-bit string with $k \sim \sqrt{mn}$

? can we efficiently query such a string ?
(without explicitly generating it!)

Rid of Random String

aim for quantum algorithm making $\sim \sqrt{mn}$ queries,
so suffices to use k-wise independent $\binom{n}{2}$-bit string with $k \sim \sqrt{mn}$

? can we efficiently query such a string ?
(without explicitly generating it!)

$\rightarrow$ use recent results on "efficient k-independent hash functions"

Rid of Random String

aim for quantum algorithm making $\sim \sqrt{mn}$ queries,
so suffices to use k-wise independent $\binom{n}{2}$-bit string with $k \sim \sqrt{mn}$

? can we efficiently query such a string ?
(without explicitly generating it!)

$\rightarrow$ use recent results on "efficient k-independent hash functions"

Theorem (Christiani-Pagh-Thorup '15)

Can construct in preprocessing time $\widetilde{O}(k)$ a k-independent oracle that simulates queries to k-wise independent string in time $\widetilde{O}(1)$ per query.

Rid of Random String

aim for quantum algorithm making $\sim \sqrt{mn}$ queries,
so suffices to use k-wise independent $\binom{n}{2}$-bit string with $k \sim \sqrt{mn}$

? can we efficiently query such a string ?
(without explicitly generating it!)

$\rightarrow$ use recent results on "efficient k-independent hash functions"

Theorem (Christiani-Pagh-Thorup '15)

Can construct in preprocessing time $\widetilde{O}(k)$ a k-independent oracle that simulates queries to k-wise independent string in time $\widetilde{O}(1)$ per query.

Corollary

Any k-query quantum algorithm that queries a uniformly random string can be simulated in time $\widetilde{O}(k)$ without random string.

Quantum Sparsification Algorithm

Quantum Sparsification Algorithm

Quantum iterative sparsification:

1. use quantum algorithm to construct $\widetilde{O}(1/\epsilon^2)$ spanners, keep these edges
2. construct k-independent oracle that marks remaining edges with probability $1/2$, and double weights

Quantum Sparsification Algorithm

Quantum iterative sparsification:

1. use quantum algorithm to construct $\widetilde{O}(1/\epsilon^2)$ spanners, keep these edges
2. construct k-independent oracle that marks remaining edges with probability $1/2$, and double weights

$\rightarrow$ per iteration: complexity $\widetilde{O}(\sqrt{mn}/\epsilon^2)$

Quantum Sparsification Algorithm

Quantum iterative sparsification:

1. use quantum algorithm to construct $\widetilde{O}(1/\epsilon^2)$ spanners, keep these edges
2. construct k-independent oracle that marks remaining edges with probability $1/2$, and double weights

$\rightarrow$ per iteration: complexity $\widetilde{O}(\sqrt{mn}/\epsilon^2)$

Theorem

There is a quantum algorithm that constructs an ϵ-spectral sparsifier with $\widetilde{O}(n/\epsilon^2)$ edges in time

$$\widetilde{O}(\sqrt{mn}/\epsilon^2)$$

A Magic Trick

Münchhausen O. Herrfurth pinx

A Magic Trick

to improve ϵ-dependency:

A Magic Trick

to improve ϵ-dependency:

1. create rough ϵ-spectral sparsifier H for $\epsilon = 1/10$
   $\rightarrow \widetilde{O}(\sqrt{mn})$ using our **quantum** algorithm

A Magic Trick

to improve ϵ-dependency:

1. create rough ϵ-spectral sparsifier H for $\epsilon = 1/10$

   $\rightarrow \widetilde{O}(\sqrt{mn})$ using our **quantum** algorithm

2. estimate effective resistances for H

   $\rightarrow \widetilde{O}(n)$ using **classical** Laplacian solving

A Magic Trick

to improve ϵ-dependency:

1. create rough ϵ-spectral sparsifier H for $\epsilon = 1/10$

   $\rightarrow \widetilde{O}(\sqrt{mn})$ using our **quantum** algorithm

2. estimate effective resistances for H

   $\rightarrow \widetilde{O}(n)$ using **classical** Laplacian solving

   = approximation of effective resistances of G !

A Magic Trick

to improve ϵ-dependency:

1. create rough ϵ-spectral sparsifier H for $\epsilon = 1/10$
   $\rightarrow \widetilde{O}(\sqrt{mn})$ using our **quantum** algorithm

2. estimate effective resistances for H
   $\rightarrow \widetilde{O}(n)$ using **classical** Laplacian solving
   = approximation of effective resistances of G !

3. sample $\widetilde{O}(n/\epsilon^2)$ edges from G using these estimates
   $\rightarrow$ in time $\widetilde{O}(\sqrt{mn/\epsilon^2})$ using **Grover search**

A Magic Trick

to improve ϵ-dependency:

1. create rough ϵ-spectral sparsifier H for $\epsilon = 1/10$
   $\rightarrow \widetilde{O}(\sqrt{mn})$ using our **quantum** algorithm
2. estimate effective resistances for H
   $\rightarrow \widetilde{O}(n)$ using **classical** Laplacian solving
   = approximation of effective resistances of G !
3. sample $\widetilde{O}(n/\epsilon^2)$ edges from G using these estimates
   $\rightarrow$ in time $\widetilde{O}(\sqrt{mn/\epsilon^2})$ using **Grover search**

Theorem (our main result)

There is a quantum algorithm that constructs an ϵ-spectral sparsifier with $\widetilde{O}(n/\epsilon^2)$ edges in time

$$\widetilde{O}(\sqrt{mn}/\epsilon)$$

A Magic Trick

to improve ϵ-dependency:

1. create rough ϵ-spectral sparsifier H for $\epsilon = 1/10$
   $\rightarrow \widetilde{O}(\sqrt{mn})$ using our **quantum** algorithm
2. estimate effective resistances for H
   $\rightarrow \widetilde{O}(n)$ using **classical** Laplacian solving
   = approximation of effective resistances of G !
3. sample $\widetilde{O}(n/\epsilon^2)$ edges from G using these estimates
   $\rightarrow$ in time $\widetilde{O}(\sqrt{mn/\epsilon^2})$ using **Grover search**

Theorem (our main result)

There is a quantum algorithm that constructs an ϵ-spectral sparsifier with $\widetilde{O}(n/\epsilon^2)$ edges in time

$$\widetilde{O}(\sqrt{mn}/\epsilon)$$

* assuming $\epsilon \geq \sqrt{n/m}$, it holds that $\widetilde{O}(\sqrt{mn}/\epsilon) \in \widetilde{O}(m)$

this work:

1. quantum algorithm to find ϵ-spectral sparsifier H in time
$$\widetilde{O}(\sqrt{mn}/\epsilon)$$
2. matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ **lower bound**
3. applications: quantum speedup for
   - max cut, min cut, min st-cut, sparsest cut, ...
   - Laplacian solving, approximating resistances and random walk properties, spectral clustering, ...

Matching Quantum Lower Bound

intuition:

finding k marked elements among M elements takes

$\Omega(\sqrt{Mk})$ quantum queries

Matching Quantum Lower Bound

intuition:

finding k marked elements among M elements takes

$\Omega(\sqrt{Mk})$ quantum queries

"hence"

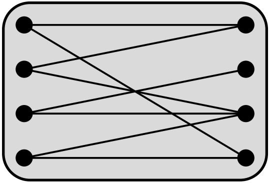
finding $\widetilde{O}(n/\epsilon^2)$ edges of sparsifier among m edges takes time

$\widetilde{\Omega}(\sqrt{mn}/\epsilon)$

Unsparsifiable Graph

Unsparsifiable Graph

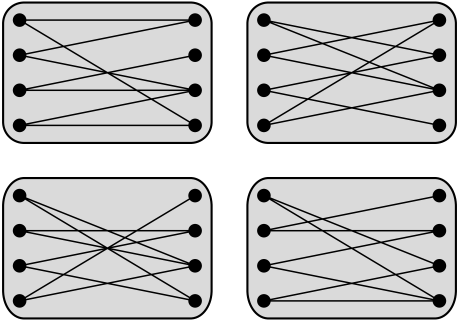
random bipartite graph on $1/\epsilon^2$ nodes



Unsparsifiable Graph

$\epsilon^2 n$ copies

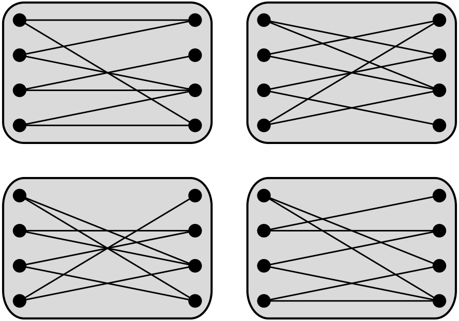
= random graph $H(n, \epsilon)$ with n nodes and $O(n/\epsilon^2)$ edges



Unsparsifiable Graph

$\epsilon^2 n$ copies

= random graph $H(n, \epsilon)$ with n nodes and $O(n/\epsilon^2)$ edges



Theorem (Andoni-Chen-Krauthgamer-Qin-Woodruff-Zhang '16)

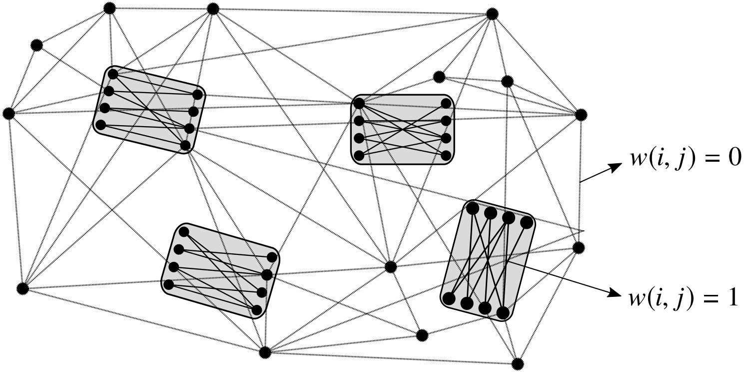
Any ϵ-spectral sparsifier of $H(n, \epsilon)$ must contain a constant fraction of its edges.

Hiding a Sparsifier

Hiding a Sparsifier

given n, m, ϵ:

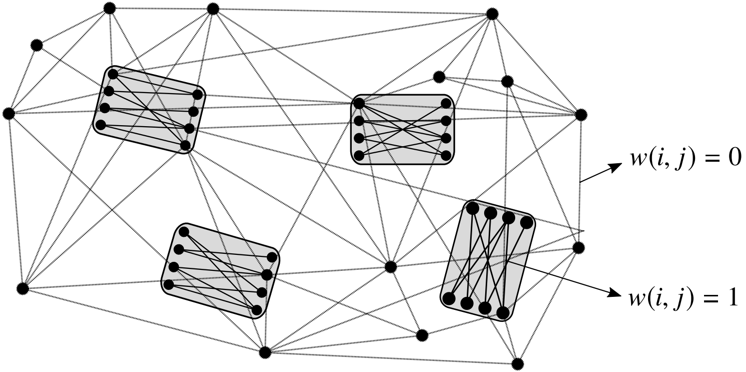
we "hide" $H(n, \epsilon)$ in larger $G(n, m, \epsilon)$ with n nodes and m edges



Hiding a Sparsifier

given n, m, ϵ:

we "hide" $H(n, \epsilon)$ in larger $G(n, m, \epsilon)$ with n nodes and m edges



$\rightarrow$ ϵ-spectral sparsifier of $G(n, m, \epsilon)$ must find constant fraction of $H(n, \epsilon)$

Proving a Lower Bound

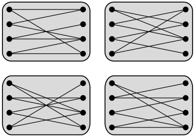
Proving a Lower Bound

“hidden” copy of random graph:
every edge of sparsifier is **hidden** among $N = m/(n\epsilon^2)$ entries

Proving a Lower Bound

"hidden" copy of random graph:
every edge of sparsifier is **hidden** among $N = m/(n\epsilon^2)$ entries

original graph:

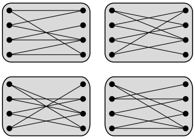


$$= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Proving a Lower Bound

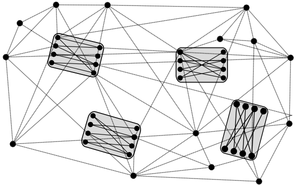
"hidden" copy of random graph:
every edge of sparsifier is **hidden** among $N = m/(n\epsilon^2)$ entries

original graph:



$$= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

hidden graph:



$$= \begin{bmatrix} 0000001000 & 0000000000 & 0000000000 & 0010000000 \\ 0001000000 & 0000000000 & 0000000010 & 0000000000 \\ 0000000000 & 0000001000 & 0000000010 & 0000000000 \\ 0000000000 & 0000000000 & 0000010000 & 0000100000 \end{bmatrix}$$

Proving a Lower Bound

forgetting about graphs:

Proving a Lower Bound

forgetting about graphs:

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \in \{0,1\}^{n \times n}$$

Proving a Lower Bound

forgetting about graphs:

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \in \{0,1\}^{n \times n}$$

$$= OR_{N,\text{blockwise}} \left(\begin{bmatrix} {\scriptstyle 000000}1{\scriptstyle 000} & {\scriptstyle 0}0{\scriptstyle 00000000} & {\scriptstyle 0000000}0{\scriptstyle 00} & {\scriptstyle 00}1{\scriptstyle 0000000} \\ {\scriptstyle 000}1{\scriptstyle 000000} & {\scriptstyle 000000000}0 & {\scriptstyle 00000000}1{\scriptstyle 0} & {\scriptstyle 000}0{\scriptstyle 000000} \\ {\scriptstyle 00000}0{\scriptstyle 0000} & {\scriptstyle 000000}1{\scriptstyle 000} & {\scriptstyle 00000000}1{\scriptstyle 0} & {\scriptstyle 00000}0{\scriptstyle 0000} \\ 0{\scriptstyle 000000000} & {\scriptstyle 00}0{\scriptstyle 0000000} & {\scriptstyle 00000}1{\scriptstyle 0000} & {\scriptstyle 0000}1{\scriptstyle 00000} \end{bmatrix} \in \{0,1\}^{Nn \times Nn} \right)$$

Proving a Lower Bound

forgetting about graphs:

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \in \{0,1\}^{n \times n}$$

$$= OR_{N,\text{blockwise}} \left(\begin{bmatrix} {\scriptstyle 000000}1{\scriptstyle 000} & {\scriptstyle 0}0{\scriptstyle 00000000} & {\scriptstyle 0000000}0{\scriptstyle 00} & {\scriptstyle 00}1{\scriptstyle 0000000} \\ {\scriptstyle 000}1{\scriptstyle 000000} & {\scriptstyle 000000000}0 & {\scriptstyle 00000000}1{\scriptstyle 0} & {\scriptstyle 000}0{\scriptstyle 000000} \\ {\scriptstyle 00000}0{\scriptstyle 0000} & {\scriptstyle 000000}1{\scriptstyle 000} & {\scriptstyle 00000000}1{\scriptstyle 0} & {\scriptstyle 00000}0{\scriptstyle 0000} \\ 0{\scriptstyle 000000000} & {\scriptstyle 00}0{\scriptstyle 0000000} & {\scriptstyle 00000}1{\scriptstyle 0000} & {\scriptstyle 0000}1{\scriptstyle 00000} \end{bmatrix} \in \{0,1\}^{Nn \times Nn} \right)$$

task:

output constant fraction of 1-bits of A, each described by OR_N-function

Proving a Lower Bound

forgetting about graphs:

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \in \{0,1\}^{n \times n}$$

$$= OR_{N,\text{blockwise}} \left(\begin{bmatrix} {\scriptstyle 000000}1{\scriptstyle 000} & {\scriptstyle 0}0{\scriptstyle 00000000} & {\scriptstyle 0000000}0{\scriptstyle 00} & {\scriptstyle 00}1{\scriptstyle 0000000} \\ {\scriptstyle 000}1{\scriptstyle 000000} & {\scriptstyle 000000000}0 & {\scriptstyle 00000000}1{\scriptstyle 0} & {\scriptstyle 000}0{\scriptstyle 000000} \\ {\scriptstyle 00000}0{\scriptstyle 0000} & {\scriptstyle 000000}1{\scriptstyle 000} & {\scriptstyle 00000000}1{\scriptstyle 0} & {\scriptstyle 00000}0{\scriptstyle 0000} \\ 0{\scriptstyle 000000000} & {\scriptstyle 00}0{\scriptstyle 0000000} & {\scriptstyle 00000}1{\scriptstyle 0000} & {\scriptstyle 0000}1{\scriptstyle 00000} \end{bmatrix} \in \{0,1\}^{Nn \times Nn} \right)$$

task:

output constant fraction of 1-bits of A, each described by OR_N-function

= relational problem composed with OR_N

Proving a Lower Bound

? quantum lower bound for composition of relational problem and OR_N-function ?

Proving a Lower Bound

? quantum lower bound for composition of relational problem and OR_N-function ?

Theorem (proof by A. Belov and T. Lee, to be published)

The quantum query complexity of an efficiently verifiable relational problem, with lower bound L, composed with the OR_N-function, is

$$\Omega(L\sqrt{N}).$$

Proving a Lower Bound

? quantum lower bound for composition of relational problem and OR_N-function ?

Theorem (proof by A. Belov and T. Lee, to be published)

The quantum query complexity of an efficiently verifiable relational problem, with lower bound L, composed with the OR_N-function, is

$$\Omega(L\sqrt{N}).$$

for $L = \widetilde{\Omega}(n)$ and $N = m/(n\epsilon^2)$:

Corollary

The quantum query complexity of explicity outputting an ϵ-spectral sparsifier of a graph with n nodes and m edges is

$$\widetilde{\Omega}(\sqrt{mn}/\epsilon).$$

this work:

1. quantum algorithm to find ϵ-spectral sparsifier H in time

$$\widetilde{O}(\sqrt{mn}/\epsilon)$$

2. matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound

3. **applications:** quantum speedup for
   - max cut, min cut, min st-cut, sparsest cut, . . .
   - Laplacian solving, approximating resistances and random walk properties, spectral clustering, . . .

Quantum Speedups by Quantum Sparsification

Quantum Speedups by Quantum Sparsification

graph quantity P,
approximately preserved under sparsification

Quantum Speedups by Quantum Sparsification

graph quantity P,
approximately preserved under sparsification

+

classical $\widetilde{O}(m)$ algorithm for P

Quantum Speedups by Quantum Sparsification

graph quantity P,
approximately preserved under sparsification

+

classical $\widetilde{O}(m)$ algorithm for P

$\downarrow$

quantum sparsify G to H in $\widetilde{O}(\sqrt{mn}/\epsilon)$

+ **classical** algorithm on H in $\widetilde{O}(n/\epsilon^2)$

Quantum Speedups by Quantum Sparsification

graph quantity P,
approximately preserved under sparsification

+

classical $\widetilde{O}(m)$ algorithm for P

$\downarrow$

quantum sparsify G to H in $\widetilde{O}(\sqrt{mn}/\epsilon)$

+ **classical** algorithm on H in $\widetilde{O}(n/\epsilon^2)$

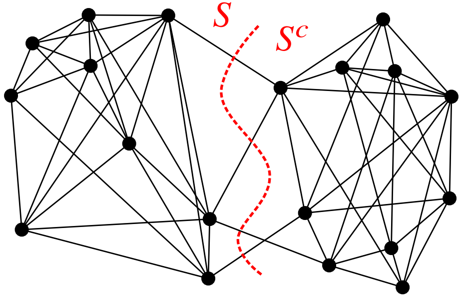
=

approximate $\widetilde{O}(\sqrt{mn}/\epsilon)$ **quantum algorithm** for P

Cut Approximation

MIN CUT:

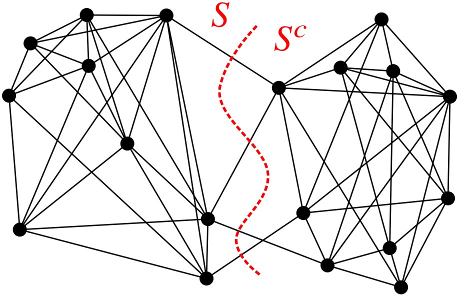
find cut (S, S^c) that minimizes cut value $\mathrm{cut}_G(S)$



Cut Approximation

MIN CUT:

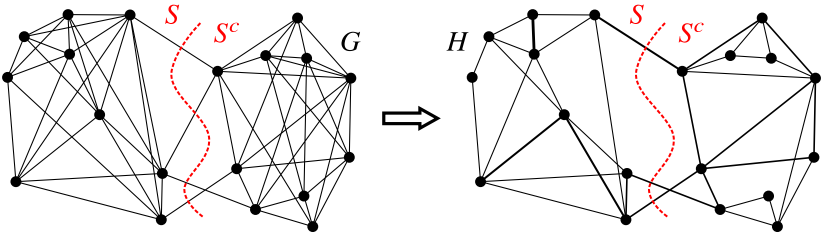
find cut (S, S^c) that minimizes cut value $\mathrm{cut}_G(S)$



classically: can find MIN CUT in time $\widetilde{O}(m)$ (Karger '00)

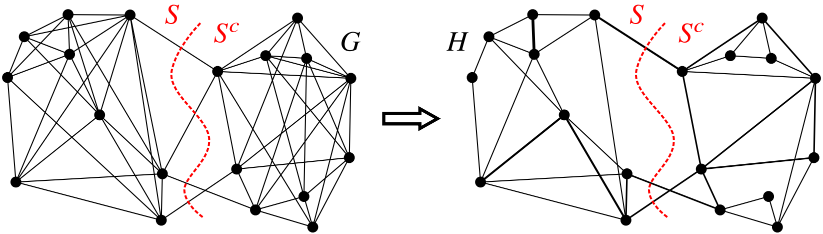
Cut Approximation

MIN CUT of ϵ-spectral sparsifier H
gives ϵ-approximation of MIN CUT of G



Cut Approximation

MIN CUT of ϵ-spectral sparsifier H
gives ϵ-approximation of MIN CUT of G



quantum sparsify G to H in $\widetilde{O}(\sqrt{mn}/\epsilon)$
+ **classical** MIN CUT on H in $\widetilde{O}(n/\epsilon^2)$ (Karger '00)

= $\widetilde{O}(\sqrt{mn}/\epsilon)$ **quantum** algorithm for ϵ-MIN CUT

Cut Approximation

	Classical	Quantum (this work)
ϵ-MIN CUT	$\widetilde{O}(m)$ (Karger'00)	$\widetilde{O}(\sqrt{mn}/\epsilon)$
ϵ-MIN st-CUT	$\widetilde{O}(m + n/\epsilon^5)$ (Peng'16)	$\widetilde{O}(\sqrt{mn}/\epsilon + n/\epsilon^5)$
$\sqrt{\log n}$-SPARSEST CUT/ -BAL. SEPARATOR	$\widetilde{O}(m + n^{1+\delta})$ (Sherman'09)	$\widetilde{O}(\sqrt{mn} + n^{1+\delta})$
.878-MAX CUT	$\widetilde{O}(m)$ (Arora-Kale'07)	$\widetilde{O}(\sqrt{mn})$

Laplacian Solving

Laplacian Solving

general linear system $Ax = b$

Laplacian Solving

general linear system $Ax = b$

given A and b, with $nnz(A) = m$,

complexity of approximating x is $\widetilde{O}(\min\{mn, n^{\omega}\})$ ($\omega < 2.373$)

Laplacian Solving

Laplacian system $Lx = b$

Laplacian Solving

Laplacian system $Lx = b$

given L and b, with $nnz(L) = m$,

complexity of approximating x is $\widetilde{O}(m)$ [Spielman-Teng '04]

Laplacian Solving

Laplacian system $Lx = b$

given L and b, with $nnz(L) = m$,

complexity of approximating x is $\widetilde{O}(m)$ [Spielman-Teng '04]

+

if H sparsifier of G then $L_H^+ b \approx L_G^+ b$

Laplacian Solving

Laplacian system $Lx = b$

given L and b, with $nnz(L) = m$,

complexity of approximating x is $\widetilde{O}(m)$ [Spielman-Teng '04]

+

if H sparsifier of G then $L_H^+ b \approx L_G^+ b$

$\downarrow$

quantum algorithm to sparsify G to H in $\widetilde{O}(\sqrt{mn}/\epsilon)$

+ solve $L_H x = b$ **classically** in $\widetilde{O}(n/\epsilon^2)$

Laplacian Solving

Laplacian system $Lx = b$

given L and b, with $nnz(L) = m$,

complexity of approximating x is $\widetilde{O}(m)$ [Spielman-Teng '04]

+

if H sparsifier of G then $L_H^+ b \approx L_G^+ b$

$\downarrow$

quantum algorithm to sparsify G to H in $\widetilde{O}(\sqrt{mn}/\epsilon)$

+ solve $L_H x = b$ **classically** in $\widetilde{O}(n/\epsilon^2)$

=

quantum algorithm for Laplacian solving in $\widetilde{O}(\sqrt{mn}/\epsilon)$

Laplacian Solving

Laplacian system $Lx = b$

given L and b, with $nnz(L) = m$,

complexity of approximating x is $\widetilde{O}(m)$ [Spielman-Teng '04]

+

if H sparsifier of G then $L_H^+ b \approx L_G^+ b$

$\downarrow$

quantum algorithm to sparsify G to H in $\widetilde{O}(\sqrt{mn}/\epsilon)$

+ solve $L_H x = b$ **classically** in $\widetilde{O}(n/\epsilon^2)$

=

quantum algorithm for Laplacian solving in $\widetilde{O}(\sqrt{mn}/\epsilon)$

(+ quantum reduction for symmetric, diagonally dominant systems)

Laplacian Solving and Friends

	Classical	Quantum (this work)
ϵ-SDD Solving	$\widetilde{O}(m)$ (ST'04)	$\widetilde{O}(\sqrt{mn}/\epsilon)$
ϵ-Effective Resistance (single)	$\widetilde{O}(m)$	$\widetilde{O}(\sqrt{mn}/\epsilon)$ prior: $\widetilde{O}(\sqrt{mn}/\epsilon^2)$
ϵ-Effective Resistance (all)	$\widetilde{O}(m + n/\epsilon^4)$ (Spielman-Srivastava'08)	$\widetilde{O}(\sqrt{mn}/\epsilon + n/\epsilon^4)$
$O(1)$-Cover Time	$\widetilde{O}(m)$ (Ding-Lee-Peres'10)	$\widetilde{O}(\sqrt{mn})$
k bottom eigenvalues	$\widetilde{O}(m + kn/\epsilon^2)$	$\widetilde{O}(\sqrt{mn}/\epsilon + kn/\epsilon^2)$ prior, $k = 1$: $\widetilde{O}(n^2/\epsilon)$
Spectral k-means clustering	$\widetilde{O}(m + n\,\mathrm{poly}(k))$	$\widetilde{O}(\sqrt{mn} + n\,\mathrm{poly}(k))$

summary:

summary:

- quantum algorithm for spectral sparsification in time $\widetilde{O}(\sqrt{mn}/\epsilon)$

summary:

- quantum algorithm for spectral sparsification in time $\widetilde{O}(\sqrt{mn}/\epsilon)$
- matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound

summary:

- quantum algorithm for spectral sparsification in time $\widetilde{O}(\sqrt{mn}/\epsilon)$
- matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound
- speedup for cut approximation, Laplacian solving, ...

summary:

- quantum algorithm for spectral sparsification in time $\widetilde{O}(\sqrt{mn}/\epsilon)$
- matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound
- speedup for cut approximation, Laplacian solving, ...

open questions:

summary:

- quantum algorithm for spectral sparsification in time $\widetilde{O}(\sqrt{mn}/\epsilon)$
- matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound
- speedup for cut approximation, Laplacian solving, ...

open questions:

- matching **lower bounds for applications?**
  e.g., $\Omega(\sqrt{mn}/\epsilon)$ for approximate min cut or Laplacian solving?

summary:

- quantum algorithm for spectral sparsification in time $\widetilde{O}(\sqrt{mn}/\epsilon)$
- matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound
- speedup for cut approximation, Laplacian solving, ...

open questions:

- matching **lower bounds for applications?**
  e.g., $\Omega(\sqrt{mn}/\epsilon)$ for approximate min cut or Laplacian solving?
- our $\widetilde{O}(\sqrt{mn}/\epsilon)$ sparsification algorithm is tight for weighted graphs.
  can we do **better for unweighted graphs?**

summary:

- quantum algorithm for spectral sparsification in time $\widetilde{O}(\sqrt{mn}/\epsilon)$
- matching $\widetilde{\Omega}(\sqrt{mn}/\epsilon)$ lower bound
- speedup for cut approximation, Laplacian solving, ...

open questions:

- matching **lower bounds for applications?**
  e.g., $\Omega(\sqrt{mn}/\epsilon)$ for approximate min cut or Laplacian solving?
- our $\widetilde{O}(\sqrt{mn}/\epsilon)$ sparsification algorithm is tight for weighted graphs.
  can we do **better for unweighted graphs?**

thank you! stay safe!