Quantum Speedup for Graph Sparsification, Cut Approximation and Laplacian Solving

Simon Apers\textsuperscript{1} \quad Ronald de Wolf\textsuperscript{2}

\textsuperscript{1}Inria, France and CWI, the Netherlands  
\textsuperscript{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”?

\[ L_G = D - A \]

with

\[ (D)_{ii} = \sum_j w(i, j) \]

and

\[ (A)_{ij} = w(i, j) \]
Graph Sparsification

what are “interesting quantities”?

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

what are “interesting quantities”? extremal cuts, eigenvalues, random walk properties, . . .

→ typically captured by graph Laplacian $L_G$
Graph Sparsification

what are “interesting quantities”?

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

→ typically captured by graph Laplacian $L_G$

$L_G = D - A$

with

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

equivalently,

$$L = \sum_{(i, j) \in E} w_{ij} L_{ij}$$

with

$$L_{ij} = (e_i - e_j) (e_i - e_j)^T$$

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

\[ L_{(i,j)} = (e_i - e_j) (e_i - e_j)^T = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} & \vdots \\ 0 & \cdots & 0 \end{bmatrix}_{(i,j)} \]
Graph Laplacian

mainly interested in \textbf{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 \textbf{quadratic forms in } \( L_G \)

\[
x^T L_G x = \sum_{(i,j)} w(i,j) x^T L_{(i,j)} x
\]
mainly interested in **quadratic forms in** $L_G$

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

$$= \sum_{(i,j)} w(i,j) \ (x(i) - x(j))^2$$
mainly interested in **quadratic forms in** \( L_G \)

\[
x^T L_G x = \sum_{(i,j)} w(i,j) \quad x^T L_{(i,j)} x = \sum_{(i,j)} w(i,j) \quad (x(i) - x(j))^2
\]
Graph Laplacian

mainly interested in \textbf{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 $x_S$ \textbf{indicator vector} on $S \subseteq V$: 

![Graph Laplacian Diagram](image-url)
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 $S \subseteq V$:

\[ x^T_S L_G x_S \]
Graph Laplacian

mainly interested in **quadratic forms in** \( L_G \)

\[
x^T L_G x = \sum_{(i,j)} w(i,j) \quad x^T L(i,j)x = \sum_{(i,j)} w(i,j) \left( x(i) - x(j) \right)^2
\]

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

\[
x^T S L_G x_S = \sum_{(i,j)} w(i,j)(x_S(i) - x_S(j))^2
\]
Graph Laplacian

mainly interested in \textbf{quadratic forms in} $L_G$

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

\[ \text{e.g., if } x_S \text{ indicator vector on } S \subseteq V: \]

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

$$x^T_S L_G x_S = \sum_{(i,j)} w(i,j) (x_S(i) - x_S(j))^2 = \sum_{i \in S, j \in S^c} w(i,j) = \text{cut}_G(S)$$
Graph Laplacian

as it turns out, quadratic forms

\[ x^T L_G x \quad \text{and} \quad x^T L_G^+ x \quad \text{for} \quad x \in \mathbb{R}^n \]

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

as it turns out, **quadratic forms**

\[ x^T L_G x \quad \text{and} \quad x^T L_G^+ x \quad \text{for} \quad x \in \mathbb{R}^n \]

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

\[ \rightarrow \text{interested in preserving quadratic forms!} \]
Spectral Sparsification

\[ H \] is an \( \epsilon \)-spectral sparsifier of \( G \) iff

\[ x^T L H x = (1 \pm \epsilon) x^T L G x \quad \text{for all} \quad x \in \mathbb{R}^n \]

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

= approximately preserve all quadratic forms

\[ H \] is \( \epsilon \)-spectral sparsifier of \( G \) iff

\[ x^T L_H x = (1 \pm \epsilon) x^T L_G x \]

for all \( x \in \mathbb{R}^n \)

equivalently:

\[ x^T (L_H + H) x = (1 \pm O(\epsilon)) x^T (L_G + G) x \]

equivalently:

\[ (1 - \epsilon) L_G \preceq L_H \preceq (1 + \epsilon) L_G \]
Spectral Sparsification

= approximately preserve all quadratic forms

definition: $H$ is $\epsilon$-spectral sparsifier of $G$
Spectral Sparsification

\[ \text{= approximately } \text{preserve all quadratic forms} \]

\[
\begin{align*}
\text{definition: } H \text{ is } \epsilon\text{-spectral sparsifier of } G \text{ iff } \\
x^T L_H x = (1 \pm \epsilon) x^T L_G x \text{ for all } x \in \mathbb{R}^n
\end{align*}
\]
Spectral Sparsification

= approximately preserve all quadratic forms

definition: $H$ is $\epsilon$-spectral sparsifier of $G$ iff

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

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 $\epsilon$-spectral sparsifier of $G$ iff

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

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
Spectral Sparsification

how sparse can we go?

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

Theorem

- Every graph has $\varepsilon$-spectral sparsifier $H$ with a number of edges

$$\tilde{O}(n/\varepsilon^2)$$
Spectral Sparsification

how sparse can we go?

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

Theorem

- every graph has $\epsilon$-spectral sparsifier $H$ with a number of edges
  \[
  \widetilde{O}(n/\epsilon^2)
  \]

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

how sparse can we go?

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

**Theorem**

- every graph has \( \epsilon \)-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 \( \epsilon \leq \sqrt{n/m} \))
important building stone of many
$\tilde{O}(m)$ cut approximation algorithms
Applications

important building stone of many

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

- max cut (Arora-Kale ’16)
- min cut (Karger ’00)
- min $st$-cut (Peng ’16)
- sparsest cut (Sherman ’09)
- ...
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 $\tilde{O}(m)$. 
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 $\tilde{O}(m)$.***
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 \( \tilde{O}(m) \).

= Gödel prize 2015
Applications

crucial component of Spielman-Teng breakthrough Laplacian solver:

<table>
<thead>
<tr>
<th>Theorem (Spielman-Teng ’04)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $G$ be a graph with $m$ edges. The Laplacian system $L_Gx = b$ can be approximately solved in time $\tilde{O}(m)$.</td>
</tr>
</tbody>
</table>

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

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

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

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

can we do better using a quantum computer?
Our Contribution

classically, $\tilde{O}(m)$ 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:

1. Quantum algorithm to find \( \epsilon \)-spectral sparsifier \( H \) in time \( \tilde{O}(\sqrt{mn}/\epsilon) \)

2. Matching \( \tilde{\Omega}(\sqrt{mn}/\epsilon) \) 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, ...
Our Contribution

this work:

1. **quantum algorithm** to find $\epsilon$-spectral sparsifier $H$ in time

$$\tilde{O}(\sqrt{mn}/\epsilon)$$
Our Contribution

this work:

1. **quantum algorithm** to find $\epsilon$-spectral sparsifier $H$ in time
   \[ \tilde{O}(\sqrt{mn}/\epsilon) \]

2. matching \(\tilde{\Omega}(\sqrt{mn}/\epsilon)\) lower bound
Our Contribution

this work:

1. **quantum algorithm** to find $\epsilon$-spectral sparsifier $H$ in time

   $\tilde{O}\left(\sqrt{mn}/\epsilon\right)$

2. matching $\tilde{\Omega}\left(\sqrt{mn}/\epsilon\right)$ 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 $\epsilon$-spectral sparsifier $H$ in time

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

2. matching $\tilde{\Omega}(\sqrt{mn}/\epsilon)$ 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

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 \( \frac{1}{p_e} \).

This ensures that \( E(w_H) = w_G \) and hence \( E(L_H) = E(\sum w_e L_e) = L_G \).

How to ensure concentration?

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

\[
\mathbb{E}(w^H_e) = w^G_e
\]

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

\[
\mathbb{E}(w^H_e) = w^G_e
\]

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

(Ohm’s law) $\equiv$ 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

(Ohm’s law) $\Rightarrow$ 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 $\tilde{O}(n)$ edges
? how to identify high-resistance edges?

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

= 

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

$$d_G(i,j) \leq d_F(i,j) \leq \log(n) \cdot d_G(i,j)$$
how to identify high-resistance edges?

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

= 

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

$$d_G(i,j) \leq d_F(i,j) \leq \log(n) \cdot d_G(i,j)$$
? how to identify high-resistance edges ?

[Koutis-Xu ’14]:

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

= 

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

\[ d_G(i, j) \leq d_F(i, j) \leq \log(n) \cdot d_G(i, j) \]

\[ \text{stretch}(e) = 4 \]
[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 $\tilde{O}(1/\epsilon^2)$ 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 $\tilde{O}(1/\epsilon^2)$ 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 $\tilde{O}(1/\epsilon^2)$ 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 $\epsilon$-spectral sparsifier with $m/2 + \tilde{O}(n/\epsilon^2)$ edges
Classical Sparsification Algorithm

Iterative sparsification:
1. construct $\tilde{O}(1/\epsilon^2)$ 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 $\epsilon$-spectral sparsifier with $m/2 + \tilde{O}(n/\epsilon^2)$ edges

\[ \rightarrow \text{repeat } O(\log n) \text{ times: } \epsilon\text{-spectral sparsifier with } \tilde{O}(n/\epsilon^2) \text{ edges} \]
Quantum Sparsification Algorithm
Quantum Sparsification Algorithm

= quantum spanner algorithm

+ $k$-independent oracle

+ a magic trick
Theorem ("easy")

There is a quantum spanner algorithm with query complexity 
\[ \tilde{O}(\sqrt{mn}) \]

**quantum 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 \( F \)

→ can prove: \( \tilde{O}(n) \) edges are found using \( \tilde{O}(\sqrt{mn}) \) queries.
Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with query complexity

\( \tilde{O}(\sqrt{mn}) \)
Theorem ("easy")

There is a quantum spanner algorithm with query complexity

\[ \tilde{O}(\sqrt{mn}) \]

- greedy spanner algorithm:

Quantum Spanner Algorithm
Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with query complexity

\[ \tilde{O}(\sqrt{mn}) \]

greedy spanner algorithm:

1. set \( F = (V, E_F = \emptyset) \)
Quantum Spanner Algorithm

Theorem ("easy")

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

\[ \tilde{O}(\sqrt{mn}) \]

- 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

\[ \tilde{O}(\sqrt{mn}) \]

- 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:
Quantum Spanner Algorithm

Theorem ("easy")

There is a quantum spanner algorithm with query complexity

\[ \tilde{O}(\sqrt{mn}) \]

- 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

\[ \tilde{O}(\sqrt{mn}) \]

- 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 \(F\)
Quantum Spanner Algorithm

Theorem ("easy")

*There is a quantum spanner algorithm with query complexity*

\[ \tilde{O}(\sqrt{mn}) \]

- **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 \( F \)

→ can prove: \( \tilde{O}(n) \) edges are found using \( \tilde{O}(\sqrt{mn}) \) queries
Quantum Spanner Algorithm

Theorem ("less easy")

There is a quantum spanner algorithm with time complexity

\[ \tilde{O}(\sqrt{mn}) \]
Quantum Spanner Algorithm

Theorem ("less easy")

There is a quantum spanner algorithm with time complexity

\(\tilde{O}(\sqrt{mn})\)

= (roughly)

[Thorup-Zwick ’01]

classical construction of a spanner by growing small shortest-path trees (SPTs)
Theorem ("less easy")

There is a quantum spanner algorithm with time complexity

\[ \tilde{O}(\sqrt{mn}) \]

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

→ after 1 iteration: “intermediate” graph with $\approx m/2$ edges
Quantum Sparsification Algorithm

Iterative sparsification:

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

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

→ 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 $x \in \{0, 1\}^{n/2}$

```
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 \( x \in \{0, 1\}^{n/2} \)

\[
100110110100
\]

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

(oblivious to the graph!)
Query Access to Random String

- maintain (offline) random string $x \in \{0, 1\}^\binom{n}{2}$
  
  \[
  \begin{array}{cccccccccccc}
  1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
  \end{array}
  \]

  edge $(i, j)$ discarded  edge $(i', j')$ kept
  (oblivious to the graph!)

  query $(i, k) \rightarrow (j, x(i, j))$
Query Access to Random String

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

  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 $(i, k) \rightarrow (j, x(i, j))$

adjacency list \[ m \] \[ \Rightarrow \] \[ \Rightarrow \] \[ \Rightarrow \] \[ \Rightarrow \] \[ \Rightarrow \] adj. list + random string \[ m/2 \] \[ \Rightarrow \] output \[ \tilde{O}(n/\epsilon^2) \]
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 $x \in \{0, 1\}^{n/2}$

- classical solution: “lazy sampling” (generate bits on demand)
Query Access to Random String

**problem:**

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

- classical solution: “lazy sampling” (generate bits on demand)
- quantum this is not possible: can address all bits in superposition
luckily, we can outsmart this quantum demon:
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 \(x \in \{0, 1\}^{n \choose 2}\)
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!)

→ 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!)

→ use recent results on “efficient $k$-independent hash functions”

**Theorem (Christiani-Pagh-Thorup ’15)**

*Can construct in preprocessing time $\tilde{O}(k)$ a $k$-independent oracle that simulates queries to $k$-wise independent string in time $\tilde{O}(1)$ per query.*
Rid of Random String

aim for quantum algorithm making \( \sim \sqrt{mn} \) queries, so suffices to use \( k \)-wise independent \( \left( \begin{array}{c} n \\ 2 \end{array} \right) \)-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 \( \tilde{O}(k) \) a \( k \)-independent oracle that simulates queries to \( k \)-wise independent string in time \( \tilde{O}(1) \) per query.

**Corollary**

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

Quantum iterative sparsification:
1. Use quantum algorithm to construct $\tilde{O}(1/\epsilon^2)$ spanners, keep these edges.
2. Construct $k$-independent oracle that marks remaining edges with probability $1/2$, and double weights per iteration: complexity $\tilde{O}(\sqrt{mn}/\epsilon^2)$.

Theorem: There is a quantum algorithm that constructs an $\epsilon$-spectral sparsifier with $\tilde{O}(n/\epsilon^2)$ edges in time $\tilde{O}(\sqrt{mn}/\epsilon^2)$. 
Quantum Sparsification Algorithm

Quantum iterative sparsification:

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

→ per iteration: complexity $\tilde{O}(\sqrt{mn}/\epsilon^2)$
Quantum Sparsification Algorithm

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

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

Theorem

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

$\tilde{O}(\sqrt{mn}/\epsilon^2)$
A Magic Trick
A Magic Trick

to improve \( \epsilon \)-dependency:
A Magic Trick

to improve $\epsilon$-dependency:

1. create rough $\epsilon$-spectral sparsifier $H$ for $\epsilon = 1/10$
   $\rightarrow \tilde{O}(\sqrt{mn})$ using our quantum algorithm
A Magic Trick

to improve $\epsilon$-dependency:

1. create rough $\epsilon$-spectral sparsifier $H$ for $\epsilon = 1/10$
   \[
   \rightarrow \tilde{O}(\sqrt{mn}) \text{ using our quantum algorithm}
   \]

2. estimate effective resistances for $H$
   \[
   \rightarrow \tilde{O}(n) \text{ using classical Laplacian solving}
   \]
A Magic Trick

to improve $\epsilon$-dependency:

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

2. estimate effective resistances for $H$
   $\rightarrow \tilde{O}(n)$ using classical Laplacian solving
   $= \text{approximation of effective resistances of } G$!
A Magic Trick

to improve $\epsilon$-dependency:

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

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

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

to improve $\epsilon$-dependency:

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

2. estimate effective resistances for $H$
   \[ \rightarrow \tilde{O}(n) \] using classical Laplacian solving
   $= \text{approximation of effective resistances of } G$

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

Theorem (our main result)

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

$\tilde{O}(\sqrt{mn/\epsilon})$
A Magic Trick

to improve $\epsilon$-dependency:

1. create rough $\epsilon$-spectral sparsifier $H$ for $\epsilon = \frac{1}{10}$
   \[ \rightarrow \tilde{O}(\sqrt{mn}) \text{ using our quantum algorithm} \]

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

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

Theorem (our main result)

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

1. quantum algorithm to find $\epsilon$-spectral sparsifier $H$ in time
   \[ \tilde{O}(\sqrt{mn}/\epsilon) \]

2. matching $\tilde{\Omega}(\sqrt{mn}/\epsilon)$ 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 $\tilde{O}(n/\epsilon^2)$ edges of sparsifier among $m$ edges takes time

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

\[ \epsilon^2 n \text{ copies} \]

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

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

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

given $n$, $m$, $\epsilon$:
we "hide" $H(n, \epsilon)$ in larger $G(n, m, \epsilon)$ with $n$ nodes and $m$ edges
$\rightarrow$ $\epsilon$-spectral sparsifier of $G(n, m, \epsilon)$ must find constant fraction of $H(n, \epsilon)$
Hiding a Sparsifier

given \( n, m, \epsilon \):

we “hide” \( H(n, \epsilon) \) in larger \( G(n, m, \epsilon) \) with \( n \) nodes and \( m \) edges.
Hiding a Sparsifier

given $n$, $m$, $\epsilon$:
we “hide” $H(n, \epsilon)$ in larger $G(n, m, \epsilon)$ with $n$ nodes and $m$ edges

→ $\epsilon$-spectral sparsifier of $G(n, m, \epsilon)$ must find constant fraction of $H(n, \epsilon)$
Proving a Lower Bound

[Matrix and text content]
Proving a Lower Bound

“hidden” copy of random graph:
every edge of sparsifier is hidden among $N = m/(n\varepsilon^2)$ entries
Proving a Lower Bound

“hidden” copy of random graph:
every edge of sparsifier is \textbf{hidden} among \( N = \frac{m}{n\epsilon^2} \) entries

original graph:

\[
\begin{bmatrix}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1
\end{bmatrix}
\]
“hidden” copy of random graph:
every edge of sparsifier is hidden among $N = m/(n\epsilon^2)$ entries

original graph:

hidden graph:
Proving a Lower Bound

forgetting about graphs:

\[
\begin{bmatrix}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
\end{bmatrix} \in \{0, 1\}^{n \times n}
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix} \in \{0, 1\}^{N \times N}
\]

task:

output constant fraction of 1-bits of \(A\), each described by \(\text{OR}\) \(N\)-function

= relational problem composed with \(\text{OR}\) \(N\)-function
Proving a Lower Bound

forgetting about graphs:

\[
A = \begin{bmatrix}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
\end{bmatrix} \in \{0, 1\}^{n \times n}
\]
Proving a Lower Bound

forgetting about graphs:

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

\[ = OR_{N, \text{blockwise}} \left( \begin{bmatrix} 000001000 & 000000000 & 000000000 & 010000000 \\ 001000000 & 0000000000 & 000000000 & 000000000 \\ 0000000000 & 0000000000 & 0000000000 & 0000000000 \\ 00000000000 & 00000000000 & 00000000000 & 00000000000 \end{bmatrix} \right) \in \{0, 1\}^{Nn \times Nn} \]
Proving a Lower Bound

forgetting about graphs:

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

\[ = OR_{N, \text{blockwise}} \left( \begin{bmatrix} 000001000 & 000000000 & 000000000 & 001000000 \\ 000000000 & 000000000 & 000000000 & 000000000 \\ 000000000 & 000000000 & 000000000 & 000000000 \\ 000000000 & 000000000 & 000000000 & 000000000 \end{bmatrix} \right) \in \{0, 1\}^{Nn \times Nn} \]

task:

output constant fraction of 1-bits of \( A \), each described by \( OR_N \)-function
Proving a Lower Bound

forgetting about graphs:

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

\[
= OR_{N, \text{blockwise}} \left( \begin{bmatrix}
000001000 & 000000000 & 000000000 & 001000000 \\
001000000 & 000000000 & 000000000 & 000000000 \\
000000000 & 000000100 & 000000000 & 000000000 \\
000000000 & 000000000 & 000001000 & 000001000 \\
\end{bmatrix} \right) \in \{0, 1\}^{Nn \times Nn}
\]

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 ?

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 $L = \tilde{\Omega}(n)$ and $N = m / (n\epsilon^2)$.

Corollary
The quantum query complexity of explicitly outputting an $\epsilon$-spectral sparsifier of a graph with $n$ nodes and $m$ edges is $\tilde{\Omega}(\sqrt{mn}/\epsilon)$. 

41
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 $L = \tilde{\Omega}(n)$ and $N = m/(n\epsilon^2)$:

Corollary

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

$$\tilde{\Omega}(\sqrt{mn}/\epsilon).$$
this work:

1. quantum algorithm to find $\epsilon$-spectral sparsifier $H$ in time $\tilde{O}(\sqrt{mn}/\epsilon)$

2. matching $\tilde{\Omega}(\sqrt{mn}/\epsilon)$ 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 $\tilde{O}(m)$ algorithm for $P$...
Quantum Speedups by Quantum Sparsification

graph quantity $P$, approximately preserved under sparsification

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

↓

quantum sparsify $G$ to $H$ in $\tilde{O}(\sqrt{mn}/\epsilon)$
+ classical algorithm on $H$ in $\tilde{O}(n/\epsilon^2)$
Quantum Speedups by Quantum Sparsification

graph quantity $P$, approximately preserved under sparsification

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

$\downarrow$

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

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

= approximate $\tilde{O}(\sqrt{mn}/\epsilon)$ quantum algorithm for $P$
Cut Approximation

MIN CUT:

find cut \((S, S^c)\) that minimizes cut value \(\text{cut}_G(S)\)
Cut Approximation

MIN CUT:

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

classically: can find MIN CUT in time \(\tilde{O}(m)\) (Karger ’00)
MIN CUT of $\epsilon$-spectral sparsifier $H$
gives $\epsilon$-approximation of MIN CUT of $G$
MIN CUT of \(\epsilon\)-spectral sparsifier \(H\) gives \(\epsilon\)-approximation of MIN CUT of \(G\)

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

\[= \tilde{O}(\sqrt{mn}/\epsilon)\]  quantum algorithm for \(\epsilon\)-MIN CUT
## Cut Approximation

<table>
<thead>
<tr>
<th></th>
<th>Classical</th>
<th>Quantum (this work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$-MIN CUT</td>
<td>$\tilde{O}(m)$ (Karger’00)</td>
<td>$\tilde{O}(\sqrt{mn}/\epsilon)$</td>
</tr>
<tr>
<td>$\epsilon$-MIN st-CUT</td>
<td>$\tilde{O}(m + n/\epsilon^5)$ (Peng’16)</td>
<td>$\tilde{O}(\sqrt{mn}/\epsilon + n/\epsilon^5)$</td>
</tr>
<tr>
<td>$\sqrt{\log n}$-SPARSEST CUT/BAL. SEPARATOR</td>
<td>$\tilde{O}(m + n^{1+\delta})$ (Sherman’09)</td>
<td>$\tilde{O}(\sqrt{mn} + n^{1+\delta})$</td>
</tr>
<tr>
<td>.878-MAX CUT</td>
<td>$\tilde{O}(m)$ (Arora-Kale’07)</td>
<td>$\tilde{O}(\sqrt{mn})$</td>
</tr>
</tbody>
</table>
Laplacian Solving
general linear system $Ax = b$
Laplacian Solving

general linear system \( Ax = b \)

given \( A \) and \( b \), with \( \text{nnz}(A) = m \),

complexity of approximating \( x \) is \( \tilde{O}(\min\{mn, n^\omega\}) \) \( (\omega < 2.373) \)
Laplacian Solving

Laplacian system $Lx = b$
Laplacian Solving

Laplacian system \( Lx = b \)

given \( L \) and \( b \), with \( \text{nnz}(L) = m \),

complexity of approximating \( x \) is \( \tilde{O}(m) \) [Spielman-Teng ’04]
Laplacian Solving

Laplacian system \( Lx = b \)

given \( L \) and \( b \), with \( \text{nnz}(L) = m \),

complexity of approximating \( x \) is \( \tilde{O}(m) \) [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 \( \text{nnz}(L) = m \),

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

\[
\text{quantum algorithm to sparsify } G \text{ to } H \text{ in } \tilde{O}(\sqrt{mn}/\epsilon)
\]

+ solve \( L_Hx = b \) \textit{classically} in \( \tilde{O}(n/\epsilon^2) \)
Laplacian Solving

Laplacian system $Lx = b$

given $L$ and $b$, with $\text{nnz}(L) = m$,

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

+ 

if $H$ sparsifier of $G$ then $L^+_H b \approx L^+_G b$

↓

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

+ solve $L^+_H x = b$ classically in $\tilde{O}(n/\epsilon^2)$

= 

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

Laplacian system \( Lx = b \)

given \( L \) and \( b \), with \( \text{nnz}(L) = m \),

complexity of approximating \( x \) is \( \tilde{O}(m) \) [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 \( \tilde{O}(\sqrt{mn}/\epsilon) \)

+ solve \( L_Hx = b \) classically in \( \tilde{O}(n/\epsilon^2) \)

= 

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

(+ quantum reduction for symmetric, diagonally dominant systems)
Laplacian Solving and Friends

<table>
<thead>
<tr>
<th></th>
<th>Classical</th>
<th>Quantum (this work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon )-SDD Solving</td>
<td>( \tilde{O}(m) ) (ST'04)</td>
<td>( \tilde{O}(\sqrt{mn}/\epsilon) )</td>
</tr>
<tr>
<td>( \epsilon )-Effective Resistance (single)</td>
<td>( \tilde{O}(m) )</td>
<td>( \tilde{O}(\sqrt{mn}/\epsilon) ) prior: ( \tilde{O}(\sqrt{mn}/\epsilon^2) )</td>
</tr>
<tr>
<td>( \epsilon )-Effective Resistance (all)</td>
<td>( \tilde{O}(m + n/\epsilon^4) ) (Spielman-Srivastava’08)</td>
<td>( \tilde{O}(\sqrt{mn}/\epsilon + n/\epsilon^4) )</td>
</tr>
<tr>
<td>( O(1) )-Cover Time</td>
<td>( \tilde{O}(m) )</td>
<td>( \tilde{O}(\sqrt{mn}) )</td>
</tr>
<tr>
<td>(Ding-Lee-Peres’10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( k ) bottom eigenvalues</td>
<td>( \tilde{O}(m + kn/\epsilon^2) )</td>
<td>( \tilde{O}(\sqrt{mn}/\epsilon + kn/\epsilon^2) ) prior, ( k = 1 ): ( \tilde{O}(n^2/\epsilon) )</td>
</tr>
<tr>
<td>Spectral ( k )-means clustering</td>
<td>( \tilde{O}(m + n \text{ poly}(k)) )</td>
<td>( \tilde{O}(\sqrt{mn} + n \text{ poly}(k)) )</td>
</tr>
</tbody>
</table>
summary:

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

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

thank you! stay safe!
summary:

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

- quantum algorithm for spectral sparsification in time $\tilde{O}(\sqrt{m n}/\epsilon)$
- matching $\tilde{\Omega}(\sqrt{m n}/\epsilon)$ lower bound
summary:

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

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

thank you! stay safe!
summary:

- Quantum algorithm for spectral sparsification in time \(\tilde{O}(\sqrt{mn}/\epsilon)\)
- Matching \(\tilde{\Omega}(\sqrt{mn}/\epsilon)\) 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 $\tilde{O}(\sqrt{mn}/\epsilon)$
- matching $\tilde{\Omega}(\sqrt{mn}/\epsilon)$ 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 $\tilde{O}(\sqrt{mn}/\epsilon)$ sparsification algorithm is tight for weighted graphs. can we do better for unweighted graphs?
summary:
- quantum algorithm for spectral sparsification in time $\tilde{O}(\sqrt{mn}/\epsilon)$
- matching $\tilde{\Omega}(\sqrt{mn}/\epsilon)$ 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 $\tilde{O}(\sqrt{mn}/\epsilon)$ sparsification algorithm is tight for weighted graphs. can we do better for unweighted graphs?

thank you! stay safe!