Quantum algorithms for learning graphs

Ashley Montanaro and Changpeng Shao
University of Bristol, UK

Merged with the talk: Troy Lee, Miklos Santha and Shengyu Zhang. "Quantum algorithms for graph problems with cut queries"

arXiv:2011.08611

Learning graphs via oracles in three models

By learning, we mean identifying all the edges.

- **OR query model**
- **Parity query model** [Troy's talk]
- **Graph state**

Learning graphs via oracles in three models

By learning, we mean identifying all the edges.

- **OR query model**
- **Parity query model** [Troy's talk]
- **Graph state**

The first two models arise from wide applications in chemical reactions, molecular biology, DNA sequencing.

Learning graphs via oracles in three models

By learning, we mean identifying all the edges.

- **OR query model**
- **Parity query model** [Troy's talk]
- **Graph state**

The first two models arise from wide applications in chemical reactions, molecular biology, DNA sequencing.

What we are given:

- the set of vertices $V = [n]$
- an oracle or access to the graph state.

Learning graphs via oracles in three models

By learning, we mean identifying all the edges.

- **OR query model**
- **Parity query model** [Troy's talk]
- **Graph state**

The first two models arise from wide applications in chemical reactions, molecular biology, DNA sequencing.

What we are given:

- the set of vertices $V = [n]$
- an oracle or access to the graph state.

Goal: use as few queries as possible, and try to get rid of the dependence on n.

Learning graphs via OR queries

OR query: the oracle returns whether a given subset of the vertices contains any edges

Learning graphs via OR queries

OR query: the oracle returns whether a given subset of the vertices contains any edges

The results ($m = \#$ edges, $n = \#$ vertices):

	Quantum	Classical	
All graphs	$\Theta(n^2)$	$\Theta(n^2)$	No speedup

Learning graphs via OR queries

OR query: the oracle returns whether a given subset of the vertices contains any edges

The results ($m = \#$ edges, $n = \#$ vertices):

	Quantum	Classical	
All graphs	$\Theta(n^2)$	$\Theta(n^2)$	No speedup
m edges	$O(m \log(m \log n))$ $\Omega(m)$	$\Omega(m \log \frac{n^2}{m})$	Speedup when $m \ll n$

Learning graphs via OR queries

OR query: the oracle returns whether a given subset of the vertices contains any edges

The results ($m = \#$ edges, $n = \#$ vertices):

	Quantum	Classical	
All graphs	$\Theta(n^2)$	$\Theta(n^2)$	No speedup
m edges	$O(m \log(m \log n))$ $\Omega(m)$	$\Omega(m \log \frac{n^2}{m})$	Speedup when $m \ll n$
Matching	$O(m^{3/4}),\ \Omega(m)$	$\Omega(m \log \frac{n}{m})$	Polynomial speedups
Cycle	$O(m^{3/4}),\ \Omega(m)$	$\Omega(m \log \frac{n}{m})$	
Star	$\Theta(\sqrt{m})$	$\Omega(m \log \frac{n}{m})$	
k-vertex clique	$\Theta(\sqrt{k})$	$\Omega(k \log \frac{n}{k})$	

Main idea for m edges, matching, cycle

Step 1. Decompose $V = V_1 \cup \cdots \cup V_k$ (disjoint union, i.e., k-coloring), such that each V_i includes no edges. (hope: k small)

Main idea for m edges, matching, cycle

Step 1. Decompose $V = V_1 \cup \cdots \cup V_k$ (disjoint union, i.e., k-coloring), such that each V_i includes no edges. (hope: k small)

A p-random set S is obtained by including each vertex independently with probability p. Then

$$\text{Prob}[S \text{ includes no edges}] \geq 1 - mp^2.$$

Main idea for m edges, matching, cycle

Step 1. Decompose $V = V_1 \cup \cdots \cup V_k$ (disjoint union, i.e., k-coloring), such that each V_i includes no edges. (hope: k small)

A p-random set S is obtained by including each vertex independently with probability p. Then

$$\text{Prob}[S \text{ includes no edges}] \geq 1 - mp^2.$$

Choose $p = 0.1/\sqrt{m}$.

1. With probability ≥ 0.99, we can find V_1.
2. In $V - V_1$, we can similarly find V_2, and so on.
3. $k \approx \sqrt{m}\log n$ (optimal, e.g. complete graph)

Main idea for m edges, matching, cycle

Step 1. Decompose $V = V_1 \cup \cdots \cup V_k$ (disjoint union, i.e., k-coloring), such that each V_i includes no edges. (hope: k small)

A p-random set S is obtained by including each vertex independently with probability p. Then

$$\text{Prob}[S \text{ includes no edges}] \geq 1 - mp^2.$$

Choose $p = 0.1/\sqrt{m}$.

1. With probability ≥ 0.99, we can find V_1.
2. In $V - V_1$, we can similarly find V_2, and so on.
3. $k \approx \sqrt{m} \log n$ (optimal, e.g. complete graph)

Step 2. Find all the edges between V_i, V_j.

Main idea for m edges, matching, cycle

Step 1. Decompose $V = V_1 \cup \cdots \cup V_k$ (disjoint union, i.e., k-coloring), such that each V_i includes no edges. (hope: k small)

A p-random set S is obtained by including each vertex independently with probability p. Then

$$\text{Prob}[S \text{ includes no edges}] \geq 1 - mp^2.$$

Choose $p = 0.1/\sqrt{m}$.

1. With probability ≥ 0.99, we can find V_1.
2. In $V - V_1$, we can similarly find V_2, and so on.
3. $k \approx \sqrt{m} \log n$ (optimal, e.g. complete graph)

Step 2. Find all the edges between V_i, V_j.

Using quantum algorithms for combinatorial group testing (Belovs, arXiv:1311.6777)

Learning an unknown graph state

Graph state model: we are given copies of the graph state corresponding to the graph.

Learning an unknown graph state

Graph state model: we are given copies of the graph state corresponding to the graph.

The results ($m = \#$ edges, $n = \#$ vertices):

	Quantum	Classical	
All graphs	$\Theta(n)$	$\Theta(n^2)$	Quadratic speedup

Learning an unknown graph state

Graph state model: we are given copies of the graph state corresponding to the graph.

The results ($m = \#$ edges, $n = \#$ vertices):

	Quantum	Classical	
All graphs	$\Theta(n)$	$\Theta(n^2)$	Quadratic speedup
m edges	$O(m \log \frac{n^2}{m})$	$\Omega(m \log \frac{n^2}{m})$	No speedup

Learning an unknown graph state

Graph state model: we are given copies of the graph state corresponding to the graph.

The results ($m = \#$ edges, $n = \#$ vertices):

	Quantum	Classical	
All graphs	$\Theta(n)$	$\Theta(n^2)$	Quadratic speedup
m edges	$O(m \log \frac{n^2}{m})$	$\Omega(m \log \frac{n^2}{m})$	No speedup
Degree d	$O(d \log \frac{m}{d})$	$\Omega(nd \log \frac{n}{d})$	Exponential speedups
Matching	$O(\log m)$	$\Omega(m \log \frac{n}{m})$	
Cycle	$O(\log m)$	$\Omega(m \log \frac{n}{m})$	
Star	$O(1)$	$\Omega(m \log \frac{n}{m})$	
k-vertex clique	$O(1)$	$\Omega(k \log \frac{n}{k})$	

Main idea

The graph state:

$$|G\rangle = \prod_{(i,j)\in E} CZ_{ij}|+\rangle^{\otimes n}.$$

Main idea

The graph state:

$$|G\rangle = \prod_{(i,j)\in E} CZ_{ij}|+\rangle^{\otimes n}.$$

Stabilizers:

$$\{X_v \prod_{w\in N(v)} Z_w : v \in V\}$$

where $N(v)$ denotes the set of vertices neighbouring v.

Main idea

The graph state:

$$|G\rangle = \prod_{(i,j)\in E} CZ_{ij}|+\rangle^{\otimes n}.$$

Stabilizers:

$$\{X_v \prod_{w\in N(v)} Z_w : v \in V\}$$

where $N(v)$ denotes the set of vertices neighbouring v.

We use a procedure called Bell sampling (Montanaro, arXiv:1707.04012), which returns a uniformly random stabilizer of $|G\rangle$

$$\prod_{v\in S} X_v \prod_{u\in N(v)} Z_u = \prod_{u\in[n]} X_u^{[u\in S]} Z_u^{|N(u)\cap S|}$$

Main result

Denote A as the adjacency matrix, then each Bell sample returns $A\mathbf{s}$ modulo 2 for a random $\mathbf{s} \in \{0,1\}^n$.

Main result

Denote A as the adjacency matrix, then each Bell sample returns $A\mathbf{s}$ modulo 2 for a random $\mathbf{s} \in \{0,1\}^n$. If we take k samples, we obtain an $n \times k$ boolean matrix B and the matrix AB modulo 2.

Main result

Denote A as the adjacency matrix, then each Bell sample returns $A\mathbf{s}$ modulo 2 for a random $\mathbf{s} \in \{0,1\}^n$. If we take k samples, we obtain an $n \times k$ boolean matrix B and the matrix AB modulo 2.

Theorem 1 (Arbitrary graphs)

Let $\mathcal{F}$ be a family of graphs. Then, for any $G \in \mathcal{F}$, G can be identified by applying Bell sampling to $O(\log |\mathcal{F}|)$ copies of $|G\rangle$.

Main result

Denote A as the adjacency matrix, then each Bell sample returns $A\mathbf{s}$ modulo 2 for a random $\mathbf{s} \in \{0,1\}^n$. If we take k samples, we obtain an $n \times k$ boolean matrix B and the matrix AB modulo 2.

Theorem 1 (Arbitrary graphs)

Let $\mathcal{F}$ be a family of graphs. Then, for any $G \in \mathcal{F}$, G can be identified by applying Bell sampling to $O(\log |\mathcal{F}|)$ copies of $|G\rangle$.

e.g. If G is a graph with at most m edges, it can be identified with $O(m \log(n^2/m))$ copies of $|G\rangle$.

Main result

Denote A as the adjacency matrix, then each Bell sample returns $A\mathbf{s}$ modulo 2 for a random $\mathbf{s} \in \{0,1\}^n$. If we take k samples, we obtain an $n \times k$ boolean matrix B and the matrix AB modulo 2.

Theorem 1 (Arbitrary graphs)

Let $\mathcal{F}$ be a family of graphs. Then, for any $G \in \mathcal{F}$, G can be identified by applying Bell sampling to $O(\log|\mathcal{F}|)$ copies of $|G\rangle$.

e.g. If G is a graph with at most m edges, it can be identified with $O(m \log(n^2/m))$ copies of $|G\rangle$.

By information-theoretic arguments, $\Omega(\log|\mathcal{F}|)$ is the lower bound to learn graphs in the classical setting. In the quantum setting, the lower bound is $\Omega(\sqrt{\log|\mathcal{F}|})$.

Main result

Theorem 2 (Bounded-degree graphs)

For an arbitrary graph G, there is a quantum algorithm which uses $O(d \log m)$ copies of $|G\rangle$, and

- *For each vertex v that has degree at most d, outputs all the neighbours of v and that v has degree at most d.*
- *For each vertex w that has degree larger than d, the algorithm outputs "degree larger than d".*

Main result

Theorem 2 (Bounded-degree graphs)

For an arbitrary graph G, there is a quantum algorithm which uses $O(d \log m)$ copies of $|G\rangle$, and

- *For each vertex v that has degree at most d, outputs all the neighbours of v and that v has degree at most d.*
- *For each vertex w that has degree larger than d, the algorithm outputs "degree larger than d".*

If G is a subgraph of a fixed bounded-degree graph, the algorithm can be made time-efficient.

Main result

Theorem 2 (Bounded-degree graphs)

For an arbitrary graph G, there is a quantum algorithm which uses $O(d \log m)$ copies of $|G\rangle$, and

- *For each vertex v that has degree at most d, outputs all the neighbours of v and that v has degree at most d.*
- *For each vertex w that has degree larger than d, the algorithm outputs "degree larger than d".*

If G is a subgraph of a fixed bounded-degree graph, the algorithm can be made time-efficient.

Thanks very much for your attention!