UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Query Algorithms for Graph Connectivity and *st*-Minimum Cut

by Arinta Primandini Auza

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of

A/Prof. Dr. Troy Lee
A/Prof. Dr. Marco Tomamichel

University of Technology Sydney
Faculty of Engineering and Information Technology

March 2023

Certificate of Original Authorship

I, Arinta Primandini Auza, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signed: Production Note: Signature removed prior to publication.

Date: 27 March 2023

Acknowledgements

I would like to express my deep and sincere gratitude to my principal supervisor, Troy Lee for the continuous support of my Ph.D study and providing invaluable guidance throughout this research. I would also like to thank Marco Tomamichel my co-supervisor who opened the door for me to join his research group at UTS. My sincere thanks also goes to Youming Qiao, who kindly offered support to fund my last semester of Ph.D that allows me the extra time to complete this thesis. I would also like to thank Lily, Robyn, and Camila for the generous help with the university paperwork. I acknowledge Sydney Quantum Academy (SQA) for giving me the opportunity to expand my knowledge in quantum technology and to meet experts and other Ph.D students in this field. I also acknowledge the SQA student committee members, Eser, and Rose for the experience to learn skills outside academia. Finally, I would like to acknowledge my mother, my father, and my brother for their continuous support throughout my life.

This is a THESIS BY COMPILATION. The content have been edited to suit the formatting of the thesis and to maintain its coherence.

LIST OF PUBLICATIONS INCLUDED IN THE THESIS

Chapter	:	Chapter 6
Name	:	On the query complexity of connectivity with global queries
Authors	:	Arinta Auza and Troy Lee
Contribution	:	AA partially proved the complexity of the main algorithm. AA proved the global query implementations. All the authors wrote and modified the paper. TL supervised the project.
Status	:	Preprint
Address	:	https://doi.org/10.48550/arXiv.2109.02115
Signature	:	Arinta Auza Troy Lee

Chapter	:	Chapter 7
Name	:	A sublinear time quantum algorithm for s-t minimum cut on dense simple graphs
Authors	:	Simon Apers, Arinta Auza and Troy Lee
Contribution	:	AA proved the query complexity of the quantum algorithms. All the authors wrote and modified the paper. TL supervised the project.
Status	:	Preprint
Address	:	https://doi.org/10.48550/arXiv.2110.15587
Signature	:	Simon Apers Arinta Auza Troy Lee

Contents

List of Figures

Abstract

Graph problems are one of the most important and interesting areas in computer science. This thesis focuses on two graph problems: *connectivity* and *s-t-minimum cut*. We study query algorithms to solve these problems. With query algorithms, we assume that we do not have access to the graph, but instead, we have access to an oracle that gives some information about the graph. There are two types of query models: *local query* and *global query* model. With local queries, we can only get information about a single vertex or edge slot, whereas with global queries, we can get information about many vertices and edge slots.

The aim of the proposed thesis is to develop query-efficient algorithms for connectivity and s-t-minimum cut problems. We use global query model to develop algorithms for connectivity problem by finding a spanning forest of the graph. Our algorithm uses the boolean matrix-vector multiplication queries as a master model, then we simulate this master query by other query models such as cut queries and bipartite independent set (BIS) queries. As a result, we give query efficient quantum cut and quantum BIS algorithms for connectivity. We also show lower bound on the zero-error randomized linear query complexity for connectivity. Lower bounds for unrestricted linear queries are difficult to show because the answer to a query can have an unbounded number of bits. We adapt certificate complexity technique to the connectivity problem and show an $\Omega(n)$ lower bound for zero-error randomized algorithms.

For the s-t-minimum cut problem, we use the local query model. We give a query-efficient quantum algorithm that follows an algorithm by Rubinstein, Schramm, and Weinberg (RSW). The idea of the algorithm is to find a sparser graph that still has approximately the same cut values as the original graph, which we call cut sparsifier, then find a minimum s-t cut in this sparser graph. We use quantum tools in our algorithm to speed up the whole procedure. We include an easier self-contained proof for the query complexity of quantum cut sparsification algorithm. We also fix the gap in the RSW analysis for cut query algorithm for the sparsifier.

Chapter 1

Introduction

1.1 Background

Graph problems are one of the fundamental areas of algorithms. Two important graph problems are connectivity and s-t-mincut. In this thesis, we study query algorithms for these two problems. We categorize the query models we study into two types; *local queries* and *global queries*. With local queries, one can access information about a vertex or edge slot. For example, in the adjacency matrix model, one can query if there is an edge between two vertices, and in the adjacency list model, one can ask the degree of a vertex. Global queries are more powerful than local queries, in which a single query can aggregate information about many vertices and edge slots. Solving graph problems with global queries have been studied extensively in the past few years, such as additive queries [1–7], cross queries [1, 4], independent set queries [7–9, 9–13], bipartite independent set queries [11], cut queries [14], matrix-vector queries [15], and linear queries [16]. The study of global queries is motivated by many applications. Matrix-vector multiplication queries have applications to streaming algorithms [17], cut queries are motivated by connections to submodular function minimization, bipartite independent set queries have been studied in connection with reductions between counting and decision problems [18], additive queries have applications in bioinformatics, and independent set queries are motivated by genome sequencing.

Let A be the adjacency matrix of a simple graph $G = (V, E)$. Some global query models can be interpreted as matrix-vector or vector-matrix-vector multiplication. For example, the answer to

one matrix cut query $S \subseteq V$ is $A\chi_S$, where χ_S is the characteristic vector of S. A cut query S, which returns the number of edges between S and $V \setminus S$, can be represented as $(\mathbf{1} - \chi_S)A\chi_S$, where $\mathbf{1}$ is all ones vector. An additive query S returns $\chi_S^T A\chi_S$, which is twice the number of edges with both endpoints in S. And a bipartite independent set query (S, T), where S and T are disjoint, will return 1 if there is at least an edge connecting S and T, that is, $|\chi_S^\intercal A\chi_T| > 0$, and 0 otherwise. There are also global queries that do not involve adjacency matrix A, for example, linear queries. Let $G = (V, w)$ be a weighted graph where w is the edge weight vector of G. A linear query $x \in \{0, 1\}^{\binom{n}{2}}$ to G returns $\langle w, x\rangle$.

The query complexity of a problem is measured by taking the minimum number of queries over all algorithms that solve the problem. Several techniques have been introduced to compute the lower bounds of graph problems. Eden and Rosenbaum [19] developed a framework for proving query complexity lower bounds for graph properties with local queries via reductions from communication complexity. Communication complexity is well-studied and has numerous applications, such as in proving lower bounds for Turing machines [20], streaming algorithms [21], circuit complexity [22], distributed algorithms [23], and algorithmic game theory [24]. We discuss Eden and Rosenbaum's technique in Chapter 4 and give a proof for local query lower bound for computing the size of the minimum cut. Communication complexity has also been used to prove global query lower bounds for connectivity. Harvey [25] proved that the deterministic cut query lower bound for connectivity is $\Omega(n)$, which follows from the deterministic communication complexity lower bound of $\Omega(n \log n)$ for connectivity [26], and the fact that the answers to cut queries can be communicated with $O(\log n)$ bits. The randomized communication lower bound for connectivity is $\Omega(n)$, giving an $\Omega(n/\log n)$ lower bound for any randomized algorithm solving connectivity with cut queries. We give a self-contained proof in Chapter 5.

The communication complexity approach, however, does not apply to the linear query model. The family of instances used to show the communication complexity lower bound for connectivity are simple graphs. We cannot use simple graphs as "hard" inputs for a linear query lower bound since a simple graph can be learned with a single linear query by querying the vector of powers of 2 of the appropriate dimension. A lower bound technique that goes beyond considering simple graphs is the cut dimension [27]. It was observed in [28] that the lower bound applies to the linear query model, and a strengthening of the technique was given, called the ℓ_1-approximate cut dimension. We observe that the ℓ_1-approximate cut dimension characterizes, up to an additive $+1$,

a well-known query complexity lower bound technique applied to the minimum cut problem, the *certificate complexity*. We further adapt this certificate complexity technique to the connectivity problem in Section 5.2, allowing us to show an $\Omega(n)$ lower bound (Corollary 5.13) for deterministic linear query algorithms solving connectivity. As far as we are aware, this is the first lower bound for connectivity in the unrestricted linear query model.

In Chapter 6, we propose global query algorithms to solve connectivity. Most algorithms that study connectivity typically use the local query model [29–33]. It is known that in the local query model, solving connectivity for n-vertex simple graphs by classical randomized algorithms requires $\Omega(n^2)$ queries [29] in the adjacency matrix model and $\Theta(m)$ queries in the adjacency list model where m is the number of edges. The quantum query complexity of connectivity is $\Theta(n^{3/2})$ in the matrix model and $\Theta(n)$ in the list model, as shown by Dürr et al. [29]. In the case of weighted graphs, finding the minimum spanning tree requires $\Theta(n^{3/2})$ queries in the matrix model and $\Theta(\sqrt{nm})$ in the list model. In the global query model, Sun et al. [15] observe an $O(\log^4(n))$ matrix-vector multiplication queries to the signed vertex-edge incidence matrix $A_{\pm} \in \{-1, 0, 1\}^{n \times \binom{n}{2}}$ for finding a spanning forest. For a *bipartite graph* G with bipartition V_1, V_2, we define the *bipartite adjacency matrix* as the submatrix of the adjacency matrix where rows are restricted to V_1 and columns to V_2. Sun et al. show that to determine if a bipartite graph is connected, we need $\Omega(n/\log n)$ matrix-vector multiplication queries to the bipartite adjacency matrix, where the multiplication of the vector is on the right. However, the family of instances they give can be solved by a single matrix multiplication query on the left side of the bipartite adjacency matrix.

We present a randomized algorithm that can output a spanning forest of a weighted graph with constant probability after $O(\log^4(n))$ matrix-vector multiplication queries to the adjacency matrix, which complements the algorithm by Sun et al. We use the matrix-vector multiplication query model as a master model that can be simulated by other query models to give non-trivial results. We then show applications to quantum algorithms using cut and BIS queries. Our quantum algorithm can output a spanning forest of an unweighted graph after $O(\log^5(n))$ cut queries, improving and simplifying a result of Lee, Santha, and Zhang [34], which gave the bound $O(\log^8(n))$. We also show that a quantum algorithm can output a spanning tree after $O(\sqrt{n})$ BIS queries.

Another problem we study is s-t-minimum cut (Chapter 8). The minimum cut problem has been

extensively studied since at least the 1950s [35]. In the global minimum cut problem, we are interested in finding a cut of minimum size, whereas in the s-t-minimum cut problem, we are looking for a minimum cut that separates the vertices s and t. For unweighted graphs, the size of the global minimum cut can be computed in nearly linear time by deterministic algorithms classically [36]. For weighted graphs with m edges, the weight of a global minimum cut can be determined in nearly linear time $\tilde{O}(m)$ by a randomized algorithm [37, 38] and in almost linear time $O(m^{1+o(1)})$ by a deterministic algorithm [39]. Recent work by Apers and Lee [40] gave a quantum algorithm to solve the global minimum cut problem for undirected and weighted graphs whose weights are at least 1 and at most τ, using $\tilde{O}(n^{3/2}\sqrt{\tau})$ queries and time in the adjacency matrix model.

By the max-flow min-cut theorem [35], the size of the s-t-mincut is equal to the maximum value of the s-t-flow. One can compute an s-t-mincut from a maximum s-t-flow in linear time. However, no such reduction is known the other way around. There has been a surprising lack of work on quantum algorithms for the exact maximum flow or minimum s-t cut problem, which motivates us to study this problem. As far as we are aware, the only work on the quantum complexity of these problems is by Ambainis and Špalek [41], who gave a quantum algorithm for max flow in a directed graph with integral capacities bounded by $W \leq n^{1/4}$ with running time $\tilde{O}(\min\{n^{7/6}\sqrt{m}W^{1/3}, m\sqrt{nW}\})$, given adjacency list access to G. However, current classical randomized algorithms give better results.

Rubinstein, Schramm, and Weinberg (RSW) [14] give query-efficient algorithms for the global min-cut and the s-t-mincut problem in simple graphs in the cut query model. A key idea of their s-t-mincut algorithm is to transform the graph G into a sparser graph G' while preserving the cut values. Following their construction, we develop a quantum algorithm for s-t-mincut that is both query and time-efficient. In our algorithm, we first find a ε-cut sparsifier of G by using the quantum algorithm of Apers and de Wolf [42] which is faster than the classical sparsification algorithm that needs $O(m)$ time. Then, we use Grover's algorithm to learn all the edges in the sparser graph G'. After we explicitly know the graph G' we use a classical randomized algorithm to compute a minimum s-t cut in G'. Our s-t-mincut algorithm computes with high probability the weight of a minimum s-t cut with $\tilde{O}(\sqrt{m}n^{5/6}W^{1/3} + n^{5/3}W^{2/3})$ queries, given adjacency list access to G. For simple graphs this bound is always $\tilde{O}(n^{11/6})$, even in the dense case when $m = \Omega(n^2)$.

Apers and de Wolf's algorithm [42] is constructing a spectral sparsifier that is stronger than a cut sparsifier. In the construction of our algorithm, we only need a cut sparsifier. Since the proof for the quantum algorithm for spectral sparsification in [42] is quite complicated, we include an easier self-contained proof for the query complexity of quantum cut sparsification algorithm in Chapter 7. We also found a gap in the RSW analysis for cut query algorithm for the sparsifier, and fixed the gap as a contribution in this thesis.

1.2 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 covers the essential mathematical background. Chapter 3 covers communication complexity, which we will use to prove some lower bounds. Chapter 4 discusses the query models that we use throughout the thesis, as well as some lower bounds for the local query model. Starting with Chapter 5, we present novel contributions in this thesis. We discuss the global query lower bound for connectivity in Chapter 5 and introduce a technique to compute the deterministic linear query lower bound for connectivity and minimum cut problems. Chapter 6 presents our results for connectivity with global queries. In Chapter 7, we discuss some algorithms to construct a sparsifier. We present our quantum algorithm for the s-t minimum cut problem in Chapter 8. Finally, in Chapter 9, we provide a summary and an outlook on the future work of this thesis.

Chapter 2

Preliminaries

This chapter provides the necessary background for presenting the results in this thesis. It begins with preliminaries from graph theory in Section 2.1. In Section 2.3, an overview of combinatorial group testing is given. Then, linear programming and its duality are discussed in Section 2.4.1. Finally, the chapter concludes with a discussion of the quantum algorithms used in this thesis in Section 2.5.

2.1 Graph theory

Let V be a finite set and $V^{(2)}$ the set of all two-element subsets of V. A *graph* is a pair $G = (V, E)$ where V is the set of vertices and $E \subseteq V^{(2)}$ is the set of edges. For $e = \{i, j\} \in E$, the vertices i and j are *neighbors* and we call them the *endpoints* of edge e. The *degree* of a vertex $v \in V$ is the number of edges incident to that vertex.

A weighted graph $G = (V, w)$ is a graph with an $\binom{n}{2}$-dimensional real vector w representing the edge weights. A *simple graph* is an undirected and unweighted graph with no self-loops and at most one edge between any pair of vertices. The *complete graph* on n vertices, denoted by K_n, is a simple graph in which every pair of vertices is connected by an edge. If the vertices of a graph $G = (V, E)$ can be partitioned into sets U_1 and U_2 such that there is no edge with both endpoints in U_1 or both endpoints in U_2, then G is said to be *bipartite*.

A subgraph of a simple graph $G = (V, E)$ is a pair (V', E') where $V' \subseteq V$ and $E' \subseteq E$, such that if $e = \{u, v\} \in E'$ then $\{u, v\} \in E$ and $u, v \in V'$. Given a subset $S \subseteq V$, $G[S] = (S, E_S)$ denotes the *induced subgraph* of G, that is, E_S is the set of all edges $e \in E$ that have both endpoints in S.

Definition 2.1 (Path)**.** A *path* is a sequence of distinct vertices $(v_1, v_2, \ldots, v_n)$ such that v_i and v_{i+1} are adjacent for $i = 1, \ldots, n-1$.

An undirected graph G is said to be *connected* if for any two vertices $i, j \in V$, there exists a path connecting i and j. Otherwise, G is *disconnected*. We say we disconnect G if we remove edges to produce a disconnected graph. A *connected component* of an undirected graph G is a maximal set of nodes such that each pair of nodes is connected by a path.

Definition 2.2. Let $G = (V, E)$ be a graph. For $\emptyset \neq S \subsetneq V$, the *cut* defined by S is $\Delta_G(S) = \{\{i, j\} : i \in S, j \in V \setminus S, \{i, j\} \in E\}$. We call S and $V \setminus S$ as the *shores* of the cut.

Definition 2.3. Let $G = (V, w)$ be a weighted graph. The *weight* of a cut is

$$w(\Delta_G(S)) = \sum_{e \in \Delta_G(S)} w(e) \tag{2.1}$$

Definition 2.4. We define $\lambda(G)$ as the minimum weight of a cut in G, that is

$$\lambda(G) = \min_{\emptyset \neq S \subsetneq V} w(\Delta_G(S)) \tag{2.2}$$

Definition 2.5 (Minimum cut)**.** Let G be a graph. A *global minimum cut* is a cut $\Delta_G(S)$ such that

$$w(\Delta_G(S)) = \lambda(G) \tag{2.3}$$

Given two vertices $s, t \in V$. A *minimum s-t cut* is a cut $\Delta_G(S)$ with minimum weight such that $s \in S$ and $t \in V \setminus S$.

A *tree* is a connected graph with no cycles. A *forest* is a disjoint union of trees. A *spanning tree* of a graph G is a subset of G that covers all of its vertices using the minimum number of edges. A *spanning forest* of a graph G is a collection of spanning trees across its connected components. We can easily prove that a weighted graph $G = (V, w)$ is connected if and only if it has a spanning tree, and that spanning tree must have exactly $|V| - 1$ edges.

2.1.1 Graph representation

The two most common representations of a graph are the adjacency matrix and the adjacency list. An adjacency matrix is a square matrix where the rows and columns correspond to the vertices of the graph. The entries of the matrix indicate whether pairs of vertices are adjacent or not in the graph, and for an undirected graph, the matrix is symmetric.

Definition 2.6 (Adjacency matrix)**.** For a simple graph $G = (V, E)$ with n vertices and m edges, the *adjacency matrix* A is an $n \times n$ matrix such that

$$A[i,j] = \begin{cases} 1 & \text{if } \{i,j\} \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Definition 2.7 (Adjacency matrix for weighted graphs)**.** Let $G = (V, w)$ be a weighted graph with n vertices and m edges. The *adjacency matrix* A of G is given by

$$A[i,j] = \begin{cases} w[\{i,j\}] & \text{if } i < j \\ w[\{j,i\}] & \text{if } i > j \\ 0 & \text{if } i = j \end{cases} \tag{2.5}$$

An adjacency list is a collection of unordered lists where each list is indexed by the vertices and describes the set of neighbors adjacent to the corresponding vertex.

Definition 2.8 (Adjacency list)**.** Let $G = (V, w)$ be a weighted graph with n vertices and m edges. The *adjacency list* L consists of n arrays such that for each array L_i where $i \in [n]$, we have $L_i[k] = j$ if j is the k-th neighbour of vertex i.

Another graph representation we consider is called the signed vertex-edge incidence matrix. The rows of this matrix are indexed by the vertices, and the columns are indexed by the edge slots. Each entry in the matrix is either 0, 1 or -1. The entry at row i and column j is 1 if vertex i is an endpoint of edge j, -1 if vertex i is the other endpoint of edge j, and 0 otherwise. This representation is particularly useful in directed graphs, as it allows us to represent the direction of the edges as well as their weights.

Definition 2.9 (Signed vertex-edge incidence matrix)**.** Let $G = (V, E)$ be a simple graph. The *signed vertex-edge incidence* matrix $A_{\pm}$ is an $n \times \binom{n}{2}$ matrix whose rows are indexed by the vertices and columns indexed by the edge slots such that,

$$A_{\pm}(u, \{v, w\}) = \begin{cases} 0 & \text{if } \{v, w\} \notin E \text{ or } u \notin \{v, w\} \\ 1 & \text{if } \{v, w\} \in E \text{ and } u \text{ is the smallest element in } \{v, w\} \\ -1 & \text{if } \{v, w\} \in E \text{ and } u \text{ is the largest element in } \{v, w\} \ . \end{cases} \tag{2.6}$$

2.1.2 Several graph problems

In this thesis, we consider two graph problems: the s-t minimum cut problem and the graph connectivity problem, which is a special case of the minimum cut problem.

Definition 2.10 (CONN)**.** The *graph connectivity* problem, denoted by CONN, is a decision problem that decides whether a graph G is connected or not. Note that a graph G is connected iff $\lambda(G) > 0$.

Definition 2.11 (MINCUT)**.** The input in the MINCUT problem is an n-vertex weighted undirected graph $G = (V, w)$ and the goal is to output $\lambda(G)$.

Definition 2.12 (s-t MINCUT)**.** Let $G = (V, w)$ be a weighted and undirected graph. Let $s, t \in V$. The goal of the s-t MINCUT is to output the minimum s-t cut of G.

2.2 Strong connectivity

Definition 2.13. A graph $G = (V, w)$ is *k-connected* if there is no cut of weight less than k in G. The *connectivity* K_G of G is the weight of G's minimum cut.

Definition 2.14. A k-strong component of G is a maximal k-connected vertex-induced subgraph of G. Individual vertices are defined to be ∞-strong components.

Definition 2.15. The *connectivity* of an edge is the minimum value of a cut separating its endpoints.

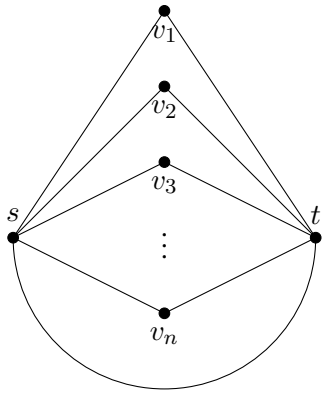


FIGURE 2.1: Graph with edge strength 2 and connectivity $n+1$

Definition 2.16 (Edge strength)**.** Let $G = (V, w)$. The *edge strength* k_e of e is the maximum k such that a k-strong component contains both endpoints of e,

$$k_e = \max_{S \subset V : u,v \in S} K_G(G[S]) \tag{2.7}$$

where $G[S]$ denotes the vertex-induced subgraph of G on S. We say that e is *k-strong* if its strength is at least k, and *k-weak* otherwise.

Note that the edge strength of e is always at most its connectivity. Consider a simple graph with set of vertices $V = \{s, t\} \cup \{v_1, \ldots, v_n\}$ and edges $E = \{\{s, t\}, \{s, v_1\}, \ldots, \{s, v_n\}, \{v_1, t\}, \ldots, \{v_n, t\}\}$ (Fig. 5.1). The connectivity of edge $\{s, t\}$ is $n + 1$, whereas its edge strength is 2 since in any induced subgraph with set of vertices $\{s, v_1, \ldots, v_k, t\}$ for $k \leq n$, we can remove 2 edges ($\{s, v_i\}$ and $\{v_i, t\}$) to disconnect the subgraph.

Lemma 2.17 ([43])**.** *Let $G = (V, w)$ be a weighted graph. Then,*

$$\sum_{e \in E} \frac{w(e)}{k_e} \leq n - 1 \tag{2.8}$$

Lemma 2.18 ([14])**.** *If a graph G has strong connectivity k and no components with strong connectivity $\geq 2k$, there G has $\Theta(nk)$ edges.*

Definition 2.19 (k-strong partition)**.** For a graph $G = (V, w)$, a k-strong partition of G is a partition $\mathcal{P} = \{S_1, \ldots, S_t\}$ of V such that

1. The subgraph $G[S_i]$ is k-connected for each $i = 1, \ldots, t$.

2. $\sum_{\{i,j\}\in[t]^{(2)}} w(S_i, S_j) = O(kn)$.

2.3 Combinatorial group testing

Robert Dorfman proposed group testing in the 1940s to help identify soldiers with syphilis during the Second World War. To identify an individual with syphilis, a blood sample must be drawn and analyzed. Testing each soldier's blood sample was very expensive at the time, and only a tiny proportion of soldiers were likely to be infected. A more efficient testing scheme was to pool soldiers into groups and combine the blood samples in each group. If the combined sample was negative, it indicated that everyone in the group was not infected, and many tests were saved. Otherwise, if the combined sample was positive, at least one person in the group was infected, and more tests were needed.

In combinatorial group testing, it is often assumed that the number of defectives among items is equal to or at most some fixed positive integer [44]. Consider a set S of n items, of which d items are defective. We perform error-free tests to identify the defective items. Each test is applied to a subset of S with size less than n. If the result is negative, then all elements in the subset are safe. But if the result is positive, then at least one element is defective. Our goal is to identify all defective items with a small number of tests. This problem is called the (d, n)-*combinatorial group testing* problem.

Input: $x \in \{0,1\}^n$. The j-th item is called *defective* iff $x_j = 1$.
Test: $T \subseteq \{1, \ldots, n\}$. The result is positive if there exists $j \in T$ such that $x_j = 1$.
Goal: Find defective items in x.

There are two types of algorithms for solving combinatorial group testing problems: adaptive and non-adaptive. With *adaptive* algorithms, the outcome of previous tests is assumed to be known at the time of determining the current test. On the other hand, in *non-adaptive* algorithms, the choice of which test to perform does not depend on the outcomes of previous tests. Suppose we want to test n items with a non-adaptive testing procedure that consists of tests $S_1, S_2, \ldots, S_t$ for some non-negative integer t. We can set up a testing matrix M of size $t \times n$ where $M[i, j] = 1$ if and only if $j \in S_i$. Let x be a vector of length n where $x_i = 1$ if and only if item i is defective and

$x_i = 0$ otherwise. Then, the result vector $y = (y_1, \ldots, y_t)$ is defined as $y_i = M[i] \vee x$ where $M[i]$ is the i-th row vector of M, and $a \vee b = a \cdot b \mod 2$, which is the dot product between a and b.

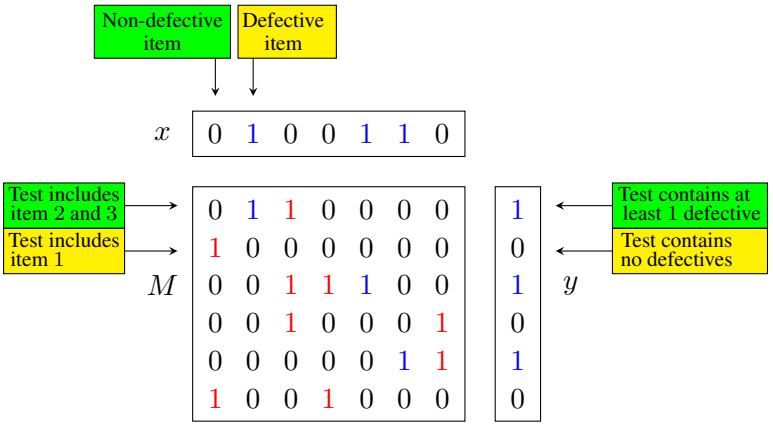


FIGURE 2.2: Non adaptive algorithm for combinatorial group testing. The color blue indicates a defective item, while the color red indicates a test item that is not defective.

2.4 Linear programming

Linear programming is a mathematical optimization problem involving a linear objective function with linear constraints. The problem can be expressed as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \ . \end{aligned} \tag{2.9}$$

where $c, b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$ are given. The value $x \in \mathbb{R}^n$ is to be determined. The first line of Eq. (2.9) is called the *objective function*, while the second and the third lines are called the *constraint functions*.

2.4.1 Linear programming duality

We can obtain a dual problem of Eq. (2.9) by computing its *Lagrange dual function* [45]. The dual problem is as the following

$$\begin{aligned} \underset{\nu}{\text{maximize}} \quad & -b^T\nu \\ \text{subject to} \quad & A^T\nu + c \geq 0 \end{aligned} \tag{2.10}$$

2.5 Quantum algorithms

A quantum query algorithm with T queries is defined as a series of unitary transformations alternating between a unitary operator and a query operation. In each query operation, the algorithm is allowed to access an input function $f : \{0,1\}^n \to \{0,1\}$ by querying the function on a quantum superposition of inputs. The goal of the algorithm is to determine some property of the input function with high probability, such as finding a satisfying assignment for a Boolean formula. The number of queries T is a measure of the efficiency of the algorithm, and a lower bound on T is often used to prove hardness results for quantum algorithms.

$$U_0 \to O_x \to U_1 \to O_x \to \ldots \to U_{T-1} \to O_x \to U_T \tag{2.11}$$

The unitary operators $U_0, \ldots, U_T$ are arbitrary and independent of the input. The oracle O_x depends on the input string $x = x_1 x_2 \ldots x_N$ and performs the following transformation:

$$O_x |i, a\rangle \mapsto |i, a \oplus x_i\rangle , \tag{2.12}$$

where $\oplus$ denotes the bitwise XOR operation. The computation starts from the initial state $|0\rangle$, and the final state is measured to obtain the outcome. We say that a quantum algorithm $\mathcal{A}$ computes a function f if the result of the measurement after termination is $f(x)$ with probability at least $2/3$. The query complexity of $\mathcal{A}$ is T if $\mathcal{A}$ terminates after T oracle calls.

2.5.1 Quantum search algorithm

Grover[46] introduced a quantum algorithm to solve the unstructured (not sorted) search problem, which provides a quadratic speed-up over the best possible classical algorithms. Let us define the *search problem* as follows:

Given a set $S = 0, 1, \ldots, N-1$ for some integer N, and a function $f : S \to 0, 1$ such that $|f^{-1}(1)| = 1$, find the unique $x \in S$ such that $f(x) = 1$. We call x the marked element.

INPUT: Oracle access to f.

OUTPUT: The marked element.

In the Grover algorithm, we have access to an oracle that adds a negative phase to the solution state:

$$U_\omega |x\rangle = \begin{cases} |x\rangle & \text{if } x \neq \omega \\ -|x\rangle & \text{if } x = \omega \end{cases} \tag{2.13}$$

Grover's search algorithm can find a marked element after $O(\sqrt{N})$ calls to U_ω [46]. If there are $0 < M \leq N$ marked elements, we can find one marked element after $O(\sqrt{N/M})$ oracle calls.

Theorem 2.20 ([47], Theorem 3)**.** *Suppose the number of marked elements M among N elements is unknown. We can find a marked element in expected time $O(\sqrt{N/M})$.*

To find all marked elements, we can repeat Grover's search algorithm and remove the element that we find in the input set after each application. Hence, to find all M marked elements, we can apply Theorem 2.20 M times. The total number of calls to the oracle is $O(\sqrt{NM})$, as stated in the following theorem.

Theorem 2.21 ([**?**], Repeated Grover Search)**.** *Let $f : \{0, 1, \ldots, N-1\} \to \{0, 1\}$. Suppose $|f^{-1}(1)| = M$. Then there is a quantum algorithm that finds all x such that $f(x) = 1$ with probability $2/3$ after $O(\sqrt{NM})$ oracle calls.*

Theorem 2.22 (cf. [40, Theorem 13])**.** *Given $t, N \in \mathbb{N}$ with $1 \leq t \leq N$ and oracle access to $x \in \{0, 1\}^N$, there is a quantum algorithm such that*

- *if $|x| \leq t$ then the algorithm outputs x with certainty, and*
- *if $|x| > t$ then the algorithm reports so with probability at least $9/10$.*

The algorithm makes $O(\sqrt{tN})$ queries to x and has time complexity $O(\sqrt{tN}\log(N))$.

2.5.2 Bernstein-Vazirani's algorithm

Given an oracle access to a function $f : \{0,1\}^n \to \{0,1\}$ and a promise that $f(x) = s \cdot x$, where s is a secret string. Bernstein-Vazirani's algorithm tries to learn this secret string. We have access to an oracle U_f that transforms $|x\rangle \to (-1)^{f(x)} |x\rangle$.

Theorem 2.23 ([48]). *Let $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = x \cdot s$ for some $s \in \{0,1\}^n$. With access to U_f, we can find s correctly with one oracle call.*

2.5.3 Belov's combinatorial group testing

Let $k \leq n$ be a fixed positive integer, and let $\mathcal{C}$ denote the set of all subsets of $[n]$ of size at most k. For each $A \in \mathcal{C}$, define $f_A : 2^{[n]} \to \{0,1\}$ as

$$f_A(S) = \begin{cases} 1 & \text{, if } A \cap S \neq \emptyset \\ 0 & \text{, otherwise} \end{cases} \tag{2.14}$$

The goal of the combinatorial group testing problem is to determine A given oracle access to f_A. Belovs showed that this problem can be solved with $\Theta(\sqrt{k})$ queries quantumly.

Theorem 2.24 (Belovs [49]). *Let $A \in \{0,1\}^n$. There is a quantum algorithm with oracle access to f_A that outputs A with probability at least $2/3$ after $O(\sqrt{k})$ queries to A.*

2.5.4 Quantum minimum finding

Let $T[0, \ldots, N-1]$ be an unordered table of N items where each item holds a value from an ordered set. The minimum search problem is to find an index y such that $T[y]$ is minimum. Classically, this problem requires linear time to solve. However, with a quantum algorithm, we can find the index of the minimum value in time $O(\sqrt{N})$ [50].

Theorem 2.25 ([50], Theorem 1). *Algorithm 1 returns the index of the minimum value in T in time $O(\sqrt{N})$ with probability at least $1/2$.*

Algorithm 1 Quantum minimum finding

Input:

Output: An element with minimum value.

1: $y \leftarrow_R \{0, 1, \ldots N-1\}$.
2: **while** Total running time $\leq 22.5\sqrt{N} + 1.4 \log^2 N$ **do**
3: Initialize $|\Psi\rangle = \sum_i \frac{1}{\sqrt{N}} |i\rangle |y\rangle$. Mark every item i for which $T[i] < T[y]$.
4: Apply **??**.
5: Observe the first register and let y' be the outcome.
6: **if** $T[y'] < T[y]$ **then**
7: $y \leftarrow y'$.
8: **end if**
9: **end while**
10: Return y.

Chapter 3

Communication Complexity

In this chapter, we discuss communication complexity, which is a useful tool for proving lower bounds for some graph problems. We begin the chapter with an introduction to Möbius inversion in Section 3.1, which will be used later in Theorem 3.5 to prove the deterministic communication complexity lower bound. In Section 3.2, we discuss both deterministic and randomized communication complexity.

3.1 Möbius inversion

A *partially ordered set* or *poset* is a set P equipped with a binary relation $\leq$ satisfying the following properties for all $x, y, z \in P$:

(1) $x \leq x$ (reflexivity).

(2) if $x \leq y$ and $y \leq z$ then $x \leq z$ (transitivity).

(3) if $x \leq y$ and $y \leq x$ then $x = y$ (anti-symmetry).

Two elements $x, y \in P$ are *comparable* if $x \leq y$ or $y \leq x$, and *incomparable* otherwise. A poset without incomparable elements is a *total* order. A *maximal* element z of a poset $(P, \leq)$ is an element that is not less than any other element in the poset, that is, for any $x \in P$ if $z \leq x$ then $x = z$. *Minimal* elements are similarly defined.

A poset P can be represented by a *Hasse diagram*, which is a graph where the vertices represent the elements of P. Each element x is placed above y if $x > y$ and x, y are joined by an edge if x covers y (there does not exist z distinct from both x and y such that $x \geq z \geq y$).

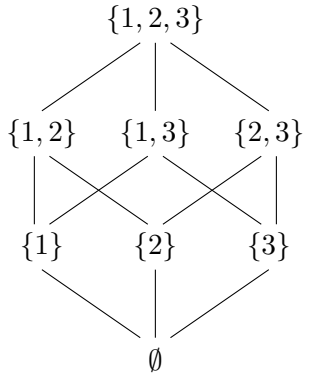


FIGURE 3.1: Hasse diagram for the set of subsets of $[3]$ ordered by inclusion.

A *lattice* is a partially ordered set $\mathcal{L} = (\mathcal{L}, \leq)$ such that for any $x, y \in \mathcal{L}$:

(1) The subset $Z = \{z : z \geq x \text{ and } z \geq y\} \subseteq \mathcal{L}$, with induced ordering has a unique minimal element, denoted by $x \vee y$. We call this element the *least upper bound* or *join* of x and y.

(2) The subset $Z' = \{z : z \leq x \text{ and } z \leq y\} \subseteq \mathcal{L}$, has a unique maximal element, denoted by $x \wedge y$. This element is called the *greatest lower bound* or *meet* of x and y. An element is *irreducible* if $x = y \vee z \implies y$ or $z = x$.

Both $\wedge$ and $\vee$ are commutative, associative, and idempotent. A lattice is said to be finite if the set $\mathcal{L}$ is finite. Every finite lattice has a unique least element denoted by 0 and a unique greatest element denoted by 1.

Definition 3.1 (Möbius function)**.** Let $\mathcal{P} = (P, \leq)$ be a partially ordered set. Möbius function $\mu : P \times P \to \mathbb{Z}$ is defined by

$$\mu(x, x) = 1 \qquad x \in P \tag{3.1}$$

$$\mu(x, y) = 0 \qquad x \not\leq y \tag{3.2}$$

$$\mu(x, y) = -\sum_{x \leq z < y} \mu(x, z) \qquad x < y \tag{3.3}$$

Theorem 3.2 (Möbius inversion)**.** *Let $\mathcal{P} = (P, \leq)$ be a finite partially ordered set and $f : P \to \mathbb{R}$ be a function.*

1. *Suppose for all $x \in P$, g is defined by*

$$g(x) = \sum_{y \leq x} f(y) \tag{3.4}$$

Then,

$$f(x) = \sum_{y \leq x} g(y)\mu(y, x) \tag{3.5}$$

2. *If g is defined by*

$$g(x) = \sum_{y \geq x} f(y) \tag{3.6}$$

Then,

$$f(x) = \sum_{y \geq x} g(y)\mu(x, y) \tag{3.7}$$

Lemma 3.3 ([51], Problem 25D)**.** *Let $\mathcal{L}$ be a finite lattice and. For any $a \in \mathcal{L} - \{1\}$ we have*

$$\sum_{x \wedge a = 0} \mu(x, 1) = 0 \tag{3.8}$$

Lemma 3.4 ([51])**.** *Let Π_n be a partially ordered set of partitions of n ordered by refinement (P is defined as the refinement of Q if all elements of P is contained in Q). Then for all $\alpha \in \Pi_n, \alpha < 1$ we have*

$$\mu_{\Pi_n}(\alpha, 1) = (-1)^{\lambda - 1}(\lambda - 1)! \tag{3.9}$$

where $\lambda = |\alpha|$.

Proof. Let $[x, y]$ denote all z such that $x \leq z \leq y$. For all $\tau \in [\alpha, 1]$, every set $S \in \tau$ is a union of some sets in α. Therefore $[\alpha, 1] \cong \Pi_\lambda$ and $\mu_{\Pi_n}(\alpha, 1) = \mu_{\Pi_\lambda}(0, 1)$. Pick a maximal element $a \in \Pi_\lambda$ such that a is a partition of the form $(i, [\lambda] - i)$. If $x \wedge a = 0$ then $x = 0$ or x contains $\lambda - 1$ blocks of $\lambda - 2$ singleton and one 2-element set where one of them is i. Let $x_1, \ldots, x_{\lambda - 1}$

be such partitions. Note that $[x_i, 1] \cong \Pi_{\lambda-1}$. Then, from Lemma 3.3 we have

$$\sum_{\substack{x \in \Pi_\lambda \\ x \wedge a = 0}} \mu_{\Pi_\lambda}(x, 1) = \mu_{\Pi_\lambda}(0, 1) + \sum_{i=1}^{\lambda-1} \mu_{\Pi_\lambda}(x_i, 1) = 0 \tag{3.10}$$

Hence,

$$\begin{aligned} \mu_{\Pi_\lambda}(0, 1) &= -\sum_{i=1}^{\lambda-1} \mu_{\Pi_\lambda}(x_i, 1) && (3.11) \\ &= -(\lambda - 1)\mu_{\Pi_{\lambda-1}}(0, 1) && (3.12) \\ &= (-1)^{\lambda-1}(\lambda - 1)! && (3.13) \end{aligned}$$

□

Let $\mathcal{L}$ be a lattice and $V(\mathcal{L})$ be the free vector space over $\mathbb{Q}$ generated by elements in $\mathcal{L}$, i.e., the set of all mappings $f : \mathcal{L} \to \mathbb{Q}$ with addition and scalar multiplication. Define I_x, J_x, K_x as:

$$I_x(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

$$J_x = \sum_{y : y \vee x = 1} I_y \tag{3.15}$$

$$K_x = \sum_{y : y \leq x} I_y \tag{3.16}$$

Theorem 3.5 ([52], Dowling-Wilson)**.** *Let $\mathcal{L}$ be a finite lattice. For each $x \in \mathcal{L}$ the following equations hold in $V(\mathcal{L})$,*

(1) $I_x = \sum_{y : y \leq x} \mu(y, x) K_y$.

(2) $J_x = \sum_{y : y \geq x} \mu(y, 1) K_y$.

(3) $\mu(x, 1) K_x = \sum_{y : y \geq x} \mu(x, y) J_y$.

(4) If $\mu(\alpha, 1) \neq 0$ for all $a \in \mathcal{L}$, then

$$I_x = \sum_y \lambda(x, y) J_y \tag{3.17}$$

where

$$\lambda(x, y) = \sum_{\alpha : \alpha \leq x \wedge y} \frac{\mu(\alpha, x)\mu(\alpha, y)}{\mu(\alpha, 1)} \tag{3.18}$$

Proof. (1) We apply Möbius inversion to the definition of K_x. From Eq. (3.19), we have

$$K_x = \sum_{y : y \leq x} I_y$$

The Möbius inversion of K_x according to Eq. (3.5) is

$$I_x = \sum_{y : y \leq x} K_y \mu(y, x) \tag{3.19}$$

(2) Observe that

$$\begin{aligned} \sum \mu(y, 1) K_y &= \sum_{y : y \geq x} \mu(y, 1) \sum_{z : z \leq y} I_z \\ &= \sum_z \left(\sum_{y : y \geq x \vee z} (\mu(y, 1)) \right) I_z = J_x \end{aligned} \tag{3.20}$$

since

$$\sum_{y : y \geq x \vee z} \mu(y, 1) = \begin{cases} 1 & \text{if } x \vee z = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

(3) Apply Möbius inversion to (2) (Similar to (1)).

(4) By applying Möbius inversion to Eq. (3.19) and from (1) and (3) we get,

$$I_x = \sum_{\alpha : \alpha \leq x} \mu(\alpha, x) K_\alpha \tag{3.22}$$

$$= \sum_{\alpha : \alpha \leq x} \frac{\mu(\alpha, x)}{\mu(\alpha, 1)} \mu(\alpha, 1) K_\alpha \tag{3.23}$$

$$= \sum_{\alpha : \alpha \leq x} \frac{\mu(\alpha, x)}{\mu(\alpha, 1)} \left(\sum_{y : y \geq \alpha} \mu(\alpha, y) J_y \right) \tag{3.24}$$

□

3.2 Communication Complexity

Suppose we have finite sets X and Y, and a function $f : X \times Y \to \{0,1\}$. Two players, Alice and Bob, are given the task of computing $f(x,y)$, where $x \in X$ and $y \in Y$. Alice holds input x while Bob holds input y, and neither knows the other's input. We are interested in determining the minimum amount of communication between Alice and Bob required for them to both know the value of $f(x,y)$. We define a communication protocol as follows.

Definition 3.6. Let $f : X \times Y \to \{0,1\}$ be a function. A t-round *communication protocol* Π is a sequence of functions $P_1, \ldots, P_t : \{0,1\}^* \to \{0,1\}^*$. For all $i \geq 1$, Alice computes $a_i = P_i(x, a_1, \ldots, a_{i-1})$ if i is odd and sends a_i to Bob. Whereas Bob computes $a_i = P_i(y, a_1, \ldots, a_{i-1})$ if i is even and sends a_i to Alice. The protocol Π is valid if $\Pi(x,y) = f(x,y)$ for every pair x, y, i.e., the last message is the value $f(x,y)$.

3.2.1 Deterministic communication

Definition 3.7. Let Π be a deterministic communication protocol. The *communication cost* of Π is defined as

$$\text{cost}(\Pi) = \max_{x,y} |\Pi(x,y)| \tag{3.25}$$

where $|\Pi(x,y)|$ is the number of bits communicated on inputs x and y, i.e., $\sum_{i=1}^{t} |a_i|$.

Definition 3.8. Let $f : X \times Y \to \{0,1\}$ be a function and Π be a deterministic communication protocol computing f. The *deterministic communication complexity* of f is defined as

$$D(f) = \min_{\Pi} \text{cost}\{\Pi | \Pi \text{ computes } f\} \tag{3.26}$$

Definition 3.9. Let $f : X \times Y \to \{0,1\}$ be a function. A *communication matrix* M_f is a $|X| \times |Y|$ matrix such that

$$M_f[x,y] = f(x,y) \tag{3.27}$$

We have the following lower bound for deterministic communication complexity.

Theorem 3.10. $D(f) \geq \log(rk(M_f))$

By using Theorem 3.10 we can compute the lower bound for graph connectivity, as we can see in the following theorem.

Theorem 3.11 ([53])**.** *The deterministic communication complexity of connectivity is* $\Theta(n \log(n))$.

Proof. Suppose we have n vertices, and we partition these vertices into three sets A, B, C. Alice gets edges in $A \times B$ and Bob gets edges in $B \times C$. Let $P = \{S_1, \ldots, S_k\}$ and $Q = \{T_1, \ldots, T_k\}$ be partitions of B. Define graph G_P as follows: for each S_i, choose $v_i \in A$ that does not connect to S_j for any j, and connect all vertices in S_i to v_i. So we have each distinct S_i connects with distinct v_i. Connect all vertices in $A - \{v_1, \ldots, v_k\}$ to some vertex in B. Define G_Q in the same way for C and partition Q. Then $G = G_P \cup G_Q$ is connected iff $P \vee Q = 1$. To see this, suppose $P \vee Q = R$ where $R \neq 1$, then there exist X, Y partitions of R and we have two disconnected graphs $(X_A - X - X_C)$ and $(Y_A - Y - Y_C)$, where Z_r is the set of vertices in $r \in \{A, C\}$ connected to vertices in $Z \in \{X, Y\}$.

Let $P_1, \ldots P_t$ be partitions of B. Let $M \in \mathbb{Z}_2^{t \times t}$ where $M[P_i, P_j] = 1$ iff $P_i \vee P_j = 1$ (i.e. the graph $G_{P_i} \cup G_{P_j}$ is connected). Let $J_1, \ldots, J_t$ be the column vectors of M. Since $J_x(y) = 1$ iff $x \wedge y = 1$ we have $J_x = \sum_{y: y \wedge x = 1} I_y$. From Theorem 3.5 and the fact that $\mu(\alpha, 1) \neq 0$ for all $\alpha < 1$ (Lemma 3.4), every basis vector I_x can be represented as a linear combination of $J_1, \ldots, J_t$. Hence, M is a full rank matrix. Consider the worst case where $|B| = \Omega(n)$. Therefore, we have

$$D(\text{CONN}) \geq \log(rk(M)) = \log(t) = \Omega(\log(B_n)) = \Omega(n \log(n)) \tag{3.28}$$

where B_n is the n-th Bell number. □

3.2.2 Randomized communication

In this setting, we allow for some error ε when computing the value $f(x, y)$. Both Alice and Bob have access to a shared random string r that they will use to determine the message they send at each round. Now, we can define the randomized communication protocol as follows.

Definition 3.12. A *randomized communication protocol* Π computes $f : X \times Y \to \{0,1\}$ iff for all $(x,y) \in X \times Y$,

$$Pr_{\varepsilon}\left[\Pi(x,y;r) = f(x,y)\right] \geq \varepsilon \tag{3.29}$$

Definition 3.13. Let Π be a randomized communication protocol. The (expected) *communication cost* of Π is defined by

$$\text{cost}_{\epsilon}(\Pi) = \sup_{x,y} \mathop{\mathbf{E}}_{r}(|\Pi(x,y;r)|) \tag{3.30}$$

Definition 3.14. The *randomized communication complexity* of f, denoted by $R(f)$, is the minimum cost of any protocol that computes f:

$$R(f) = \min\{\text{cost}_{\epsilon}(\Pi) | \Pi \text{ computes } f\} \tag{3.31}$$

Babai, Frankl, and Simon [26] proved that the randomized communication complexity for the graph connectivity problem is $\Omega(n)$.

Theorem 3.15 ([26], Corollary 7.5)**.** *The randomized communication complexity of graph connectivity is $\Omega(n)$, where n is the number of vertices.*

The following problem will be useful in Chapter 4 where we prove the lower bound for the size of minimum cut with local queries.

Definition 3.16 (k-Intersection)**.** Let k, N such that $k \leq N$. Let $S = \{(x,y) \in \{0,1\}^N \times \{0,1\}^N : \sum_{i=1}^{N} x_i y_i = k \text{ or } k-1\}$. The **k-Intersection** function $\text{INT}_k^N : S \to \{0,1\}$ is defined as

$$\text{INT}_k^N(x,y) = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} x_i y_i = k \\ 0 & \text{otherwise} \end{cases} \tag{3.32}$$

Theorem 3.17 ([54])**.** *The randomized communication complexity of the k-Intersection function on N bits is $\Omega(N)$.*

Chapter 4

Query Models

In the *query access model*, the algorithm only has limited access to the input. The input is concealed behind an oracle, and to access it, an algorithm is allowed to query the oracle for some information about the input. In the query complexity model, the goal is to minimize the number of queries. One important assumption is that, unlike the time complexity model, we do not have any constraints on the amount of computation between queries. We divide the query model into two categories: the *local* query model and the *global* query model. In Section 4.1, we show some examples of local query models. Then we provide some lower bounds for graph problems for both classical and quantum local queries in Section 4.2. In Section 4.3, we discuss the global query model and the relationship between global queries in Section 4.4.

4.1 Local queries

Graph algorithms with local queries have been studied extensively. Local queries allow us to access information about one vertex or edge slot with a single query. There are two models that we consider: adjacency matrix and adjacency list models.

Let $G = (V, E)$ be a graph with $|V| = n$ and $|E| = m$. Let $\mathcal{G}_n$ denote the set of all graphs with n vertices and we define $\mathcal{G} = \bigcup_{n \in \mathbb{N}} \mathcal{G}_n$. A *query* is defined as an arbitrary function $q : \mathcal{G} \to \{0, 1\}^*$.

Adjacency matrix

- *Pair queries*: A function $\text{pair} : V^{(2)} \to \{0, 1\}$, where $\text{pair}(u, v)$ returns 1 if $\{u, v\} \in E$ and 0 otherwise.

Adjacency list

- *Degree queries*: A function $d : V \to [n - 1]$, where $d(u)$ is the degree of vertex u.
- *Neighbor queries*: A function $\text{nbr}_i : V \to V \cup \emptyset$, where $\text{nbr}_i(u)$ returns the i-th neighbor of vertex u if exists and $\emptyset$ otherwise.

The local query model has been extensively used to solve graph problems. Goldreich, Goldwasser, and Ron [55] solved the property testing problem using the adjacency matrix model, which is appropriate for dense graphs. Later, Goldreich and Ron introduced the adjacency list model [56], which allows accessing the graph through neighbor queries, assuming the maximum degree of the graph is bounded. Another model for sparse graphs without the bounded degree assumption was introduced by Parnas and Ron [57] through degree queries.

4.2 Lower bounds for graph problems

4.2.1 Classical lower bound

Eden and Rosenbaum (ER) [19] introduced a technique for proving lower bounds on the number of queries required to solve various graph problems, including counting subgraphs, sampling edges, estimating the number of triangles, and estimating edge-connectivity. Their technique adapts the property testing lower bound technique introduced by Blais, Brody, and Matulef [58], which established a connection between communication complexity and property testing. Bishnu et al. [54] recently adapted ER's lower bound technique to prove a query lower bound for computing the exact global minimum cut of a graph.

Our focus is on the local query model, where we aim to characterize the query complexity of graph problems. We consider randomized algorithms where the randomness is provided by a random string $r \in \{0, 1\}^*$. We use the worst-case query complexity.

Definition 4.1. A *graph parameter* is a function $g : \mathcal{G} \to \mathbb{R}$ that is invariant under any permutation of vertices of each $G \in \mathcal{G}$.

Definition 4.2. Let $g : \mathcal{G} \to \mathbb{R}$ be a graph parameter. We say that an algorithm $\mathcal{A}$ computes a $(1 \pm \varepsilon)$-approximation of g for $\varepsilon > 0$, if for all $G \in \mathcal{G}$, the output of $\mathcal{A}$ satisfies

$$\Pr_r(|\mathcal{A}(G) - g(G)| \leq \varepsilon |g(G)|) \geq \frac{2}{3} \tag{4.1}$$

Definition 4.3. Let $P \subseteq \{0,1\}^N \times \{0,1\}^N$. Suppose $f : P \to \{0,1\}$ is an arbitrary (partial) function and let $g : \mathcal{G}_n \to \{0,1\}$. Let $\mathcal{E} : \{0,1\}^N \times \{0,1\}^N \to \mathcal{G}_n$. We call the pair $(\mathcal{E}, g)$ an *embedding* of f if for all $(x, y) \in P$ we have $f(x, y) = g(\mathcal{E}(x, y))$.

Definition 4.4. Let $q : \mathcal{G}_n \to \{0,1\}^*$ be a query and $(\mathcal{E}, g)$ be an embedding of f. The query q has *communication cost* $\mathrm{cost}_{\mathcal{E}}(q) \leq B$ if there exists a (zero-error) communication protocol Π_q such that for all $(x, y) \in P$ we have $\Pi_q(x, y) = q(\mathcal{E}(x, y))$ and $|\Pi_q(x, y)| \leq B$.

Theorem 4.5. *Let Q be the set of queries, $f : P \to \{0,1\}$ and $(\mathcal{E}, g)$ be an embedding of f. Suppose $\mathrm{cost}_{\mathcal{E}}(q) \leq B$, for all $q \in Q$. Then any randomized algorithm $\mathcal{A}$ that computes g with probability at least $2/3$ using queries from Q will have expected query complexity of $\Omega(R(f)/B)$.*

Proof. Suppose $\mathcal{A}$ computes g using T queries. Define a randomized communication protocol Π_f with a shared randomness r where for all $x, y \in P$, $\Pr_r[\Pi_f(x, y) = f(x, y)] \geq 2/3$ from $\mathcal{A}$ as follows: Let x, y be inputs for Alice and Bob respectively. Both of them invoke $\mathcal{A}$ on $\mathcal{E}(x, y)$ and let their shared randomness r be the randomness of $\mathcal{A}$. They invoke Π_q to compute the response to a query q. The protocol terminates when $\mathcal{A}$ halts and returns $\mathcal{A}(\mathcal{E}(x, y))$. Since $\Pr_r(\mathcal{A}(\mathcal{E}(x, y)) = g(\mathcal{E}(x, y))) \geq 2/3$ and $g(\mathcal{E}(x, y)) = f(x, y)$, we know that Π_f computes f. Since $\mathrm{cost}_{\mathcal{E}}(q) \leq B$, the communication cost of Π_f is at most BT. Hence, $T \geq R(f)/B$. □

From Theorem 4.5, we can construct the following framework for computing the lower bound of graph problems:

1. Choose a "hard" communication problem f.
2. Define $\mathcal{E} : f \to \mathcal{G}_n$ and $g : \mathcal{G}_n \to \{0,1\}$ such that $(\mathcal{E}, g)$ is an embedding of f.
3. For each query $q \in Q$ bound B.

We can now use the above technique to compute the lower bound for the minimum cut.

Theorem 4.6 (Size of minimum cut[54]). *The randomized lower bound to decide if $\lambda(G) = t$ or $\lambda(G) = t - 2$ with probability 2/3 with degree, neighbor, and pair queries is $\Omega(m)$.*

To prove the lower bound, we choose the **k-Intersection** problem (Definition 3.16) as our hard communication problem.

Proof of Theorem 4.6. We will prove by giving a reduction from $\text{INT}^N_{t/2}$ (Definition 3.16). Let $s = t + \sqrt{t^2 + (m - nt)/2}$ and $N = s^2$. Then we have $s = \Theta(\sqrt{m})$. We identify $\{0,1\}^N$ as $\{0,1\}^{s \times s}$, so that elements of $x \in \{0,1\}^N$ are indexed by two parameters $x = (x_{ij})$ where $1 \leq i, j \leq s$. Suppose x is Alice's input and y is Bob's input and $\sum_{i=1}^N x_i y_i = t/2$. Construct the graph $\mathcal{E}(x, y) = (V, E)$ as follows:

1. Partition V into sets A, A', B, B', C such that $|A| = |A'| = |B| = |B'| = s$ and $|C| = n - 4s$.
2. Each $v \in C$ is connected to $2t$ distinct vertices in A arbitrarily.
3. Construct the following edges:

$$\begin{cases} \{a_i, b'_j\}, \{b_i, a'_j\} \in E & \text{if } x_{ij} = y_{ij} = 1 \\ \{a_i, a'_j\}, \{b_i, b'_j\} \in E & \text{otherwise} \end{cases} \tag{4.2}$$

We define the partial function $g : \mathcal{G}_n \to \{0,1\}$ by

$$g(G) = \begin{cases} 1 & \text{if } \lambda(G) = t \\ 0 & \text{if } \lambda(G) = t - 2 \end{cases} \tag{4.3}$$

We claim that $(\mathcal{E}, g)$ is an embedding of $\text{INT}^N_{t/2}$.

By construction, the degree of every $v \in C$ is $2t$. For any $v \notin C$, the neighbors of every v in C is fixed irrespective of x and y, and the number of neighbors outside C is $s \geq 2t$. In the case where $\text{INT}^N_{t/2} = 0$, there are $t - 2$ edges between $C \cup A \cup A'$ and $B \cup B'$, whereas for $\text{INT}^N_{t/2} = 1$, there are t edges, and removing them will disconnect $\mathcal{E}(x, y)$.

We need to prove the following claims to show that $(\mathcal{E}, g)$ is an embedding of $\mathrm{INT}^N_{t/2}$.

Claim 4.7. *Every pair of vertices in $A \cup A' \cup C$ is connected by at least $3t/2$ edge-disjoint paths. And every pair of vertices in $B \cup B'$ is also connected by at least $3t/2$ edge disjoint path.*

Proof. We consider the following cases:

Case 1: $u, v \in A$. Since $\frac{t}{2} \geq \sum_{i=1}^{N} x_i y_i \geq \frac{t-2}{2}$, then u, v have at least $s - t \geq 3t/2$ common neighbors in A'. Thus, there are at least $3t/2$ edge-disjoint paths connecting them.

Case 2: $u \in A$ and $v \in A'$. Let $u_1, \ldots, u_{3t/2}$ be $3t/2$ distinct neighbors of $v \in A$. Since u has $3t/2$ common neighbors with each $u_r, r \in [3t/2]$, there is a matching of size $3t/2$. Denote this matching by (v_r, u_r). Thus, $\{u, v_r\}, \{v_r, u_r\}, \{u_r, v\}$ forms a set of edge-disjoint paths of size $3t/2$ from u to v, each of length 3. If u is one of the neighbors of v, then one of the $3t/2$ paths gets reduced to $\{u, v\}$, a length 1 path that is edge-disjoint from the remaining paths.

Case 3: $u, v \in C$. Let $u_1, \ldots, u_{2t} \in A$ and $v_1, \ldots, v_{2t} \in A'$ be the neighbors of u and v respectively. If for some $i, j \in [2t], u_i = v_j$, then $\{u, u_i\}, \{u_i, v_j\}, \{v_j, v\}$ is a path. Thus, assume $u_i \neq v_j$ for all $i, j \in [2t]$. Since u_i and v_i have at least $3t/2$ common neighbors in A', we can find $3t/2$ edge disjoint paths $\{u_i, v'\}, \{v', v_i\}$ where $v' \in A'$. The existence of $3t/2$ edge disjoint paths from $u \in C$ to $v \in A$ can be proved as in Case 1, and from $u \in C$ to $v \in A'$ as in Case 2.

The proof for every pair of vertices in $B \cup B'$ is similar.

□

Claim 4.8. *Let $X = C \cup A \cup A'$ and $Y = B \cup B'$. The following statements hold:*

(1) If $\mathrm{INT}^N_{t/2} = 1$, then the set of t edges between X and Y forms the unique global minimum cut of $\mathcal{E}(x, y)$.

(2) If $\mathrm{INT}^N_{t/2} = 0$, then the set of $t - 2$ edges between X and Y forms the unique global minimum cut of $\mathcal{E}(x, y)$.

(3) For any $i, j \in [s]$, $x_{ij} = y_{ij} = 1$ if and only if $\{a_i, b'_j\}, \{b_i, a'_j\}$ are edges in the unique global minimum cut.

Proof. To prove (1), if $\mathrm{INT}^N_{t/2} = 1$, there are t edges between $C \cup A \cup A'$ and $B \cup B'$ that disconnect $\mathcal{E}(x, y)$. By Claim 4.7, it is clear that $\lambda(\mathcal{E}(x, y)) = t$, and thus these edges form the unique global minimum cut. We prove (2) similarly. (3) follows from the construction of $\mathcal{E}(x, y)$ and (1) & (2). □

Now, the degree, neighbor, and pair queries can be simulated by at most 2 bits of communication. Hence, by Theorem 3.17 and Theorem 4.5, any algorithm $\mathcal{A}$ requires $\Omega(N) = \Omega(s^2) = \Omega(m)$ queries to compute the size of minimum cut of G.

□

4.2.2 Quantum lower bound

Lower bounds for connectivity in the quantum setting were proven by Dürr et al. [29]. In the adjacency list model, the proof for the connectivity lower bound is by reduction from the parity problem.

Definition 4.9. Let $x \in \{1, -1\}^n$. The PARITY problem is defined as

$$\mathrm{PARITY}(x) = \prod_{i=1}^{n} x_i \tag{4.4}$$

Theorem 4.10 ([59])**.** *The quantum query complexity of PARITY is* $\Omega(n)$.

Theorem 4.11 ([29])**.** *The graph connectivity problem requires* $\Omega(n)$ *in the adjacency list model.*

Proof. Let $x \in \{1, -1\}^p$ be an instance of PARITY. We construct a permutation π on $V = \{v_0, \ldots, v_{2p-1}\}$ according to x where for all $i \in [p]$ and $b = x_i$,

$$\pi(v_{2i}) = v_{2i+\frac{5}{2}-\frac{b}{2}}$$
$$\pi(v_{2i+1}) = v_{2i+\frac{5}{2}+\frac{b}{2}}$$

The graph defined by π has two levels and p columns where $v_1, v_3, \ldots v_{2p-1}$ are on the upper level and $v_0, \ldots, v_{2p-2}$ are on the lower level, as we can see in Fig. 4.1. A walk starting from v_0 and

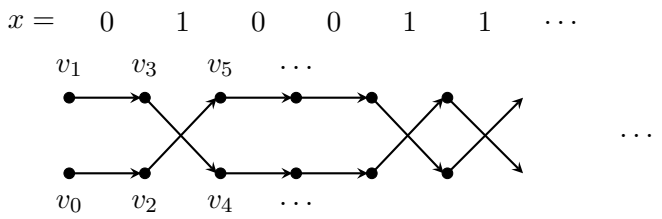


FIGURE 4.1: Reduction from parity

v_1 corresponds to π. If $x_i = 1$, then it stays on the same level, otherwise, it changes level. Then, if the parity of x is even, the walk will return to v_0 after exploring half of the graph, otherwise it returns to v_1, and after another p more steps, it connects back to v_0. Theorem 4.10 concludes the proof.

□

For the adjacency matrix model, we use the Ambainis' adversary bound technique [60] to prove the connectivity lower bound.

Theorem 4.12 (Basic Adversary Bound, [60])**.** *Let $f : [q]^n \to \{0,1\}^n$ be a function. Let $X \subseteq f^{-1}(1), Y \subseteq f^{-1}(0)$, and $R \subseteq X \times Y$ be a relation such that*

- *for every $x \in X$ there are at least m different $y \in Y$ such that xRy.*
- *for every $y \in Y$ there are at least m' different $x \in X$ such that xRy.*
- *for every $x \in X$ and $i \in [n]$ there are at most $l_{x,i}$ different $y \in Y$ such that xRy and $x_i \neq y_i$.*
- *for every $y \in Y$ and $i \in [n]$ there are at most $l'_{y,i}$ different $x \in X$ such that xRy and $x_i \neq y_i$.*

Then any quantum algorithm computing f uses $\Omega(\frac{mm'}{l_{max}})$ where l_{max} is the maximum of $l_{x,i}l'_{y,i}$ subject to xRy and $x_i \neq y_i$.

The graph connectivity lower bound in the adjacency matrix model is proven in the following theorem.

Theorem 4.13 ([29]). *The problem* CONN *requires* $\Omega(n^{3/2})$ *queries in the adjacency matrix model.*

Proof. We use the basic adversary method to prove the lower bound. Let X be the set of n-vertex graphs with one cycle going through all the vertices and Y be the set of graphs formed by two cycles with each cycle has length between $n/3$ and $2n/3$. We can see that X is connected and Y is disconnected. Let $R \subseteq X \times Y$ be a relation such that for all $x \in X, y \in Y, xRy$ if there exists $a, b, c, d \in [n]$ such that ab, cd are edges in x but not in y and ac, bd are edges in y but not in x.

For any $x \in X$ and any edge ab in x, we have $l_{x,ab} = n/3$ because there are $n/3$ edges that we can choose as cd. If ab is not an edge in x then $l_{x,ab} \leq 2$. For any $y \in Y, l'_{y,ac} = \Theta(n)$ if ac is an edge and $l'_{y,ac} \leq 4$ otherwise. We have $m = O(n^2)$ because we can choose $n-1$ edges for the first edge and $n/3$ edges for the second edge. Also, $m' = O(n^2)$ because we have to choose one edge in each cycle, and cycle length is at least $n/3$. If $x_{ab} \neq y_{ab}$, then one of $l_{x,ab}, l'_{y,ab}$ is $O(1)$ and the other is $O(n)$, so $l_{max} = O(n)$. Thus, the query complexity is $\Omega\left(\sqrt{\frac{mm'}{l_{max}}}\right) = \Omega(n^{3/2})$. □

4.3 Global queries

The global query model is a more powerful model compared to the local query model. This model involves the relation between sets of vertices or edge slots. There is a rich literature on graph algorithms with global queries. For example, learning a graph via additive and cross queries has been studied in [1–7], with Choi [1] showing a $O(\frac{m \log n}{\log m})$ query adaptive randomized algorithm. Choi and Kim [4] proved the existence of a non-adaptive deterministic algorithm with the same number of queries. However, constructing an algorithm with this optimal query complexity is still an open problem.

There are also studies involving independent set queries (which is also referred to as *edge-detecting queries*) [7–9, 9, 10] with $O(1)$-round algorithm that asks $m \log n + \sqrt{m} \log^k n$ queries, and $O(\log n)$-round algorithm that asks $O(m \log n)$ queries [8]. No deterministic algorithm can solve this problem in a constant number of rounds. A stronger model, called bipartite independent set queries, has been used to solve the problem of counting edges of a graph [11], matching [12], and

triangle estimation [13]. Recently, Assadi, Chakrabarty, and Khanna [16] studied the connectivity problem via linear and OR queries.

There are many applications that motivate the study of global queries. For example, the cross query model is motivated by the problem of finding Fourier coefficients of bounded pseudo-boolean functions, which has applications to evolutionary computation and population genetics. Additive queries have applications in bioinformatics, while the study of independent set queries is motivated by genome sequencing. Matrix-vector multiplication queries have applications to streaming algorithms [17], and cut queries are motivated by connections to submodular function minimization. Additionally, bipartite independent set queries have been studied in connection with reductions between counting and decision problems [18].

4.3.1 Matrix-vector multiplication queries

Let $G = (V, E)$ be a graph with n vertices and m edges and A be its adjacency matrix. Suppose we want to query some set of vertices $S \subseteq [n]$. Let $\chi_S \in \{0,1\}^n$ be the characteristic vector of S where $\chi_S[i] = 1$ iff $i \in S$. The matrix-vector queries involve both A and χ_S. Some examples of matrix-vector queries are as follows:

1. *Matrix-vector multiplication* $\mathrm{mv} : 2^V \to \mathbb{Z}$ where $\mathrm{mv}(S) = A\chi_S$.
2. *Boolean multiplication queries* $b : 2^S \to \{0,1\}^n$ where $b(S) = A \vee \chi_S$.

4.3.2 Vector-matrix-vector queries

Let A be the adjacency matrix of $G = (V, E)$ with n vertices and m edges. We want to query two sets of vertices S, T to A. Suppose $\chi_S \in \{0,1\}^n$ is the characteristic vector of S and $\chi_T \in \{0,1\}^n$ is the characteristic vector of T. Vector-matrix-vector queries involve the two vectors χ_S, χ_T (χ_S and χ_T can be equal) and the adjacency matrix A where we multiply the vectors on the left and the right side of A. The following are examples of vector-matrix-vector queries:

1. *Matrix cut queries* $\mathrm{mc} : 2^V \times 2^V \to \mathbb{Z}$ where $\mathrm{mc}(S, T) = \chi_S^\intercal A \chi_T$. A matrix cut query will return the total weight of edges with one endpoint is in S and the other endpoint is in T.

2. *Additive queries* $\mathrm{add} : 2^V \to \mathbb{Z}$ where $\mathrm{add}(S) = \chi_S^\intercal A \chi_S$. The answer to an additive query $S \subseteq V$ is the total weight of edges (the number of edges if the graph is unweighted) with both endpoints in S times two.

3. *Cross queries* $\mathrm{cross} : 2^V \times 2^V \to \mathbb{Z}$ where for $S, T \subseteq V$ such that $S \cap T = \emptyset$, $\mathrm{cross}(S,T) = \chi_S^\intercal A \chi_T$. A cross query will return the total weight of edges between two independent sets of vertices.

4. *Cut queries* $\mathrm{cut} : 2^V \to \mathbb{Z}$ where $\mathrm{cut}(S) = (1 - \chi_S)^\intercal A \chi_S$. This can be interpreted as $\lambda_G(S)$.

5. *Independent set queries* $\mathrm{is} : 2^V \to \mathbb{Z}$ where $\mathrm{is}(S) = 1$ if $|\chi_S^\intercal A \chi_S| > 0$, and 0 otherwise. This model is also called *edge-detecting queries* in several literatures. The query will answer if a set of vertices induces an edge of the graph.

6. *Bipartite independent set queries* $\mathrm{bis} : 2^V \times 2^V \to \mathbb{Z}$ where for $S, T \subseteq V$ such that $S \cap T = \emptyset$, $\mathrm{bis}(S,T) = 1$ if $|\chi_S^\intercal A \chi_T| > 0$, and 0 otherwise. One BIS query will answer if there is an edge between two independent sets of vertices.

4.3.3 Other global queries

Some global queries do not involve the adjacency matrix A. For example *linear queries*. Let $G = (V, w)$ be a weighted graph with n vertices and m edges. Suppose we want to query a set of edge slots S. Let $\chi_S \in \{0,1\}^{\binom{n}{2}}$ be the characteristic vector of S. A single linear query χ_S will return the inner product $\langle w, \chi_S \rangle$. Another query model is *OR queries*. A single OR query χ_S returns 1 if $\langle w, \chi_S \rangle > 0$ and 0 otherwise.

4.4 Relationship between global query models

Relationship between global query models is illustrated in Fig. 4.2 and we prove some of them in Lemma 4.14 - 4.17.

Lemma 4.14. *Cross queries and cut queries are constant equivalents.*

Proof. To simulate a cut query $x^\intercal A(1-x)$ we only need one cross query $x^\intercal Ay$ by choosing $y = 1 - x$. For the reverse direction, let $z = x + y$. Since x, y are disjoint, we have $z \in \{0,1\}^n$ and $A(x+y) = Ax + Ay$. Then, to simulate one cross query $x^T Ay$, we need three cut queries, as we can see below.

$$\frac{1}{2}\left(x^\intercal A(1-x) + y^\intercal A(1-y) - z^\intercal A(1-z)\right) = \frac{1}{2}\left(x^\intercal Ay + y^\intercal Ax\right) \tag{4.5}$$

$$= x^\intercal Ay \tag{4.6}$$

□

Lemma 4.15 ([34], Lemma 23)**.** *Matrix cut queries and additive queries are constant equivalent.*

Proof. One additive query $x^\intercal Ax$ can be simulated by one matrix cut query $x^\intercal Ay$ by choosing $y = x$. We will prove that we can simulate one matrix cut query with five additive queries. Let $z = x \circ y, \bar{x} = x - z$, and $\bar{y} = y - z$. Then,

$$x^\intercal Ay = (\bar{x} + z)^\intercal A(\bar{y} + z) \tag{4.7}$$

$$= \bar{x}^\intercal A\bar{y} + \bar{x}^\intercal Az + z^\intercal A\bar{y} + z^\intercal Az \tag{4.8}$$

$$= \frac{1}{2}\left((\bar{x}+\bar{y})^\intercal A(\bar{x}+\bar{y}) - x^\intercal Ax - y^\intercal Ay\right) - \bar{x}^\intercal A\bar{x} - \bar{y}^\intercal A\bar{y} \tag{4.9}$$

We can get Eq. (4.8) from Eq. (4.7) because $\bar{x}, \bar{y}, z$ are pairwise disjoint. Eq. (4.9) is obtained from the fact that A is symmetric and $u^\intercal Av = \frac{1}{2}\left((u+v)^\intercal A(u+v) - u^\intercal Au - v^\intercal Av\right)$ for disjoint u, v. □

Lemma 4.16 ([34], Lemma 25)**.** *A cut query can be simulated by a constant number of additive queries.*

Proof. A cut query can be simulated by three additive queries.

$$(1-x)^\intercal Ax = \frac{1}{2}\left(1^\intercal A1 - x^\intercal Ax - (1-x)^\intercal(1-x)\right) \tag{4.10}$$

□

Lemma 4.17 ([34],Theorem 26)**.** *Let $G = (V, E)$ be an n-vertex graph with A as its adjacency matrix. We need at least $n/2$ cut queries to simulate an additive query.*

Proof. Let x be the characteristic vector of a subset $X \subseteq V$ and $\mathrm{add}(X) = x^\intercal Ax$. Let Sym_n^0 be a vector space of symmetric $n \times n$ matrices with zeros along the diagonal. Define $\mathrm{symvec} : \mathrm{Sym}_n^0 \to \mathbb{R}^{\binom{n}{2}}$ by $\mathrm{symvec}(C) = [C(2:n,1); C(3:n,2); \ldots; C[n,n-1]]^\intercal$. Let $B = J - I$ where J is the all-ones matrix and I is the identity matrix, and $b = \mathrm{symvec}(B)$. Then, we have $\mathrm{add}(V) = \mathrm{symvec}(A)^\intercal b$ and

$$(1-x)^\intercal Ax = \mathrm{symvec}(A)^\intercal \mathrm{symvec}(x(1-x)^\intercal + (1-x)x^\intercal) \tag{4.11}$$

Note that the rank of $(x(1-x)^\intercal + (1-x)x^\intercal)$ is 2.

Consider a deterministic cut query algorithm making $k < n/2$ queries. Let $m = n^{2k+1} \cdot k^{k/2}$ and G' be a graph whose adjacency matrix is $A' = m \cdot B$. Set $a = \mathrm{symvec}(A')$. Suppose the algorithm makes queries $X_1, \ldots, X_k \subseteq V$ in G' with characteristic vectors $x_1, \ldots, x_k$ respectively. Let $d_i = \mathrm{symvec}(x_i(1-x_i)^\intercal + (1-x_1)x_1^\intercal)$ for $i = 1, \ldots, k$ and D be an $\binom{n}{2}$ matrix whose i^{th}-column is d_i. Since $\mathrm{rk}(B) = n$ and $k < n/2$, there is no solution y to $Dy = b$, otherwise b can be expressed as linear combination of k matrices of rank 2.

Let $\hat{y} \in \mathbb{Z}^{\binom{n}{2}}$ be the vector given by Lemma 4.18 satisfying $\hat{y}^\intercal D = 0$ and $\hat{y}^\intercal b \neq 0$. By the choice of m and Lemma 4.18, $a + \hat{y}$ will be a non-negative integer vector. Let G'' be a graph with adjacency matrix A'' such that $\mathrm{symvec}(A''G'') = a_1 + \hat{y}$. Since $a_1^\intercal D = (a_1 + \hat{y})^\intercal D$, both G' and G'' give the same answers on all queries. But $a_1^\intercal b \neq (a_1 + \hat{y})^\intercal b$, which means the total sum of weights in G' and G'' are different. Thus, a contradiction and any deterministic algorithm must make at least $n/2$ cut queries. □

Lemma 4.18 (Fredholm alternative)**.** *Let $A \in \{0,1\}^{N \times k}$ have independent columns and let $b \in \{0,1\}^N$. Suppose $Ax = b$ has no solutions $x \in \mathbb{R}^k$. Then the integer vector $y = \det(A^\intercal A)(I - A(A^\intercal A)^{-1}A^\intercal)b \in \mathbb{Z}_N$ satisfies*

(1) $y^\intercal A = 0$.

(2) $y^\intercal b \neq 0$.

(3) $||y||_\infty \leq N^{k+1/2}k^{k/2}$.

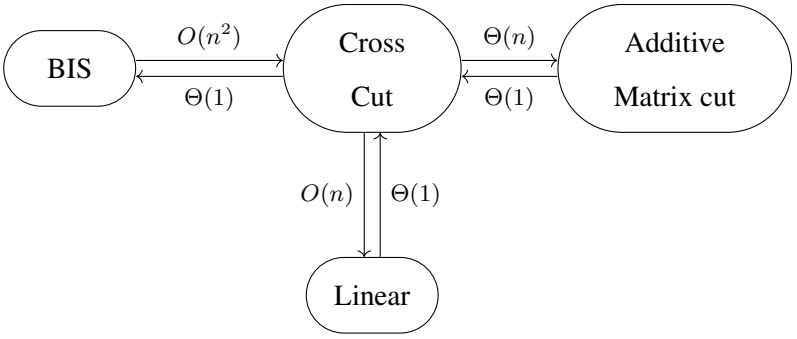


FIGURE 4.2: Relationship between global queries

Chapter 5

Global query lower bounds

This chapter provides a deterministic lower bound for graph connectivity problem with cut queries in Section 5.1. In Section 5.2, we adapt the technique from certificate complexity to prove a deterministic linear query lower bound.

5.1 Cut queries

Harvey [25] shows a deterministic cut query lower bound for the connectivity problem through communication complexity lower bound.

Theorem 5.1. *Let $G = (V, E)$ be a graph on n vertices. Any deterministic algorithm $\mathcal{A}$ requires $\Omega(n)$ cut queries to solve connectivity. Furthermore, any randomized algorithm solving connectivity requires $\Omega(n/\log(n))$ cut queries.*

Proof. Let G_A and G_B be graphs with vertex set V such that every edge $(u, v) \in E$ is exactly in one of G_A and G_B. Suppose Alice is given the subgraph $H_A = (V, E_A) \subseteq G_A$ and Bob is given the subgraph $H_B = (V, E_B) \subseteq G_B$. From Theorem 3.11, the deterministic communication complexity to decide the connectivity of graph $H = (V, E_A \cup E_B)$ is $\Omega(n \log n)$.

For $U \subseteq V$, let $\delta(U)$ denotes the set of edges with one endpoint in U and the other in $V \setminus U$ and $\mathrm{cut}(U)$ denotes the cut function of U. Then

$$\mathrm{cut}(U) = |\delta_H(U)| = |\delta_{H_A}(U)| + |\delta_{H_B}(U)| \tag{5.1}$$

We can transform the algorithm $\mathcal{A}$ with access to $\mathrm{cut}(U)$ into a communication protocol for connectivity. The protocol works as follows: Alice and Bob simulate the algorithm independently. Every time the algorithm makes query $\mathrm{cut}(U)$, Alice communicates $|\delta_{H_A}(U)|$ to Bob and Bob communicates $|\delta_{H_B}(U)|$ to Alice. Since $|E_A|, |E_B| \leq n^2$, the number of bits transmitted each time is less than $4 \log n$. Suppose the algorithm makes q queries in total. Hence, the number of bits in the communication protocol is $4q \log n$. The lower bound $D(\mathrm{CONN}) = \Omega(n \log n)$ implies $q = \Omega(n)$.

By the same argument, the number of cut queries for any randomized algorithm is at least $q = \Omega(n/\log(n))$, since $R(\mathrm{CONN}) = \Omega(n)$ by Theorem 3.15. □

5.2 Linear queries

This section is taken from the following arXiv paper [61]

In this section, we show that any deterministic, or even zero-error randomized, algorithm for connectivity must make $\Omega(n)$ linear queries. As far as we are aware, this is the first lower bound of any kind for connectivity in the unrestricted linear query model. The reason lower bounds against linear queries are difficult to show is because the answer to a query can have an unbounded number of bits. The "hard" inputs for a linear query lower bound cannot be limited to simple graphs, as given the promise that a graph is simple, it can be learned with a single linear query, by querying the vector $[1, 2, 4, \ldots, 2^{\binom{n}{2}-1}]^T$. Then, the graph can be obtained from the binary expansion of the answer.

Most global query lower bounds for connectivity are derived from communication complexity. It is known that the communication complexity of connectivity is $\Theta(n \log n)$ in the deterministic case [53] and $\Omega(n)$ in the bounded-error randomized case [26]. Moreover, the family of instances used to show these lower bounds are all simple graphs. If the answer to a global query on a simple graph

can be communicated with at most b bits, then these communication results imply $\Omega(n \log(n)/b)$ and $\Omega(n/b)$ deterministic and randomized lower bounds on the query complexity of connectivity in this model, respectively. In particular, this method gives an $\Omega(n)$ deterministic and $\Omega(n/\log n)$ randomized lower bound on the cut query complexity of connectivity. This approach, however, cannot show *any* lower bound against linear queries.

A lower bound technique that goes beyond considering simple graphs is the *cut dimension* [27]. This technique was originally developed by Graur et al. for showing cut query lower bounds against deterministic algorithms solving the minimum cut problem. In [28], it was observed that the lower bound also applies to the linear query model, and a strengthening of the technique was given called the ℓ_1-*approximate cut dimension*. Here we observe that the ℓ_1-approximate cut dimension characterizes, up to an additive $+1$, a well-known query complexity lower bound technique applied to the minimum cut problem, the *certificate complexity*. We further adapt this certificate complexity technique to the connectivity problem, allowing us to show an $\Omega(n)$ lower bound for deterministic linear query algorithms solving connectivity. By its nature, certificate complexity also lower bounds the query complexity of zero-error randomized algorithms. Thus we get a lower bound of $\Omega(n)$ for connectivity in this model as well.

We show that any zero-error randomized algorithm that correctly solves connectivity must make $\Omega(n)$ linear queries in expectation (Corollary 5.13). We do this by building on the ℓ_1-approximate cut dimension approach. The ℓ_1-approximate cut dimension was originally applied to show lower bounds on the deterministic linear query complexity of the minimum cut problem. We show that this method actually characterizes, up to an additive $+1$, the linear query *certificate complexity* of the minimum cut problem. Certificate complexity is a well-known lower bound technique for query complexity. The certificate complexity of a function f on input x is the minimum number of queries $q_1, \ldots, q_k$ needed such that any input y which agrees with x on these queries $q_1(x) = q_1(y), \ldots, q_k(x) = q_k(y)$ must also satisfy $f(x) = f(y)$. The maximum certificate complexity over all inputs x is a lower bound on the query complexity of deterministic and even zero-error randomized algorithms for f.

In our context, the linear query certificate complexity of *connectivity* on a graph $G = (V, w)$ is the minimum k for which there are queries $q_1, \ldots, q_k$ such that for any graph $G' = (V, w')$, if $\langle w|q_i\rangle = \langle w'|q_i\rangle$ for all $i = 1, \ldots, k$ then G' is connected iff G is. We adapt the ℓ_1-approximate

cut dimension approach to give a linear algebraic characterization of the linear query certificate complexity of connectivity. We then show that the linear query certificate complexity for connectivity of the simple n-vertex cycle is $\Omega(n)$, giving the $\Omega(n)$ linear query lower bound for zero-error algorithms solving connectivity.

We first define the linear query certificate complexity in Section 5.2.1 and show that it is equivalent to the ℓ_1-approximate cut dimension of [28]. Then in Section 5.2.1.1 we apply this method to show an $\Omega(n)$ lower bound on the linear query certificate complexity of connectivity.

5.2.1 Certificate complexity

A deterministic algorithm correctly solves the MINCUT problem if it outputs $\lambda(G)$ for every n-vertex input graph G. We let $D_{\text{lin}}(\text{MINCUT})$ denote the minimum, over all deterministic linear query algorithms $\mathcal{A}$ that correctly solve MINCUT, of the maximum over all n-vertex input graphs $G = (V, w)$ of the number of linear queries made by $\mathcal{A}$ on G. The deterministic linear query complexity of connectivity, $D_{\text{lin}}(\text{CONN})$, is defined analogously.

We will also consider zero-error randomized linear query algorithms. A zero-error randomized linear query algorithm for MINCUT is a probability distribution over deterministic linear query algorithms that correctly solve MINCUT. The cost of a zero-error randomized algorithm $\mathcal{A}$ on input $G = (V, w)$ is the expected number of queries made by $\mathcal{A}$. We let $R_{0,\text{lin}}(\text{MINCUT})$ denote the minimum over all zero-error randomized algorithms $\mathcal{A}$ for MINCUT of the maximum over all G of the cost of $\mathcal{A}$ on G. $R_{0,\text{lin}}(\text{CONN})$ is defined analogously. Clearly $R_{0,\text{lin}}(\text{MINCUT}) \leq D_{\text{lin}}(\text{MINCUT})$ and $R_{0,\text{lin}}(\text{CONN}) \leq D_{\text{lin}}(\text{CONN})$

We will investigate a lower bound technique called certificate complexity.

Definition 5.2 (Certificate complexity)**.** Let $f : \{0,1\}^n \to \{0,1\}$. A *certificate* for input x is a set $S \subseteq \{1, \ldots, n\}$ such that for all input y, if $y_i = x_i$ for all $i \in S$ then $f(x) = f(y)$. The *certificate complexity* of x for f is the minimum size of a certificate for x.

Now, we can define certificate complexity for the minimum-cut problem.

Definition 5.3. Let $G = (V, w)$ be an n-vertex graph. The minimum-cut linear-query certificate complexity of G, denoted $\text{mincut-cert}(G)$, is the minimum k such that there is a matrix $A \in \mathbb{R}^{k \times \binom{n}{2}}$ with the property that, for any graph $G' = (V, w')$, if $Aw = Aw'$ then $\lambda(G) = \lambda(G')$.

The connectivity linear-query certificate complexity, denoted $\text{con-cert}(G)$, is the minimum k such that there is a matrix $A \in \mathbb{R}^{k \times \binom{n}{2}}$ with the property that, for any graph $G' = (V, w')$, if $Aw = Aw'$ then $\lambda(G') > 0$ iff $\lambda(G) > 0$.

It is a standard fact that certificate complexity is a lower bound on zero-error randomized query complexity [62]. We reproduce the proof here for completeness.

Lemma 5.4. *For any n-vertex graph $G = (V, w)$*

$$R_{0,\text{lin}}(\text{MINCUT}) \geq \text{mincut-cert}(G)$$
$$R_{0,\text{lin}}(\text{CONN}) \geq \text{con-cert}(G) \ .$$

Proof. We treat the minimum cut case, the connectivity case follows similarly.

First consider a deterministic linear query algorithm for minimum cut on input G. This algorithm must make at least $\text{mincut-cert}(G)$ many queries. If not, there is a graph G' which agrees with G on all the queries but such that $\lambda(G') \neq \lambda(G)$. As the algorithm does not distinguish G from G' it cannot answer correctly.

Consider now a zero-error randomized linear query algorithm $\mathcal{A}$ for minimum cut. Each deterministic algorithm in the support of $\mathcal{A}$ correctly solves MINCUT and so must make at least $\text{mincut-cert}(G)$ many queries on input G. Thus expected number of queries made by $\mathcal{A}$ on input G is at least $\text{mincut-cert}(G)$ as well. □

Let $A \in \mathbb{R}^{k \times \binom{n}{2}}$ be a minimum cut certificate for G. Then for any $G' = (V, w')$ with $Aw = Aw'$ it must hold that $\lambda(G) = \lambda(G')$. The condition that $\lambda(G') \leq \lambda(G)$ can be certified with a single query. If S is the shore of a minimum cut in G then we can query $\chi_S \in \{0,1\}^{\binom{n}{2}}$, the characteristic vector of $\Delta_{K_n}(S)$ where K_n is the complete graph. Then $\langle \chi_S | w \rangle = \lambda(G)$, and including χ_S as a row of A guarantees that if $Aw = Aw'$ then $\lambda(G') \leq \lambda(G)$.

The more challenging problem is certifying that the minimum cut of any $G' = (V, w')$ with $Aw = Aw'$ is at least $\lambda(G)$. We single out a more general version of this lower bound certification problem, which will be useful to treat the minimum cut and connectivity cases together.

Definition 5.5. Let $G = (V, w)$ be a graph and let $0 \leq \tau \leq \lambda(G)$ be a parameter. The $\text{at-least-}\tau\text{-cert}$ of G is the least k such that there is a matrix $A \in \mathbb{R}^{k \times \binom{n}{2}}$ such that for any $G' = (V, w')$ with $Aw = Aw'$ it holds that $\lambda(G') \geq \tau$.

Lemma 5.6. *Let $G = (V, w)$ be a graph. Then*

$$\textit{at-least-}\lambda(G)\textit{-cert}(G) \leq \text{mincut-cert}(G) \leq \textit{at-least-}\lambda(G)\textit{-cert}(G) + 1 \ .$$

If G is connected then $\text{con-cert}(G) = \inf_{\tau > 0} \text{at-least-}\tau\text{-cert}(G)$.

Proof. If A is a mincut-cert for G then in particular it certifies that $\lambda(G') \geq \lambda(G)$ for any $G' = (V, w')$ with $Aw = Aw'$. This shows the first lower bound. For the other direction, suppose A is an at-least-$\lambda(G)$-cert for G. Let S be the shore of a minimum cut in G and $\chi_S \in \{0,1\}^{\binom{n}{2}}$, the characteristic vector of the cut corresponding to S in the complete graph. Then adjoining χ_S as an additional row to A creates a minimum cut certificate for G.

Now we show the second part of the lemma. Let G be a connected graph. For any $0 < \tau \leq \lambda(G)$, if A is an $\text{at-least-}\tau\text{-cert}$ certificate for G then it holds that for any $G' = (V, w')$ with $Aw = Aw'$ that $\lambda(G') \geq \tau > 0$ so G' is also connected. This shows that $\text{con-cert}(G) \leq \inf_{\tau > 0} \text{at-least-}\tau\text{-cert}(G)$.

For the other direction, let A be a connectivity certificate for G. We claim that A is also a $\text{at-least-}\tau\text{-cert}$ for some $\tau > 0$. For $\emptyset \neq S \subsetneq V$ let $\chi_S \in \{0,1\}^{\binom{n}{2}}$ be the characteristic vector of $\Delta_{K_n}(S)$, where K_n is the complete graph on n vertices. Define

$$\begin{aligned} \iota(S) = \underset{w'}{\text{minimize}} \quad & \left\langle \chi_S \middle| w' \right\rangle \\ \text{subject to} \quad & Aw = Aw' \\ & w' \geq 0 \ . \end{aligned}$$

As $\iota(S)$ is defined by a linear program, the minimum is achieved, and is strictly positive as A is a connectivity certificate. Thus letting $\tau = \min_{\emptyset \neq S \subsetneq V} \iota(S)$ we see that A is an at-least-τ-cert. This shows $\inf_{\tau>0} \text{at-least-}\tau\text{-cert}(G) \leq \text{con-cert}(G)$. □

We now proceed to give a linear-algebraic characterization of these certificate complexities. First a definition.

Definition 5.7 (universal cut-edge incidence matrix)**.** Let K_n be the complete graph on vertex set $\{1, \ldots, n\}$. The universal cut-edge incidence matrix M_n is a Boolean $(2^{n-1} - 1)$-by-$\binom{n}{2}$ matrix with rows indexed by non-empty sets S with $1 \notin S$ and columns indexed by edges e and $M_n(S, e) = 1$ iff $e \in \Delta_{K_n}(S)$.

Definition 5.8. Let $G = (V, w)$ be a graph on n vertices and M_n the universal cut-edge incidence matrix. For $\tau \in \mathbb{R}$ with $0 \leq \tau \leq \lambda(G)$ define

$$\begin{aligned} \tau\text{-cut-rank}(G) = \underset{X}{\text{minimize}} \quad & \text{rank}(X) \\ \text{subject to} \quad & X \leq M_n \\ & Xw \geq \tau \cdot \mathbf{1} \ . \end{aligned}$$

where $X \leq M_n$ denotes coordinate-wise inequality.

Lemma 5.9. *The $\lambda(G)$-cut-rank(G) is equivalent to the ℓ_1 approximate cut dimension defined in [28].*

Proof. We have defined the cut rank where X is a matrix with $\binom{n}{2}$ columns as this will be more convenient in the proof of Theorem 5.10. If G has m edges, one could instead restrict M_n and X to matrices with m columns, corresponding to the edges of G.

Indeed, let $\widehat{M_n}$ be the $(2^{n-1} - 1)$-by-m matrix that is M_n with columns restricted to edges of G and similarly $\hat{w} \in \mathbb{R}^m$ be w restricted to edges of G. One can see that the program

$$\begin{aligned} \underset{Y}{\text{minimize}} \quad & \text{rank}(Y) \\ \text{subject to} \quad & Y \leq \widehat{M_n} \\ & Y\hat{w} \geq \tau \cdot \mathbf{1} \end{aligned}$$

has value equal to τ-cut-rank(G). It is clear that τ-cut-rank(G) is at most the value of this program as any feasible Y can be turned into a feasible X by populating the additional columns with zeros.

For the other direction, let let X^* realize τ-cut-rank(G) and let Y^* be X^* with columns corresponding to non-edges of G deleted. Then $\mathrm{rk}(Y^*) \leq \mathrm{rk}(X^*)$, $Y^* \leq \widehat{M}_n$ and $Y^*\hat{w} \geq \tau$ because w is zero on all entries corresponding to the deleted columns. When we are actually proving lower bounds in Section 5.2.1.1, it will be more convenient to use this formulation where columns are restricted to edges of G. □

Theorem 5.10. *For any n-vertex graph $G = (V, w)$ and $0 \leq \tau \leq \lambda(G)$ we have* at-least-τ-cert$(G) =$ τ-cut-rank(G).

Proof. We first show at-least-τ-cert$(G) \geq \tau$-cut-rank(G). Suppose the at-least-τ-cert certificate complexity of G is k and let A be a k-by-$\binom{n}{2}$ matrix realizing this. Let $G' = (V, w')$ be such that $Aw = Aw'$. Let us write $w' = w - z \geq 0$ where $Az = 0$.

Consider a shore $\emptyset \neq S \subsetneq V$. Let $\chi_S \in \{0,1\}^{\binom{n}{2}}$ be the characteristic vector of $\Delta_{K_n}(S)$, where K_n is the complete graph. Thus $w(\Delta_G(S)) = \langle \chi_S | w \rangle$ and

$$w'(\Delta_{G'}(S)) = \langle \chi_S | w - z \rangle = w(\Delta_G(S)) - \langle \chi_S | z \rangle \quad .$$

As A is an at-least-τ-cert certificate, we must have $w(\Delta_G(S)) - \langle \chi_S | z \rangle \geq \tau$ which means $\langle \chi_S | z \rangle \leq w(\Delta_G(S)) - \tau$.

Consider the optimization problem

$$\begin{aligned} \alpha(S) = \text{maximize}_{z} \quad & \langle \chi_S | z \rangle \\ \text{subject to} \quad & Az = 0 \\ & w - z \geq 0 \ . \end{aligned}$$

As we have argued, $\alpha(S) \leq w(\Delta_G(S)) - \tau$. The dual of this problem is

$$\begin{aligned} \beta(S) = \text{minimize}_{v} \quad & \langle w | \chi_S - A^T v \rangle \\ \text{subject to} \quad & \chi_S - A^T v \geq 0 \ . \end{aligned}$$

The primal is feasible with $z = 0$ so we have strong duality and $\alpha(S) = \beta(S)$. This means there exists a vector X_S in the row space of A with $\chi_S \geq X_S$ and $\langle w|\chi_S - X_S\rangle \leq w(\Delta_G(S)) - \tau$, which implies $\langle X_S|w\rangle \geq \tau$.

As S was arbitrary, this holds for every shore. Package the vectors X_S as the rows of a matrix X. This matrix satisfies $X \leq M_n$, since $X_S \leq \chi_S$, and $Xw \geq \tau$. Further, $\mathrm{rk}(X) \leq k$ as every row of X is in the row space of A.

Now we show τ-cut-rank$(G) \geq$ at-least-τ-cert(G). Let X be a matrix realizing τ-cut-rank(G) and let $k = \mathrm{rk}(X)$. Let A be a matrix whose rows are a basis for the row space of X, and so $\mathrm{rk}(A) = k$. To show that A is an at-least-τ certificate for G it suffices to show that $\alpha(S) \leq w(\Delta_G(S)) - \tau$ for every shore S.

Let X_S be the row of X corresponding to shore S. We have that $X_S \leq \chi_S$ and that X_S is in the row space of A. Thus $\alpha(S) = \beta(S) \leq \langle w|\chi_S - X_S\rangle \leq w(\Delta_G(S)) - \tau$. This completes the proof. □

Corollary 5.11. *Let $G = (V, w)$ be a graph. Then*

$$\lambda(G)\text{-cut-rank}(G) \leq \text{mincut-cert}(G) \leq \lambda(G)\text{-cut-rank}(G) + 1 \ .$$

If G is connected then $\text{con-cert}(G) = \inf_{\tau>0} \tau\text{-cut-rank}(G)$.

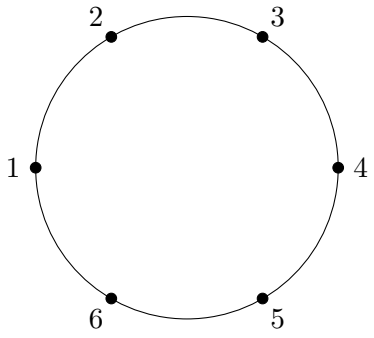
5.2.1.1 Application to connectivity

Theorem 5.12. *Let n be even and $C_n = ([n], w)$ be a cycle graph on n vertices where all edge weights are 1. Then* $\text{con-cert}(C_n) \geq n/4$.

Proof. We will show that $\inf_{\tau>0} \tau\text{-cut-rank}(C_n) \geq n/4$ which will give the theorem by Corollary 5.11.

Let w be the weight vector of C_n and let X be a $(2^{n-1} - 1)$-by-$\binom{n}{2}$ matrix satisfying $X \leq M_n$ and $Xw > 0$. We want to show that $\mathrm{rk}(X) \geq n/4$. Let Y be a $n/2$-by-n submatrix of X where the rows are restricted to the singleton shores $\{2\}, \{4\}, \ldots, \{n\}$ at even numbered vertices, and

the columns are restricted to the edges $e_{12}, e_{23}, \ldots, e_{n1}$ of C_n. To show a lower bound on the rank of X it suffices to show a lower bound on the rank of Y.



$$Y = \begin{array}{c} \\ \{2\} \\ \{4\} \\ \{6\} \end{array} \begin{array}{c} \begin{array}{cccccc} e_{12} & e_{23} & e_{34} & e_{45} & e_{56} & e_{61} \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array}$$

FIGURE 5.1: Example with C_6

The cuts $\Delta(\{2\}), \Delta(\{4\}), \ldots, \Delta(\{n\})$ partition the edges of C_n, therefore every pair of rows of Y are disjoint. Note that Xw only depends on the columns of X corresponding to edges of C_n. These are the columns we have restricted to in Y, therefore since $Xw > 0$, every row sum of Y must be positive. Also note that entries of Y in the row corresponding to shore $\{2i\}$ can only potentially be positive in the columns labeled by $e_{2i-1,2i}$ and $e_{2i,2i+1}$, where addition on the subscripts is taken modulo n.

We further modify Y to a square $n/2$-by-$n/2$ matrix Y' by summing together columns $2i-1$ and $2i$ for $i = 1, \ldots, n/2$. Note that $\mathrm{rank}(Y') \leq \mathrm{rank}(Y)$ and now Y' has strictly positive entries on the diagonal, and must be non-positive everywhere else. Next we multiply each row by the appropriate number so that the diagonal entry becomes 1. This preserves the property that in each row the sum of the positive entries (which is now 1) is strictly greater than the sum of the negative entries, and does not change the rank. Thus in each row the ℓ_1 norm of the off-diagonal entries is at most 1, and by the Gershgorin circle theorem all eigenvalues of Y' are at most 2. On the other hand, the trace of Y' is $n/2$, thus the rank must be at least $n/4$. □

Corollary 5.13. $R_{0,\mathrm{lin}}(\mathrm{CONN}) \geq n/4$.

Proof. Follows by Lemma 5.4 and Theorem 5.12. □

Chapter 6

Connectivity with Global Queries

This chapter is taken from the following arXiv paper: [61]

Abstract We study the query complexity of determining if a graph is connected with global queries. The first model we look at is matrix-vector multiplication queries to the adjacency matrix. Here, for an n-vertex graph with adjacency matrix A, one can query a vector $x \in \{0,1\}^n$ and receive the answer Ax. We give a randomized algorithm that can output a spanning forest of a weighted graph with constant probability after $O(\log^4(n))$ matrix-vector multiplication queries to the adjacency matrix. This complements a result of Sun et al. (ICALP 2019) that gives a randomized algorithm that can output a spanning forest of a graph after $O(\log^4(n))$ matrix-vector multiplication queries to the signed vertex-edge incidence matrix of the graph. As an application, we show that a quantum algorithm can output a spanning forest of an unweighted graph after $O(\log^5(n))$ cut queries, improving and simplifying a result of Lee, Santha, and Zhang (SODA 2021), which gave the bound $O(\log^8(n))$.

6.1 Introduction

Determining whether or not a graph is connected is a fundamental problem of computer science, which has been studied in many different computational models [63]. In this chapter, we study query algorithms for connectivity that make use of *global* queries. Traditionally, the most common

query models for studying graph problems have been *local* query models. It is known that the randomized query complexity of connectivity on a graph with n vertices and m edges is $\Theta(n^2)$ in the adjacency matrix model and $\Theta(m)$ in the adjacency array model [29, 54].

There has recently been a lot of interest in the complexity of graph problems with various global queries: matrix-vector multiplication queries to the adjacency or (signed) vertex-edge incidence matrix [15, 64], cut queries [14, 16, 34], and bipartite independent set (BIS) queries [11], to name a few. Motivation to study each of these models comes from different sources: matrix-vector multiplication queries have applications to streaming algorithms [17], cut queries are motivated by connections to submodular function minimization, and bipartite independent set queries have been studied in connection with reductions between counting and decision problems [18].

Let us take the example of matrix-vector multiplication queries to the adjacency matrix, which is the primary model we study. This is a very powerful model, as one learns an n-dimensional vector with a single query. On the other hand, this model can also lead to very efficient algorithms: we show that a randomized algorithm making $O(\log^4(n))$ many matrix-vector queries to the adjacency matrix of G can output a spanning forest of G with constant probability (see Corollary 6.9).

This result answers a natural question left open by Sun et al. [15]. They introduce the study of the complexity of graph problems with matrix-vector multiplication queries, and ask if the particular representation of a graph by a matrix makes a difference in the complexity of solving certain problems.

Besides the adjacency matrix, another natural representation of an n-vertex simple graph $G = (V, E)$ is the signed vertex-edge incidence matrix $A_{\pm} \in \{-1, 0, 1\}^{n \times \binom{n}{2}}$ (Definition 2.9). Sun et al. observe that a beautiful sketching algorithm of Ahn, Guha, and McGregor [17] to compute a spanning forest of a graph G via $O(n \log^3(n))$ non-adaptive linear measurements also gives a non-adaptive randomized query algorithm that finds a spanning forest after $O(\log^4(n))$ matrix-vector multiplication queries to $A_{\pm}$.

For a *bipartite* graph G with bipartition V_1, V_2, the *bipartite adjacency matrix* is the submatrix of the adjacency matrix where rows are restricted to V_1 and columns to V_2. Sun et al. show that

$\Omega(n/\log n)$ matrix-vector multiplication queries to the *bipartite* adjacency matrix (where multiplication of the vector is on the right) can be needed to determine if a bipartite graph is connected or not. However, the family of instances they give can be solved by a single query to the full adjacency matrix of the graph (or a single matrix-vector multiplication query *on the left* to the bipartite adjacency matrix). The arguably more natural question of comparing the complexity of matrix-vector multiplication queries in the signed vertex-edge incidence matrix versus the adjacency matrix model was left open. Our results mean that connectivity cannot show a large separation between these models.

While the matrix-vector multiplication query model is very powerful, as one can also hope for very efficient algorithms it still has interesting applications to weaker models, like cut queries. As an example, it is not hard to see that each entry of $A_{\pm}x$ can be computed with a constant number of cut queries, and therefore a matrix-vector multiplication query to $A_{\pm}$ can be simulated by $O(n)$ cut queries. Thus the randomized non-adaptive $O(\log^4(n))$ matrix-vector query algorithm for connectivity in this model implies a randomized non-adaptive $O(n\log^4(n))$ algorithm for connectivity in the cut query model, which is state-of-the-art for the non-adaptive cut query model [16]. For the adaptive case, recent work by [65] showed that the cut query complexity of graph connectivity is $O(n)$ for zero-error randomized algorithms. The deterministic query complexity for connectivity is $O(n\log n)$ as shown by Harvey [[25], Theorem 5.10].

We look at applications of matrix-vector multiplication algorithms to *quantum* algorithms using cut and BIS queries. Note that our algorithms are adaptive. Recent work of Lee, Santha, and Zhang showed that a quantum algorithm can output a spanning forest of a graph with high probability after $O(\log^8(n))$ cut queries [34]. This is in contrast to the randomized case where $\Omega(n/\log n)$ cut queries can be required to determine if a graph is connected [26]. A key observation made in [34] is that a quantum algorithm with cut queries can efficiently simulate a restricted version of a matrix-vector multiplication query to the adjacency matrix. Namely, if A is the adjacency matrix of an n-vertex simple graph, a quantum algorithm can compute $Ax \circ (1-x)$ for $x \in \{0,1\}^n$ with $O(\log n)$ cut queries (Corollary 6.11). Here $\circ$ denotes the Hadamard or entrywise product. Let $z = Ax \circ (1-x)$. Then we can compute Ax from $z \circ (1-x)$. In other words, the quantum algorithm can efficiently compute the entries of Ax where x_i is 0.

We quantitatively improve the Lee et al. result to show that a quantum algorithm can output a spanning tree of a simple graph with constant probability after $O(\log^5(n))$ cut queries (Theorem 6.12). We do this by showing that the aforementioned $O(\log^4(n))$ adjacency-matrix-vector multiplication query algorithm to compute a spanning forest also works with the more restrictive $Ax \circ (1-x)$ queries. In addition to quantitatively improving the result of [34], this gives a much shorter proof and nicely separates the algorithm into a quantum part, simulating $Ax \circ (1-x)$ queries, and a classical randomized algorithm using $Ax \circ (1-x)$ queries.

This modular reasoning allows us to extend the argument to other models as well. We also show that a quantum algorithm can compute a spanning forest with constant probability after $\tilde{O}(\sqrt{n})$ BIS queries. This is done by simulating an $Ax \circ (1-x)$ by a quantum algorithm making $O(\sqrt{n})$ BIS queries (Theorem 6.8).

6.2 Basic spanning forest algorithm

In this section, we give a template spanning forest algorithm based on Borůvka's algorithm [66]. In Borůvka's algorithm on input a graph $G = (V, w)$, one maintains the invariant of having a partition $\{S_1, \ldots, S_k\}$ of V and a spanning tree for each S_i. At the start of the algorithm this partition is simply V itself. In a generic round of the algorithm, the goal is to find one outgoing edge from each S_i which has one. One then selects a subset H of these edges which does not create a cycle among $S_1, \ldots, S_k$, and merges sets connected by edges from H while updating the spanning trees for each set accordingly. If q of the k sets have an outgoing edge, then the cycle free subset H will satisfy $|H| \geq q/2$. Every edge from H decreases the number of sets by at least 1, so the number of sets after this round is at most $k - q/2$. In this way we see that k minus the number of connected components of G at least halves with every round, and the algorithm terminates with a spanning forest of G after $O(\log n)$ rounds.

The main work of a round of this algorithm is the problem of finding an outgoing edge from each S_i. We abstract out performing this task by a primitive called $\mathrm{RecoverOneFromAll}$. Given oracle access to a non-negative matrix $A \in \mathbb{R}^{m \times n}$, a set of rows R, a set of columns S, and an error parameter δ, with probability at least $1 - \delta$ $\mathrm{RecoverOneFromAll}$ does the following: for every

$i \in R$ for which there is a $j \in S$ with $A(i,j) > 0$ it outputs a pair (i,j') with $A(i,j') > 0$. We record the input/output behavior of RecoverOneFromAll in this code header.

Algorithm 2 RecoverOneFromAll$[A](R, S, \delta)$

Input: Oracle access to a non-negative matrix $A \in \mathbb{R}^{m \times n}$, subsets $R \subseteq [m], S \subseteq [n]$, and an error parameter δ.

Output: With probability at least $1 - \delta$ output a set $\mathcal{T} \subseteq R \times S$ such that for every $i \in R$ for which $\exists j \in S$ with $A(i,j) > 0$ there is a pair $(i,t) \in \mathcal{T}$ with $A(i,t) > 0$.

Assadi, Chakrabarty, and Khanna [16] study a similar primitive for connectivity called *single element recovery*. In this problem one is given a non-zero non-negative vector $x \in \mathbb{R}^N$ and the goal is to find a coordinate j with $x_j > 0$. RecoverOneFromAll is exactly the matrix version of this problem, where one wants to solve the single element recovery problem on every row of a matrix. This version is more suitable for the matrix-vector multiplication query algorithms we study here where we want to implement a round of Borůvka's algorithm in a parallel fashion.

In the subsequent sections we will see how RecoverOneFromAll can be implemented in various global models. First we analyze how many calls to RecoverOneFromAll are needed to find a spanning forest.

Theorem 6.1. *Let $G = (V, w)$ be a weighted graph with n vertices, and A its adjacency matrix. There is a randomized algorithm that finds a spanning forest of G with probability at least $49/50$ after making $3(\log(n) + 10)$ calls to* RecoverOneFromAll$[A](R, S, \delta)$ *with error parameter $\delta = (300(\log(n) + 10))^{-1}$. Moreover, all calls to* RecoverOneFromAll *involve sets $R, S \subseteq V$ that are disjoint.*

Proof. The algorithm is given in Algorithm 4. We use a Union-Find data structure to maintain a partition of the vertices into sets which are connected. This data structure has the operations MakeSet to create a set of the partition, FindSet which takes as argument a vertex and returns a representative of the set containing the vertex, and Union which takes as arguments two vertices and merges their corresponding sets. We also suppose the data structure supports the operation Elts which, given a vertex $v \in V$, returns all the elements in the same set of the partition as v.

The algorithm makes one call to RecoverOneFromAll in each iteration of the for loop. There are $T = 3(\log(n) + 10)$ iterations of the for loop, giving the upper bound on the complexity. Let us

Algorithm 3 FindSpanningForest

Input: Subroutine RecoverOneFromAll$[A](R, S)$, where A is the adjacency matrix of a weighted graph $G = (V, w)$.

Output: Spanning forest F of G.

$\mathcal{B} \leftarrow \emptyset, F \leftarrow \emptyset, \delta \leftarrow (300(\log(n) + 10))^{-1}, T = 3(\log(n) + 10)$
for $v \in V$ **do**
  $\mathcal{B} \leftarrow \mathcal{B} \cup \mathrm{MakeSet}(v)$ ▷ Set up Union-Find data structure
end for
for $i = 1$ to T **do**
  $R \leftarrow \emptyset, S \leftarrow \emptyset$
  for $v \in \mathcal{B}$ **do**
    Flip a fair coin. If heads $R \leftarrow R \cup \mathrm{Elts}(v)$, else $S \leftarrow S \cup \mathrm{Elts}(v)$.
  end for
  $\mathrm{RoundEdges} \leftarrow \mathrm{RecoverOneFromAll}[A](R, S, \delta)$ ▷ Note $R \cap S = \emptyset$
  $\mathrm{EdgesToAdd} \leftarrow \emptyset$ ▷ EdgesToAdd will hold one outgoing edge from each set
  for $v \in \mathcal{B}$ **do**
    $\mathrm{Outgoing} \leftarrow \{e \in \mathrm{RoundEdges} : |e \cap \mathrm{Elts}(v)| = 1\}$
    if $\mathrm{Outgoing} \neq \emptyset$ **then**
      Add one edge from Outgoing to EdgesToAdd.
    end if
  end for
  $F \leftarrow F \cup \mathrm{EdgesToAdd}$
  for $\{u, v\} \in \mathrm{EdgesToAdd}$ **do**
    $\mathrm{Union}(u, v)$ ▷ Merge sets connected by an edge of EdgesToAdd
  end for
  $\mathrm{Temp} \leftarrow \emptyset$
  for $v \in \mathcal{B}$ **do**
    $\mathrm{Temp} \leftarrow \mathrm{Temp} \cup \mathrm{FindSet}(v)$ ▷ Create updated partition
  end for
  $\mathcal{B} \leftarrow \mathrm{Temp}$
end for
Return F

now show correctness. We will first show correctness assuming the procedure RecoverOneFromAll has no error.

In an arbitrary round of the algorithm we have a partition $B_1, \ldots, B_k$ of V and a set of edges F which consists of a spanning tree for each B_i. For $i = 1, \ldots, k$ we color B_i red or blue independently at random with equal probability. All vertices in B_i are given the color of B_i. Let R be the set of red vertices and S the set of blue vertices. We then call RecoverOneFromAll$[A](R, S, \delta)$. Note that $R \cap S = \emptyset$, showing the "moreover" part of the theorem. Assuming that RecoverOneFromAll returns without error, this gives us an edge in the red-blue cut for every red vertex which has such an edge. The set of all these edges is called RoundEdges. We then select from RoundEdges one

edge incident to each red B_i which has one, and let the set of these edges be $\mathrm{EdgesToAdd}$. This set of edges necessarily does not create a cycle among the sets $B_1, \ldots, B_k$ as we have chosen at most one edge from only the red B_i. We add $\mathrm{EdgesToAdd}$ to F and merge the sets connected by these edges giving a new partition $B'_1, \ldots, B'_{k'}$. Assuming that $\mathrm{RecoverOneFromAll}$ returns without error, so that all added edges are valid edges, and as we have added a cycle free set of edges, we maintain the invariant that F contains a spanning tree for each set of the partition.

To show correctness it remains to show that with constant probability at the end of the algorithm the partition $B_1, \ldots, B_k$ consists of connected components. The key to this is to analyze how the number of sets in the partition decreases with each round. Let Q_i be a random variable denoting the number of sets of the partition that have an outgoing edge at the start of round i. Thus, for example Q_1 is equal to n minus the number of isolated vertices with probability 1. In particular $\mathbb{E}[Q_1] \leq n$.

Consider a generic round i and say that at the start of this round q_i many sets of the partition have an outgoing edge. In expectation $q_i/2$ of the sets with an outgoing edge are colored red, and each outgoing edge has probability $1/2$ of being in the red-blue cut. Thus the expected size of $\mathrm{EdgesToAdd}$ is at least $q_i/4$. As each edge in $\mathrm{EdgesToAdd}$ reduces the number of sets in the partition of the next round by at least one this means $\mathbb{E}[Q_{i+1} \mid Q_i = q_i] \leq 3q_i/4$. Therefore we have

$$\begin{aligned}\mathbb{E}[Q_{i+1}] = \sum_{q_i} \mathbb{E}[Q_{i+1} \mid Q_i = q_i] \Pr[Q_i = q_i] &\leq \frac{3}{4} \sum_{q_i} q_i \Pr[Q_i = q_i] \\ &= \frac{3}{4}\mathbb{E}[Q_i] \ .\end{aligned}$$

From this it follows by induction that $\mathbb{E}[Q_i] \leq (3/4)^{i-1}n$. Taking $T = 3(\log(n) + 10)$ we have $\mathbb{E}[Q_T] \leq 1/100$, and therefore by Markov's inequality the algorithm will terminate with all sets being connected components except with probability at most $1/100$.

Finally, let us remove the assumption that $\mathrm{RecoverOneFromAll}$ returns without error. We have set the error parameter in $\mathrm{RecoverOneFromAll}$ to be $\delta = (300(\log(n)+10))^{-1}$. As $\mathrm{RecoverOneFromAll}$ is called T times, by a union bound the probability it ever makes an error is at most $T\delta \leq 1/100$. Therefore adding this to our error bound, the algorithm will be correct with probability at least $49/50$. □

6.3 Master Model

Given Algorithm 4, the task now becomes to implement $\mathrm{RecoverOneFromAll}[A](R, S, \delta)$. There is a natural approach to do this following the template of ℓ_0 samplers [67]. Let $B = A(R, S)$ be the submatrix of interest, and assume for the moment that $A \in \{0,1\}^{n \times n}$ is Boolean.

1. Estimate the number of ones in each row of B.
2. Bucket together the rows whose estimate is in the range $(2^{i-1}, 2^i]$ into a set G_i.
3. For each G_i, randomly sample $\Theta(n \log(n)/2^i)$ columns of B. With high probability every row of G_i will have at least one and at most $c \cdot \log(n)$ ones in this sample, for a constant c.
4. For each i and row in G_i learn the $O(\log(n))$ ones in the sampled set.

We now want to find a "master query model" that can efficiently implement this template, yet is sufficiently weak that simulations of this model by other global query models give non-trivial results. To do this we take inspiration from combinatorial group testing. In combinatorial group testing one is given *OR query* access to a string $x \in \{0,1\}^n$. This means that one can query any subset $S \subseteq [n]$ and receive the answer $\vee_{i \in S} x_i$. Both the problems of estimating the number of ones in x we need for step 1 and learning a sparse string x we need for step 4 have been extensively studied in the group testing setting. A natural kind of query to implement group testing algorithms in the matrix setting is a Boolean matrix-vector multiplication query, that is for $A \in \{0,1\}^{n \times n}$ one can query $x \in \{0,1\}^n$ and receive the answer $y = A \vee x \in \{0,1\}^n$ where $y_i = \vee_j (A(i,j) \wedge x_j)$. With this kind of query one can implement a *non-adaptive* group testing algorithm in parallel on all rows of A.

We will define the master model to be a weakening of the Boolean matrix-vector multiplication query, in order to accomodate quantum cut queries later on. In the master model one can query $x \in \{0,1\}^n$ and receive the answer $(A \vee x) \circ (1 - x)$. Here $\circ$ denotes the Hadamard or entrywise product of vectors. In other words, in the master model one only learns the entries of $A \vee x$ in those coordinates where x is zero. This restriction allows one to efficiently simulate these queries by a quantum algorithm making cut queries.

If A is non-Boolean then in the master model one can again query any $x \in \{0,1\}^n$ and receive the answer $[Ax]_{>0} \circ (1-x)$, where $[y]_{>0} \in \{0,1\}^n$ is a Boolean vector whose i^{th} entry is 1 if $y_i > 0$ and 0 otherwise. The following proposition shows that it suffices to restrict our attention to Boolean matrices.

Proposition 6.2. *Let $A \in \mathbb{R}^{n\times n}$ be a non-negative matrix and $x \in \{0,1\}^n$. Then $[Ax]_{>0} = [A]_{>0} \vee x$.*

Algorithm 4 Implementation of RecoverOneFromAll in the master model.

Input: $(A \vee x) \circ (1-x)$ query access to a Boolean matrix $A \in \{0,1\}^{n\times n}$, disjoint subsets $R, S \subseteq [n]$, and an error parameter δ

Output: A set $\mathcal{T} \subseteq R \times S$ such that for every $i \in R$ if there is a $j \in S$ with $A(i,j) = 1$ then there is a pair $(i,t) \in \mathcal{T}$ with $A(i,t) = 1$, except with error probability at most δ.

1: Estimate $|A(i,S)|$ for all $i \in R$ using Theorem 6.6.
2: **for** $i = 1, \ldots, \lceil \log n \rceil$ **do**
3: Let G_i be the set of rows of R whose estimate is in the range $(2^{i-1}, 2^i]$
4: Randomly sample $\min\{n, \lceil 32|S| \ln(n)/2^i \rceil\}$ elements from S with replacement, and let the selected set be H_i
5: Learn the submatrix $A(G_i, H_i)$
6: **end for**

The main result of this section is the following.

Lemma 6.3. *Let $A \in \mathbb{R}^{n\times n}$ be a non-negative matrix.* RecoverOneFromAll$[A](R, S, 1/n^2)$ *on disjoint subsets $R, S \subseteq [n]$ can be implemented with $O(\log(n)^3)$ many $[Av]_{>0} \circ (1-v)$ queries.*

The procedure RecoverOneFromAll is implemented using $(A \vee x) \circ (1-x)$ queries. Suppose $S = i : x_i > 0$. Then the query $(A \vee x) \circ (1-x)$ can only access the sub-matrix of A where the columns are in S and the rows are in $R = \{j : x_j = 0\}$. In other words, S and R are disjoint.

At a high level, Lemma 6.3 will follow the four steps given at the beginning of the section. Step 2 does not require any queries. Step 3 follows easily by a Chernoff bound, as encapsulated in the following lemma.

Lemma 6.4 ([34, Lemma 8])**.** *Let $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^\ell$ be such that $\frac{t}{8} \leq |x^{(i)}| \leq 2t$ for all $i \in \{1, \ldots, k\}$. For $\delta > 0$, sample with replacement $\frac{8\ell \ln(k/\delta)}{t}$ elements of $\{1, \ldots, n\}$, and call the resulting set R. Then*

- $\Pr_R[\exists i \in \{1, \ldots, k\} : |x^i(R)| = 0] \leq \delta$

- $\Pr_R[\exists i \in \{1,\ldots,k\} : |x^i(R)| > 64 \ln(k/\delta)] \leq \delta$

The interesting part are steps 1 and 4, both of which will be done using non-adaptive group testing algorithms. Towards step 1 we make the following definition.

Definition 6.5 (Good estimate)**.** We say that $b \in \mathbb{R}^k$ is a good estimate of $c \in \mathbb{R}^k$ iff $b(i) \leq c(i) \leq 2b(i)$ for all $i \in \{1,\ldots,k\}$.

The next theorem gives a randomized non-adaptive group testing algorithm for estimating the number of ones in a string x.

Theorem 6.6 ([68, Theorem 4])**.** *Let n be a positive integer and $0 < \delta < 1$ an error parameter. There is a non-adaptive randomized algorithm that on input $\alpha \in \{0,1\}^n$ outputs a good estimate of $|\alpha|$ with probability at least $1-\delta$ after $O(\log(1/\delta)\log(n))$ many OR queries to α.*

The problem of learning a string α by means of OR queries is the central problem of combinatorial group testing and has been extensively studied [69]. It is known that non-adaptive *deterministic* group testing algorithms must make $\Omega\left(\frac{d^2\log(n)}{\log(d)}\right)$ queries in order to learn a string with at most d ones [70, 71]. However, there are non-adaptive *randomized* group testing algorithms that learn any $\alpha \in \{0,1\}^n$ with $|\alpha| \leq d$ with probability at least $1-\delta$ after $O(d(\log(n)+\log(1/\delta)))$ queries. This is what we will use.

Theorem 6.7 ([72, Theorem 2])**.** *Let d, n be positive integers with $d \leq n$, and $0 < \delta < 1$. There is a distribution $\mathcal{D}$ over n-by-k Boolean matrices for $k = O(d(\log(n)+\log(1/\delta)))$ such that for any $x \in \{0,1\}^n$ with $|x| \leq d$, the string x can be recovered from $[x^T R]_{>0}$ with probability at least $1-\delta$ when R is chosen according to $\mathcal{D}$.*

We are now ready to give the proof of Lemma 6.3

Proof of Lemma 6.3. By Proposition 6.2 we may assume that A is Boolean and we have $(A \vee x) \circ (1-x)$ query access to A. Let $k = |R|, \ell = |S|$ and $B = A(R,S)$ be the submatrix of interest. Let $b \in \mathbb{N}^k$ where $b(i)$ is the number of ones in the i^{th} row of B. By Theorem 6.6 applied with $\delta = 1/n^4$ and a union bound, there is a distribution $\mathcal{R}_{count}$ over Boolean ℓ-by-t matrices with $t = O(\log(n)^2)$ such that from $[BK]_{>0}$ we can recover a good estimate of b with probability at

least $1 - 1/n^3$ when K is drawn from $\mathcal{R}_{count}$. We add $1/n^3$ to our error bound and continue the proof assuming that we have a good estimate of b. As R and S are disjoint we can simulate the computation of $[BK]_{>0}$ with $O(\log(n)^2)$ many $(A \vee x) \circ (1 - x)$ queries.

Next, for $i = 1, \ldots, \lceil \log n \rceil$ we bucket together rows of B where the estimated number of non-zero entries is in the interval $(2^{i-1}, 2^i]$ into a set G_i. As the estimate is good, for every $j \in G_i$ the number of non-zero entries in the j^{th} row of B is actually in the interval $(2^{i-3}, 2^{i+1}]$. For each $i = 1, \ldots, \lceil \log n \rceil$ randomly sample $\frac{32\ell \ln(n)}{2^i}$ elements of S with replacement and let H_i be the resulting set. By Lemma 6.4, with probability at least $1 - 1/n^3$ there is at least 1 and at most $O(\log(n))$ ones in each row of the submatrix $A(G_i, H_i)$. We add $\log(n)/n^3$ to our error total and assume this is the case for all $i = 1, \ldots, \lceil \log n \rceil$.

For each $i = 1, \ldots, \lceil \log n \rceil$ we next learn the non-zero entries of $A(G_i, H_i)$ via Theorem 6.7. This theorem states that for $t = O(\log(n)^2)$ there is a family of $|H_i|$-by-t matrices $\mathcal{D}$ such that from $[A(G_i, H_i)D]_{>0}$ we can learn the positions of all the ones of $A(G_i, H_i)$ with probability at least $1 - \frac{1}{n^3}$, when D is chosen from $\mathcal{D}$. Again by a union bound this computation will be successful for all i except with probability $\log(n)/n^3$. As R and S are disjoint we can simulate the computation of $[A(G_i, H_i)D]_{>0}$ with $O(\log^2 n)$ many $(A \vee x) \circ (1 - x)$ queries. Over all i, the total number of queries is $O(\log(n)^3)$ and the error is at most $3\log(n)/n^3 = O(1/n^2)$. □

Theorem 6.8. *Let $G = (V, w)$ be an n-vertex weighted graph and A its adjacency matrix. There is a randomized algorithm that outputs a spanning forest of G with probability at least $4/5$ after $O(\log(n)^4)$ many $[Av]_{>0} \circ (1 - v)$ queries.*

Proof. This follows from Theorem 6.1 and Lemma 6.3. □

6.4 Applications

In this section we look at consequences of Theorem 6.8 to algorithms with matrix-vector multiplication queries to the adjacency matrix, quantum algorithms with cut queries, and quantum algorithms with bipartite independent set queries.

6.4.1 Matrix-vector multiplication queries

Let $G = (V, E)$ be a simple n-vertex graph. There are several different ways one can represent G by a matrix, for example by its adjacency matrix $A \in \{0,1\}^{n\times n}$, or its *signed vertex-edge incidence matrix* $A_\pm \in \{-1,0,1\}^{n\times\binom{n}{2}}$. Sun et al. study the complexity of graph problems with matrix-vector multiplication queries, and ask the question if some matrix representations allow for much more efficient algorithms than others [15, Question 4]. They point out that the sketching algorithm for connectivity of Ahn, Guha, and McGregor [17] can be phrased as a matrix-vector multiplication query algorithm to the signed vertex-edge incidence matrix $A_\pm$. Specifically, the AGM algorithm gives a randomized *non-adaptive* algorithm that can output a spanning forest of an n-vertex graph after $O(\log^4(n))$ matrix-vector multiplication queries to $A_\pm$, where the queries are Boolean vectors.[1]

In contrast, Sun et al. show that there is a bipartite graph G such that when one is given matrix-vector multiplication query access *on the right* to the *bipartite* adjacency matrix of G, $\Omega(n/\log(n))$ matrix-vector multiplication queries are needed to determine if G is connected. However, the instances they give can be solved by a *single* matrix-vector multiplication query to the full adjacency matrix of G (or a single matrix-vector multiplication query on the left to the bipartite adjacency matrix).

This leaves open the natural question if connectivity can be used to separate the matrix-vector multiplication query complexity of the adjacency matrix versus the signed vertex-edge incidence matrix representation. As one can clearly simulate a query $[Ax]_{>0} \circ (1-x)$ in the master model by a matrix-vector multiplication query, our results show that connecitivity does not separate these models with the current state-of-the-art.

Corollary 6.9. *Let $G = (V, w)$ be an n-vertex weighted graph. There is an randomized algorithm that outputs a spanning forest of G with probability at least $49/50$ after $O(\log(n)^4)$ matrix-vector multiplication queries to the adjacency matrix of G.*

The $O(\log(n)^4)$ complexity matches the upper bound in the signed vertex-edge incidence matrix representation, however, the latter algorithm is non-adaptive while the adjacency matrix algorithm

[1] The literal translation of the AGM algorithm uses $O(\log^3(n))$ many queries by vectors whose entries have $O(\log n)$ bits. When translated to queries by Boolean vectors this results in $O(\log^4(n))$ queries.

is not. It is still possible that connectivity provides a large separation between the non-adaptive adjacency matrix-vector multiplication and non-adaptive signed vertex-edge incidence matrix-vector multiplication models.

6.4.2 Quantum cut queries

Let $G = (V, w)$ be an n-vertex weighted graph. In this section, we will assume that the edge weights are natural numbers in $\{0, \ldots, M-1\}$. Let $A \in \{0, \ldots, M-1\}^{n \times n}$ be the adjacency matrix of G. In a cut query, one can ask any $z \in \{0,1\}^n$ and receive the answer $(1-z)^T A z$. A very similar query is a *cross query*. In a cross query one can query any $y, z \in \{0,1\}^n$ that are *disjoint* (i.e. $y \circ z = 0$) and receive the answer $y^T A z$. It is clear that a cross query can simulate a cut query. One can also simulate a cross query with 3 cut queries because for disjoint y, z

$$y^T A z = \frac{1}{2}\left((1-y)^T A y + (1-z)^T A z - (1-(y+z))^T A (y+z)\right) \quad .$$

Because of this constant factor equivalence we will make use of cross queries when it is more convenient.

Lee, Santha, and Zhang [34] give a quantum algorithm to find a spanning forest of a simple n-vertex graph after $O(\log^8(n))$ cut queries. A key observation they make is that a quantum algorithm can efficiently simulate a restricted version of a matrix-vector multiplication query to the adjacency matrix. This follows from the following lemma, which is an adaptation of the Bernstein-Vazirani algorithm [48].

Lemma 6.10 ([34, Lemma 9])**.** *Let $x \in \{0, 1, \ldots, K-1\}^n$ and suppose we have access to an oracle that returns $\sum_{i \in S} x_i \mod K$ for any $S \subseteq [n]$. Then there exists a quantum algorithm that learns x with $O(\lceil \log(K) \rceil)$ queries without any error.*

Corollary 6.11 (cf. [34, Lemma 11])**.** *Let $G = (V, w)$ be an n-vertex weighted graph with integer weights in $\{0, 1, \ldots, M-1\}$, and let A be its adjacency matrix. There is a quantum algorithm that for any $z \in \{0,1\}^n$ perfectly computes $x = Az \circ (1-z)$ with $O(\log(Mn))$ cut queries to A.*

Proof. Let $x = Az \circ (1-z)$. The entries of x are in $\{0, \ldots, n(M-1)\}$, thus we can work modulo $K = 2Mn$ and preserve all entries of x. Moreover, x is zero wherever z is one. Thus for any

$y \in \{0,1\}^n$ we have $y^T x = (y \circ (1-z))^T x = (y \circ (1-z))^T Az$, which can be computed with one cross query. Thus we can apply Lemma 6.10 to learn x perfectly with $O(\log(Mn))$ many cut queries to A. □

Using Corollary 6.11, classical algorithms using $Az \circ (1-z)$ queries give rise to quantum algorithm using cut queries. Thus we can apply Theorem 6.8 to obtain the following quantitative improvement of the result of [34].

Theorem 6.12. *Given cut query access to an n-vertex graph G with integer weights in $\{0,1,\ldots,M-1\}$ there is a quantum algorithm making $O(\log^4(n)\log(Mn))$ queries that outputs a spanning forest of G with at least probability $49/50$.*

Proof. The proof follows from Corollary 6.11 and Theorem 6.8. □

6.4.3 Quantum bipartite independent set queries

Let $G = (V, E)$ be a simple graph and A its adjacency matrix. In a bipartite independent set query, one can query any *disjoint* $y, z \in \{0,1\}^n$ and receive the answer $[y^T Az]_{>0} \in \{0,1\}$. A query in the master model can be simulated by a quantum algorithm making $\tilde{O}(\sqrt{n})$ bipartite independent set queries. The key to this result is Belovs' theorem (Theorem 2.24) about combinatorial group testing with quantum algorithms.

Corollary 6.13. *Let $G = (V, E)$ be an n-vertex simple graph, and let A be its adjacency matrix. There is a quantum algorithm that for any $z \in \{0,1\}^n$ computes $Az \circ (1-z)$ with probability at least $1 - 1/n^3$ after $O(\sqrt{n}\log(n))$ bipartite independent set queries to G.*

Proof. Let $x = Az \circ (1-z)$. For any $y \in \{0,1\}^n$ we have $y^T x = (y \circ (1-z))^T x = (y \circ (1-z))Az$, thus $[y^T x]_{>0}$ can be computed with a single bipartite independent set query to G as $y \circ (1-z)$ and z are disjoint. Therefore by Theorem 2.24 and error reduction, with $O(\sqrt{n}\log(n))$ bipartite independent set queries we can compute x with probability at least $1 - 1/n^3$. □

Theorem 6.14. *Let $G = (V, E)$ be an n-vertex simple graph. There is a quantum algorithm which outputs a spanning forest of G with probability at least $4/5$ after $O(\sqrt{n}\log(n)^5)$ bipartite independent set queries to G.*

Proof. Let A be the adjacency matrix of G. By Theorem 6.8 there is an algorithm that outputs a spanning forest of G with probability at least $49/50$ after $O(\log(n)^4)$ many $[Av]_{>0} \circ (1-v)$ queries. We can implement an $[Av]_{>0} \circ (1-v)$ with error at most $1/n^3$ by a quantum algorithm making $O(\sqrt{n}\log(n))$ bipartite independent set queries by Corollary 6.13. The probability any of the $O(\log(n)^4)$ many $[Av]_{>0} \circ (1-v)$ queries is computed incorrectly is at most $O(\log(n)^4/n^3)$. Thus there is a quantum algorithm with success probability at least $4/5$ that outputs a spanning forest of G after $O(\sqrt{n}\log(n)^5)$ bipartite independent set queries. □

Chapter 7

Sparsifier

In this chapter, we discuss sparsifiers, which are reweighted subgraphs that approximately preserve some properties of the original graph. Initially, works in this area focused on sparsifiers that preserve the cut values of the original graph, which is called a cut sparsifier [73]. Benczur and Karger showed that randomly sampling $O(n \log n)$ edges and reweighting them appropriately preserves the cut values up to some approximation factor. Solving problems involving cuts in the resulting sparsified graph gives better running time as the graph has fewer edges. The first idea to achieve this (which does not achieve the desired number of edges) is to sample each edge in the graph with probability inversely proportional to the size of a minimum cut in the graph. Let G be a weighted undirected graph with minimum cut value c. If we sample each edge with probability $\tilde{\Omega}(1/c)$, we get a new graph with an expected number of edges $\tilde{O}(m/c)$ with cut values close to their expected values. To reduce the number of sampled edges, Karger and Benczur extend this idea by sampling edges with different probabilities, namely with a probability that is inversely proportional to the strength of an edge, which is the maximum k such that a component with a minimum cut of at least k contains both endpoints of this edge. Suppose edge e in graph G has strength k_e. We sample each edge with probability $\tilde{\Omega}(1/k_e)$ and reweight the edge to $w(e)/k_e$ if sampled. This will give us a graph with $O(n \log n/\varepsilon^2)$ edges, where each cut value approximates the original value within a $(1 \pm \varepsilon)$ factor.

Spielman and Teng [74] strengthened the notion of cut sparsifiers and introduced *spectral sparsifiers*, which sparsify the graph and preserve the spectral structure or quadratic form defined by the

Laplacian of the graph. They proved that a $(1 \pm \varepsilon)$-spectral approximation with $\tilde{O}(n/\varepsilon^2)$ edges can be constructed in $\tilde{O}(m)$ time.

One of the existing cut-sparsification algorithms developed by Rubinstein, Schramm, and Weinberg (RSW) [14] relies on a sampling procedure by Benczur and Karger [43]. However, they did not prove that their sampling procedure gives the same result as the one from Benczur and Kager. Instead of sampling every edge from the original graph, RSW algorithm only subsamples some edges. We show in Theorem 7.10 that their method still gives the same cut sparsifier as in Benczur and Karger's algorithm.

We begin this chapter by a brief overview of spectral sparsifiers in Section 2.2. Then, we discuss various works on cut sparsifiers in Section 7.2. In Section 7.2.1, we discuss the cut sparsifier algorithm by Rubinstein, Schramm, and Weinberg and fix the gap in their algorithm. Additionally, we present another contribution in this thesis which is to show the query complexity of the quantum cut sparsifier for unweighted graphs in the adjacency list model in Theorem 7.17

7.1 Spectral sparsifier

Definition 7.1 (Laplacian)**.** Let $G = (V, w)$ be a connected weighted undirected graph with edge set E. We define the *Laplacian* L of G as $L = B^T W B$ where W is an $m \times m$ diagonal matrix such that $W(e, e) = w(e)$ and B is the signed edge-vertex incidence matrix given by

$$B(\{v, w\}, u) = \begin{cases} 0 & \text{if } \{v, w\} \notin E \text{ or } u \notin \{v, w\} \\ 1 & \text{if } \{v, w\} \in E \text{ and } u \text{ is the smallest element in } \{v, w\} \\ -1 & \text{if } \{v, w\} \in E \text{ and } u \text{ is the largest element in } \{v, w\} \end{cases}. \tag{7.1}$$

L is positive semi-definite since for all $x \in \mathbb{R}^n$,

$$\begin{aligned} x^T L x &= x^T B^T W B x \\ &= ||W^{1/2} B x||_2^2 \\ &= \sum_{\{u,v\} \in E} w(\{u, v\})(x_u - x_v)^2 \geq 0 \end{aligned}$$

Definition 7.2. A subgraph H of G is called an *ε-spectral sparsifier* of G if

$$(1-\varepsilon)L_G \preccurlyeq L_H \preccurlyeq (1+\varepsilon)L_G \tag{7.2}$$

where L_H and L_G are the Laplacian of H and G, respectively.

Spielman and Teng [74] prove that a $(1 \pm \varepsilon)$-spectral approximation with $\tilde{O}(n/\varepsilon^2)$ edges can be constructed in $\tilde{O}(m)$ time.

7.2 Cut sparsifier

The cut sparsifier is the main focus of this thesis. We use a cut sparsifier to solve the s-t minimum cut problem with quantum algorithm in Chapter 8.

Definition 7.3. A subgraph H of G is called a *cut-sparsifier* if for all $\emptyset \neq S \subsetneq V$,

$$(1-\varepsilon)w_G(\Delta_G(S)) \leq w_H(\Delta_H(S)) \leq (1+\varepsilon)w_G(\Delta_G(S)) \tag{7.3}$$

The value of any cut can be expressed by evaluating the quadratic forms in the Laplacian at $\{0,1\}$-vectors. Let $S \subseteq V$ and $x(u) = 1$ if $u \in S$ and $x(u) = 0$ otherwise. Then, $(x(u) - x(v))^2 w(\{u,v\}) = w(\{u,v\})$ iff one of u, v is in S and the other is in $V \setminus S$. The quadratic form of the Laplacian $\frac{1}{2}x^T Lx = \frac{1}{2}\sum_{\{u,v\}\in E} x(u) - x(v))^2 w(\{u,v\}) = w(\Delta_G(S))$. Therefore, spectral sparsifiers are also cut sparsifiers.

Karger [75] shows that if we sample edges with probability $p = \Omega(\log(n)/c)$ from unweighted graphs of n vertices and m edges with minimum cut c, then cut values can be computed in the sparser sample because with high probability all cuts have values close to their expectations. The sparsified graph will have expected number of edges $\tilde{O}(m/c)$.

Lemma 7.4 ([73])**.** *Let $G = (V, E)$ be a multigraph with minimum cut c, and let $G' = (V, E')$ be the result of sampling each edge of E with probability $p \geq \min\left(\frac{40 \ln n}{\varepsilon^2 c}, 1\right)$. Then with high probability, every cut of value k in G has value $(1 \pm \varepsilon)pk$ in G'.*

This approach, however, does not give much advantage when the graph has a large number of edges and small minimum cut value, since we only reduce the number of edges to $\tilde{O}(m/c)$. The result can be even worse in weighted graphs. But a graph with a large total edge weight necessarily contains a subgraph with large connectivity, as we can see in Lemma 7.5.

Lemma 7.5 ([73]). *A graph with total edge weight at least $k(n-1)$ has a k-strong component.*

Proof. We prove this by contradiction. Suppose G is a smallest counterexample with n vertices. Since G is not k-connected, it has a cut C of size less than k. By removing C we have two connected components G_1 and G_2 with n_1 and $n_2 = n - n_1$ vertices respectively. Since G is the smallest and G_i is not k-strong for $i \in \{1, 2\}$, G_i must have total edge weight less than $k(n_i - 1)$. Thus, the total edge weight of G is less than $k(n_1 - 1) + k(n_2 - 1) + k = k(n - 1)$, a contradiction. □

From this observation, Benczur and Karger show that we can sample each edge with different probability according to its edge strength. Suppose an n-vertex graph G with minimum cut c has an induced subgraph K with connectivity $k \gg c$. We can sample edges in K with probability $\tilde{\Omega}(1/k)$ and reweight this edge to $w(e)/k$ if sampled, while preserving cut values in K (up to some approximation factor). Thus, the sampled graph will have less edges and all cut values in the sampled graph are concentrated around their expectations. In general, for each edge e with strength k_e, we sample e with probability $\tilde{\Omega}(1/k_e)$ and change the weight to $w(e)/k_e$ if sampled. They show that the sampled graph will have $\tilde{O}(n/\varepsilon^2)$ edges.

Theorem 7.6 (Theorem 6.2 [43]). *Let $G = (V, w)$ be a weighted graph and k_e be the edge strength of $e \in E$. Let H be a graph formed by sampling each edge e with probability $p_e = \min(\frac{\rho_\varepsilon w(e)}{k_e}, 1)$, where $\rho_\varepsilon = 3(d+3)\ln n/\varepsilon^2$ for some integer d, and including it with weight $w(e)/p_e$. Then with high probability:*

1. *H has $O(n\rho_\varepsilon)$ edges.*
2. *Every cut in H is within $(1 \pm \varepsilon)$-factor of its value in G.*

In the above theorem, the edge strength k_e is known for all edge e which can be computed by maximum flow algorithm. However, we can also work without the exact edge strengths. Theorem 7.7 shows that there exists an algorithm with running time $\tilde{O}(m)$ to produce edge strength

estimates. Then, Theorem 7.8 shows that Theorem 7.6 remains asymptotically correct even if we use the edge strength estimates. After computing the edge strength estimates, ε-cut sparsifier can be constructed by sampling each of the m edges of G. The total running time is $\tilde{O}(m)$.

Theorem 7.7 (Theorem 6.5 [43])**.** *Let $G = (V, w)$ be a weighted graph with n vertices and m edges. There is a deterministic algorithm that in time $O(m \log^3(n))$ can produce edge strength estimates $\tilde{k}_e$ such that*

1. *$\tilde{k}_e \leq k_e$ for all $e \in E(G)$ and*
2. *$\sum_{e \in E(G)} \frac{w(e)}{\tilde{k}_e} = O(n)$.*

Theorem 7.8. *Let $G = (V, w)$ be a weighted graph. For each edge $e \in E$, let k_e be the edge strength of $e \in E$. Suppose we are given $\{\tilde{k}_e\}_{e \in E}$ such that $\frac{1}{4}k_e \leq \tilde{k}_e \leq k_e$. Let H be a graph formed by sampling each edge e with probability $p_e = \min(\frac{\rho_\varepsilon w(e)}{\tilde{k}_e}, 1)$, where $\rho_\varepsilon = 3(d+3)\ln n/\varepsilon^2$ and including it with weight $w(e)/p_e$. Then with high probability, H has $O(n \ln n/\varepsilon^2)$ edges, and every cut in H has value $(1 \pm \varepsilon)$ of the original value in G.*

Proof. The bound on the number of edges comes from the fact that $\sum_{e \in E(G)} \frac{w(e)}{\tilde{k}_e} = O(n)$. The expected number of edges is $\sum_{e \in E} \frac{\rho_\varepsilon w(e)}{\tilde{k}_e} = O(\rho_\varepsilon n)$. □

7.2.1 RSW cut sparsifier

Inspired by Benczur and Karger's algorithms, Rubinstein, Schramm, and Weinberg (RSW)[14] develop an algorithm to compute a cut sparsifier in an unknown graph with cut query access. They showed that a cut sparsifier can be constructed from this graph with $\tilde{O}(n/\varepsilon^2)$ cut queries. We have seen that Benczur and Karger's algorithm efficiently computes approximate edge strengths by looking at each edge in a known graph. In contrast, the RSW algorithm (shown in Algorithm 7.9) subsamples from the original graph in each iteration to avoid the expense of sampling each edge. They refer to Theorem 7.6 to prove the correctness of their algorithm. However, the sampling in Theorem 7.6 is done to every edge in the original graph, which is not the case in the RSW's algorithm. Therefore, to prove the correctness of their algorithm, we need to show that subsampling some edges still yields the same result. This is shown in Theorem 7.10.

Algorithm 7.9 (Approximating edge strengths)**.**

Input: *An accuracy parameter ε, and a cut query oracle for unweighted graph G.*

Output: *Edge strength approximate $\{k'_e\}_{e\in E}$.*

1. *Initialize an empty graph H on n vertices.*
2. *For $j = 0, \cdots, \log n$, set $\kappa_j = n/2^j$ and:*
   (a) *Subsample G' from G by taking each edge of G with probability $q_j = \min(100 \cdot 40 \cdot \frac{\ln n}{\kappa_j}, 1)$.*
   (b) *In each connected component of G':*
      i. *While there exists a cut of size $\leq q_j \cdot \frac{4}{5}\kappa_j$, remove the edges from that cut, and then recurse on the two sides. Let the connected components induced by removing the cut edges be $C_1, \ldots, C_r$.*
      ii. *For every $i \in [r]$, set $k'_e = \frac{1}{2}\kappa_j$ for all e with endpoints in C_i, and subsample every edge in $C_i \times C_i$ with probability $2q_j/\varepsilon^2$ and add it to H with weight $\frac{\varepsilon^2}{2q_j}$.*
      iii. *Update G by contracting C_i for each $i \in [r]$.*

Theorem 7.10. *Let $G = (V, w)$ be a weighted undirected graph. Sample an edge of G with probability $p_e = \min(1, \rho_\varepsilon w(e)/k_e)$ add it to G' with weight $w(e)/p_e$, where k_e is the edge strength of e and $\rho_\varepsilon = 3(d+2)\ln n/\varepsilon^2$. If we sample $O(n \ln n/\varepsilon^2)$ many times, then with probability $1 - O(n^{-d})$, every cut in G' has value within $(1 \pm \varepsilon)$ of its expectation.*

To prove Theorem 7.10, we need the following lemmas.

Lemma 7.11 (Chernoff-Hoeffding bound)**.** *Given any set of independent random variables X_i with values in $[0, 1]$, let $\mu = E[\sum X_i]$ and $\varepsilon < 1$. Then*

$$\Pr\left(\sum X_i \notin [(1-\varepsilon)\mu, (1+\varepsilon)\mu]\right) \leq 2e^{-\varepsilon^2\mu/3} \tag{7.4}$$

Lemma 7.12 ([76])**.** *Over all n-vertex graphs with min-cut c an even integer, and for any deletion probability p, the quantity $E[2^R]$ is maximized by a cycle on n vertices, where each adjacent pair is connected by $c/2$ edges.*

Proof of Theorem 7.10. Let $T = \sum_{e \in G} p_e$ The probability that an edge e in G is added to G' is p_e/T. Fix a cut C. Let X be the probability that we select an edge at C. Then, $X = \sum_{e \in C} p_e/T$. If we repeat this procedure T many times, then

$$\mu = E[X] = T \sum_{e \in C} \frac{p_e}{T} \tag{7.5}$$

$$= \sum_{e \in c} p_e \tag{7.6}$$

Now, since $\min(1, \rho_\varepsilon w_e/k_e)$ and by applying Lemma 2.17, we have

$$T = \sum_{e \in G} p_e = \sum_{e \in G} \rho_\varepsilon \frac{u_e}{k_e} = O(n\rho_\varepsilon)$$

$$= O\left(\frac{n \ln n}{\varepsilon^2}\right)$$

We apply the Chernoff-Hoeffding bound to each cut of G'. From Lemma 7.11, the probability that this cut deviates more than $\varepsilon < 1$ of its expected value is at most

$$Pr(TX \notin [(1-\varepsilon)\mu, (1+\varepsilon)\mu]) \leq 2e^{-\varepsilon^2 \mu/3} \tag{7.7}$$

Let $p_C = 2e^{-\varepsilon^2 \mu/3}$. Then the deviation probability in Eq. (7.7) is at most p_C. From the choice of ρ_ε have $p_C < 2n^{-(d+2)}$. So each cut is close to its expectation with high probability. However, we wish to bound the probability that any cut deviates. By union bound, the probability is at most $\sum_C p_C$. We consider an experiment to bound this probability.

Write $q_e = e^{-\varepsilon^2 p_e/3}$ and thus we have

$$p_C = 2 \prod_{e \in C} q_e \tag{7.8}$$

Consider G and suppose that edge e is deleted with probability q_e. Then p_C is twice the probability that every edge in C is deleted, i.e., that the graphs is disconnected at cut C. Thus, $\sum p_C$ is twice the expected number of cuts that are left empty with the edge deletions in this experiment.

Consider the connected components induced by the edges that are not deleted. Then, the empty cuts are those that partition the connected components into two parts without cutting any component. Suppose there are R connected components. The number of empty cuts is $2^{R-1} - 1$ (place each component in one of the two sides, divide the total by two since reversing the sides gives the same cut, and subtract by 1 since we want to avoid all components on the same side). Thus, the expected number of the empty cuts is $E[2^{R-1} - 1] = \frac{1}{2}E[2^R] - 1$, where R is the random variable denoting the number of connected components of G after deleting each edge e by probability q_e. We want to bound this quantity.

Suppose $q_e = p$ and the min-cut c is an even integer. Then, by Lemma 7.12, the n-vertex cycle maximizes $E[2^R]$. The number of connected components R on this cycle is equal to the number of $c/2$-edge "bundles" (edges connecting the same endpoints), and 1 if no bundle is deleted. Each bundle is deleted with probability $p^{c/2}$. Thus,

$$
\begin{aligned}
E[2^R] &= (1-p^{c/2})^n \cdot 2^1 + \sum_{r=1}^{n} \binom{n}{r} (p^{c/2})^r (1-p^{c/2})^{n-r} \cdot 2^r \\
&= (1-p^{c/2})^n + (1-p^{c/2})^n + \sum_{r=1}^{n} \binom{n}{r} (2p^{c/2})^r (1-p^{c/2})^{n-r} \\
&= (1-p^{c/2})^n + \sum_{r=0}^{n} \binom{n}{r} (2p^{c/2})^r (1-p^{c/2})^{n-r} \\
&= (1-p^{c/2})^n + (2p^{c/2} + 1 - p^{c/2})^n \\
&= (1-p^{c/2})^n + (1+p^{c/2})^n \\
&= 2 + O(n^2 p^c) \quad \text{when } n^2 p^c \leq 1
\end{aligned}
$$

Therefore,

$$
E[2^{R-1} - 1] = O(n^2 p^c) \tag{7.9}
$$

Since $2n^{-(d+2)} > p_C = 2p^c$, we have $p^c < n^{-(d+2)}$. Thus,

$$\sum_C p_C = 2E[2^{R-1} - 1] = O(n^{-d}) \tag{7.10}$$

Now, we want to generalize from $p_e = p$ to arbitrary probabilities. We can simulate edge e with deletion probability p_e by bundle of $k = 2\lceil \ln p_e / 2 \ln p \rceil$ parallel edges with deletion probability p for each edge. The probability that the entire bundle will be deleted is approaching p_e from below as $p \to 1$ (consequently, $k \to \infty$). Thus, each p_C in the simulated graph converges to its value in the original graph. It follows that $\sum_C p_C$ also approaches the desired limit and $\max_C p_C \leq 1$. Let $\delta = \max_C p_C$. In this simulated graph, we replace all edges with edges of uniform probability, thus, the result in Eq. (7.9) applies and we have $\sum_C p_C = O(n^2\delta)$. And since $p_C = 2e^{-\varepsilon^2\mu/3}$, we get the desired bound for the Chernoff bound in Eq. (7.7). □

After we establish Theorem 7.10, we can now analyse Algorithm 7.9. The following theorem shows that the algorithm outputs an ε-cut sparsifier with $\tilde{O}(n/\varepsilon^2)$ edges after $\tilde{O}(n/\varepsilon^2)$ cut queries.

Theorem 7.13. *Algorithm 7.9 produces an ε-cut sparsifier H of G with $\tilde{O}(n/\varepsilon^2)$ cut queries and the number of edges of H is $O(n \log n/\varepsilon^2)$.*

To prove the theorem, we use the following claims [14].

Claim 7.14. *At iteration $j = \lceil \log(n/k_e) \rceil$, e is either assigned $k'_e = \frac{1}{2}\kappa_j = n/2^{j+1} \geq k_e/4$ or has been assigned a larger value of k'_e.*

Claim 7.15. *At iteration j, no edges e with $k_e < \frac{1}{2}\kappa_j$ are assigned a strength approximation.*

Proof of Theorem 7.13. From the two claims above, the approximate edge strength is within a factor of two of the true strength. Step 1, 2(b)i, and 2(b)iii do not require any query calls. Hence, we only need to analyze Step 2a and Step 2(b)ii. In Step 2a, all components with strong connectivity larger than the current connectivity (κ_j) have been contracted, so there are no $2\kappa_j$-connected components. By Lemma 2.18, the current G has at most $O(n\kappa_j)$ edges. Therefore, we have $q_j = \tilde{O}(n/|E_G|)$, and the expected number of sampled edges is $\tilde{O}(n)$. This step can be computed in $\tilde{O}(n)$ cut queries.

Let $C = C_1 \cup \ldots C_r$ where $C_1, \ldots C_r$ are the connected components of G' obtained in step 2(b)i. By Lemma 2.18, the number of edges in C is $O(n\kappa_j)$. In Step 2(b)ii, every edge is sampled with probability $2q_j/\varepsilon^2 = \tilde{O}\left(\frac{n}{\varepsilon^2|E_C|}\right)$. Thus, the number of sampled edges is $\tilde{O}(n/\varepsilon^2)$ and the number of cut queries to sample these edges is $\tilde{O}(n/\varepsilon^2)$. Since the number of iterations is $\log n$, the total number of queries is $\tilde{O}(n/\varepsilon^2)$. The correctness of the algorithm follows from Theorem 7.10.

□

For weighted graphs, we set $q_j = \min(\frac{w(e)\ln n}{\kappa_j}, 1)$ in step 2a of Algorithm 7.9, and in step 2(b)ii, the edge sampled is added with weight $\frac{w(e)\varepsilon^2}{2q_j}$.

Theorem 7.16. *Let $G = (V, w)$ be a weighted undirected graph. We can construct an ε-cut sparsifier H of G with $\tilde{O}(n/\varepsilon^2)$ cut queries.*

Proof. The expected number of edges that we select in step 2a of Algorithm 7.9 is $O(\sum_{e\in G} \frac{w_G(e)}{\kappa_j} \ln n)$. At this point, all edges with strength greater than $2\kappa_j$ have been contracted. Thus, by Lemma 2.17, the number of expected edges is $\tilde{O}(n)$. By the algorithm described in **??**, we can find these edges with $\tilde{O}(n)$ cut queries.

In step 2(b)ii, the number of expected edges that we sample from $C_i \times C_i$ is $O(\frac{\sum_{e\in G} w_G(e)}{\varepsilon^2\kappa_j} \ln n)$. All edges in $C = C_1 \cup \ldots \cup C_r$ have strength less than $2\kappa_j$. Thus, again by Lemma 2.17, the number of edges is $\tilde{O}(n/\varepsilon^2)$. Therefore, the number of queries to sample these edges is $\tilde{O}(n/\varepsilon^2)$.

The total number of queries in the algorithm is $\tilde{O}(n/\varepsilon^2)$. □

7.2.2 Quantum cut sparsifier

By following the construction in Algorithm 7.9, we show the query complexity for quantum algorithm in the adjacency list model in the following theorem.

Theorem 7.17. *A quantum algorithm can output an ε-cut sparsifier H of an unweighted graph G with $O(\sqrt{mn}/\varepsilon)$ queries in the adjacency list model.*

Proof. The proof is similar to Theorem 7.13. Thus, we only need to analyze the number of queries in step 2a and 2(b)ii. The expected number of sampled edges in step 2a is $\tilde{O}(n)$, hence with

repeated Grover's algorithm, we can find all these edges after $\tilde{O}(\sqrt{mn})$ queries. In Step 2(b)ii, the number of edges to sample is $\tilde{O}(n/\varepsilon^2)$. Thus, we can sample these edges after $\tilde{O}(\sqrt{mn}/\varepsilon)$ and the total number of queries is $\tilde{O}\sqrt{mn}/\varepsilon$. □

This bound matches the upper bound shown by Apers and de Wolf [42], who give a quantum algorithm that can produce an ε-cut sparsifier in time $\tilde{\Omega}(\sqrt{mn}/\varepsilon)$ in the adjacency list model.

Theorem 7.18 ([42, Theorem 1])**.** *Let G be a weighted and undirected graph with n vertices and m edges. There exists a quantum algorithm that outputs with high probability the explicit description of an ε-cut sparsifier H of G with $\tilde{O}(n/\varepsilon^2)$ edges in time $\tilde{O}(\sqrt{mn}/\varepsilon)$ in the adjacency list model and time $\tilde{O}(n^{3/2}/\varepsilon)$ in the adjacency matrix model. Moreover, if G has integral weights then so does H, and if the largest weight of an edge of G is τ then the largest weight of an edge of H is $\tilde{O}(\tau\varepsilon^2 n)$.t*

Apers and de Wolf also show a matching lower bound for constructing a sparsifier.

Theorem 7.19 ([42], Quantum lower bound)**.** *Let G be a weighted undirected graph with n vertices and m vertices, and $\varepsilon \geq \sqrt{n/m}$. Given adjacency list access to G, a quantum algorithm can output an ε-cut sparsifier of G with $\tilde{\Omega}(\sqrt{mn}/\varepsilon)$ queries.*

Chapter 8

s-t Minimum Cut

This chapter is taken from the following arXiv paper: [77]

Abstract An s-t minimum cut in a graph corresponds to a minimum weight subset of edges whose removal disconnects vertices s and t. Finding such a cut is a classic problem that is dual to that of finding a maximum flow from s to t. In this work we describe a quantum algorithm for the minimum s-t cut problem on undirected graphs. For an undirected graph with n vertices, m edges, and integral edge weights bounded by W, the algorithm computes with high probability the weight of a minimum s-t cut after $\tilde{O}(\sqrt{m}n^{5/6}W^{1/3})$ queries, given adjacency list access to G. For simple graphs this bound is always $\tilde{O}(n^{11/6})$, even in the dense case when $m = \Omega(n^2)$. In contrast, a randomized algorithm must make $\Omega(m)$ queries to the adjacency list of a simple graph G even to decide whether s and t are connected.

8.1 Introduction

Let G be a graph with n vertices and m edges, and let s, t be two vertices of G. The problem of determining the maximum amount of flow $\lambda_{st}(G)$ that can be routed from s to t while respecting the capacity constraints of G is one of the most fundamental problems in theoretical computer science, whose study goes back at least to the 1950s and the pioneering work of Ford and Fulkerson [35].

By the max-flow min-cut theorem, $\lambda_{st}(G)$ is equal to the minimum weight of a cut separating s and t in G. From a maximum s-t flow one can compute a minimum s-t cut in linear time, but no such reduction is known the other way around. The Goldberg-Rao [78] algorithm, with running time $O(\min\{\sqrt{m}, n^{2/3}\}m\log(n)\log(W))$, where W is the largest weight of an edge, stood as the best bound for both problems for many years. In the past decade, however, beginning with work of Christiano et al. [79] there has been tremendous progress in max-flow algorithms by incorporating techniques from continuous optimization [80–86]. With a recent $\tilde{O}((m+n^{3/2})\log W)$ time max-flow algorithm by Brand et al. [87], there are now near-linear time randomized max-flow algorithms for dense graphs with polynomial weights.

Given its fundamental nature, there has been a surprising lack of work on quantum algorithms for the exact maximum flow or minimum s-t cut problem. As far as we are aware, the only work on the quantum complexity of these problems is by Ambainis and Špalek [41], who gave a quantum algorithm for max flow in a directed graph with integral capacities bounded by $W \leq n^{1/4}$ with running time $\tilde{O}(\min\{n^{7/6}\sqrt{m}W^{1/3}, m\sqrt{nW}\})$, given adjacency list access to G. This bound is completely subsumed by current classical randomized algorithms.

More recent work of Apers and de Wolf [42] gives a $\tilde{O}(\sqrt{mn}/\varepsilon)$ time quantum algorithm for finding a $(1+\varepsilon)$-approximation of the minimum s-t cut.[1] This algorithm works by first constructing an ε-cut sparsifier H of the input graph G with a quantum algorithm in time $\tilde{O}(\sqrt{mn}/\varepsilon)$. An ε-cut sparsifier is a re-weighted subgraph of G that has only $\tilde{O}(n/\varepsilon^2)$ edges but for which the weight of every cut agrees with that of G up to a factor of $1\pm\varepsilon$. One can then run a classical randomized s-t minimum cut algorithm on the sparse graph H to find an approximate s-t minimum cut of G. The quantum sparsification algorithm also plays a key role in our work.

In this chapter, we give a quantum algorithm that computes $\lambda_{st}(G)$ on dense simple graphs. More generally, for an undirected and integral weighted graph G with maximum weight W, we give a quantum algorithm with adjacency list access to G that with high probability outputs $\lambda_{st}(G)$ after $\tilde{O}(\sqrt{mn}^{5/6}W^{1/3}+n^{5/3}W^{2/3})$ queries (see Theorem 8.19). Thus for unweighted graphs the number of queries is $\tilde{O}(n^{11/6})$ even for dense instances with $m\in\Omega(n^2)$ edges. On the other hand, it is easy to show that in the worst case a randomized algorithm requires $\Omega(m)$ queries to

[1] The bound quoted in [42, Claim 9] for finding a $(1+\varepsilon)$-approximate minimum s-t cut is $\tilde{O}(\sqrt{mn}/\varepsilon+n/\varepsilon^5)$. However, plugging into the proof the improved randomized approximate s-t minimum cut algorithm of [88] with running time $\tilde{O}(m+\sqrt{mn}/\varepsilon)$ improves this to $\tilde{O}(\sqrt{mn}/\varepsilon)$.

the adjacency list of a graph with m edges even to decide whether s and t are connected (see Theorem 8.24). Our algorithm also works with adjacency matrix access to G, in which case the number of queries becomes $\tilde{O}(n^{11/6}W^{1/3})$ (see Corollary 8.20). In addition to $\lambda_{st}(G)$, our algorithm can output the edges of a minimum s-t cut and the corresponding bipartition of the vertex set. It does not however compute a maximum s-t flow, and finding a sublinear quantum algorithm for this problem remains a tantalizing open question.

Our quantum algorithm follows an algorithm of Rubinstein, Schramm and Weinberg (RSW) [14], which computes a minimum s-t cut in the *cut query* model. In the cut query model one can query a subset of vertices S and receives as an answer the total weight of edges with exactly one endpoint in S. The crux of the algorithm is a procedure to transform the input graph G into a sparser graph G' while preserving minimum s-t cuts. We use two quantum tools to improve this procedure. The first is to use the quantum algorithm from [42] to find an ε-cut sparsifier of G better than a classical sparsification algorithm. The second is to use Grover's algorithm to learn all the edges in the sparser graph G'. Once we explicitly know the graph G' we can use a classical randomized algorithm to compute a minimum s-t cut in G'. This blueprint is similar to the global min cut algorithm from [40], which is also based on a cut query algorithm by [14]. There as well a cut sparsifier of G is used to identify edges that can be contracted while preserving (non-trivial) global minimum cuts, and then Grover search is applied to learn the edges in the sparser contracted graph.

The contracted graph in the RSW procedure is obtained by contracting edges that cannot participate in any minimum s-t cut. To get an intuition for the idea, suppose that G is a simple graph and first imagine that we compute a maximum s-t flow F in G. If we let G_F be the graph whose edge weights are those of G minus the flow F, then s and t become disconnected in G_F—otherwise we could route more flow from s to t. Contracting all connected components of G_F then results in a sparser graph while preserving any edge that is part of a minimum s-t cut. This routine is also how one can compute a minimum s-t cut from a maximum s-t flow: the connected component of s in G_F gives one side of a vertex bipartition corresponding to a minimum s-t cut.

Of course we did not save anything in this example as we had to compute a maximum s-t flow in G. What RSW actually do is to first obtain an ε-cut sparsifier H of G. This is where we use the $\tilde{O}(\sqrt{mn}/\varepsilon)$ time quantum sparsification algorithm from [42]. As H has only $\tilde{O}(n/\varepsilon^2)$ edges it is much cheaper to carry out the above plan on H instead: we compute a maximum flow

F in H and consider the graph H_F whose weights are those of H minus the flow F. As cuts of H only approximate those in G we cannot fully contract the connected components in H_F and hope to preserve all minimum s-t cut of G. However, a key insight of RSW is that we can still safely contract induced subgraphs of H_F that are highly connected. Doing so results in the desired contraction G' of G that is relatively sparse—one can argue it has only $O(\varepsilon n^2)$ edges—yet preserves all minimum s-t cuts. We can learn the $O(\varepsilon n^2)$ edges of G' among the m edges of G using Grover search with $\tilde{O}(n\sqrt{m\varepsilon})$ queries. We then again run a randomized max flow algorithm on the now explicitly known G' to compute $\lambda_{st}(G)$. Balancing the terms $\sqrt{mn}/\varepsilon$ from the sparsifier computation and $n\sqrt{m\varepsilon}$ for learning the edges of G' leads to the choice $\varepsilon = n^{-1/3}$, and total queries of $\tilde{O}(\sqrt{m}n^{5/6})$.

8.2 Maximum flow

A flow network is a directed graph with two nodes marked as a source s and a sink t. Each edge has an individual capacity which is the maximum limit of flow that edge could allow.

Definition 8.1. The *capacity* function of a graph G is a map $c : V \times V \to \mathbb{R}_{\geq 0}$. If $e \notin E$, the capacity $c(e) = 0$.

Definition 8.2. A *flow network* is a tuple (G, s, t, c), where $G = (V, E)$ is a directed graph, $s, t \in V$ are the source and the sink node respectively, and c is the capacity function.

Definition 8.3 (Flow)**.** A *flow* is a function $f : V \times V \to \mathbb{R}$ satisfying the following constraints:

1. (capacity) $\forall u, v \in V, f(u, v) \leq c(u, v)$
2. (conservation) $\forall u \notin \{s, t\}, \sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u)$.
3. (skew symmetry) $\forall u, v \in V, f(u, v) = -f(v, u)$

Definition 8.4. The *value of flow* is the amount of flow passing from the source to the sink.

$$|f| = \sum_{v} f(s, v) \tag{8.1}$$

Definition 8.5 (Residual network)**.** Let (G, s, t, c) be a flow network with flow f. The *residual capacity* w.r.t flow f is $c_f(u, v) = c(u, v) - f(u, v)$. The *residual network* is given by $G_f = (V, E_f)$ with capacity function c_f where the arc set is $E_f = \{(u, v) : c(u, v) > 0\}$.

Definition 8.6 (Augmenting path)**.** An augmenting path is a path $(u_1, u_2, ..., u_k)$ in the residual network, where $u_1 = s, u_k = t$, and $c_f(u_i, u_{i+1}) > 0$.

Definition 8.7 (Maximum flow)**.** Given a flow network (G, s, t, c). The *maximum flow* problem is to find a flow f that maximizes $|f|$.

Theorem 8.8 ([35], Min-cut/max-flow)**.** *Let $G = (V, E)$ with a source $s \in V$ and a sink $t \in V$ and capacity c. Then the maximum value of a flow is equal to the minimum value of a cut.*

Given a graph G with n vertices and m edges, and a flow network (G, s, t, c), the maximum flow problem is an optimization problem to find a flow of maximum value from the source s to the sink t. This problem was formulated by Harris and Ross [] to model the Soviet railway traffic flow. The first known algorithm to solve the maximum flow problem is due to Ford and Fulkerson []. The idea behind the Ford-Fulkerson algorithm is to push the maximum possible flow along one of the paths from the source s to the sink t as long as there exists an s-t path of positive residual capacity. Suppose F is the value of the maximum flow. If the capacities are rational, the algorithm is guaranteed to find the maximum flow in time $O(F(m + n))$.

Algorithm 5 Ford-Fulkerson

Input: Flow network (G, s, t, c).
Output: Maximum flow f.

Initialize $f(u, v) \leftarrow 0$ for all $u, v \in V$.

while $\exists s$-t path P in G_f **do**
    Find $g = \min\{c_f(u, v) : (u, v) \in P\}$.
    $f \leftarrow f + g$.
    Update G_f based on f.
end while
Return f.

Theorem 8.9. *Let (G, s, t, c) be a flow network with integer capacities and maximum flow F. The Ford-Fulkerson algorithm (Algorithm 5) terminates in time $O(F(m + n))$.*

The state of the art for maximum flow problem is an algorithm by Brand et al. [] with runtime of $\tilde{O}((m+n^{1.5})\log W)$ where W is the maximum capacity. Their algorithm is a reduction from the minimum cost maximum flow problem where the costs are set to 1.

Definition 8.10 (Minimum cost maximum flow)**.** Let (G, s, t, c) be a flow network and $u : E \to \mathbb{R}$ be a cost function of G. The *minimum cost maximum flow* problem computes a maximum flow with minimum cost.

Theorem 8.11. *Let $G = (V, E)$ be a graph with n vertices and m edges, and (G, s, t, c) be a network flow with integer capacities and costs $u \in \mathbb{Z}^E$. There exists a randomized algorithm that with high probability computes a minimum cost maximum flow in time $O(m \log(||c||_\infty ||u||_\infty) + n^{1.5} \log^2(||c||_\infty ||u||_\infty))$.*

Let G_f be the residual graph of G w.r.t t, and $G_f(\Delta)$ is a graph obtained by removing all edges with capacity less than Δ in G_f, for some $\Delta \in \mathbb{Z}$.

Algorithm 6 Brand et al. Maximum flow algorithm

Input: Flow network (G, s, t, c) and cost $u \in \mathbb{Z}^E$.
Output: Maximum flow f.

1: Find a flow f in G.
2: $\Delta = 2^{\lfloor \log_2 ||c||_\infty \rfloor}$
3: **while** $\Delta \geq 1$ **do**
4: Find a maximum flow f' in $G_f(\Delta)$.
5: $f \leftarrow f + f'$.
6: $\Delta \leftarrow \Delta/2$.
7: Update G_f based on f.
8: **end while**
9: Return f.

Theorem 8.12. *Algorithm 6 computes a maximum flow in time $\tilde{O}(m + n^{1.5} \log ||c||_\infty)$ with high probability.*

Proof. We can see that at the last iteration when $\Delta = 1$, there is no augmenting path we can find in G_f. Therefore, the algorithm correctly computes a maximum flow.

To compute the time complexity, first we will prove that in the beginning of each iteration, the maximum flow in G_f is at most $2m\Delta$. This is true in the first iteration. For other iterations, note that all edges in the augmenting paths in G_f have capacity less than 2Δ because of the previous

iteration. Therefore, the maximum flow of $G_f(2\Delta)$ is zero. Since G_f can be obtained from $G_f(2\Delta)$ by adding at most m edges with capacity at most 2Δ, the maximum flow of G_f is at most $2m\Delta$.

Following the above observation, we can bound the capacity of each edge by $2m\Delta$ to compute a maximum flow in $G_f(\Delta)$. Therefore, by Theorem 8.11 and the fact that the ratio between the maximum and the minimum capacity is at most $2m$, we can compute a maximum flow in $G_f(\Delta)$ in $\tilde{O}(m + n^{1.5})$. Since the number of iterations in the algorithm is $\log ||c||_\infty$, the total running time of the algorithm is $\tilde{O}((m + n^{1.5}) \log ||c||_\infty)$. □

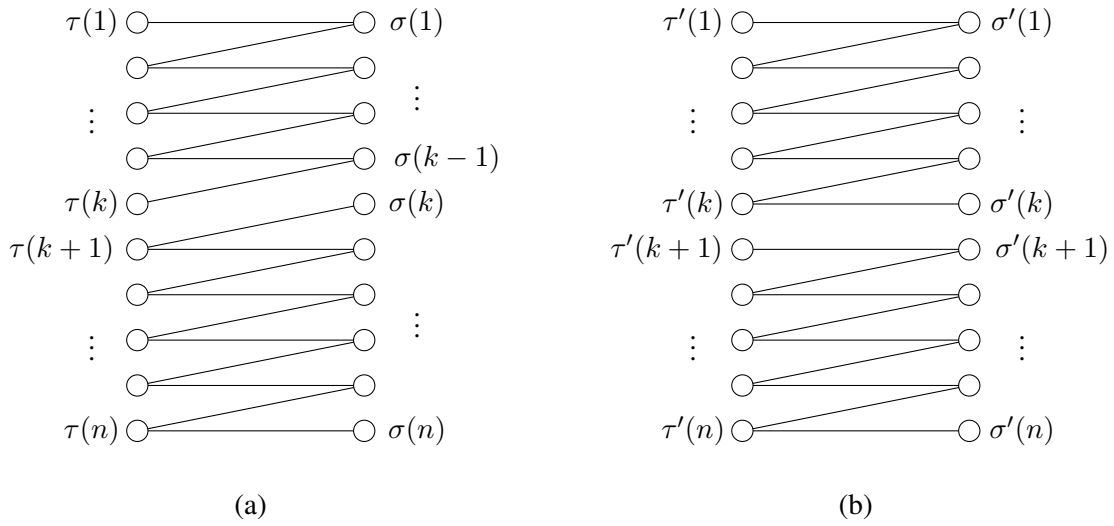
Lemma 8.13 ([14, Lemma 5.4])**.** *Let $G = (V, w)$ be a graph with integral weights from $[0, W]$ and let $s, t \in V$. Let F be a non-circular $s - t$ flow of value f in G. Then the total weight of flow $\sum_{e \in E(G)} F(e) \leq 10 \cdot n\sqrt{fW}$.*

8.3 Lower bound for s-t mincut

Theorem 8.14. *Any quantum algorithm that solves s-t mincut in the adjacency matrix model requires at least $\Omega(n^{1.5})$ queries.*

Proof. We will prove this by reduction from bipartite matching problem. Let B be an undirected bipartite graph with n vertices in each side and we are interested in the maximum matching in B. To reduce this problem to s-t maximum flow, we modify the graph as follows. Add new vertices s, t and direct s to the left side of the graph, then direct all edges from the left side to the right side, and finally direct all edges from the right side to t. We can see that there is a matching of size k in B if and only if there is a flow of value at least k in $B + \{s, t\}$. We use directed graph in the construction because if we do the same construction for undirected s-t maxflow, then we cannot guarantee that if there exists a flow of value k then a matching of size k exists.

Let X be the set of the bipartite graphs in Fig. 8.1(a) where τ and σ are permutations of $\{1, \ldots, n\}$, and $\frac{n}{3} \leq k \leq \frac{2n}{3}$. Let Y be the set of the bipartite graphs in Fig. 8.1(b) where τ' and σ' are permutations of $\{1, \ldots, n\}$, and $\frac{n}{3} \leq k' \leq \frac{2n}{3}$. Let $R \subseteq X \times Y$ where graph $y \in Y$ is obtained from $x \in X$ by choosing horizontal edges $(\tau(i), \sigma(i)), (\tau(j), \sigma(j))$, removing them, and adding two edges $(\tau(i), \sigma(j)), (\tau(j), \sigma(i))$.



(a) (b)

FIGURE 8.1: X and Y

We can see that the graphs in $X + \{s, t\}$ have maximum flow n, therefore these graphs have minimum cut of value n. On the other hand, the graphs in $Y + \{s, t\}$ have minimum cut of value $n - 1$. Therefore, these two instances have different minimum cut values.

From the proof of Theorem 4 in [89] and basic adversary bound (Theorem 4.12) , we get $m = m' = O(n^2)$ and $l_{max} = O(n)$ (the values m, m, l_{max} refer to Theorem 4.12). Then, the quantum lower bound for bipartite matching is $\Omega(n^{1.5})$. Hence, this proves the theorem. □

8.4 Algorithm for s-t mincut

Our s-t mincut algorithm is based on the following algorithm by Rubinstein, Schramm, and Weinberg [14, Algorithm 5.1], who used it to give a randomized cut query algorithm for the minimum s-t cut problem.

Algorithm 7 Algorithm for s-t-mincut on a weighted graph G

Input: Oracle access to a weighted graph $G = (V, w_G)$ with largest weight W, vertices $s, t \in V$, and a parameter $0 < \varepsilon < 1/3$.

Output: The shore of a minimum s-t cut in G and the value $\lambda_{st}(G)$.

1: Compute an ε-cut sparsifier H of G.

2: Compute a maximum s-t flow F in H and subtract F from H. Denote the result as H'.

3: Compute a $3\varepsilon nW$-strong partition $\mathcal{P} = \{A_1, \ldots, A_t\}$ of H'.

4: Let G' be the graph formed from G by contracting the vertices in each A_i, and let $a, b \in \mathcal{P}$ be the sets containing s, t respectively. Compute a minimum a-b cut $\Delta_{G'}(X')$ in G' and return $\lambda_{ab}(G')$ and the shore $X = \cup_{A \in X'} A$.

For completeness we show correctness of the template, largely following the discussion of [14]. In the next section we analyze the running time for a quantum algorithm that implements this algorithm with adjacency list access to G.

Theorem 8.15. *If every step of Algorithm 7 is performed correctly then the algorithm returns a minimum s-t cut of G.*

We first prove two claims from which the proof of Theorem 8.15 will easily follow.

Claim 8.16. *Let $G = (V, w_G)$ be a weighted graph with the largest edge weight W and $s, t \in V$. For $0 < \varepsilon < 1/3$, let $H = (V, w_H)$ be an ε-cut sparsifier of G and let F be a maximum s-t flow in H. Form the graph $H' = (V, w_{H'})$ where $w_{H'}(e) = w_H(e) - F(e)$ for all $e \in V^{(2)}$. Let $\mathcal{P}$ be a $3\varepsilon nW$-strong partition of H'. Then for any minimum s-t cut $\Delta_G(X)$ of G and all $A \in \mathcal{P}$ it holds that either $A \subseteq X$ or $A \subseteq \overline{X}$.*

Proof. Let $f_G = \lambda_{st}(G)$ and $f_H = \lambda_{st}(H)$. As H is an ε-cut sparsifier of G we have $f_G \leq f_H/(1-\varepsilon)$. Let $\Delta_G(X)$ be a minimum s-t cut of G, i.e. such that $s \in X, t \notin X$ and $w_G(\Delta_G(X)) = f_G$. Then $w_H(X) \leq (1+\varepsilon) f_G \leq \frac{1+\varepsilon}{1-\varepsilon} f_H$. Further, we have $w_H(X) = w_{H'}(X) + w_F(X)$ by definition of H' and F. We must have $w_F(X) \geq f_H$ since there is a flow of value f_H from s to t in H. Thus $w_H(X) \geq f_H + w_{H'}(X)$. Comparing inequalities we have $w_{H'}(X) \leq \frac{2\varepsilon}{1-\varepsilon} f_H < 3\varepsilon nW$, as $\varepsilon < 1/3$.

Now let $A \in \mathcal{P}$ and let $B = A \cap X$. If B is nontrivial, i.e. $\emptyset \neq B \subsetneq A$, then there is a cut of $H'[A]$ of weight less than $3\varepsilon nW$, a contradiction to the assumption that $\mathcal{P}$ is a $3\varepsilon nW$-strong partition of H'. Thus B must be trivial and either $A \subseteq X$ or $A \subseteq \overline{X}$. □

Claim 8.17. *Let G, H, H' and ε be as in Claim 8.16. Let $\mathcal{P} = \{A_1, \ldots, A_t\}$ be a $3\varepsilon nW$-strong partition of H' and G' be the graph formed from G by contracting vertices in the same set of $\mathcal{P}$. Let $a, b \in \mathcal{P}$ be the sets containing s and t respectively. Then $\lambda_{st}(G) = \lambda_{ab}(G')$. Moreover, if X' is the shore of a minimum a-b cut of G' then $X = \cup_{A \in X'} X$ is the shore of a minimum s-t cut of G.*

Proof. Let $G = (V, w_G)$ and note that by the definition of G' we have $G' = (\mathcal{P}, w)$ where $w(A_i, A_j) = w_G(E(A_i, A_j))$ for $i \neq j$.

Let us show that $\lambda_{st}(G') \leq \lambda_{st}(G)$. Let $\Delta_G(X)$ be a minimum s-t cut in G. For every $A_i \in \mathcal{P}$ we either have $A_i \subseteq X$ or $A_i \subseteq \overline{X}$ by Claim 8.16. Note that in particular this means $a \subseteq X$ and $b \cap X = \emptyset$. From X we define a set X' where for every $A_i \in \mathcal{P}$ we put A_i in X' iff $A_i \subseteq X$. Note that $\Delta_{G'}(X')$ is an a-b cut of G' and $w_G(\Delta_G(X)) = w(\Delta_{G'}(X'))$, thus $\lambda_{ab}(G') \leq \lambda_{st}(G)$.

We next show the general fact that contraction cannot decrease the minimum s-t cut value. For any set $X' \subseteq \mathcal{P}$ we can define a set $X \subseteq V$ by $X = \cup_{A \in X'} A$. Further $w(\Delta_{G'}(X')) = w_G(\Delta_G(X))$ by the definition of w. Thus $\lambda_{ab}(G') \geq \lambda_{st}(G)$.

This establishes $\lambda_{st}(G) = \lambda_{ab}(G')$. To see the "moreover" part of the claim, let X' be a minimum a-b cut of G'. We have just shown that $w(\Delta_{G'}(X')) = w_G(\Delta_G(X))$ for $X = \cup_{A \in X'} A$, thus X will be a minimum s-t cut of G. □

Proof of Theorem 8.15. If the steps of the algorithm are implemented correctly then H, H' satisfy the hypotheses of Claim 8.16. We can therefore invoke Claim 8.16 and Claim 8.17 to conclude that $\lambda_{st}(G) = \lambda_{ab}(G')$ and that X will be the shore of a minimum s-t cut in G. □

8.4.1 Implementation by a quantum algorithm

We now discuss the implementation of Theorem 8.19 by a quantum algorithm with adjacency list access to G. For the running time it is important to have an upper bound on the number of edges

in the contracted graph G' formed in step 4 of Algorithm 7. We do this by means of the following claim.

Claim 8.18. *Let G, H, H' and ε be as in Claim 8.16, with the following additional conditions:*

1. *The largest edge weight of G is $W \geq 1$.*
2. *H has integral weights and the largest edge weight is $W_H \in O(\varepsilon^2 nW)$.*

Let $\mathcal{P} = \{A_1, \ldots, A_t\}$ be a $3\varepsilon nW$-strong partition of H'. Then $\sum_{i=1}^{t} w_G(\Delta_G(A_i)) = O(\varepsilon n^2 W)$.

Proof. As H is an ε-cut sparsifier of G we have

$$\begin{aligned} w_G(\Delta_G(A_i)) &\leq \frac{w_H(\Delta_H(A_i))}{1-\varepsilon} \\ &= \frac{w_{H'}(\Delta_H(A_i)) + w_F(\Delta_H(A_i))}{1-\varepsilon} \ . \end{aligned}$$

By the definition of a $3\varepsilon n$-strong partition (Definition 2.19), we have $\sum_{i=1}^{t} w_{H'}(\Delta_{H'}(A_i)) = O(\varepsilon n^2)$. Note also that $w_{H'}(\Delta_{H'}(A_i)) = w_{H'}(\Delta_H(A_i))$ because $e \in E(H')$ implies $e \in E(H)$, and so $\sum_{i=1}^{t} w_{H'}(\Delta_H(A_i)) = O(\varepsilon n^2 W)$. By Lemma 8.13 we also have that

$$\begin{aligned} \sum_{i=1}^{t} w_F(\Delta_H(A_i)) &\leq 20n\sqrt{\lambda_{st}(H) W_H} \\ &= O(\varepsilon n^2 W) \ , \end{aligned}$$

using that $\lambda_{st}(H) \leq (1+\varepsilon)nW$ and $W_H \in O(\varepsilon^2 nW)$. Thus, $\sum_{i=1}^{t} w_G(\Delta_G(A_i)) = O(\varepsilon n^2 W)$. □

Theorem 8.19. *Let G be a graph with n vertices and m edges where all edge weights are integral and the largest weight of an edge is W. Given adjacency list access to G, there is a quantum algorithm that with high probability computes $\lambda_{st}(G)$ and the shore of a minimum s-t cut of G with $\tilde{O}(\sqrt{mn}n^{5/6}W^{1/3})$ queries.*

Proof. We follow the algorithm given in Algorithm 7 with the choice $\varepsilon = (nW)^{-1/3}$. As the choice of ε depends on W, we first need to know what the maximum weight of an edge of G is. We can do this by the quantum minimum finding routine of Dürr and Høyer [50] (Algorithm 1)

with high probability in time $\tilde{O}(\sqrt{m})$. With that settled, we now go over each step of the algorithm to explain how it is implemented and its running time.

Line 1 We run quantum algorithm of Apers and de Wolf [42] given in Theorem 7.18 to find an ε-cut sparsifier H of G with $\tilde{O}(n/\varepsilon^2)$ edges. As all edge weights of G are integral and the largest weight of an edge is W, by the "moreover" part of this theorem the edge weights of H are integral and the largest weight of an edge of H is $O(\varepsilon^2 nW)$. This step succeeds with high probability and takes $\tilde{O}(\sqrt{mn}/\varepsilon)$ queries.

Line 2 For this step we run the classical maximum s-t flow algorithm of [87] quoted in **??** to compute a max flow F in H. This algorithm requires a graph with integral weights, which is satisfied by H. This step does not need any oracle calls. As $\lambda_{st}(H) \leq (1+\varepsilon)nW$, the total weight of edges in F is $O(\varepsilon n^2 W)$ by Lemma 8.13. Then we can compute $H' = H - F$.

Line 3 This step does not need any oracle calls.

Line 4 Let $\mathcal{P} = \{A_1, \ldots, A_t\}$ be the $3\varepsilon nW$-strong partition of H' computed in the previous step. The graph G' is formed by contracting vertices in the same set of this partition. By Claim 8.18 the graph G' has $O(\varepsilon n^2 W)$ edges, assuming all previous steps have succeeded. Consider the concatenation of all the lists in the adjacency list representation of G. This gives a vector of dimension $2m$ and defines an ordering of edges of G where every edge appears twice. Define a string $x \in \{0,1\}^{2m}$ using the same ordering where $x(e) = 1$ if the endpoints of e are in distinct sets of the partition $\mathcal{P}$ and $x(e) = 0$ otherwise. A query to a bit of x can be answered with a single query to the adjacency list of G. By Theorem 2.22 via Grover search with $\tilde{O}(n\sqrt{m\varepsilon W})$ queries, with high probability we can determine if x has at most $O(\varepsilon n^2 W)$ ones, and if so learn the positions of all these ones. If the number of ones in x is larger than $O(\varepsilon n^2 W)$ then we abort. With high probability we do not abort and once we learn the positions of all the ones in x we can then classically learn the weight of the corresponding edges in G with $O(\varepsilon n^2 W)$ more queries. Then we have explicitly learned the graph G'.

Once we have an explicit description of G' we can run the max flow algorithm of [87] to compute a maximum a-b flow F' in G' where $a, b \in \mathcal{P}$ are the sets of the partition containing s, t respectively. The flow of F' gives $\lambda_{ab}(G')$. To also compute the shore of a minimum a-b cut of G' we consider the residual graph of the flow F' and compute the connected component containing a.

Correctness and total queries Each step individually succeeds with high probability and so with high probability all steps will be correct. We can thus invoke Theorem 8.15 to conclude that the algorithm is correct with high probability.

Queries to G are only made in the quantum steps as all classical steps work on graphs explicitly constructed by the algorithm. Thus the total number of queries is $\tilde{O}(\sqrt{m}n^{5/6}W^{1/3})$. □

We have the following result for the adjacency matrix model.

Corollary 8.20. *Let G be a graph with n vertices and m edges where all edge weights are integral and the largest weight of an edge is W. Given oracle access to the adjacency matrix of G, there is a quantum algorithm that computes $\lambda_{st}(G)$ and the shore of a minimum s-t cut of G with high probability after $\tilde{O}(n^{11/6}W^{1/3})$ queries.*

Proof. As in the proof of Theorem 8.19 we first compute the maximum weight of an edge of G. This can be done by the quantum minimum finding routine of Dürr and Høyer [50].

In Line 1 we run the adjacency matrix version of the sparsifier algorithm from [42] (quoted in Theorem 7.18), which takes $\tilde{O}(n^{3/2}/\varepsilon) = \tilde{O}(n^{11/6}W^{1/3})$ queries. In Line 4 we find the $O(\varepsilon n^2 W)$ edges of G' by applying Theorem 2.22 over the $O(n^2)$ entries of the adjacency matrix, rather than the $O(m)$ entries of the adjacency list. This takes $\tilde{O}(n^2\sqrt{\varepsilon W}) = \tilde{O}(n^{11/6}W^{1/3})$ queries.

The remaining steps are performed classically on graphs explicitly constructed by the algorithm.

Thus, the total number of queries is $\tilde{O}(n^{11/6}W^{1/3})$. □

Remark 8.21. *After finding the shore of a minimum s-t cut via Theorem 8.19 or Corollary 8.20 one can also learn the edges in the cut by a quantum algorithm in the same asymptotic running time. If the maximum weight of an edge is W then in a graph with integral weights the number of edges in a minimum s-t cut is at most nW. We can learn these $O(nW)$ edges by Grover's*

algorithm (Theorem 2.22) in time $\tilde{O}(\sqrt{mnW})$ in the adjacency list model or time $\tilde{O}(n^{3/2}\sqrt{W})$ in the adjacency matrix model. In both cases this is low order to the $\tilde{O}(n^{5/3}W^{2/3})$ term.

8.5 Randomized lower bound

In this section we show that a randomized algorithm that computes $\lambda_{st}(G)$ with success probability at least $9/10$ on simple graphs G with m edges must make $\Omega(m)$ queries to the adjacency list of G in the worst case. This holds true even for just determining if s and t are connected in G, which we call the USTCON problem.

Definition 8.22 (USTCON)**.** In the USTCON problem one is given adjacency list access to an undirected graph $G = (V, E)$ and two distinguished vertices $s, t \in V$. The problem is to determine if s and t are connected in G.

To show a lower bound on USTCON we will use the classical adversary method for randomized query complexity [90–92], a randomized analog of the quantum adversary method [60]. We will just need a simple unweighted version of the method, adapted from [90, Theorem 4.3].

Lemma 8.23 (cf. [90, Theorem 4.3])**.** *For a finite set Σ and $S \subseteq \Sigma^n$, let $f : S \to \{0, 1\}$. Let $X \subseteq f^{-1}(0)$ and $Y \subseteq f^{-1}(1)$. Let $R \subseteq X \times Y$ be such that for every $x \in X$ there are at least t different $y \in Y$ such that $(x, y) \in R$ and for every $x \in X$ and $k \in [n]$ there are at most ℓ different $y \in Y$ such that $(x, y) \in R$ and $x_k \neq y_k$. Then any randomized algorithm that computes f with success probability $9/10$ must make $\Omega(t/\ell)$ queries.*

Theorem 8.24. *A randomized algorithm that solves USTCON with success probability at least $9/10$ on graphs with m edges edges requires $\Omega(m)$ queries.*

Proof. First we show the lower bound in the dense case where $m = \Omega(n^2)$ using Lemma 8.23. We construct sets of negative instances X and positive instances Y. Suppose that n is even. Excluding s and t we partition the remaining $n - 2$ vertices into two sets A and B each of size $(n - 2)/2$. In all instances, s will be connected to all vertices in A and t will be connected to all vertices in B. X will consist of a single negative instance G where we additionally put a clique on the vertices in A and a clique on the vertices in B and no further edges. Thus s and t will not be connected in G. Note that apart from s and t, every vertex has degree $(n - 2)/2$ in G.

For Y we create a family of $2\binom{n-2}{2}^2$ positive instances. A positive instance will be labeled by $a = \{a_1, a_2\} \in A^{(2)}, b = \{b_1, b_2\} \in B^{(2)}$ and a bit $c \in \{0, 1\}$. Associated to a, b, c we construct a graph $G_{a,b,c}$ as follows. Take the graph G and remove the edges a and b. If $c = 0$ then add edges $\{\min\{a_1, a_2\}, \min\{b_1, b_2\}\}$ and $\{\max\{a_1, a_2\}, \max\{b_1, b_2\}\}$ to form $G_{a,b,c}$. Otherwise if $c = 1$ then add edges $\{\min\{a_1, a_2\}, \max\{b_1, b_2\}\}$ and $\{\max\{a_1, a_2\}, \min\{b_1, b_2\}\}$ to form $G_{a,b,c}$. In both cases all vertices in $A \cup B$ have degree $(n-2)/2$ in $G_{a,bc}$ and there will be two edges connecting A and B so $\lambda_{st}(G_{a,b,c}) = 2$. This completes the description of the instances.

The degree sequences of all instances are the same, thus we may assume this is known to the algorithm and the algorithm does not need to make any degree queries. We thus focus on queries to the name of the i^{th} neighbor of a vertex v. For this it is important to specify the ordering of vertices given in the adjacency lists. For all vertices we will use the same ordering in their list. Let $k = (n-2)/2$ and label the vertices of A as $a_1, a_2, \ldots, a_k$ and the vertices of B as $b_1, b_2, \ldots, b_k$. We use the ordering $s < t < a_1 < b_1 < \cdots < a_k < b_k$.

We are now ready to show the lower bound using Lemma 8.23. We put G in relation with all $2\binom{n-2}{2}^2$ of the $G_{a,b,c}$. Now consider how many of the $G_{a,b,c}$ differ from G in a specific location of the adjacency list, specifically consider the i^{th} neighbor of a vertex $a_j \in A$. In G the i^{th} neighbor of a_j is

1. s if $i = 1$.
2. a_{i-1} if $i \leq j$.
3. a_i if $i > j$.

The i^{th} neighbor of a_j in $G_{a,b,c}$ will be the same unless

1. $i \leq j$ and $a = \{a_{i-1}, a_j\}$.
2. $i > j$ and $a = \{a_j, a_i\}$.

In either case, the number of a, b, c for which $G_{a,b,c}$ differs from G on the name of the i^{th} neighbor of a_j is $2\binom{n-2}{2}$, as one only has free choice of the edge b and the bit c. A similar argument holds when considering the i^{th} neighbor of a vertex $b_j \in B$. Thus for any position of the adjacency list

the number of $G_{a,b,c}$ that differ from G is at most $2\binom{n-2}{2}$ and by Lemma 8.23 we obtain a lower bound of $\binom{n-2}{2} = \Omega(m)$, proving the theorem in the dense case.

Finally, let us treat the general case of graphs with at most m edges. We choose the largest integer p such that $2(p + \binom{p}{2}) \leq m$ and then take disjoint sets of vertices A and B both of size p. We then repeat the construction from the dense case on s, t, A, B. The lower bound will be $\binom{p}{2} = \Omega(m)$ as desired. □

Chapter 9

Discussion

Connectivity and s-t minimum cut problems are fundamental in graph theory and have been studied extensively in several different models. We show interesting results in the global query model for connectivity and the local query model for the s-t problem. We consider the following problems for future research.

1. We have shown that connectivity has an efficient algorithm with matrix-vector multiplication queries to the adjacency matrix. It remains an interesting question to find an example of a graph problem that can be solved much more efficiently with matrix-vector multiplication queries to the signed vertex-edge incidence matrix than with matrix-vector multiplication queries to the adjacency matrix. Sun et al. [15] show that one can find a spectral sparsifier of a graph with $\mathrm{polylog}(n)$ matrix-vector multiplication queries to the signed vertex-edge incidence matrix. This means that one can solve the problem of determining if the edge connectivity of a simple graph is at least $2k$ or at most k with $\mathrm{polylog}(n)$ matrix-vector multiplication queries to the signed vertex-edge incidence matrix. It is an interesting open question if this can also be done with $\mathrm{polylog}(n)$ matrix-vector multiplication queries to the adjacency matrix.

2. What is the randomized *non-adaptive* complexity of connectivity with matrix-vector multiplication queries to the adjacency matrix? The $O(\log^4(n))$ query algorithm in the signed

vertex-edge incidence matrix model is non-adaptive, but we do not see how to design a non-adaptive algorithm making $\mathrm{polylog}(n)$ matrix-vector multiplication queries to the adjacency matrix.

3. How large can the minimum-cut linear-query certificate complexity of a graph be?

4. What is the bounded-error randomized linear query complexity of connectivity? It seems that new ideas are needed to show lower bounds for the bounded-error model.

5. We show a query efficient quantum algorithm for the s-t minimum cut problem in the local query model. However, a time efficient quantum algorithm is still an open problem.

Bibliography

[1] Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proceedings of the 26th Annual Conference on Learning Theory*, volume 30 of *Proceedings of Machine Learning Research*, pages 797–818, Princeton, NJ, USA, 12–14 Jun 2013. PMLR.

[2] Nader H. Bshouty and Hanna Mazzawi. Reconstructing weighted graphs with minimal query complexity. *Theor. Comput. Sci.*, 412(19):1782–1790, apr 2011. ISSN 0304-3975. doi: 10.1016/j.tcs.2010.12.055.

[3] Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theor. Comput. Sci.*, 417:23–35, feb 2012. ISSN 0304-3975. doi: 10.1016/j.tcs.2011.09.005.

[4] Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 749–758, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580470. doi: 10.1145/1374376.1374484.

[5] Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. pages 246–258, 04 2006. ISBN 978-3-540-63397-6. doi: 10.1007/3-540-63397-9$_1$9.

[6] Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 608–615. SIAM, 2010. doi: 10.1137/1.9781611973075.51.

[7] Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proceedings of the 18th International Conference on Algorithmic Learning Theory*, ALT '07, page 285–297, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 9783540752240. doi: 10.1007/978-3-540-75225-7$_2$4.

[8] Hasan Abasi and Nader H. Bshouty. On learning graphs with edge-detecting queries. In *ALT*, 2019.

[9] Dana Angluin and Jiang Chen. Learning a hidden graph using o(logn) queries per edge. *Journal of Computer and System Sciences*, 74(4):546–556, 2008. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2007.06.006. Carl Smith Memorial Issue.

[10] Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM J. Discret. Math.*, 18:697–712, 2005.

[11] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. doi: 10.1145/3404867.

[12] Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM J. Comput.*, 33(2):487–501, 2004. doi: 10.1137/S0097539702420139.

[13] Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Triangle estimation using polylogarithmic queries. *ArXiv*, abs/1808.00691, 2018.

[14] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 39:1–39:16. LIPICS, 2018. doi: 10.4230/LIPIcs.ITCS.2018.39.

[15] Xiaoming Sun, David P. Woodruff, Guang Yang, and Jialin Zhang. Querying a matrix through matrix-vector products. *ACM Trans. Algorithms*, 17(4), oct 2021. ISSN 1549-6325. doi: 10.1145/3470566.

[16] Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph connectivity and single element recovery via linear and OR queries. In Petra Mutzel, Rasmus Pagh, and Grzegorz

Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 7:1–7:19, 2021.

[17] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 459–467. SIAM, 2012. doi: 10.1137/1.9781611973099.40.

[18] Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. *ACM Trans. Comput. Theory*, 13(2):8:1–8:24, 2021. doi: 10.1145/3442352.

[19] Talya Eden and Will Rosenbaum. Lower Bounds for Approximating Graph Parameters via Communication Complexity. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)*, volume 116 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-085-9. doi: 10.4230/LIPIcs.APPROX-RANDOM.2018.11.

[20] R. Paturi and J. Simon. Probabilistic communication complexity. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 118–126, 1984. doi: 10.1109/SFCS.1984.715908.

[21] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 20–29, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237823.

[22] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 539–550, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912640. doi: 10.1145/62212.62265.

[23] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proceedings of the Forty-Third Annual ACM Symposium on*

Theory of Computing, STOC '11, page 363–372, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306911. doi: 10.1145/1993636.1993686.

[24] Yannai A. Gonczarowski, Noam Nisan, Rafail Ostrovsky, and Will Rosenbaum. A stable marriage requires communication. *Games and Economic Behavior*, 118:626–647, 2019. ISSN 0899-8256. doi: https://doi.org/10.1016/j.geb.2018.10.013.

[25] Nicholas J. A. Harvey. *Matchings, matroids and submodular functions*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2008. URL http://hdl.handle.net/1721.1/44416.

[26] László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *27th Annual Symposium on Foundations of Computer Science, FOCS 1986*, pages 337–347, 1986. doi: 10.1109/SFCS.1986.15.

[27] Andrei Graur, Tristan Pollner, Vidhya Ramaswamy, and S. Matthew Weinberg. New query lower bounds for submodular function minimization. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 64:1–64:16, 2020.

[28] Troy Lee, Tongyang Li, Miklos Santha, and Shengyu Zhang. On the cut dimension of a graph. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021*, volume 200 of *LIPIcs*, pages 15:1–15:35, 2021.

[29] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM J. Comput.*, 35(6):1310–1328, 2006. doi: 10.1137/050644719.

[30] Aleksandrs Belovs and Ben Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. volume 7501, 03 2012. ISBN 978-3-642-33089-6. doi: 10.1007/978-3-642-33090-2$_1$8.

[31] Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Information and Computation*, 18, 10 2016. doi: 10.26421/QIC18.1-2-2.

[32] Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1, 04 2017. doi: 10.22331/q-2017-08-17-26.

[33] Kai DeLorenzo, Shelby Kimmel, and R. Teal Witter. Applications of the quantum algorithm for st-connectivity. In *Theory of Quantum Computation, Communication, and Cryptography*, 2019.

[34] Troy Lee, Miklos Santha, and Shengyu Zhang. Quantum algorithms for graph problems with cut queries. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 939–958. SIAM, 2021.

[35] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[36] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic global minimum cut of a simple graph in near-linear time. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 665–674, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335362. doi: 10.1145/2746539.2746588.

[37] Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Minimum cut in $O(m \log^2 n)$ time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.ICALP.2020.57.

[38] Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 496–509, 2020.

[39] Jason Li. *Deterministic Mincut in Almost-Linear Time*, page 384–395. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450380539.

[40] Simon Apers and Troy Lee. Quantum complexity of minimum cut. In *Proceedings of the 36th Computational Complexity Conference (CCC)*. LIPICS, 2021.

[41] Andris Ambainis and Robert Špalek. Quantum algorithms for matching and network flows. In *Proceedings of the 23rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 172–183. Springer, 2006. doi: 10.1007/11672142_13.

[42] Simon Apers and Ronald de Wolf. Quantum speedup for graph sparsification, cut approximation and Laplacian solving. In *Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020.

[43] András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015. doi: 10.1137/070705970.

[44] Du Ding-Zhu and Frank K Hwang. *Combinatorial group testing and its applications*. World Scientific Publishing Company, 1993.

[45] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511804441.

[46] L. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 78:325–328, 1997.

[47] Michel Boyer, Gilles Brassard, Peter Hoyer, and Alain Tappa. *Tight Bounds on Quantum Searching*, volume 46, pages 187 – 199. 01 2005. ISBN 9783527603091. doi: 10.1002/3527603093.ch10.

[48] Ethan Bernstein and Umesh V. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26 (5):1411–1473, 1997. doi: 10.1137/S0097539796300921.

[49] Aleksandrs Belovs. Quantum algorithms for learning symmetric juntas via the adversary bound. *Computational Complexity*, 24(2):255–293, 2015. doi: 10.1007/s00037-015-0099-2.

[50] Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *CoRR*, quant-ph/9607014, 1996.

[51] J. H. van Lint and R. M. Wilson. *A course in combinatorics*. Cambridge University Press, 2nd ed edition, 2001. ISBN 0521803403; 9780521803403; 9780521006019; 0521006015; 9780511672897; 0511672896.

[52] Dominic J. A. Welsh. *Matroid Theory*. L.M.S. Monographs 8. Academic Press, 1976.

[53] András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC 1988*, pages 186–191, 1988. doi: 10.1145/62212.62228.

[54] Arijit Bishnu, Arijit Ghosh, Gopinath Mishra, and Manaswi Paraashar. Query complexity of global minimum cut. In *Electron. Colloquium Comput. Complex.*, 2020.

[55] Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. 45(4), 1998. ISSN 0004-5411. doi: 10.1145/285055.285060.

[56] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32, 01 2004. doi: 10.1145/258533.258627.

[57] Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Struct. Algorithms*, 20(2):165–183, mar 2002. ISSN 1042-9832. doi: 10.1002/rsa.10013.abs.

[58] Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. In *2011 IEEE 26th Annual Conference on Computational Complexity*, pages 210–220, 2011. doi: 10.1109/CCC.2011.31.

[59] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Limit on the speed of quantum computation in determining parity. *Phys. Rev. Lett.*, 81:5442–5444, Dec 1998. doi: 10.1103/PhysRevLett.81.5442.

[60] Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. doi: 10.1006/jcss.2002.1826.

[61] Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries, 2021.

[62] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. doi: 10.1017/CBO9780511804090.

[63] Avi Wigderson. The complexity of graph connectivity. In Ivan M. Havel and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS'92*, volume 629 of *Lecture Notes in Computer Science*, pages 112–132. Springer, 1992. doi: 10.1007/3-540-55808-X_10.

[64] Andrew M. Childs, Shih-Han Hung, and Tongyang Li. Quantum query complexity with matrix-vector products. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 55:1–55:19, 2021.

[65] S. Apers, Y. Efron, P. Gawrychowski, T. Lee, S. Mukhopadhyay, and D. Nanongkai. Cut query algorithms with star contraction. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 507–518, Los Alamitos, CA, USA, nov 2022. IEEE Computer Society. doi: 10.1109/FOCS54457.2022.00055.

[66] Jaroslav Nesetril, Eva Milková, and Helena Nesetrilová. Otakar Borůvka on the minimum spanning tree problem: Translation of both the 1926 papers, comments, history. *Discret. Math.*, 233(1-3):3–36, 2001. doi: 10.1016/S0012-365X(00)00224-7.

[67] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for L_p samplers, finding duplicates in streams, and related problems. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011*, pages 49–58. ACM, 2011.

[68] Nader H. Bshouty. Lower bound for non-adaptive estimation of the number of defective items. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPIcs*, pages 2:1–2:9, 2019.

[69] Ding-Zhu Du and Frank K Hwang. *Combinatorial Group Testing and Its Applications*. World Scientific, 2nd edition, 1999. doi: 10.1142/4252.

[70] A. G. D'yachkov and V. V. Rykov. Bounds on the length of disjunctive codes. *Probl. Peredachi Inf.*, 18:7–13, 1982.

[71] Miklós Ruszinkó. On the upper bound of the size of the r-cover-free families. *J. Comb. Theory, Ser. A*, 66(2):302–310, 1994.

[72] Nader H. Bshouty, Nuha Diab, Shada R. Kawar, and Robert J. Shahla. Non-adaptive randomized algorithm for group testing. In Steve Hanneke and Lev Reyzin, editors, *Proceedings of the 28th International Conference on Algorithmic Learning Theory*, volume 76 of *Proceedings of Machine Learning Research*, pages 109–128, Kyoto University, Kyoto, Japan, 15–17 Oct 2017. PMLR.

[73] András Benczúr and David R. Karger. Approximating $s - t$ minimum cuts in $o(n^2)$ time. In *Proceedings of the 28th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 47–55, 1996.

[74] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011. doi: 10.1137/08074489X.

[75] David R. Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, page 648–657, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916638. doi: 10.1145/195058.195422.

[76] M. V. Lomonosov and V. P. Poleskii. Lower bound of network reliability. *Problems of Information Transmission 8*, pages 118–123, 1972.

[77] Simon Apers, Arinta Auza, and Troy Lee. A sublinear time quantum algorithm for s-t minimum cut on dense simple graphs, 2021. URL https://arxiv.org/abs/2110.15587.

[78] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, 1998. doi: 10.1145/290179.290181.

[79] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 273–282. ACM, 2011. doi: 10.1145/1993636.1993674.

[80] Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–269. IEEE, 2013. doi: 10.1109/FOCS.2013.36.

[81] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 253–262. IEEE, 2013. doi: 10.1109/FOCS.2013.35.

[82] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 217–226. SIAM, 2014. doi: 10.1137/1.9781611973402.16.

[83] Richard Peng. Approximate undirected maximum flows in $O(m\,\mathrm{polylog}(n))$ Time. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1862–1867. SIAM, 2016. doi: 10.1137/1.9781611974331.ch130.

[84] Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt(rank) linear system solves. *CoRR*, abs/1910.08033, 2019.

[85] Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, (STOC)*, pages 803–814. ACM, 2020. doi: 10.1145/3357713.3384247.

[86] Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than Goldberg-Rao. In *IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2021.

[87] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances. In *53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 859–869. ACM, 2021. doi: 10.1145/3406325.3451108.

[88] Aaron Sidford and Kevin Tian. Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow. In *59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 922–933. IEEE, 2018.

[89] Shengyu Zhang. On the power of ambainis lower bounds. *Theoretical Computer Science*, 339(2):241–256, 2005. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2005.01.019.

[90] Scott Aaronson. Lower bounds for local search by quantum arguments. *SIAM Journal of Computing*, 35(4):804–824, 2006. doi: 10.1137/S0097539704447237.

[91] Sophie Laplante and Frédéric Magniez. Lower bounds for randomized and quantum query complexity using kolmogorov arguments. *SIAM Journal on Computing*, 38(1):46–62, 2008. doi: 10.1137/050639090.

[92] Andris Ambainis, Martins Kokainis, Krisjanis Prusis, Jevgenijs Vihrovs, and Aleksejs Zajakins. All classical adversary methods are equivalent for total functions. *ACM Transactions on Computational Theory*, 13(1):7:1–7:20, 2021. doi: 10.1145/3442357.