Quantum Algorithms for Boolean Matrix Product Verification

Yongshan Ding

Advisor: Ryan O'Donnell

April 29, 2016

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

An undergraduate thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science with Honors.

Keywords: Quantum algorithms, matrix multiplication, polynomial method, span program

Abstract

We studied the quantum query complexities of the problems related to matrix multiplications. We surveyed recent literature on the upper and lower bound of the quantum algorithms that compute or verify the product of two $n \times n$ matrices under ring, semi-ring, or Boolean semi-ring.

Then we present our partial results on the upper and lower bound of the Boolean matrix product verification problem, from the perspectives of span programs and polynomial method, respectively.

Acknowledgments

I am indebted to my research advisor, Prof. Ryan O'Donnell, for first introducing me to this wonderful research topic, and providing me with continuous support. At every point in time, he was there available to chat, sharing his advice and wisdom.

I would also like to thank my parents for their lifelong love and encouragement.

Contents

Chapter 1

Introduction

1.1 Query Complexity

One reason to study query complexity is that it is simple to analyze. As the the internal structure of an algorithm gets more and more sophisticated, it is often very hard to prove strong lower bounds on the time complexity of the function that the algorithm solves. In the query model that we will describe below, we only consider the accesses of the algorithm to the input string. This simplification will thus allow us prove nontrivial lower bounds. For example the well-known $\Omega(n \log n)$ lower bound for any comparison-based sorting algorithm uses exactly the model of query complexity.

Another reason that we are interested in the model of query complexity is that it is often the same as the time complexity model, in the sense that there is often an efficient implementation of the query function. In the query model, instead of accessing the input string x directly, it is done through some *black-box* or some *oracle* function $f(z)$. The query complexity of an algorithm is simply the number of queries it makes to f. So an efficient implementation of the function f would mean the query complexity of the algorithm being the same as its time complexity.

1.1.1 Deterministic

Consider the problem of computing a Boolean function $f : \{0,1\}^n \to \{0,1\}$ on an n-bit input string $z \in \{0,1\}^n$. Then any deterministic algorithm can be generalized as below. The algorithm follows some steps of computation (including reading input bits, or making decisions based on the input) and returns some value as the result. We have therefore simplified the model of computation to a deterministic *decision tree*, where each internal node represents reading an index of the input string, and each leaf node is the output value. Starting from the root, we decide a brach to go down, based on the input value we read at that node, until we reach the leaf of this decision tree. It is, therefore, natural to consider the depth of the decision tree as the *complexity*.

Formally, we call $D(f)$ the *deterministic query complexity* of f, which is defined as the minimal depth of a decision tree that evaluates f.

Example 1.1.1 (OR function)**.** *Let* $f : \{0,1\}^n \to \{0,1\}$ *be* $f(x) = OR_n(x) = \bigvee_{i=1}^{n} x_i$*. Then*

we have $D(OR_n) = n$.

The algorithm can deterministically output 1, as soon as it queried a bit that is 1. However, in the worst case, it still have to query every bit. This is not too hard to see, suppose a deterministic algorithm have queried all but one bit, and they all appear to be zero. Should it output 0 or 1? In fact, an adversary can always give an input string such that the algorithm output an incorrect result every time, because the correctness of the algorithm depends heavily on the last bit that it chose to ignore.

1.1.2 Randomized

We can do the same simple generalization for randomized algorithms. The bounded-error randomized query complexity $R(f)$ of a function f is defined as the complexity of the best randomized algorithm that computes $f(z)$ on input string z, with probability at least $2/3$.

Example 1.1.2 (OR function)**.** *The best randomized algorithm needs only* $\frac{2}{3}n$ *queries to the input oracle, i.e.* $R(OR_n) = \frac{2}{3}n$.

The algorithm chooses $\frac{2}{3}n$ uniformly random input bits to examine. In the case that $OR_n(x) = 0$, it will always output 0. In the case that $OR_n(x) = 1$, we have probability at least $2/3$ to find a bit 1 in the $\frac{2}{3}n$ chosen bits.

1.1.3 Quantum

The quantum query complexity $Q(f)$ of a function $f : \{0,1\}^n \to \{0,1\}$ is defined as the query complexity of the best quantum algorithm evaluating the function $f(x)$, where "best" means having the smallest query complexity over the worst possible input $z \in \{0,1\}^n$. So far, we have been using Boolean total function consistently. In fact, the definitions on query complexity also apply to functions that are not necessarily Boolean, and they don't have to be total. In other words, we can write the function in its most general form $f : [\ell]^n \supseteq \mathcal{D} \to [k]^m$ on input $z \in \mathcal{D}$.

To depict the model of quantum computation better, let's take a look at what does an arbitrary algorithm A actually looks like as a quantum circuits:

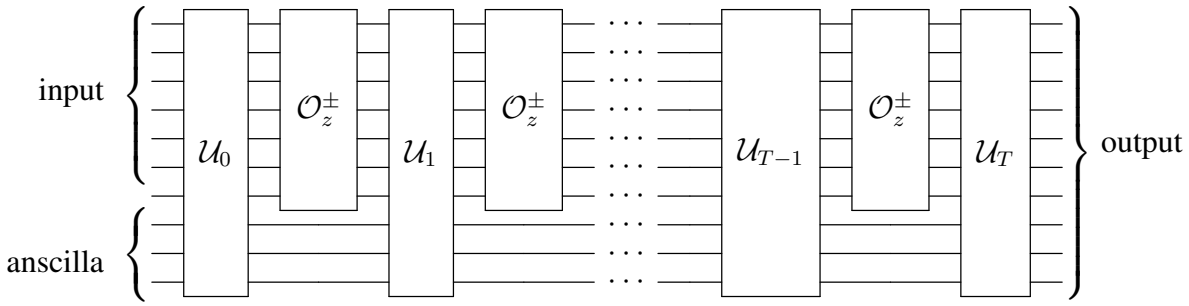

Figure 1.1: A typical quantum circuit with query complexity T, where $\mathcal{U}_i$'s are arbitrary unitary transformations and $\mathcal{O}_z^{\pm}$'s are queries to the input oracle.

In this example, the quantum algorithm with query complexity T is modeled as a sequence of $T + 1$ unitary transformations, interleaving with some number of queries to the input oracle. Notice that we call the oracle T times, with some unitary transformations in between. Since

all quantum operations are unitary, we can combine any number of intermediate operations and make a giant unitary transformation. After all the transformations have been performed, we will then read the output by measuring at the end.

We say that the quantum circuit evaluates f on input $z \in \mathcal{D}$ when the circuit outputs result $f(z)$ with probability at least $2/3$. The quantum query complexity $Q(f)$ of the algorithm is the smallest number of queries the circuit made to evaluate f.

One very useful fact about quantum query complexity is the Boolean function composition. Given a Boolean function $g : D^{n\times m} \supseteq \mathcal{D} \to \{0,1\}$ and another function $f : \{0,1\}^n \to \mathcal{E}$, then we can denote $f \circ g$ as the composed function, defined as follows:

$$(f \circ g)(x) = f(g(x_{1,1}, \ldots, x_{1,m}), \ldots, g(x_{n,1}, \ldots, x_{n,m})), \tag{1.1}$$

where $x \in \mathcal{D} \subseteq D^n$, and $x_{i,j} \in D$ for all $i \in [n], j \in [m]$. Then similar to the result $D(f \circ g) = D(f)D(g)$ in deterministic query complexity [Mon13], we have, in quantum query complexity, the following theorem:

Theorem 1.1.3 (Boolean function composition)**.** *If f and g are functions where the input of f is Boolean and the output of g is Boolean, then $Q(f \circ g) = \Theta(Q(f)Q(g))$.*

1.2 Matrix Multiplication

Matrix Multiplication has been playing important roles in research studies across different fields, for it is essential to many computational tasks, such as applications in linear systems of equations, group theory, and representation theory. Yet, many basic questions that are related to matrix multiplication still remain open, such as how efficiently can we compute or verify the product of two matrices, and can we do better in quantum setting?

1.2.1 Background

In this section, we will survey recent literatures that focused on problems centered around matrix multiplications. We will also looked at many variants of the original problem that arise in recent studies. They are shown to be as important in many aspects.

Classically, we have gone a long way from the most straightforward $O(n^3)$ algorithm of computing the product of two matrices, denoted MM. In 1969, Strassen first broke the $O(n^3)$ algorithm and proposed a better $O(n^{2.807})$ time algorithm [Str69]. Since then, various techniques that led to improvement in complexity have been discovered. Coppersmith and Winograd improved it to $O(n^{2.376})$ [CW87], and Williams to $O(n^{2.3727})$ [Wil12].

A common variant to the MM problem is the matrix product verification problem, denoted MPV. Instead of computing the resulting multiplication, now we are given three matrices, A, B, and C. Our job is to check if $C = AB$. Again classically, we have an algorithm from 1979, when Freivalds published an *optimal* $O(n^2)$ bounded-error probabilistic algorithm [Fre79].

Now we are interested in the same problems in the quantum setting. Before we do that, let's take a look at what are the common variants of the MM and MPV problem that have been studied.

Related problems

Definition 1.2.1 (MM: Matrix Multiplication)**.** Given input $A, B \in \boldsymbol{S}^{n\times n}$. Compute AB.
Definition 1.2.2 (MPV: Matrix Product Verification)**.** Given input $A, B, C \in \boldsymbol{S}^{n\times n}$. Check if $C = AB$.
Definition 1.2.3 (MvM: Matrix-vector Multiplication)**.** Given input $A \in \boldsymbol{S}^{n\times n}$ and $v \in \boldsymbol{S}^n$. Compute Av.
Definition 1.2.4 (MvPV: Matrix-vector Product Verification)**.** Given input $A \in \boldsymbol{S}^{n\times n}$ and $v, w \in \boldsymbol{S}^n$. Check if $w = Av$.

In some cases, we often simplify the problems by considering matrix A as part of the problem specification, instead of an input. Then, the problems become:

Definition 1.2.5 (MM^A: Matrix Multiplication)**.** For some known matrix $A \in \boldsymbol{S}^{n\times n}$, we are given input $B \in \boldsymbol{S}^{n\times n}$. Compute AB.
Definition 1.2.6 (MPV^A: Matrix Product Verification)**.** For some known matrix $A \in \boldsymbol{S}^{n\times n}$, we are given input $B, C \in \boldsymbol{S}^{n\times n}$. Check if $C = AB$.
Definition 1.2.7 (MvM^A: Matrix-vector Multiplication)**.** For some known matrix $A \in \boldsymbol{S}^{n\times n}$, we are given input $v \in \boldsymbol{S}^n$. Compute Av.
Definition 1.2.8 (MvPV^A: Matrix-vector Product Verification)**.** For some known matrix $A \in \boldsymbol{S}^{n\times n}$, we are given input $v, w \in \boldsymbol{S}^n$. Check if $w = Av$.

We are often interested in matrix multiplications under Boolean semi-ring. In particular:

Definition 1.2.9 (BMM^A: Boolean Matrix Multiplication)**.** For some known Boolean matrix $A \in \{0,1\}^{n\times n}$, we are given input $B \in \{0,1\}^{n\times n}$. Compute AB.
Definition 1.2.10 (BMPV^A: Boolean Matrix Product Verification)**.** For some known Boolean matrix $A \in \{0,1\}^{n\times n}$, we are given input $B, C \in \{0,1\}^{n\times n}$. Check if $C = AB$.
Definition 1.2.11 (BMvM^A: Boolean Matrix-vector Multiplication)**.** For some known Boolean matrix $A \in \{0,1\}^{n\times n}$, we are given input $v \in \{0,1\}^n$. Compute Av.
Definition 1.2.12 (BvPV^A: Boolean Matrix-vector Product Verification)**.** For some known Boolean matrix $A \in \{0,1\}^{n\times n}$, we are given input $v, w \in \{0,1\}^n$. Check if $w = Av$.

So in the above cases, the multiplication is replaced by a logical AND, and the addition is replaced by a logical OR. For example, in BMPV problem, we want to check if

$$C[i,j] = \bigvee_{k=1}^{n} A[i,k] \wedge B[k,j], \text{ for all } i,j \in [n]. \tag{1.2}$$

Known results

The reason that we are interested in so many variants of the problem is that they are shown to be very useful in the reduction of other problems, like the triangle-detection problem [MSS07, LMS13, LG14] and the graph-collision problem [ABIO13]. Here is a table of the complexities of some of the selected problems:

Problem	Input	Output	Complexity
MM	A, B	AB	$\Theta(n^2)$
MPV	A, B, C	(AB=C)?	$\Omega(n^{3/2}), O(n^{5/3})$
MPV^A	B, C	(AB=C)?	$\Theta(n^{3/2})$
MvPV^A	v, w	(Av=w)?	$\Theta(n)$
BvPV^A	v, w	(Av=w)?	$\Omega(n^{0.555}), O(n^{3/4})$
BMPV^A	B, C	(AB=C)?	$\Omega(n^{1.055}), O(n^{5/4})$
BMPV	A, B, C	(AB=C)?	$\Omega(n^{1.055}), O(n^{3/2})$

Table 1.1: Quantum query complexities for several selected problems.

1.2.2 Verifying Boolean matrix product

In particular, we will describe the ideas behind the current upper and lower bound proof on the BMPV problem. First, let's take a closer look at the Boolean formula stated above in Equation (1.2). Visually, the BMVP problem can be expressed as a constant-depth AND-OR formula as follows:

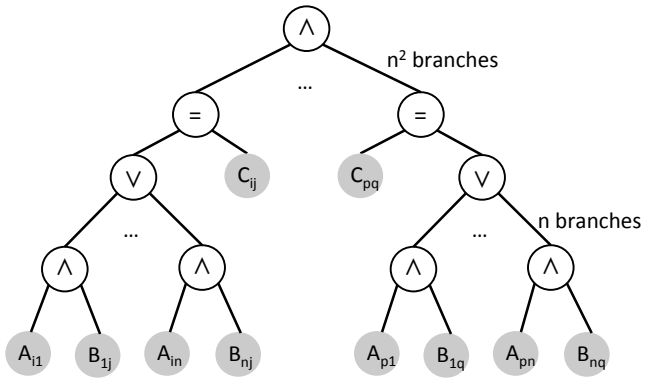

Figure 1.2: A constant-depth read-many AND-OR tree representing the BMPV problem.

Lower bound

Childs, Kimmel, and Kothari show that evaluating depth-2 Boolean circuits requires $\Omega(n^{0.555})$ queries [CKK12], and how this problem related to the Boolean matrix product verification problem. First, notice that the Boolean vector product verification problem with matrix A known (BvPV^A) can be expressed essentially as a depth-2 circuits. In particular, on input vector v, we want to verify $Av = 1$, where 1 is the all-one vector. Then the BvPV^A problem becomes a depth-2 circuit with n variables, n OR gates and 1 AND gate at the top.

$$BvPV^A(v) = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} A_{i,j} v_j \tag{1.3}$$

Thus it follows that $Q(BvPV^A) = \Omega(n^{0.555})$.

Theorem 1.2.13. *For any known matrix A,* $Q(BMPV^A) = \Omega(\sqrt{n}Q(BvPV^A)) = \Omega(n^{1.055})$

Proof sketch. In particular, we describe the proof for the special case of BMPV where matrix C is all-one matrix, checking whether AB = C. We can therefore think of the problem as n *independent* instances of the BvPV^A problem: Let b_i denote the i^{th} column of the input matrix B. Then

$$BMPV^A(B) = \bigwedge_i BvPV^A(b_i). \tag{1.4}$$

In other words, it is a composition of logical AND with the BvPV^A problem. Thus, we have from Theorem 1.1.3 that $Q(BMPV^A) = \Theta(Q(AND_n)Q(BvPV^A)) = \Omega(n^{1.055})$. □

Upper bound

Buhrman and Spalek [BS06] shows that the BMPV problem can be solved in $O(n^{3/2})$ queries. Notice that checking 1 entry of C requires $O(\sqrt{n})$ queries, so searching over n^2 entries for an incorrect entry by Grover's algorithm brings in another $O(n)$ factor. More formally, we want to check

$$C_{i,j} = \bigvee_k A_{i,k} \wedge B_{k,j}, \text{ for all } i, j \in [n] \tag{1.5}$$

And it requires at most $O(n^{3/2})$ queries as described above.

In the following chapters, we will show several techniques on proving upper and lower query complexities, and present our partial results on the Boolean matrix product verification problem.

Chapter 2

Lower Bound and the Polynomial Method

In this chapter, we describe a general technique for proving quantum query complexity lower bounds, called the Polynomial Method. And then apply this method to obtain partial results for the Boolean matrix product verification problem.

2.1 Polynomial Method

2.1.1 Quantum algorithms and polynomials

Recall from the previous section, we denote $\mathcal{O}_z$ the oracle (unitary transformation) on the input string $z \in [\ell]^n$. So quantumly, accessing the i^{th} index of the input string z is given by

$$|i\rangle |y\rangle \rightarrow |i\rangle |y \oplus z_i\rangle \tag{2.1}$$

Then we have the following:

Theorem 2.1.1. *Let $\mathcal{A}$ be a quantum query algorithm making t queries to $\mathcal{O}_z$ on input $z \in [\ell]^n$. If we denote $\widetilde{z_{i,c}} = \begin{cases} 1 & : z_i = c \in [\ell] \\ 0 & : otherwise \end{cases}$, then the amplitude of any basis state is a polynomial in $\widetilde{z_{i,c}}$ of degree less than or equal to t.*

Proof. We can show this by induction on t. Recall from the previous section, we had the model of computation being a sequence of unitary transformations $\mathcal{U}_i$, each of which followed by an query to the oracle $\mathcal{O}_z$. Suppose we start with all-zero inputs, then the base case is to apply unitary transformation $\mathcal{U}_0$ on the state $|0\rangle^n |0\rangle^m |0\rangle^a$ (assuming we reserve n qubits for the input, m qubits for the output, and a qubits for the anscilla.) We get the resulting state in its general form:

$$\sum_{j\in[n]} \sum_{b\in\{0,1\}^m} \sum_{i\in\{0,1\}^a} \alpha_{j,b,i} |j\rangle |b\rangle |i\rangle . \tag{2.2}$$

Since $\alpha_{j,b,i}$ is a degree-0 polynomial, the statement holds for base case.

Then assuming after k steps of oracle transformation, we arrive at some state:

$$|\psi\rangle = \sum_{j,b,i} P_{j,b,i}(\tilde{z}) |j\rangle |b\rangle |i\rangle \tag{2.3}$$

where each $P_{j,b,i}(\tilde{z})$ has degree at most k. Now, applying another oracle gate, we get,

$$\mathcal{O}_z |\psi\rangle = \sum_{j,b,i} P_{j,b,i}(\tilde{z}) |j\rangle |b \oplus z_j\rangle |i\rangle \tag{2.4}$$

$$= \sum_{j,b,i} \Big[\sum_{b' \in [\ell]} \widetilde{z_{j,b}} P_{j,b-b',i}(\tilde{z}) \Big] |j\rangle |b\rangle |i\rangle . \tag{2.5}$$

Thus the amplitudes after the application of $\mathcal{O}_z$ is a polynomial of degree at most $k+1$. Therefore, we have shown that for a t query algorithm, the amplitude of any bases state is a polynomial in $\widetilde{w_{i,c}}$ of degree at most t. □

Corollary 2.1.2. *The acceptance probability of a t-query algorithm on input $z \in [\ell]^n$ is a polynomial in z_i of degree at most $2t$.*

Proof. The probability of measuring some state $|j\rangle |b\rangle |i\rangle$ is $P_{j,b,i} P^*_{j,b,i}$, and both $P_{j,b,i}$ and $P^*_{j,b,i}$ are of degree at most t by the theorem above. □

Consider a Boolean function $f : \{0,1\}^n \to \{0,1\}$, then we call $P(z)$ of degree at most $2t$ a approximating polynomial when we have $|P(z) - f(z)| \leq \epsilon$, for all $z \in \{0,1\}^n$ and some small error bound ϵ.

Now to prove a lower bound on the quantum query complexity of $\mathcal{A}$, it suffice to show a lower bound on the approximating degree of the polynomial P.

2.1.2 Symmetrization

The idea behind the Polynomial Method is to simplified the analysis on quantum query complexity by transforming it to the one on polynomials, since polynomials are well-studied objects. However in many cases, bounding the degree of a multivariate polynomial is still rather complicated. To get around with that, we often proceed by transforming the multivariate polynomial into a univariate one. We call it the symmetrization step.

Let P be a polynomial in variables $z = (z_1, \ldots, z_n)$. Denote $k = |z|$ the Hamming weight of z, and let σ be a permutation on n elements, so $\sigma(z) = (z_{\sigma(1)}, z_{\sigma(2)}, \ldots, z_{\sigma(n)})$. Define the symmetrization polynomial $Q(k)$ as follows:

$$Q(k) := \mathop{\mathbf{E}}_{|z|=k} [P(z)]. \tag{2.6}$$

In other words, we take the average over all possible permutations σ of z.

Lemma 2.1.3. *Given any multilinear polynomial P, there exists a univariate polynomial Q of degree at most $deg(P)$ such that $Q(k) = \mathbf{E}_{|z|=k}[P(z)]$, for all $z \in \{0,1\}^n$.*

Proof. Given a multilinear polynomial P, which looks like:

$$\sum_{S\subseteq[n]} c_S\Big(\prod_{j\in S} z_j\Big). \tag{2.7}$$

We take any of its monomial term $\prod_{j\in S} z_j$, where $S \subseteq [n]$. Let $d = |S|$. Then the expectation of the monomial is,

$$\mathop{\mathbf{E}}_{k=|z|}\Big[\prod_{j\in S} z_j\Big] = \mathop{\mathbf{Pr}}_{k=|z|}\Big[z_j = 1, \forall j \in S\Big] \tag{2.8}$$

$$= \frac{\binom{n-d}{k-d}}{\binom{n}{k}} \tag{2.9}$$

$$= k(k-1)\cdots(k-d+1)\frac{(n-d)!}{n!} \tag{2.10}$$

Thus the entire symmetrization polynomial Q is a degree-d polynomial in k. □

2.2 Polynomials on verifying Boolean matrix product

In this section, we present the partial results that we obtained from studying the potentials of the polynomial method on verifying Boolean matrix products. In BMPV^A problem, even though the matrix A is considered to be part of the problem specification, instead of the problem input, we observe that some matrices are easier to solve and some of them are very hard. So one motivation of this section is to find matrix A such that the BMPV^A problem is not trivial to solve, while still possessing enough internal symmetry structure so that we can apply the polynomial method.

We start by considering the input A as the adjacency matrix of various graphs. One graph that is particularly interesting is the Johnson graph.

2.2.1 Johnson Graph

Definition 2.2.1. The Johnson graph $J(n, k)$ has $\binom{n}{k}$ vertices, each of which has degree $k(n-k)$, so there are $\frac{k(n-k)}{2}\binom{n}{k}$ edges.

Given the graph $J(n, k)$, we know that its adjacency matrix is of dimension $m \times m$, where $m = \binom{n}{k}$.

Consider the BvPV^A problem: $Ax = y$, where matrix A is known and x, y are input vectors of length m. If we take A as the adjacency matrix of a Johnson graph, then pictorially, we are essentially computing: for each x_i, OR across the x values of all neighbors of i, and compare the disjunction with y_i. As an example, this is $J(5, 2)$:

We have:

$$\begin{aligned}
&(y_{12} = x_{13} \vee x_{14} \vee x_{15} \vee x_{23} \vee x_{24} \vee x_{25}) \\
\wedge\ &(y_{13} = x_{12} \vee x_{14} \vee x_{15} \vee x_{23} \vee x_{34} \vee x_{35}) \\
&\vdots \\
\wedge\ &(y_{45} = x_{14} \vee x_{24} \vee x_{34} \vee x_{15} \vee x_{25} \vee x_{35})
\end{aligned}$$

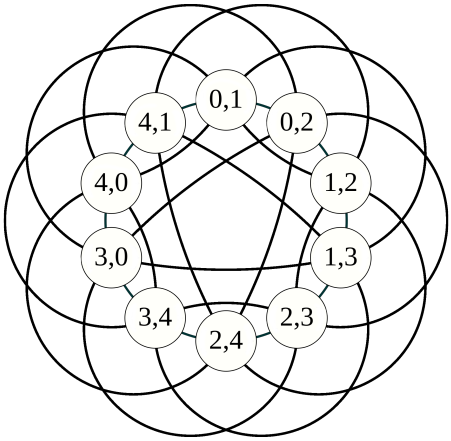

Figure 2.1: The Johnson(5,2) graph, which is the complement to the Petersen graph.

There are $m = \binom{n}{k}$ number of equality tests, and each row has $k(n-k)$ terms in the disjunction.

In order to use polynomial method, we proceed as follows: let $f : \{0,1\}^{2m} \to \{0,1\}$ on inputs $x \in \{0,1\}^m$, $y \in \{0,1\}^m$ be

- f(x,y) = 1, if $Ax = y$
- f(x,y) = 0, if $Ax \neq y$

Suppose there is a t-query algorithm for f, then there exists a polynomial P of degree $d = 2t$, such that

$$|P(x,y) - f(x,y)| \leq \frac{1}{3}$$

Then, all we need to do is to find a symmetric polynomial Q of the same degree as that of P, and find a lower bound for d.

Suppose we are trying to symmetrize a polynomial of degree 2 containing the terms $x_{12}x_{34}, \ldots,$ then swapping $x_{12}x_{34}$ will likely produce terms $(x_{12} + x_{34})^2 \ldots$ in the resulting polynomial.

Trying to exploit the symmetry in a Johnson graph, I tried to re-label the graph. In our case, swapping node i with j means, at the same time:

- swapping the i^{th} row of A with the j^{th} row, and the i^{th} col with the j^{th} col.
- swapping the i^{th} entry of x with the j^{th} entry.
- swapping the i^{th} entry of y with the j^{th} entry.

Consider A as known. In other words, $m = \binom{n}{k}$ is fixed if we are going to consider the entire matrix as the adj matrix of a graph. We don't really have much freedom on choosing n, k. In fact, if the matrix A is given, any parameters that specify the graph will a some constant, not some variable. But it would still be interesting to find some graphs that are more general.

2.2.2 Generalized Kneser graph

Definition 2.2.2. A generalized Kneser graph $KG(n,k,s)$ is the graph whose vertices are the k-subset of an n-element set, two vertices being adjacent when they share s or fewer items.

Thus, for $k = 2$, the Johnson graph $J(n, 2)$ is the complement of the generalized Kneser graph $KG(n, 2, 0)$. In $KG(n, k, 0)$, two vertices are adjacent if they represent disjoint sets.

For example, this is $KG(5, 2, 0)$, which is isomorphic to Petersen graph:

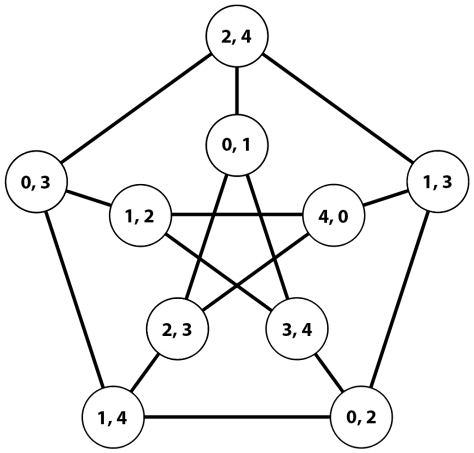

Figure 2.2: The generalized Kneser(5, 2, 0) graph, which is isomorphic to the Petersen graph.

Therefore, for Petersen graph: We have $KG(5, 2, 0)$:

$$\begin{aligned} & (y_{12} = x_{34} \vee x_{35} \vee x_{45}) \\ \wedge\ & (y_{13} = x_{24} \vee x_{25} \vee x_{45}) \\ & \vdots \\ \wedge\ & (y_{45} = x_{12} \vee x_{13} \vee x_{23}) \end{aligned}$$

There are still $m = \binom{n}{k} = 10$ number of equality tests, but each row has $\binom{n-k}{k} = 3$ terms in the disjunction now.

2.2.3 Symmetrization

1. Since relabeling doesn't change the correctness of $Ax = y$, we can consider a permutation $\sigma_{i,j}$ to be: swapping the i^{th}, j^{th} rows and cols of A, swapping the i^{th}, j^{th} entries of x, and swapping the i^{th}, j^{th} entries of y, as shown earlier.
2. One can also apply symmetrization that does not change A (i.e. one way to swap the entries in x, y while preserving the correctness of $Ax = y$) We proceed by taking a closer look at the boolean formula for BvPV^A on $KG(5, 2, 0)$ (a.k.a. the Petersen graph), and we get:

$$\begin{aligned} & (y = a \vee b \vee c) \\ \Leftrightarrow\ & (y \wedge a) \vee (y \wedge b) \vee (y \wedge c) \vee (\bar{y} \wedge \bar{a} \wedge \bar{b} \wedge \bar{c}) \end{aligned}$$

Furthermore, in our case, the following:

$$(y_{12} = x_{34} \vee x_{35} \vee x_{45})$$

becomes

$$\begin{aligned}
&(x_{34} \wedge y_{12} \wedge y_{15} \wedge x_{25}) \\
\vee\ &(x_{35} \wedge y_{12} \wedge y_{14} \wedge x_{24}) \\
\vee\ &(x_{45} \wedge y_{12} \wedge y_{13} \wedge x_{23}) \\
\vee\ &(\bar{y}_{12} \wedge \bar{x}_{34} \wedge \bar{x}_{35} \wedge \bar{x}_{45})
\end{aligned}$$

Notice that the top three rows cannot be true when the forth row is true. Also, there are exactly $2\binom{n}{k}$ distinct sets of the form: $(\cdot \wedge \cdot \wedge \cdot \wedge \cdot)$.

In simpler words, we are verifying that for each entry of y,

- If $y_i = 0$, then we want to verify that its three neighbors x_a, x_b, x_c are all zeroes.
- If $y_i = 1$, then we want to verify that at least one of its three neighbors x_a, x_b, x_c is one.

Or alternatively, first negate the bits of y, denote it y'. Then

- If $y_i = 0$, then we want to verify that its three neighbors x_a, x_b, x_c are all zeroes.
- If $y'_i = 0$, then we want to fail the case that its three neighbors x_a, x_b, x_c are all zeroes.

Therefore, it looks like, for each y_i, we only need to look at its three neighbors in x. And based on the observation above, we only verify/fail the case where y_i and three x_i's are all zeroes. So any permutation within the four items wouldn't change the correctness.

Chapter 3

Upper Bound and the Span Program

In this Chapter, we will describe a technique of constructing Quantum query algorithms that stems from the Adversary method of proving lower bound. We will show that the span program, a linear-algebraic model of computation, is closely related to the dual adversary semi-definite program. We will thereafter show its potential applications on verifying Boolean matrix products.

3.1 Span program

It might first seem surprising that the Adversary bound (a lower bound technique) is eventually used to formulate query algorithms, and hence transforming into an upper bound proof. We will describe this application of the adversary bound and its dual to help define the span program. The primary technique we use is to define high level span programs to reproduce and improve the best quantum query algorithms. Span programs have been shown to be as powerful as quantum query algorithms [Rei09]. It is therefore possible that, for any existing quantum query algorithm, we construct its span program counterpart that runs as efficiently. The significant difference of span programs from other models, on the other hand, provides us a new way of developing and improving quantum algorithms. Since the span program is in nature a linear-algebraic model of computation, it is natural to look at its applications on matrix multiplication and matrix product verification.

3.1.1 Basic adversary bound

Let's first take a look at a simple version of the adversary bound, defined by Ambainis [Amb02]. Consider a Boolean function f. The idea behind the basic adversary bound is to find a set Y of YES-instances (i.e. from $f^{-1}(1)$), and a set Z of NO-instances (i.e. from $f^{-1}(0)$) that are very hard to distinguish. More formally:

Theorem 3.1.1 ([HLS07])**.** *Let $f : [\ell]^n \supseteq \mathcal{D} \to \{0,1\}$ be a Boolean function, and $Y \subseteq f^{-1}(1)$ and $Z \subseteq f^{-1}(0)$, and a binary relation $R \subseteq Y \times Z$ such that:*

- *$\forall y \in Y$ there exists at least m different $z \in Z$ such that $(y,z) \in R$*
- *$\forall z \in Z$ there exists at least m' different $y \in Y$ such that $(y,z) \in R$*
- *$\forall y \in Y$ and $i \in [n]$, let $\ell_{y,i}$ be the number of $z \in Z$ such that $(y,z) \in R$ and $y_i \neq z_i$*

- $\forall z \in Z$ *and* $i \in [n]$*, let* $\ell_{z,i}$ *be the number of* $y \in Y$ *such that* $(y, z) \in R$ *and* $y_i \neq z_i$

Then any quantum algorithms that evaluate function f *requires* $\Omega(\sqrt{\frac{mm'}{\ell\ell'}})$ *queries, where* $\ell = \max \ell_{y,i}$ *and* $\ell' = \max \ell_{z,i}$.

Example 3.1.2 (OR of ANDs)**.** *Given a AND-OR formula of* n^2 *variables* $(z_{i,j})_{i,j\in[n]}$ *in this form:*

$$(z_{1,1} \wedge \cdots \wedge z_{1,n}) \vee (z_{2,1} \wedge \cdots \wedge z_{2,n}) \vee \cdots \vee (z_{n,1} \wedge \cdots \wedge z_{n,n}) \tag{3.1}$$

Then the formula requires $\Omega(n)$ *queries to evaluate.*

Proof. Let Y be the set of inputs with one disjunct evaluating to 1, while all other disjuncts being 0 due to exactly one variable equal to 0. Let Z be the set of inputs with all disjunct containing exactly one variable equal to 0. By definition, $f(Y) = 1$, and $f(Z) = 0$. We denote that $y \in Y$ and $z \in Z$ being in the relation, if y and z differ in exactly one position.

We get $m = m' = n$ and $\ell = \ell' = 1$. We have $m = n$ because for each $y \in Y$, we can flip any bit in the disjuncts that evaluates to 1 to find the $z \in Z$ that is in relation, similarly for $m' = n$. By theorem above, we have the $\Omega(n)$ query complexity. □

3.1.2 General adversary bound

The basic adversary bound does not always gives the optimal lower bound. Thus a generalization (the weighted adversary bound) was proposed, based on the important observation that the distinguishability of the input pairs is different.

Definition 3.1.3. Let $(\Gamma_{y,z})_{y\in Y,z\in Z}$ be a (real, symmetric) weight matrix for all pairs of input $y \in Y$ and $z \in Z$. We let $\Gamma_{y,z} = 0$ when $f(y) = f(z)$.

Analogous to the $\sqrt{mm'}$ term in the basic adversary bound, we denote $||\Gamma||$ as the maximum of the absolute values of the eigenvalues of Γ. We also define Γ_i as the matrix Γ with those entries where $y_i = z_i$ zeroed out.

Definition 3.1.4. Let $f : [\ell]^n \supseteq \mathcal{D} \to [k]$ be a function. Then we define the adversary bound as:

$$ADV^{\pm}(f) = \max_{\Gamma} \frac{||\Gamma||}{\max_{i\in[n]} ||\Gamma_i||} \tag{3.2}$$

Theorem 3.1.5 ([HLS07], [Rei09])**.** *The quantum query complexity of a function* $f : [\ell]^n \supseteq \mathcal{D} \to [k]$ *is* $\Theta(ADV^{\pm}(f))$*. Hence the weighted adversary method gives an upper and lower bound for the quantum query complexity of the function* f*.*

3.1.3 Span program

The span program is a computational model of specifying Boolean functions [KW93]. This model has been discovered to have promising applications on various problems. Many span-program-based quantum algorithms have recently been developed, for example for formulae evaluation [Rei11a], matrix rank [Bel11], subgraph-finding [Bel12], k-distinctness problem [BL11], and st-connectivity and claw detection [BR09]. We are interested in showing that span programs are useful for other problems in quantum computation.

From the theorem above, we can therefore develop quantum algorithms based on the duality of the adversary bound.

Definition 3.1.6 (Span program [KW93])**.** Given a decision function $f : \{0,1\}^n \to \{0,1\}$. We define a span program $\mathcal{P} = \{|\tau\rangle, \{(V_{i,0}, V_{i,1})\}_{i\in[n]}, V_{free}\}$ as thus:

- A non-zero target vector $|\tau\rangle \in \boldsymbol{R}^d$
- n pairs $(V_{i,0}, V_{i,1})$ of sets of input vectors in $\boldsymbol{R}^d$, for $i \in [n]$
- A set V_{free} of free input vectors in $\boldsymbol{R}^d$

With this definition of a span program $\mathcal{P}$, we can then evaluate $\mathcal{P}$ on any given input string $z \in \{0,1\}^n$:

- If $z_i = 0$, mark vectors $v \in V_{i,0}$ as *available* and $v \in V_{i,1}$ as *unavailable*
- If $z_i = 1$, mark vectors $v \in V_{i,1}$ as *available* and $v \in V_{i,0}$ as *unavailable*

The span program evaluates to 1 *if and only if* the target vector $|\tau\rangle$ lies within the span of the set of *available* or free vectors, i.e. $|\tau\rangle \in \mathcal{C}(V(x) \cup V_{free})$.

Definition 3.1.7 (Witness size [RS08])**.** To measure the complexity of a span program, we use the following notion of witness size:

- If $\mathcal{P}$ evaluates to 1 on input y, the *positive witness* for y is a pair of vectors $(|w_y\rangle, |w_{free}\rangle)$ such that $V(x)|w_y\rangle + V_{free}|w_{free}\rangle = |\tau\rangle$. The *size* of the witness $|w_y\rangle$ is defined as $||\,|w_y\rangle\,||^2$, which is the squared length of the positive witness.
- If $\mathcal{P}$ evaluates to 0 on input z, the *negative witness* for z is any vector $|w'_z\rangle \perp \mathcal{C}(V(z))$ such that $\langle w'_z|\tau\rangle = 1$. The size of the witness $|w'_z\rangle$ is defined as $||V^\dagger|w'_z\rangle||^2$, which is the sum of the inner products of the negative witness with all *unavailable* input vectors.

If we denote $size(w_x)$ to be the minimal size across all witness for input x, then the witness size of the the span program $\mathcal{P}$ on domain $\mathcal{D} \subseteq \{0,1\}^n$ is given by:

$$wsize(\mathcal{P}) = \sqrt{wsize_1(\mathcal{P})wsize_0(\mathcal{P})} \tag{3.3}$$

where

$$wsize_1(\mathcal{P}) = \max_{y\in f^{-1}(1)} size(w_y) \tag{3.4}$$

$$wsize_0(\mathcal{P}) = \max_{z\in f^{-1}(0)} size(w_z) \tag{3.5}$$

Then we have the following theorem that shows the close relation between span programs and quantum query algorithms:

Theorem 3.1.8 ([Rei09], [Rei11b])**.** *For any boolean function $f : \{0,1\}^n \supseteq \mathcal{D} \to \{0,1\}$, if P is a span program evaluating f, then there exists a quantum algorithm that evaluates f with two-sided bounded error using $O(wsize(\mathcal{P}))$ queries.*

3.2 Span program on verifying Boolean matrix product

In this section, we present our partial results on the span program for the Boolean matrix product verification problem. Inspired by the span program for $st-$connectivity problem [BR09]. We want to construct a graph, based on the input $n \times n$ Boolean matrices A, B, and C using the following rule:

- Create two dummy node, labeled as source s and target t.
- Create two sets of n^2 nodes, and label them $a_{i,j}$ or $c_{i,j}$, for all $i, j \in [n]$.
- If $A[i, j] = 1$, draw an edge from the source s to $a_{i,j}$.
- If $B[i, j] = 1$, draw edges from $a_{k,i}$ to $c_{k,j}$, for all $k \in [n]$.
- If $C[i, j] = 0$, draw an edge from the source $c_{i,j}$ to the target t.

We have therefore constructed the graph, based on the input matrices A, B, and C. Then, to solve the BMPV problem using this graph, we have two goals:

- Verify that no nodes $c_{i,j}$ reachable from the sources are connected to the target t.
- Verify that all nodes $c_{i,j}$ not reachable from the sources are connected to the target t.

Let's take a look at a concrete example. Given three Boolean matrices,

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, C = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

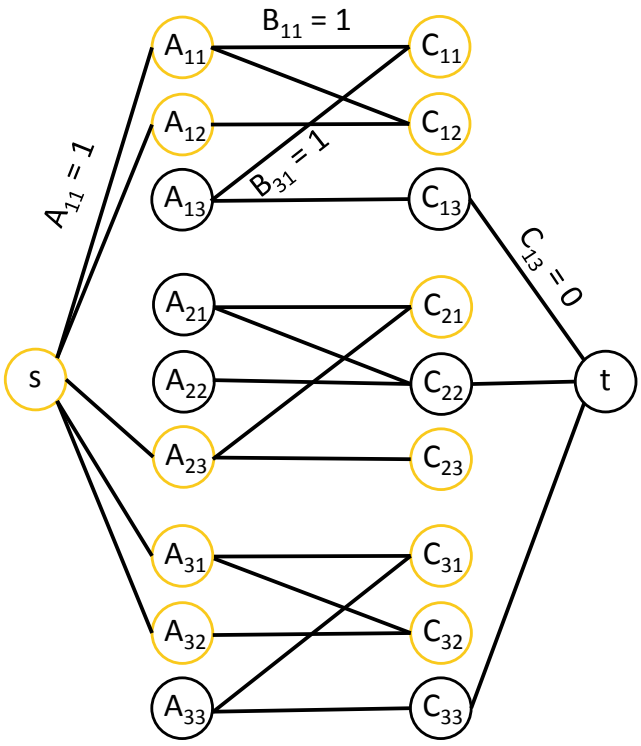

Figure 3.1: The graph constructed for the span program for Boolean matrix product verification. All nodes reachable from the source s are labelled in yellow, and all non-reachable in black.

Then we can define more formally the span program, given input matrices A, B, and C as follows: The span program $\mathcal{P}$ uses the vector space with orthonormal basis

$$\{|s\rangle, |t\rangle\} \cup \{|a_{i,j}\rangle : i, j \in [n]\} \cup \{|c_{i,j}\rangle : i, j \in [n]\} \tag{3.6}$$

The target vector is $|\tau\rangle = |s\rangle - |t\rangle$. There are the following *available* input vectors for the span program $\mathcal{P}$:

- $|s\rangle - |a_{i,j}\rangle$ for all $A[i, j] = 1$.
- $|a_{k,i}\rangle - |c_{k,j}\rangle$ for all $B[i, j] = 1, \forall k \in [n]$.
- $|c_{i,j}\rangle - |t\rangle$ for all $C[i, j] = 0$.

Notice that, by definition, the span program will successfully verify all the zero entries of the matrix C, because if any node $c_{i,j}$ that is reachable from the source s is connected to target t, say we have a path $s \to a_{1,2} \to c_{1,1} \to t$, then the target vector $|\tau\rangle$ will fall within the span of the available input vectors

$$|\tau\rangle = |s\rangle - |t\rangle = (|s\rangle - |a_{1,2}\rangle) + (|a_{1,2}\rangle - |c_{1,1}\rangle) + (|c_{1,1}\rangle - |t\rangle) \tag{3.7}$$

Chapter 4

Future work

- In the lower bound analysis, there are many other family of graphs that we can further investigate. Since the difficulty of the BMPV problem depends on the specification matrix A, the goal is thus to find a graph whose adjacency matrix is not too easy to solve yet still contains enough symmetry.
- To find the symmetrization polynomial $Q(i)$, one possible direction is to look for a parameter i that can specify some permutations that preserve the correctness.
- It might also be worth to look at the output-sensitive lower bound for the BMVP problem. In particular, what is the lower bound of the problem when there is only one wrong entry in matrix C.
- In the span program we developed, the 1-entries of the matrix C are still yet to be verified. Apparently, if we change of the rule of connecting $c_{i,j}$ with t from $C[i,j] = 0$ to $C[i,j] = 1$, then we can verify the one-entries. The potential research direction is then to combine the two span programs to one that verifies all entries of the matrix C.

Bibliography

[ABIO13] A. Ambainis, K. Balodis, J. Iraids, and R. Ozols. Parameterized quantum query complexity of graph collision. In *Proceedings of the Workshop on Quantum and Classical Complexity*, pages 5–16, 2013. 1.2.1

[Amb02] A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. 3.1.1

[Bel11] A. Belovs. Span-program-based quantum algorithm for the rank problem. Technical report, arXiv:1103.0842, 2011. 3.1.3

[Bel12] A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proc. of 44th ACM STOC*, pages 77–84, 2012. 3.1.3

[BL11] A. Belovs and T. Lee. Quantum algorithm for k-distinctness with prior knowledge on the input. Technical report, arXiv:1108.3022, 2011. 3.1.3

[BR09] A. Belovs and B. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proc. 50th IEEE FOCS*, pages 544–551, 2009. 3.1.3, 3.2

[BS06] H. Buhrman and R. Spalek. Quantum verification of matrix products. In *SODA 06: Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 880–889, 2006. 1.2.2

[CKK12] A. Childs, S. Kimmel, and R. Kothari. The quantum query complexity of read-many formulas. In *Algorithms - ESA 2012, Lecture Notes in Computer Science*, volume 7501, pages 337–348. Springer, 2012. 1.2.2

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 1987. 1.2.1

[Fre79] R. Freivalds. Fast probabilistic algorithms. In *Mathematical Foundations of Computer Science, Lecture Notes in Computer Science*, volume 74, pages 57–69. Springer, 1979. 1.2.1

[HLS07] P. Hoyer, T. Lee, and R. Spalek. Negative weights make adversaries stronger. In *Proceedings of the 39th ACM Symposium on the Theory of Computing*, pages 526–535, 2007. 3.1.1, 3.1.5

[KW93] M. Karchmer and A. Wigderson. On span programs. In *Proc. 8th IEEE Symp. Structure in Complexity Theory*, pages 102–111, 1993. 3.1.3, 3.1.6

[LG14] F. Le Gall. Improved quantum algorithm for triangle finding via combinatorial ar-

guments. In *Proc. 55th Annual Symp. Foun- dations of Computer Science*, pages 216–225, 2014. 1.2.1

[LMS13] T. Lee, F. Magniez, and M. Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1486–1502, 2013. 1.2.1

[Mon13] A. Montanaro. A composition theorem for decision tree complexity. *arXiv preprint arXiv:1302.4207*, 2013. 1.1.3

[MSS07] F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007. 1.2.1

[Rei09] B. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *Proc. 50th IEEE FOCS*, pages 544–551, 2009. 3.1, 3.1.5, 3.1.8

[Rei11a] B. Reichardt. Faster quantum algorithm for evaluating game trees. In *Proc. 22nd ACM- SIAM Symp. on Discrete Algorithms (SODA)*, pages 546–559, 2011. 3.1.3

[Rei11b] B. Reichardt. Reflections for quantum query algorithms. In *Proc. 22nd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 560–569, 2011. 3.1.8

[RS08] B. Reichardt and R. Spalek. Span-program-based quantum algorithm for evaluating formulas. In *Proc. 40th ACM STOC*, pages 103–112, 2008. 3.1.7

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. 1.2.1

[Wil12] V. Williams. Multiplying matrices faster than coppersmith-winograd. *Symposium on the Theory of Computing*, 2012. 1.2.1