Improving Quantum Query Complexity of Boolean Matrix Multiplication Using Graph Collision

Stacey Jeffery[*1,2], Robin Kothari[†1,2], and Frédéric Magniez[‡3]

[1]David R. Cheriton School of Computer Science, University of Waterloo, Canada
[2]Institute for Quantum Computing, University of Waterloo, Canada
[3]LIAFA, Univ. Paris Diderot, CNRS; Paris, France

Abstract

The quantum query complexity of Boolean matrix multiplication is typically studied as a function of the matrix dimension, n, as well as the number of 1s in the output, ℓ. We prove an upper bound of $\widetilde{O}(n\sqrt{\ell})$ for all values of ℓ. This is an improvement over previous algorithms for all values of ℓ. On the other hand, we show that for any $\varepsilon < 1$ and any $\ell \leq \varepsilon n^2$, there is an $\Omega(n\sqrt{\ell})$ lower bound for this problem, showing that our algorithm is essentially tight.

We first reduce Boolean matrix multiplication to several instances of graph collision. We then provide an algorithm that takes advantage of the fact that the underlying graph in all of our instances is very dense to find all graph collisions efficiently.

1 Introduction

Quantum query complexity has been of fundamental interest since the inception of the field of quantum algorithms [BBBV97, Gro96, Sho97]. The quantum query complexity of Boolean matrix multiplication was first studied by Buhrman and Špalek [BŠ06]. In the Boolean matrix multiplication problem, we want to multiply two $n \times n$ matrices A and B over the Boolean semiring, which consists of the set $\{0, 1\}$ with logical OR ($\vee$) as the addition operation and logical AND ($\wedge$) as the multiplication operation.

For this problem it is standard to consider an additional parameter in the complexity: The number of 1s in the product $C := AB$, which we denote by ℓ. We study the query complexity as a function of both n and ℓ, and obtain improvements for all values of ℓ.

The problem of Boolean matrix multiplication is of fundamental interest, in part due to its relationship to a variety of graph problems, such as the triangle finding problem and the all-pairs shortest path problem.

---

[*]sjeffery@uwaterloo.ca

[†]rkothari@cs.uwaterloo.ca

[‡]frederic.magniez@univ-paris-diderot.fr

Classically, it was shown by Vassilevska Williams and Williams that "practical advances in triangle detection would imply practical [Boolean matrix multiplication] algorithms" [VW10]. The previous best quantum algorithm for Boolean matrix multiplication, by Le Gall, is based on a subroutine for finding triangles in graphs *with a known tripartition* [Gal12], already suggesting that the relationship between Boolean matrix multiplication and triangle finding might be more complex for quantum query complexity. We give further evidence for this by bypassing the triangle finding subroutine entirely.

Despite its fundamental importance, much has remained unknown about the quantum query complexity of Boolean matrix multiplication and its relationship with other query problems in the quantum regime. Even for the simpler decision problem of Boolean matrix product verification, where we are given oracle access to three $n \times n$ Boolean matrices, A, B and C, and must decide whether or not $AB = C$, the quantum query complexity is unknown. The best upper bound is $O(n^{3/2})$ [BŠ06], whereas the lower bound was recently improved from the trivial $\Omega(n)$ to $\Omega(n^{1.055})$ by Childs, Kimmel, and Kothari [CKK11].

A better understanding of these problems may lead to an improved understanding of quantum query complexity in general. We contribute to this by closing the gap (up to logarithmic factors) between the best known upper and lower bounds for Boolean matrix multiplication for all $\ell \leq \varepsilon n^2$ for any constant $\varepsilon < 1$.

Previous Work. We are interested in the query complexity of Boolean matrix multiplication, where we count the number of accesses (or *queries*) to the input matrices A and B. Buhrman and Špalek [BŠ06, Section 6.2] describe how to perform Boolean matrix multiplication using $\widetilde{O}(n^{3/2}\sqrt{\ell})$ queries, by simply quantum searching for a pair $(i, j) \in [n] \times [n]$ such that there is some $k \in [n]$ for which $A[i, k] = B[k, j] = 1$, where $[n] = \{1, \ldots, n\}$. By means of a classical reduction relating Boolean matrix multiplication and triangle finding, Vassilevska Williams and Williams [VW10] were able to combine the quantum triangle finding algorithm of Magniez, Santha and Szegedy [MSS07] with a classical strategy of Lingas [Lin09] to get a quantum algorithm for Boolean matrix multiplication with query complexity $\widetilde{O}(\min\{n^{1.3}\ell^{17/30}, n^2 + n^{13/15}\ell^{47/60}\})$.

Recently, Le Gall [Gal12] improved on their work by noticing that the triangle finding needed for Boolean matrix multiplication involves a tripartite graph with a known tripartition. He then recast the known quantum triangle finding algorithm of [MSS07] for this special case and improved the query complexity of Boolean matrix multiplication. He then further improved the algorithm for large ℓ by adapting the strategy of Lingas to the quantum setting.

Our Contributions. Since previous quantum algorithms for Boolean matrix multiplication are based on a triangle finding subroutine, a natural question to ask is whether triangle finding is a bottleneck for this problem. We show that this is not the case by bypassing the triangle finding problem completely to obtain a nearly tight result for Boolean matrix multiplication.

A key ingredient of the best known quantum algorithm for triangle finding is an efficient algorithm for the graph collision problem. Our main contribution is to build an algorithm directly

2

on graph collision instead, bypassing the use of a triangle finding algorithm. Surprisingly, we do not use the graph collision algorithm that is used as a subroutine in the best known quantum algorithm for triangle finding. That algorithm is based on Ambainis' quantum walk for the element distinctness problem [Amb04]. Our algorithm, on the other hand, does not have any quantum walks.

There are two main ideas we would like to impress upon the reader. First, we can reduce the Boolean matrix multiplication problem to several instances of the graph collision problem. Second, the instances of graph collision that arise depend on ℓ; in particular, they have at most ℓ non-edges. Furthermore, we need to find all graph collisions, not just one. We provide an algorithm to find a graph collision in query complexity $\widetilde{O}(\sqrt{\ell} + \sqrt{n})$, or to find all graph collisions in time $\widetilde{O}(\sqrt{\ell} + \sqrt{n\lambda})$, where $\lambda \geq 1$ is the number of graph collisions. Combining these ideas yields the aforementioned $\widetilde{O}(n\sqrt{\ell})$ upper bound.

A lower bound of $\Omega(n\sqrt{\ell})$ for all values of $\ell \leq \varepsilon n^2$ for any constant $\varepsilon < 1$ follows from a simple reduction to ℓ-THRESHOLD, which we state formally in Theorem 4.2.

This paper is organized as follows. After presenting some preliminaries in Section 2, we describe in Section 3, the graph collision problem, its relationship to Boolean matrix multiplication, and a subroutine for finding all graph collisions when there are at most ℓ non-edges. In Section 4, we apply our graph collision subroutine to get the stated upper bound for Boolean matrix multiplication, and then describe a tight lower bound that applies to all values of $\ell \leq \varepsilon n^2$ for $\varepsilon < 1$.

2   Preliminaries

2.1   Quantum Query Framework

For a more thorough introduction to the quantum query model, see [BBC+01]. For Boolean matrix multiplication, we assume access to two query operators that act as follows on a Hilbert space spanned by $\{|i, j, b\rangle : i, j \in [n], b \in \{0, 1\}\}$:

$$\mathcal{O}_A : |i, j, b\rangle \mapsto |i, j, b \oplus A[i, j]\rangle \quad \mathcal{O}_B : |i, j, b\rangle \mapsto |i, j, b \oplus B[i, j]\rangle$$

In the quantum query model, we count the uses of $\mathcal{O}_A$ and $\mathcal{O}_B$, and ignore the cost of implementing other unitaries which are independent of A and B. We call $\mathcal{O}_A$ and $\mathcal{O}_B$ the *oracles*, and each access a *query*. The query complexity of an algorithm is the maximum number of oracle accesses used by the algorithm, taken over all inputs.

We denote a problem P by a map $\mathcal{X} \to 2^{\mathcal{Y}}$, where $P(x) \subseteq \mathcal{Y}$ denotes the set of valid outputs on input x. We say a quantum algorithm A solves a problem P: $\mathcal{X} \to 2^{\mathcal{Y}}$ with bounded error $\delta(|x|)$ if for all $x \in \mathcal{X}$, $\Pr[A(x) \in P(x)] \geq 1 - \delta(|x|)$, where $|x|$ is the size of the input. The quantum query complexity of P is the minimum query complexity of any quantum algorithm that solves P with bounded error $\delta(|x|) \leq 1/3$.

We will use the phrase *with high probability* to mean probability at least $1 - \frac{1}{\text{poly}}$ for some super-linear polynomial. We ensure that all our subroutines succeed with high probability, to achieve

3

a bounded-error algorithm at the end. To achieve such high probability, we will necessarily incur polylog factors. We will use the notation $\widetilde{O}$ to indicate that we are suppressing polylog factors. More precisely, $f(n) \in \widetilde{O}(g(n))$ means $f(n) \in O(g(n) \log^k n)$ for some constant k.

Boolean Matrices. We let $\mathbb{B}$ denote the *Boolean semiring*, which is the set $\{0,1\}$ under the operations $\vee, \wedge$. The problem we will be considering is formally defined as the following:

> BOOLEAN MATRIX MULTIPLICATION
> *Oracle Input:* Two Boolean matrices $A, B \in \mathbb{B}^{n \times n}$.
> *Output:* $C \in \mathbb{B}^{n \times n}$ such that $C = AB$.

In $\mathbb{B}^{n \times n}$, we say that $C = AB$ if for all $i, j \in [n]$, $C[i,j] = \bigvee_{k=1}^{n} A[i,k] \wedge B[k,j]$. We will use the notation $A + B$ to denote the entry-wise $\vee$ of two Boolean matrices.

2.2 Quantum Search Algorithms

In this section we examine some well-known variations of the search problem that we require. The reader familiar with quantum search algorithms may skip to Section 3.

Any search problem can be recast as searching for a marked element among a given collection, U. In order to formalize this, let $f : U \to \{0,1\}$ be a function whose purpose is to identify marked elements. An element is marked if and only if $f(x) = 1$. Define $t_f = |f^{-1}(1)|$. In Grover's search algorithm, the algorithm can directly access f, and the overall complexity can be stated as the number of queries to f. In the following $t \geq 1$ is an integer parameter.

Theorem 2.1 ([Gro96]). *There is a quantum algorithm,* GroverSearch(t), *with query complexity* $\widetilde{O}(\sqrt{|U|/t})$ *to* f, *such that, if* $t/2 \leq t_f \leq t$, *then* GroverSearch(t) *finds a marked element with probability at least* $1 - 1/\text{poly}(|U|)$.
Moreover, if $t_f = 0$, *then* GroverSearch(t) *declares with probability* 1 *that there is no marked element.*

There are several ways to generalize the above statement when no approximation of t is known. Most of the generalizations in the literature are stated in terms of expected query complexity, such as in [BBHT98]. Nonetheless, one can derive from [BBHT98, Lemma 2] an algorithm in terms of worst case complexity, when only a lower bound t on t_f is known. The algorithm consists of T iterations of one step of the original Grover algorithm where T is chosen uniformly at random from $[0, \sqrt{|U|/t}]$. This procedure is iterated $O(\log |U|)$ times in order to get bounded error $1/\text{poly}(|U|)$.

Corollary 2.2. *There is a quantum algorithm* Search(t) *with query complexity* $\widetilde{O}(\sqrt{|U|/t})$ *to* f, *such that, if* $t_f \geq t$, *then* Search(t) *finds a marked element with probability at least* $1 - 1/\text{poly}(|U|)$.
Moreover, if $t_f = 0$, *then* Search(t) *declares with probability* 1 *that there is no marked element.*

One consequence of Corollary 2.2 is that we can always apply Search(t) with $t = 1$, when no lower bound on t_f is given. In that case, we simply refer to the resulting algorithm as Search. Its query complexity to f is then $\widetilde{O}(\sqrt{|U|})$.

Another simple generalization is for finding all marked elements. This generalization is stated in the literature in various ways for expected and worst case complexity. For the sake of clarity we explicitly describe one version of this procedure using GroverSearch as a subroutine. This version is robust in the sense that it works even when the number of marked elements may decrease arbitrarily. This may occur, for example, when the finding of one marked element may cause several others to become unmarked. This situation will naturally occur in the context of Boolean matrix multiplication. Then the complexity will only depend on the number of elements that are actually in the output, as opposed to the number of elements that were marked at the beginning of the algorithm.

SearchAll

1. Let $t = |U|$, and $V = U$

2. While $t \geq 1$

   (a) Apply GroverSearch(t) to V

   (b) If a marked element x is found: Output x; Set $V \leftarrow V - \{x\}$ and $t \leftarrow t - 1$
       Else: $t \leftarrow t/2$

3. If no marked element has been found, declare 'no marked element'

Corollary 2.3. SearchAll *has query complexity* $\widetilde{O}(\sqrt{|U|(t_f + 1)})$ *to* f, *and finds all marked elements with probability at least* $1 - 1/\mathrm{poly}(|U|)$.
Moreover, if $t_f = 0$, *then* SearchAll *declares with probability* 1 *that there is no marked element.*

We end this section with an improvement of GroverSearch when we are looking for an optimal solution for some notion of maximization.

Theorem 2.4 ([DH96, DHHM06]). *Given a function* $g : U \to \mathbb{R}$, *there is a quantum algorithm,* FindMax(g), *with query complexity* $\widetilde{O}(\sqrt{|U|})$ *to* f, *such that* FindMax(g) *returns* $x \in f^{-1}(1)$ *such that* $g(x) = \max_{x' \in f^{-1}(1)} g(x')$ *with probability at least* $1 - 1/\mathrm{poly}(|U|)$. *Moreover, if* $t_f = 0$, *then* FindMax(g) *declares with probability* 1 *that there is no marked element.*

3 Graph Collision

In this section we describe the graph collision problem, and its relation to Boolean matrix multiplication. We then describe a method for solving the special case of graph collision in which we are interested.

3.1 Problem Description

Graph collision is the following problem. Let $G = (\mathcal{A}, \mathcal{B}, E)$ be a balanced bipartite graph on $2n$ vertices. We will suppose $\mathcal{A} = [n]$ and $\mathcal{B} = [n]$, though we note that in the bipartite graph, the vertex labelled by i in $\mathcal{A}$ is distinct from the vertex labelled by i in $\mathcal{B}$.

> GRAPH COLLISION(G)
> *Oracle Input:* A pair of Boolean functions $f_A : \mathcal{A} \to \{0, 1\}$ and $f_B : \mathcal{B} \to \{0, 1\}$.
> *Output:* $(i, j) \in \mathcal{A} \times \mathcal{B}$ such that $f_A(i) = f_B(j) = 1$ and $(i, j) \in E$, if such a pair exists, otherwise reject.

The graph collision problem was introduced by Magniez, Santha and Szegedy as a subproblem in triangle finding [MSS07]. The subroutine used to solve an instance of graph collision is based on Ambainis' quantum walk algorithm for element distinctness [Amb04], and has query complexity $O(n^{2/3})$. The same subroutine is used in the current best triangle finding algorithm of Belovs [Bel11]. However, the best known lower bound for this problem is $\Omega(\sqrt{n})$. It is an important open problem to close this gap.

To obtain our upper bound, we do not use the quantum walk algorithm for graph collision, but rather, a new algorithm that takes advantage of two special features of our problem. The first is that we always know an upper bound, ℓ, on the number of non-edges. When $\ell \leq n$, we can find a graph collision in $O(\sqrt{n})$ queries. The second salient feature of our problem is that we need to find all graph collisions.

3.2 Relation to Boolean Matrix Multiplication

Recall that the Boolean matrix product of A and B, can be viewed as the sum (entry-wise $\vee$) of n outer products: $C = \sum_{k=1}^{n} A[\cdot, k] B[k, \cdot]$, where $A[\cdot, k]$ denotes the kth column of A and $B[k, \cdot]$ denotes the kth row of B.

For a fixed k, if there exists some $i \in [n]$ and some $j \in [n]$ such that $A[i, k] = 1$ and $B[k, j] = 1$, then we know that $C[i, j] = 1$, and we say that k is a *witness* for (i, j). We are interested in finding all such pairs (i, j). For each index k, we could search for all pairs (i, j) with $A[i, k] = B[k, j] = 1$; however, this could be very inefficient, since a pair (i, j) may have up to n witnesses. Instead, we will keep a matrix $\widetilde{C}$ such that $\widetilde{C}[i, j] = 1$ if we have already found a one at position (i, j). Thus, we want to find a pair (i, j) such that $A[i, k] = B[k, j] = 1$ and $\widetilde{C}[i, j] = 0$. That is, we want to find a graph collision in the graph with bi-adjacency matrix $\overline{\widetilde{C}}$, the entry-wise complement of $\widetilde{C}$, and $f_A = A[\cdot, k]$, $f_B = B[k, \cdot]$.

This gives the following natural algorithm for Boolean matrix multiplication, whose details and full analysis can be found in Section 4.1:

> First, let $\widetilde{C} = 0$.
> Search for an index k such that the graph collision problem on k with $\overline{\widetilde{C}}$ as the underlying graph has a collision.

If no such k is found then we are done, and $\widetilde{C}$ is the product of A and B.

Otherwise, find all the graph collisions on the graph defined by $\overline{\overline{C}}$ with oracles $A[\cdot, k]$ and $B[k, \cdot]$ and record them in $\widetilde{C}$.

Eliminate this k from future searches and search for another index k again.

3.3 Algorithm for Graph Collision

When G is a complete bipartite graph, then the relation between $\mathcal{A}$ and $\mathcal{B}$ defined by G is trivial. In that case, there is a very simple algorithm to find a graph collision: Search for some $i \in [n]$ such that $f_A(i) = 1$. Then search for some $j \in [n]$ such that $f_B(j) = 1$. Then (i, j) is a graph collision pair. The query complexity of this is $O(\sqrt{n} + \sqrt{n})$. However, when G is not a complete bipartite graph, there is a nontrivial relation between $\mathcal{A}$ and $\mathcal{B}$. The best known algorithm solves this problem using a quantum walk.

In our case, we can take advantage of the fact that the graph we are working with always has at most ℓ non-edges — it is never more than distance ℓ from the complete bipartite graph, which we know is easy to deal with. We are therefore interested in the query complexity of finding a graph collision in some graph with m non-edges, which we denote $\mathfrak{GC}(n, m)$. In our case, ℓ will always be an upper bound on m.

For larger values of ℓ, we will also make use of the fact that for some k, we will have multiple graph collisions to find. We let $\mathfrak{GC}_{\text{all}}(n, m, \lambda)$ denote the query complexity of finding all graph collisions in a graph with m non-edges, where λ is the number of graph collisions. It is not necessary to know λ a priori.

Again we note that if G is a complete bipartite graph, then we can accomplish the task of finding all graph collisions using SearchAll to search for all marked elements on each of f_A and f_B, and output $f_A^{-1}(1) \times f_B^{-1}(1)$. Letting $t_A = \left| f_A^{-1}(1) \right|$ and $t_B = \left| f_B^{-1}(1) \right|$, so the total number of graph collision pairs is $\lambda = t_A t_B$, the query complexity of this method is $O(\sqrt{n t_A} + \sqrt{n t_B}) \in O(\sqrt{n \lambda})$. So if G is close to being a complete bipartite graph, we would like to argue that we can do nearly as well. This motivates the following algorithm.

AllGC$_G(f_A, f_B)$

Let m denote the number of non-edges in G. Let d_i be the degree of the ith vertex in $\mathcal{A}$, and let $c_i := n - d_i$. Let the vertices in $\mathcal{A}$ be arranged in decreasing order of degree, so that $d_1 \geq d_2 \geq \ldots \geq d_n$. We will say a vertex i in $\mathcal{A}$ (resp. $\mathcal{B}$) is *marked* if $f_A(i) = 1$ (resp. $f_B(i) = 1$).

1. Find the highest degree marked vertex in $\mathcal{A}$ using FindMax. Let r denote the index of this vertex. $\widetilde{O}(\sqrt{n})$

2. Case 1: If $c_r \leq \sqrt{m}$

   (a) Find all marked neighbors of r by SearchAll. Output any graph collisions found. $\widetilde{O}(\sqrt{n\lambda})$

   (b) Delete all unmarked neighbors of r. Read the values of all non-neighbors of r. $O(\sqrt{m})$

   (c) Let $\mathcal{A}'$ denote the subset of $\mathcal{A}$ consisting of all $i \in \mathcal{A}$ with a marked neighbour in $\mathcal{B}$. Find all marked vertices in $\mathcal{A}'$ by SearchAll. $\widetilde{O}(\sqrt{n\lambda})$

3. Case 2: If $c_r \geq \sqrt{m}$

   (a) Delete the first $r - 1$ vertices in $\mathcal{A}$ since they are unmarked.

   (b) Read the values of all remaining vertices in $\mathcal{A}$. $O(\sqrt{m})$

   (c) Let $\mathcal{B}'$ denote the subset of $\mathcal{B}$ consisting of all $j \in \mathcal{B}$ with a marked neighbour in $\mathcal{A}$. Find all marked vertices in $\mathcal{B}'$ by SearchAll. $\widetilde{O}(\sqrt{n\lambda})$

Theorem 3.1. *For all $\lambda \geq 1$, $\mathfrak{GC}_{\text{all}}(n, m, \lambda) \in \widetilde{O}(\sqrt{n\lambda} + \sqrt{m})$ and $\mathfrak{GC}(n, m) \in \widetilde{O}(\sqrt{n} + \sqrt{m})$.*

Proof. We will analyze the complexity of AllGC$_G(f_A, f_B)$ step by step.

Step 1 has query complexity $\widetilde{O}(\sqrt{n})$ by Theorem 2.4. Steps 2a, 2c and 3c have query complexity $\widetilde{O}(\sqrt{n\lambda})$ by Corollary 2.3. In Case 1, r has $c_r \leq \sqrt{m}$ non-neighbours, so we can certainly query them all in step 2b with $O(\sqrt{c_r}) \in O(\sqrt{m})$ queries.

Consider Case 2, when $c_r \geq \sqrt{m}$. We can ignore the first $r-1$ vertices, since they are unmarked. Since the remaining $n - r + 1$ vertices all have $c_i \geq c_r \geq \sqrt{m}$, and the total number of non-edges is m, we have $(n - r + 1) \times \sqrt{m} \leq m \Rightarrow (n - r + 1) \leq \sqrt{m}$. Thus, there are at most $\sqrt{m}$ remaining vertices and querying them all costs at most $O(\sqrt{m})$ queries.

The query complexity of this algorithm is therefore $\widetilde{O}(\sqrt{n\lambda} + \sqrt{m})$, and it outputs all graph collisions. To check if there is at least one graph collision, instead of finding them all, we can replace finding all marked vertices using SearchAll in steps 2a, 2c and 3c, with a procedure to check if there is any marked vertex, Search, and this only requires $\widetilde{O}(\sqrt{n})$ queries by Corollary 2.2, rather than $\widetilde{O}(\sqrt{n\lambda})$. $\square$

4 Boolean Matrix Multiplication

In this section we show how the graph collision algorithm from the previous section can be used to obtain an efficient algorithm for Boolean matrix multiplication and then prove a lower bound.

4.1 Algorithm

What follows is a more precise statement of the high level procedure described in Section 3.2.

$\mathsf{BMM}(A, B)$

1. Let $\widetilde{C} = 0$, $t = n$, and $V = [n]$

2. While $t \geq 1$

   (a) $\mathsf{GroverSearch}(t)$ for an index $k \in V$ such that the graph collision problem on k with $\overline{\overline{\widetilde{C}}}$ as the underlying graph has a collision.

   (b) If such a k is found

   > Compute AllGC on the graph defined by $\overline{\overline{\widetilde{C}}}$ with oracles $A[\cdot, k]$ and $B[k, \cdot]$ and record all output graph collisions in $\widetilde{C}$.
   >
   > Set $V \leftarrow V - \{k\}$ and $t \leftarrow t - 1$.

   (c) Else: $t \leftarrow t/2$

3. Output $\widetilde{C}$.

Theorem 4.1. *The query complexity of* BOOLEAN MATRIX MULTIPLICATION *is* $\widetilde{O}(n\sqrt{\ell})$.

Proof. We will analyze the complexity of the algorithm $\mathsf{BMM}(A, B)$. We begin by analyzing the cost of all the iterations in which we don't find a marked k. We have by Theorem 2.1 that $\mathsf{GroverSearch}(t)$ costs $\widetilde{O}(\sqrt{n/t})$ queries to a procedure that checks if there is a collision in the graph defined by $\overline{\overline{\widetilde{C}}}$ with respect to $A[\cdot, k]$ and $B[k, \cdot]$, each of which costs $\mathfrak{GC}(n, m_i)$, where $m_i \leq \ell$ is the number of 1s in $\widetilde{C}$ at the beginning of the ith iteration. The cost of these steps is at most the following:

$$\widetilde{O}\left(\sum_{i=0}^{\log n} \sqrt{\frac{n}{2^i}} \mathfrak{GC}(n, m_i)\right) \in \widetilde{O}\left(\sum_{i=0}^{\log n} \sqrt{\frac{n}{2^i}}(\sqrt{n} + \sqrt{m_i})\right)$$

$$\in \widetilde{O}\left((n + \sqrt{n\ell}) \sum_{i=0}^{\log n} \left(\frac{1}{\sqrt{2}}\right)^i\right) \in \widetilde{O}\left(n + \sqrt{n\ell}\right)$$

We now analyze the cost of all the iterations in which we do find a marked witness k. Let T be the number of witnesses found by BMM. Of course, T is a random variable that depends on which witnesses k are found, and in which order. We always have $T \leq \min\{n, \ell\}$.

9

Let $i_1, \ldots, i_T$ be the indices of rounds where we find a witness. Let t_j be the value of t in round j. Since there must be at least 1 marked element in the last round in which we find a marked element, we have $t_{i_T} \geq 1$. Since we find and eliminate at least 1 marked element in each round, we also have $t_{i_{(T-j-1)}} \geq t_{i_{(T-j)}} + 1$, which yields $t_{i_{(T-j)}} \geq j + 1 \Rightarrow t_{i_j} \geq T - j + 1$.

Let λ_j be the number of graph collisions found on the jth successful iteration, that is, the number of pairs witnessed by the jth witness, k_j, that have not been recorded in $\widetilde{C}$ *at the time we find* k_j. Then λ_j is also a random variable depending on which other witnesses k have been found already, but we always have $\sum_{j=1}^{T} \lambda_j = \ell$.

Then we can upper bound the cost of all the iterations in which we do find a witness by the following:

$$\widetilde{O}\left(\sum_{j=1}^{T}\left(\sqrt{\frac{n}{t_{i_j}}}\mathfrak{GC}(n, m_{i_j}) + \mathfrak{GC}_{\text{all}}(n, m_{i_j}, \lambda_j)\right)\right) \tag{1}$$

$$\in \quad \widetilde{O}\left(\sum_{j=1}^{T}\left(\sqrt{\frac{n}{T-j+1}}\mathfrak{GC}(n, m_{i_j}) + \mathfrak{GC}_{\text{all}}(n, m_{i_j}, \lambda_j)\right)\right) \tag{2}$$

$$\in \quad \widetilde{O}\left(\sqrt{nT}\mathfrak{GC}(n, m_{i_j}) + \sum_{j=1}^{T}\mathfrak{GC}_{\text{all}}(n, m_{i_j}, \lambda_j)\right) \tag{3}$$

$$\in \quad \widetilde{O}\left(\sqrt{nT}\left(\sqrt{\ell} + \sqrt{n}\right) + \sum_{j=1}^{T}\left(\sqrt{n\lambda_j} + \sqrt{\ell}\right)\right) \tag{4}$$

$$\in \quad \widetilde{O}\left(\sqrt{nT\ell} + n\sqrt{T} + \sqrt{n\ell T} + T\sqrt{\ell}\right) \tag{5}$$

$$\in \quad \widetilde{O}\left(\sqrt{n\min\{n,\ell\}\ell} + n\sqrt{\min\{n,\ell\}} + \sqrt{\min\{n,\ell\}n\ell} + \min\{n,\ell\}\sqrt{\ell}\right) \tag{6}$$

$$\in \quad \widetilde{O}\left(n\sqrt{\ell}\right) \tag{7}$$

In (4), we use the fact that $m_{i_j} \leq \ell$, and in (5), we use $\sum_{j=1}^{T}\sqrt{\lambda_j} \leq \sqrt{T}\sqrt{\sum_j \lambda_j} = \sqrt{\ell T}$, which follows from the Cauchy–Schwarz inequality. $\qquad\square$

4.2 Lower Bound

Theorem 4.2. *The query complexity of* BOOLEAN MATRIX MULTIPLICATION *is* $\Omega(n\sqrt{\min\{\ell, n^2 - \ell\}})$.

Proof. We will reduce the problem of ℓ-THRESHOLD, in which we must determine whether an input oracle f has $\geq \ell$ or $< \ell$ marked elements, to BOOLEAN MATRIX MULTIPLICATION.

Consider an instance of ℓ-THRESHOLD of size n^2, $f : [n^2] \to \{0, 1\}$. We can construct an instance of BOOLEAN MATRIX MULTIPLICATION as follows. Set A to the identity, and let B

encode f. Finding AB then gives the solution to the ℓ-Threshold instance. By [BBC$^+$01], ℓ-Threshold (with inputs of size n^2) requires $\Omega(\sqrt{n^2\min\{\ell, n^2-\ell\}})$ queries to solve with bounded error. $\qquad\square$

This lower bound implies that our algorithm is tight for any $\ell \leq \varepsilon n^2$ for any constant $\varepsilon < 1$. However, it is not tight for $\ell = n^2 - o(n)$. We can search for pairs (i,j) such that there is no $k \in [n]$ that witnesses (i,j) in cost $n^{3/2}$. If there are m 0s, we can find them all in $\widetilde{O}(n^{3/2}\sqrt{m})$, which is $o(n\sqrt{\ell})$ when $m \in o(n)$. It remains open to close the gap between $\widetilde{O}(n^{3/2}\sqrt{m})$ and $\Omega(n\sqrt{m})$ when $m \in o(n^2)$.

5　Acknowledgements

References

[Amb04]　A. Ambainis. Quantum walk algorithm for element distinctness. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 22–31, 2004. 3, 6

[BBBV97]　C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing (special issue on quantum computing)*, 26:1510–1523, 1997. `arXiv:quant-ph/9701001v1`. 1

[BBC$^+$01]　R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48, 2001. 3, 11

[BBHT98]　M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998. 4

[Bel11]　A. Belovs. Span programs for functions with constant-sized 1-certificates. Technical Report arXiv:1105.4024, arXiv, 2011. 6

[BŠ06]　H. Buhrman and R. Špalek. Quantum verification of matrix products. In *Proceedings of the 17th ACM-SIAM Symposium On Discrete Algorithms*, pages 880–889, 2006. 1, 2

[CKK11]　A. Childs, S. Kimmel, and R. Kothari. The quantum query complexity of read-many formulas. Technical Report arXiv:1112.0548v1, arXiv, 2011. 2

[DH96]　C. Dürr and P. Høyer. A quantum algorithm for finding the minimum. Technical Report arXiv:quant-ph/9607014v2, arXiv, 1996. 5

[DHHM06] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. 5

[Gal12]   F. Le Gall. Improved output-sensitive quantum algorithms for Boolean matrix multiplication. In *Proceedings of the 23rd ACM-SIAM Symposium On Discrete Algorithms*, pages 1464–1476, 2012. 2

[Gro96]   L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 212–219, 1996. 1, 4

[Lin09]   A. Lingas. A fast output-sensitive algorithm for Boolean matrix multiplication. In *Proceedings of the 17th European Symposium on Algorithms*, pages 408–419, 2009. 2

[MSS07]   F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007. 2, 6

[Sho97]   P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, October 1997. 1

[VW10]   V. Vassilevska Williams and R. Williams. Sub-cubic equivalences between path, matrix and triangle problems. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science*, pages 645–654, 2010. 2