Improving Quantum Query Complexity of Boolean Matrix Multiplication Using Graph Collision[*]

Stacey Jeffery[1,2], Robin Kothari[1,2], François Le Gall[3], and Frédéric Magniez[4]

[1]David R. Cheriton School of Computer Science, University of Waterloo, Canada
[2]Institute for Quantum Computing, University of Waterloo, Canada
[3]Graduate School of Information Science and Technology, The University of Tokyo, Japan
[4]CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris-Cité, 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 $\widetilde{\mathrm{O}}(n\sqrt{\ell+1})$ 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.

Using similar ideas, we also show that the time complexity of Boolean matrix multiplication is $\tilde{O}(n\sqrt{\ell+1} + \ell\sqrt{n})$.

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 applications in several areas in computer science. Boolean matrix multiplication is used for instance to construct efficient algorithms for computing the transitive closure of a graph [FM71, Fur70,

[*]This paper is an extended and combined version of [JKM12], [Le 12a] and [Le 12b]. This work was partially supported by NSERC, MITACS, QuantumWorks, the French ANR Blanc project ANR-12-BS02-005 (RDAM), and the European Commission IST STREP project 25596 (QCS).

Mun71], recognizing context-free languages [Ryt85, Val75], detecting if a graph contains a triangle [IR78, VW10], solving all-pairs path problems [DHZ00, GM97, Sei95, SZ99], or speeding up data mining tasks [AP09].

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* [Le 12a], 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 $\mathrm{O}(n^{3/2})$ [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 $\widetilde{\mathrm{O}}(n^{3/2}\sqrt{\ell})$ 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 [Lin11] to get a quantum algorithm for Boolean matrix multiplication with query complexity $\widetilde{\mathrm{O}}(\min\{n^{1.3}\ell^{17/30}, n^2 + n^{13/15}\ell^{47/60}\})$.

Recently, Le Gall [Le 12a] 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, and obtained the following query complexity:

$$\begin{cases} \tilde{O}(n^{1.3}\sqrt{\ell}) & \text{if } 1 \leq \ell \leq n^{1/5}, \\ \tilde{O}(n^{9/7}\ell^{4/7}) & \text{if } n^{1/5} \leq \ell \leq n^{3/8}, \\ \tilde{O}(n^{3/2}) & \text{if } n^{3/8} \leq \ell \leq n^{16/21}, \\ \tilde{O}(n^{13/14}\ell^{3/4}) & \text{if } n^{16/21} \leq \ell \leq n^{8/7}, \\ \tilde{O}(n^{3/2}\ell^{1/4}) & \text{if } n^{8/7} \leq \ell \leq n^{6/5}, \\ \tilde{O}(n^{6/7}\ell^{11/14}) & \text{if } n^{6/5} \leq \ell \leq n^{16/11}, \\ O(n^2) & \text{if } n^{16/11} \leq \ell \leq n^2. \end{cases}$$

These upper bounds are illustrated in Figure 1.

5

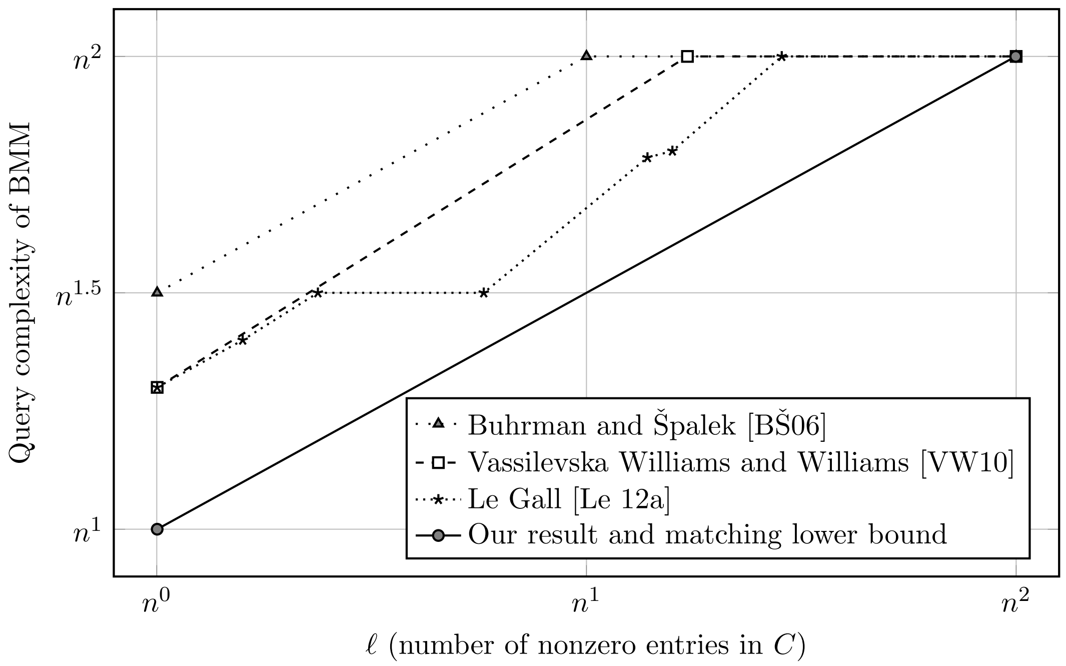

Figure 1: Quantum query complexity of Boolean matrix multiplication (BMM).

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 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.

We would like to emphasize two main ideas. 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. Moreover, we need to find all graph collisions, not just one. We present 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 $\widetilde{O}(n\sqrt{\ell+1})$ upper bound on the query complexity of Boolean matrix multiplication, which is represented in Figure 1 as well. In addition, these ideas yield an algorithm with time complexity $\widetilde{O}(n\sqrt{\ell+1} + \ell\sqrt{n})$.

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 several instances of the search problem, which we state 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 $\varepsilon < 1$. Finally, in Section 5, we prove an upper bound of $\tilde{O}(n\sqrt{\ell+1} + \ell\sqrt{n})$ on the time complexity of Boolean matrix multiplication.

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 that 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.

A search problem P is a map $\mathcal{X} \to 2^{\mathcal{Y}}$, where $\mathrm{P}(x) \subseteq \mathcal{Y}$ 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 $x \in \mathcal{X}$, $\Pr[\mathsf{A}(x) \in \mathrm{P}(x)] \geq 1 - \delta(|x|)$, 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 $1 - \frac{1}{\mathrm{poly}}$ for some super-linear polynomial. We ensure that all of our subroutines succeed with high probability, to finally achieve a bounded-error algorithm. Consequently, we will necessarily incur polylog factors in the query complexity. We will use the notation $\widetilde{\mathrm{O}}$ to indicate that we are suppressing polylog factors. More precisely, $f(n) \in \widetilde{\mathrm{O}}(g(n))$ means $f(n) \in \mathrm{O}(g(n) \log^k n)$ 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. We will use ℓ to denote the number of non-zero entries in C.

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,* $\mathsf{GroverSearch}(t)$*, with query complexity* $\widetilde{\mathrm{O}}(\sqrt{|U|/t})$ *to* f*, such that, if* $t/2 \leq t_f \leq t$*, then* $\mathsf{GroverSearch}(t)$ *finds a marked element with probability at least* $1 - 1/\mathrm{poly}(|U|)$.
Moreover, if $t_f = 0$*, then* $\mathsf{GroverSearch}(t)$ *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 $\mathrm{O}(\log |U|)$ times in order to get bounded error $1/\mathrm{poly}(|U|)$.

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

One consequence of Corollary 2.2 is that we can always apply $\mathsf{Search}(t)$ 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 $\widetilde{\mathrm{O}}(\sqrt{|U|})$.

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 $\mathsf{GroverSearch}$ 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* $\widetilde{\mathrm{O}}(\sqrt{|U|(t_f+1)})$ *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* $\widetilde{\mathrm{O}}(\sqrt{|U|})$ *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 algorithm they used to solve graph collision is based on Ambainis' quantum walk algorithm for element distinctness [Amb04], and has query complexity $\mathrm{O}(n^{2/3})$. The same algorithm is also used in other triangle finding algorithms [Bel12, LMS13]. However, the best known lower bound for graph collision 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 k^{th} column of A and $B[k, \cdot]$ denotes the k^{th} 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 $\overline{\widetilde{C}}$ 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 $\mathrm{O}(\sqrt{nt_A} + \sqrt{nt_B}) \in \mathrm{O}(\sqrt{n\lambda})$. 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 $d_i$ be the degree of the $i^{\text{th}}$ 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
$d_1 \geq d_2 \geq \ldots \geq d_n$.

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

2. Case 1: If $c_r \leq \sqrt{m}$
   (a) Find all marked neighbours of $r$ by SearchAll. Output any
       graph collisions found.                                         $\widetilde{\mathrm{O}}(\sqrt{n\lambda})$
   (b) Delete all unmarked neighbours of $r$. Read the values of all
       non-neighbours of $r$.                                          $\mathrm{O}(\sqrt{m})$
   (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.                                                      $\widetilde{\mathrm{O}}(\sqrt{n\lambda})$

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 $\mathcal{A}$.     $\mathrm{O}(\sqrt{m})$
   (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.                                                      $\widetilde{\mathrm{O}}(\sqrt{n\lambda})$
```

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

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

Step 1 has query complexity $\widetilde{\mathrm{O}}(\sqrt{n})$ by Theorem 2.4. Steps 2a, 2c and 3c have query complexity $\widetilde{\mathrm{O}}(\sqrt{n\lambda})$ 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 $\mathrm{O}(\sqrt{c_r}) \in \mathrm{O}(\sqrt{m})$ 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 $\widetilde{O}(\sqrt{n\lambda})$. □

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.

```
BMM$(A, B)$

1. Let $\widetilde{C} = 0$, $t = n$, and $V = [n]$.
2. While $t \geq 1$:
    (a) GroverSearch$(t)$ for an index $k \in V$ such that the graph collision problem on $k$ with $\overline{\widetilde{C}}$ as the underlying graph has a collision.
    (b) If such a $k$ is found:
            Compute AllGC on the graph defined by $\overline{\widetilde{C}}$ 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* $\widetilde{O}(n\sqrt{\ell + 1})$.

Proof. We will analyze the complexity of the algorithm 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 GroverSearch(t) costs $\widetilde{O}(\sqrt{n/t})$ queries to a procedure that checks if there is a collision in the graph defined by $\overline{\widetilde{C}}$ 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:

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

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

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, that is, the number of times we execute step **2(b)**. Of course, T is a random variable that depends on which witnesses k are found, and in which order. We always have $T \leq \min\{n,\ell\}$.

Let $i_1,\dots,i_T$ 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 j^{th} successful iteration, that is, the number of pairs witnessed by the j^{th} 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:

$$\begin{aligned}
& \widetilde{\mathrm{O}}\left(\sum_{j=1}^{T}\left(\sqrt{\frac{n}{t_{i_j}}}\mathfrak{GC}(n,m_{i_j})+\mathfrak{GC}_{\text{all}}(n,m_{i_j},\lambda_j)\right)\right) && (1)\\
\in\ & \widetilde{\mathrm{O}}\left(\sum_{j=1}^{T}\left(\sqrt{\frac{n}{T-j+1}}\mathfrak{GC}(n,m_{i_j})+\mathfrak{GC}_{\text{all}}(n,m_{i_j},\lambda_j)\right)\right) && (2)\\
\in\ & \widetilde{\mathrm{O}}\left(\sqrt{nT}\mathfrak{GC}(n,\ell)+\sum_{j=1}^{T}\mathfrak{GC}_{\text{all}}(n,m_{i_j},\lambda_j)\right) && (3)\\
\in\ & \widetilde{\mathrm{O}}\left(\sqrt{nT}\left(\sqrt{\ell}+\sqrt{n}\right)+\sum_{j=1}^{T}\left(\sqrt{n\lambda_j}+\sqrt{\ell}\right)\right) && (4)\\
\in\ & \widetilde{\mathrm{O}}\left(\sqrt{nT\ell}+n\sqrt{T}+\sqrt{n\ell T}+T\sqrt{\ell}\right) && (5)\\
\in\ & \widetilde{\mathrm{O}}\left(\sqrt{n\min\{n,\ell\}\ell}+n\sqrt{\min\{n,\ell\}}+\sqrt{\min\{n,\ell\}n\ell}+\min\{n,\ell\}\sqrt{\ell}\right) && (6)\\
\in\ & \widetilde{\mathrm{O}}\left(n\sqrt{\ell}\right) && (7)
\end{aligned}$$

In (3), 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. When $\ell = 0$, the algorithm makes $\widetilde{\mathrm{O}}(\sqrt{n})$ queries, giving an upper bound of $\widetilde{\mathrm{O}}(n\sqrt{\ell+1})$ queries. □

4.2 Lower Bound

Theorem 4.2. *The quantum query complexity of* Boolean Matrix Multiplication *is* $\Omega(n\sqrt{\min\{\ell,n^2-\ell\}})$*, even if the value of* ℓ *is known.*

Proof. Consider an instance of BOOLEAN MATRIX MULTIPLICATION where we know that B is the identity matrix and A has exactly ℓ ones. In this case, the output C is equal to A. We will prove an $\Omega(n\sqrt{\min\{\ell, n^2-\ell\}})$ lower bound for this problem. This is the problem of learning an n^2-bit string promised that it has ℓ ones. Since the problem is symmetric under complementation, i.e., learning A is equivalent to learning the complement of A, we can assume without loss of generality that $\ell \leq n^2/2$.

Now our problem is to learn an n^2-bit string given that it has $\ell \leq n^2/2$ ones. Divide the string into ℓ parts, each of size $\lfloor n^2/\ell \rfloor$, which is at least 2 since $\ell \leq n^2/2$. Now restrict to the inputs where we are promised that each part contains exactly one 1. Finding the locations of all these ones is equivalent to ℓ instances of the SEARCH problem on n^2/ℓ bits each. Since each instance of the search problem requires $\Omega(\sqrt{n^2/\ell})$ queries to solve, and there are ℓ copies of the problem, using the composition theorem proved by [LMR+11], we get a lower bound of $\Omega(\sqrt{n^2\ell})$. Using the observation that the complement of the problem is equally hard gives us a lower bound of $\Omega(n\sqrt{\min\{\ell, n^2-\ell\}})$. $\square$

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 $\widetilde{\mathrm{O}}(n^{3/2}\sqrt{m})$, which is $o(n\sqrt{\ell})$ when $m \in o(n)$. It remains open to close the gap between $\widetilde{\mathrm{O}}(n^{3/2}\sqrt{m})$ and $\Omega(n\sqrt{m})$ when $m \in o(n^2)$.

5 Time Efficient Implementation

In this section, we will prove the following theorem.

Theorem 5.1. *The time complexity of* BOOLEAN MATRIX MULTIPLICATION *is* $\tilde{O}\left(n\sqrt{\ell+1} + \ell\sqrt{n}\right)$.

To prove Theorem 5.1, we will slightly modify $\mathsf{AllGC}_G(f_A, f_B)$. First, we discuss how we store the data to enable efficient access to stored data.

We suppose that the graph G is stored as a data structure $\mathcal{M}$ that contains the following information:

- for each vertex u in $\mathcal{A}$, the degree of u;
- for each vertex u in $\mathcal{A}$, a list of all the vertices of $\mathcal{B}$ not connected to u.

The size of $\mathcal{M}$ is at most $\tilde{O}(n^2)$, but the key idea is that its size will be much smaller when G is "close to" a complete bipartite graph. Using adequate data structures to implement $\mathcal{M}$ (e.g., using self-balancing binary search trees), we can perform the following four access operations in poly$(\log n)$ time.

`get-degree(`u`)`: get the degree of a vertex $u \in \mathcal{A}$

`check-connection(`u, v`)`: check if the vertices $u \in \mathcal{A}$ and $v \in \mathcal{B}$ are connected

`get-vert`$_{\mathcal{A}}$`(`i, d`)`: get the i-th smallest vertex in $\mathcal{A}$ that has degree lower than d

`get-vert`$_{\mathcal{B}}$`(`i, u`)`: get the i-th smallest vertex in $\mathcal{B}$ not connected to $u \in \mathcal{A}$

For the latter two access operations, the order on the vertices refers to the usual order $\leq$ obtained when considering vertices in $\mathcal{A}$ and $\mathcal{B}$ as integers in $\{1, \ldots, n\}$. We assume that these two access operations output an error message when the query is not well-defined (i.e, when i is too large).

We can also update $\mathcal{M}$ in poly$(\log n)$ time to take into consideration the removal of one edge (u, v) from E (i.e., update the degree of u and update the list of vertices not connected to u). This low complexity will be crucial since our main algorithm (in Theorem 5.2 below) will remove successively edges from E.

Theorem 5.2. *For all $\lambda \geq 1$, $\mathfrak{GC}^t_{\text{all}}(n, m, \lambda) \in \widetilde{\mathrm{O}}(\sqrt{n}\lambda + \sqrt{m\lambda})$ and $\mathfrak{GC}^t(n, m) \in \widetilde{\mathrm{O}}(\sqrt{n} + \sqrt{m})$.*

Proof. We will use the following algorithm to find all graph collisions.

```
AllGC$^t_G(f_A, f_B)$
Let $d_i$ be the degree of the $i^{\text{th}}$ 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
$d_1 \geq d_2 \geq \ldots \geq d_n$.

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

  2. Let $\Gamma(r)$ denote the set of neighbours of $r$. Find all marked neigh-
     bours of $r$ using SearchAll. Output any graph collisions found.
     $\widetilde{\mathrm{O}}(\sqrt{n\lambda})$

  3. Let $\mathcal{B}'$ denote the set of marked neighbours of $r$. Find all graph
     collisions in $\mathcal{A} \times \mathcal{B}'$ using SearchAll. Output any graph collisions
     found.                                                            $\widetilde{\mathrm{O}}(\sqrt{n}\lambda)$

  4. Let $\mathcal{A}' = \{u \in \mathcal{A} : d_u < d_r\}$ be the set of vertices in $\mathcal{A}$ with lower
     degree than $r$. Find all graph collisions in $\mathcal{A}' \times (\mathcal{B} \setminus \Gamma(r))$ using
     SearchAll. Output all graph collisions found.                     $\widetilde{\mathrm{O}}(\sqrt{m\lambda})$
```

We analyze the complexity of $\mathsf{AllGC}^t_G(f_A, f_B)$ step by step.

Step 1 requires one call to FindMax, where we search over all vertices for a marked vertex maximizing the function `get-degree`(u), which we can compute using $\mathcal{M}$ in time $\widetilde{\mathrm{O}}(1)$. The cost is therefore $\widetilde{\mathrm{O}}(\sqrt{n})$.

In Step 2, we perform SearchAll for elements of $\mathcal{B}$ that are both marked, and a neighbour of r. We can check if a vertex in $\mathcal{B}$ is marked by querying f_B, and we can check if a vertex in $\mathcal{B}$ is a neighbour of r using `check-connection`$(\cdot, r)$ in time $\widetilde{\mathrm{O}}(1)$. The cost to find all of $\mathcal{B}'$ is therefore $\widetilde{\mathrm{O}}(\sqrt{n\,|\mathcal{B}'|})$. However, $\mathcal{B}'$ contains at most λ vertices, since r is marked, and so each marked neighbour we find corresponds to a distinct graph collision. The cost is therefore $\widetilde{\mathrm{O}}(\sqrt{n\lambda})$.

In Step 3, we perform SearchAll for elements of $\mathcal{A} \times \mathcal{B}'$ that are graph collisions. To check if a pair $(u, v) \in \mathcal{A} \times \mathcal{B}'$ is a graph collision, we must query $f_A(u)$ and $f_B(v)$, and compute `check-connection`(u, v), which can be done in time $\widetilde{\mathrm{O}}(1)$, so the total time complexity of the step is $\widetilde{\mathrm{O}}(\sqrt{|\mathcal{A} \times \mathcal{B}'|\,\lambda}) = \widetilde{\mathrm{O}}(\sqrt{n}\lambda)$, since $|\mathcal{A} \times \mathcal{B}'| \leq n\lambda$.

We have now found all graph collisions in $\mathcal{A} \times \Gamma(r)$, and we know that there are no graph collisions in $(\mathcal{A} \setminus \mathcal{A}') \times \mathcal{B}$, since no vertex in $\mathcal{A} \setminus \mathcal{A}'$ is marked. Thus, all remaining graph collisions must lie in $\mathcal{A}' \times (\mathcal{B} \setminus \Gamma(r))$. In Step 4, we find all of these remaining graph collisions using SearchAll, where, again, we can check if some (u,v) is a graph collision in time complexity $\widetilde{\mathrm{O}}(1)$. In order to implement this step time-efficiently, the search is actually done over the set $\{1,\ldots,|\mathcal{A}'|\}\times\{1,\ldots,|\mathcal{B}\setminus\Gamma(r)|\}$, using the bijection $(i,j)\mapsto(\texttt{get-vert}_{\mathcal{A}}(i,d_r),\texttt{get-vert}_{\mathcal{B}}(j,r))$ from this set to $\mathcal{A}'\times(\mathcal{B}\setminus\Gamma(r)|)$ and observing that this bijection can be evaluated in time complexity $\widetilde{\mathrm{O}}(1)$ from the data structure $\mathcal{M}$. Thus, this search costs $\widetilde{\mathrm{O}}(\sqrt{|\mathcal{A}'\times(\mathcal{B}\setminus\Gamma(r))|\,\lambda})$. Note that $|\mathcal{B}\setminus\Gamma(r)|\le n-d_r=c_r$, and so

$$\left|\mathcal{A}'\right|\left|\mathcal{B}\setminus\Gamma(r)\right|\le\left|\mathcal{A}'\right|c_r\le m,$$

since every vertex in $\mathcal{A}'$ has at least c_r non-edges. Thus Step 4 costs $\widetilde{\mathrm{O}}(\sqrt{m\lambda})$.

The time complexity of this algorithm is therefore $\widetilde{\mathrm{O}}(\sqrt{n}\lambda+\sqrt{m\lambda})$, 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 elements with $\mathsf{SearchAll}$ with an application of Search to check if there is any marked vertex in Step 2, and an application of Search to check if there is any graph collision in Steps 3 and 4. These procedures require $\widetilde{\mathrm{O}}(\sqrt{n})$, $\widetilde{\mathrm{O}}(\sqrt{n})$, and $\widetilde{\mathrm{O}}(\sqrt{m})$ time respectively, by Corollary 2.2. Thus, $\mathfrak{GC}^t(n,m)$ is $\widetilde{\mathrm{O}}(\sqrt{n}+\sqrt{m})$. □

We are now ready to prove the main result of this section, Theorem 5.1.

Proof of Theorem 5.1. This proof is similar to that of Theorem 4.1, which analyzed the query complexity of the algorithm $\mathsf{BMM}(A,B)$. Instead, we will analyze the time complexity of this algorithm.

We first discuss the complexity of creating the data structure $\mathcal{M}$ (remember that updating $\mathcal{M}$ to take in consideration the removal of one edge from E has polylogarithmic cost). Initially the graph considered is the complete bipartite graph, so each vertex of $\mathcal{A}$ has the same degree n and, for each vertex $u\in\mathcal{A}$, there is no vertex in $\mathcal{B}$ not connected to u. The cost for creating $\mathcal{M}$ is thus $\tilde{O}(n)$ time.

Then, we note that the only quantum subroutines used in the algorithm $\mathsf{BMM}(A,B)$ are $\mathsf{GroverSearch}(t)$ and AllGC. The rest of the algorithm is classical and time efficient. Thus the only change that has to be made to the analysis is the substitution of time complexities of these algorithms into the final calculation, instead of their query complexities. However, Grover's algorithm and its variants used in this paper are all time-efficient, i.e., their time complexity is the same as the query complexity up to polylogarithmic factors. Since we are neglecting these factors, the only change we need to make in the calculation is the substitution of values of $\mathfrak{GC}^t_{all}(n,m,\lambda)$ and $\mathfrak{GC}^t(n,m)$ from Theorem 5.2.

Thus the calculation of time complexity is exactly the same as the calculation of query complexity at the end of Theorem 4.1 up to equation (3).

We continue the calculation here starting from equation (3). The time complexity is upper bounded by the following.

$$\begin{aligned}
&\widetilde{\mathrm{O}}\left(\sqrt{nT}\mathfrak{GC}^t(n,\ell)+\sum_{j=1}^{T}\mathfrak{GC}^t_{\mathrm{all}}(n,m_{i_j},\lambda_j)\right) && (8)\\
\in\ &\widetilde{\mathrm{O}}\left(\sqrt{nT}\left(\sqrt{\ell}+\sqrt{n}\right)+\sum_{j=1}^{T}\left(\sqrt{n}\lambda_j+\sqrt{\ell\lambda_j}\right)\right) && (9)\\
\in\ &\widetilde{\mathrm{O}}\left(\sqrt{nT\ell}+n\sqrt{T}+\sqrt{n}\ell+\ell\sqrt{T}\right) && (10)\\
\in\ &\widetilde{\mathrm{O}}\left(n\sqrt{\ell}+\ell\sqrt{n}\right) && (11)
\end{aligned}$$

□

Theorem 4.2 shows that the time complexity given in Theorem 5.1 is optimal in the case $\ell \leq n$. We conclude this section by discussing the optimality in the case $\ell > n$. We first present a simple reduction from general Boolean matrix multiplication (i.e., without any sparsity condition on the output matrix) to several instances of Boolean matrix multiplications with sparse output matrices.

Proposition 5.3. *Let L be any value such that $n \leq L \leq n^2$. Assume the existence of an algorithm that multiplies two $n \times n$ Boolean matrices with at most L non-zero entries in the product in time $T(n, L)$, where $T(n, L) \geq L$. Then there exists another algorithm that computes the product of two arbitrary $n \times n$ Boolean matrices in time $\tilde{O}(\frac{n^2}{L} \times T(n, L))$.*

Proof. Partition the set $\{1, \ldots, n\}$ into $t = O(n^2/L)$ subsets $S_1, \ldots, S_t$, each of size at most L/n. For each $r \in \{1, \ldots, t\}$, let B_r be the $n \times n$ Boolean matrix such that $B_r[i, j] = 1$ if and only if $B[i, j] = 1$ and $j \in S_r$. Observe that

$$AB = \sum_{r=1}^{t} AB_r,$$

where the sum represents the entry-wise OR operation. In order to compute the Boolean product AB it is thus sufficient to compute the t Boolean products AB_r. Finally, observe that each product AB_r can have at most $n \times L/n = L$ non-zero entries. □

In the time complexity setting, the asymptotically fastest algorithms [CW90, Vas12] computing the product of two $n \times n$ Boolean matrices have complexity $O(n^{2.38})$ but use an algebraic approach. This algebraic approach has many disadvantages, the main being that the huge constants involved in the complexities make these algorithms impractical. Indeed, in the classical setting, much attention has focused on algorithms that do not use reductions to matrix multiplication over rings, but instead are based on search or on combinatorial arguments. Such algorithms are often called *combinatorial algorithms*, and the main open problem in this field is to understand whether a $O(n^{3-\varepsilon})$-time combinatorial algorithm, for some constant $\varepsilon > 0$, exists for Boolean matrix multiplication. Unfortunately, there has been little progress on this question. The best known combinatorial classical algorithm for Boolean matrix multiplication, by Bansal and Williams [BW09], has time complexity $O(n^3/\log^{2.25}(n))$.

The reduction stated in Proposition 5.3 is actually classical and combinatorial: the resulting algorithm uses only classical combinatorial operations and $O(n^2/L)$ calls to the first algorithm.

Thus Proposition 5.3 implies that, if for a given value $\ell \geq n$ the complexity of Theorem 5.1 can be improved to $O\left(\ell\sqrt{n}/n^{\delta}\right)$ for some constant $\delta > 0$ using techniques similar to ours (i.e., based on quantum search), then there exists an algorithm based on quantum search and classical combinatorial operations that computes the product of two $n \times n$ Boolean matrices with time complexity $\tilde{O}(n^{5/2-\delta})$, i.e., an amortized cost of $\tilde{O}(n^{1/2-\delta})$ per entry of the output matrix. Since quantum search can typically be simulated classically with a quadratic increase in the time complexity, such an algorithm may lead to a combinatorial way of computing, classically, each entry of the output matrix in (amortized) time $\tilde{O}(n^{1-\varepsilon})$, for some constant $\varepsilon > 0$, giving the first truly subcubic-time classical combinatorial algorithm for Boolean matrix multiplication.

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.

[AP09] Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *Proceedings of Database Theory - ICDT*, pages 121–126, 2009.

[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.

[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.

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

[Bel12] A. Belovs. Span programs for functions with constant-sized 1-certificates: extended abstract. In *Proceedings of the 44th Symposium on Theory of Computing Conference*, pages 77–84, 2012.

[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.

[BW09] N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 745–754, 2009.

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

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

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

[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.

[DHZ00] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[FM71] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory*, pages 129–131, 1971.

[Fur70] M. E. Furman. Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph. *Soviet Mathematics Doklady (English translation)*, 11(5):1252, 1970.

[GM97] Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997.

[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.

[IR78] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.

[JKM12] S. Jeffery, R. Kothari, and F. Magniez. Improving quantum query complexity of boolean matrix multiplication using graph collision. In *Proceedings of 39th International Colloquium on Automata, Languages and Programming*, pages 522–532, 2012.

[Le 12a] 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.

[Le 12b] F. Le Gall. A time-efficient output-sensitive quantum algorithm for Boolean matrix multiplication. In *Proceedings of the 23rd International Symposium on Algorithms and Computation*, pages 639–648, 2012.

[Lin11] A. Lingas. A fast output-sensitive algorithm for Boolean matrix multiplication. *Algorithmica*, pages 36–50, 2011.

[LMR$^+$11] Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Spalek, and Mario Szegedy. Quantum query complexity of state conversion. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 344–353, 2011.

[LMS13] Troy Lee, Frédéric Magniez, and Miklos Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *Proceedings of the 24th ACM-SIAM Symposium On Discrete Algorithms (SODA 2013)*, pages 1486–1502, 2013.

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

[Mun71] J. Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971.

[Ryt85] Wojciech Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and Control*, 67(1-3):12–22, 1985.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[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.

[SZ99] Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 605–615, 1999.

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975.

[Vas12] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th ACM Symposium on Theory of Computing*, pages 887–898, 2012.

[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.