An optimal quantum algorithm for the oracle identification problem

Robin Kothari

David R. Cheriton School of Computer Science and Institute for Quantum Computing, University of Waterloo, Waterloo, Canada
rkothari@cs.uwaterloo.ca

Abstract

In the oracle identification problem, we are given oracle access to an unknown N-bit string x promised to belong to a known set $\mathcal{C}$ of size M and our task is to identify x. We present a quantum algorithm for the problem that is optimal in its dependence on N and M. Our algorithm considerably simplifies and improves the previous best algorithm due to Ambainis et al. Our algorithm also has applications in quantum learning theory, where it improves the complexity of exact learning with membership queries, resolving a conjecture of Hunziker et al.

The algorithm is based on ideas from classical learning theory and a new composition theorem for solutions of the filtered γ_2-norm semidefinite program, which characterizes quantum query complexity. Our composition theorem is quite general and allows us to compose quantum algorithms with input-dependent query complexities without incurring a logarithmic overhead for error reduction. As an application of the composition theorem, we remove all log factors from the best known quantum algorithm for Boolean matrix multiplication.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases quantum algorithms, quantum query complexity, oracle identification

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.482

1 Introduction

Query complexity is a model of computation where quantum computers are provably better than classical computers. Some of the great breakthroughs of quantum algorithms have been conceived in this model (e.g., Grover's algorithm [11]). In this paper we study the query complexity of the oracle identification problem, the very basic problem of completely determining a string given oracle access to it.

In the oracle identification problem, we are given an oracle for an unknown N-bit string x, promised to belong to a known set $\mathcal{C} \subseteq \{0,1\}^N$, and our task is to identify x while minimizing the number of oracle queries. For a set $\mathcal{C}$, we denote this problem $\textsc{oip}(\mathcal{C})$. As usual, classical algorithms have access to an oracle that outputs x_i on input i, while quantum algorithms have access to a unitary O_x that maps $|i,b\rangle$ to $|i,b \oplus x_i\rangle$ for $b \in \{0,1\}$. For a function $f : D \to E$, where $D \subseteq \{0,1\}^N$, let $Q(f)$ denote the bounded-error quantum query complexity of computing $f(x)$. Then $\textsc{oip}(\mathcal{C})$ corresponds to computing the identity function $f(x) = x$ with $D = E = \mathcal{C}$.

For example, let $\mathcal{C}_N := \{0,1\}^N$. Then the classical query complexity of $\textsc{oip}(\mathcal{C}_N)$ is N, since every bit needs to be queried to learn x, even with bounded error. A surprising result of van Dam shows that $Q(\textsc{oip}(\mathcal{C}_N)) = N/2 + O(\sqrt{N})$ [19]. As another example, consider the set $\mathcal{C}_{\mathrm{H1}} = \{x : |x| = 1\}$, where $|x|$ is the Hamming weight of x. This is the search problem with 1 marked item and thus $Q(\textsc{oip}(\mathcal{C}_{\mathrm{H1}})) = \Theta(\sqrt{N})$ [6, 11].

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 482–493
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Due to the generality of the problem, it has been studied in contexts such as quantum query complexity [1, 2], quantum machine learning [18, 5, 13] and post-quantum cryptography [8]. Several well-known problems are special cases of oracle identification, e.g., the search problem with one marked item [11], the Bernstein-Vazirani problem [7], the oracle interrogation problem [19] and hidden shift problems [20]. For some applications, generic oracle identification algorithms are almost as good as algorithms tailored to the specific application [9]. Consequently, this result improves some of the upper bounds stated in [9].

Ambainis et al. [1, 2] studied the oracle identification problem in terms of N and $M := |\mathcal{C}|$. They exhibited algorithms whose query complexity is close to optimal in its dependence on N and M. For a given N and M, we say an oracle identification algorithm is optimal in terms of N and M if it solves all N-bit oracle identification problems with $|\mathcal{C}| = M$ making at most Q queries and there exists some N-bit oracle identification problem with $|\mathcal{C}| = M$ that requires $\Omega(Q)$ queries. This does not, however, mean that the algorithm is optimal for each set $\mathcal{C}$ individually, since these two parameters do not completely determine the query complexity of the problem. For example, all oracle identification problems with $M = N$ can be solved with $O(\sqrt{N})$ queries, and this is optimal since this class includes the search problem with 1 marked item ($\mathcal{C}_{\mathrm{H1}}$ above). However there exists a set $\mathcal{C}$ of size $M = N$ with query complexity $\Theta(\log N)$, such as the set of all strings with arbitrary entries in the first $\log N$ bits and zeroes elsewhere.

Let $\textsc{oip}(M, N)$ denote the set of oracle identification problems with $\mathcal{C} \subseteq \{0,1\}^N$ and $|\mathcal{C}| = M$. Let the query complexity of $\textsc{oip}(M, N)$ be the maximum query complexity of any problem in that set. Then the classical query complexity of $\textsc{oip}(M, N)$ is easy to characterize:

▶ Proposition 1. The classical (bounded-error) query complexity of $\textsc{oip}(M, N)$ is $\Theta(\min\{M, N\})$.

For $M \leq N$, the upper bound follows from the observation that we can always eliminate at least one potential string in $\mathcal{C}$ with one query. For the lower bound, consider any subset of $\mathcal{C}_{\mathrm{H1}}$ of size M. For $M > N$, the lower bound follows from any set $\mathcal{C} \supseteq \mathcal{C}_{\mathrm{H1}}$ and the upper bound is trivial since any query problem can be solved with N queries.

Now that the classical query complexity is settled, we can move to quantum query complexity. When quantum queries are permitted, the $M \leq N$ case is fully understood. For a lower bound, we consider (as before) any subset of $\mathcal{C}_{\mathrm{H1}}$ of size M, which is as hard as the search problem on M bits and requires $\Omega(\sqrt{M})$ queries. For an upper bound, we can reduce this to the case of $M = N$ by selecting M bits such that the strings in $\mathcal{C}$ are distinct when restricted to these bits. (A proof of this fact appears in [9, Theorem 11].) Thus $Q(\textsc{oip}(M, N)) \leq Q(\textsc{oip}(M, M))$, which is $O(\sqrt{M})$ [1, Theorem 3].

▶ Proposition 2. For $M \leq N$, $Q(\textsc{oip}(M, N)) = \Theta(\sqrt{M})$.

For the hard regime, where $M > N$, the best known lower and upper bounds are the following, from [1, Theorem 2] and [2, Theorem 2] respectively.

▶ **Theorem 1** ([1, 2]). *If $N < M \leq 2^{N^d}$ for some constant $d < 1$, then $Q(\textsc{oip}(M, N)) = O(\sqrt{N \log M / \log N})$ and for all $M > N$, $Q(\textsc{oip}(M, N)) = \Omega(\sqrt{N \log M / \log N})$.*

When M gets closer to 2^N, their algorithm no longer gives nontrivial upper bounds. For example, if $M \geq 2^{N/\log N}$, their algorithm makes $O(N)$ queries. While not stated explicitly, an improved algorithm follows from the techniques of [3, Theorem 6], but the improved algorithm also does not yield a nontrivial upper bound when $M \geq 2^{N/\log N}$. Ambainis et al. [2] left open two problems, in increasing order of difficulty: to determine whether it is always possible to solve the oracle identification problem for $M = 2^{o(N)}$ using $o(N)$ queries and to design a single algorithm that is optimal in the entire range of M.

In this paper we resolve both open problems by completely characterizing the quantum query complexity of the oracle identification problem in the full range $N < M \leq 2^N$.

▶ **Theorem 2.** *For $N < M \leq 2^N$, $Q(\text{OIP}(M,N)) = \Theta\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$.*

The lower bound follows from the ideas in [1], but needs additional calculation. We provide a proof in the full version of this paper [15]. The lower bound also appears in an unpublished manuscript [3, Remark 1]. The $+1$ term in the denominator is relevant only when M gets close to 2^N; it ensures that the complexity is $\Theta(N)$ in that regime.

Our main result is the algorithm, which is quite different from and simpler than that of [2]. It is also optimal in the full range of M as it makes $O\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$ queries when $M \geq N$ and $O(\sqrt{M})$ queries when $M \leq N$. Our algorithm has two main ingredients:

First, we use ideas from classical learning theory, where the oracle identification problem is studied as the problem of exact learning with membership queries [4]. In particular, our quantum algorithm is based on Hegedűs' implementation of the halving algorithm [12]. Hegedűs characterizes the number of queries needed to solve the classical oracle identification problem in terms of the "extended teaching dimension" of $\mathcal{C}$. While we do not use that notion, we borrow some of the main ideas. This is further explained in Section 2.

We now present a high-level overview of the algorithm. Say we know that the string in the black box, x, belongs to a set S. We can construct from S a string s, known as the "majority string," which is 1 at position i if at least half the strings in S are 1 at position i. Importantly, for any i, the set of strings in S that disagree with s at position i is at most half the size of S. Now we search for a disagreement between x and s using Grover's algorithm. If the algorithm finds no disagreement, then $x = s$. If it does, we have reduced the size of S by a factor of 2. This gives a suboptimal algorithm with query complexity $O(\sqrt{N} \log M)$. We improve the algorithm by taking advantage of two facts: first, that Grover's algorithm can find a disagreement faster if there are many disagreements to be found, and second, that there exists an order in which to find disagreements that reduces the size of S as much as possible in each iteration. The existence of such an order was shown by Hegedűs [12].

The second ingredient of our upper bound is a composition theorem for solutions of the filtered γ_2-norm semidefinite program (SDP) introduced by Lee et al. [16] that preserves input-dependent query complexities. We need such a result to resolve the following problem: Our algorithm consists of k bounded-error quantum algorithms that must be run sequentially because each algorithm requires as input the output of the previous algorithm. Let the query complexities of the algorithms be $Q_1(x), Q_2(x), \ldots, Q_k(x)$ on input x. If these were exact algorithms, we could merely run them one after the other, giving one algorithm's output to the next as input, to obtain an algorithm with worst-case query complexity $O(\max_x \sum_i Q_i(x))$. However, since these are bounded-error algorithms, we cannot guarantee that all k algorithms will give the correct output with high probability. One option is to apply standard error reduction, but this would yield an algorithm making $O(\max_x \sum_i Q_i(x) \log k)$ queries. Instead, we prove a general composition theorem for the filtered γ_2-norm SDP that gives an algorithm making $O(\max_x \sum_i Q_i(x))$ queries, as if the algorithms had no error. A similar result is known for worst-case query complexity, but that gives a suboptimal upper bound of $O(\sum_i \max_x Q_i(x))$ queries. We prove this result in Section 3.

The oracle identification problem was also studied by Atıcı and Servedio [5], who studied algorithms that are optimal for a given set $\mathcal{C}$. The query complexity of their algorithm depends on a combinatorial parameter of $\mathcal{C}$, $\hat{\gamma}^{\mathcal{C}}$, which satisfies $2 \leq 1/\hat{\gamma}^{\mathcal{C}} \leq N+1$. They prove $Q(\text{OIP}(\mathcal{C})) = O(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} \log M \log\log M)$. Our algorithm for oracle identification, without modification, makes fewer queries than this. Our algorithm makes $O\left(\sqrt{\frac{1/\hat{\gamma}^{\mathcal{C}}}{\log 1/\hat{\gamma}^{\mathcal{C}}}} \log M\right)$

queries, which resolves a conjecture of Hunziker et al. [13]. We show this in Section 4.1. Our composition theorem can also be used to remove unneeded log factors from existing quantum query algorithms. As an example, we show how to improve the almost optimal Boolean matrix multiplication algorithm that makes $O(n\sqrt{l}\,\mathrm{poly}(\log n))$ queries [14], where n is the size of the matrices and l is the output sparsity, to an algorithm with query complexity $O(n\sqrt{l})$. We show this in Section 4.2. We conclude with open questions in Section 5. Proofs omitted due to space constraints appear in the full version of this paper [15].

2 Oracle identification algorithm

In this section we explain the ideas that go into our algorithm and prove its correctness. We also prove the query upper bound assuming we can compose bounded-error quantum algorithms without incurring log factors, which we justify in Section 3.

Throughout this section, let $x \in \mathcal{C}$ be the string we are trying to identify. For any set $S \in \{0,1\}^N$, let $\textsc{maj}(S)$ be an N-bit string such that $\textsc{maj}(S)_i$ is 1 if $|\{y \in S : y_i = 1\}| \geq |\{y \in S : y_i = 0\}|$ and 0 otherwise. In words, $\textsc{maj}(S)_i$ is b if the majority of strings in S have bit i equal to b. Note that the string $\textsc{maj}(S)$ need not be a member of S. In this paper, all logarithms are base 2 and for any positive integer k, we define $[k] := \{1, 2, \ldots, k\}$.

2.1 Basic halving algorithm

We begin by describing a general learning strategy called the halving algorithm, attributed to Littlestone [17]. Say we currently know that the oracle contains a string $x \in S \subseteq \mathcal{C}$. The halving algorithm tests if the oracle string x is equal to $\textsc{maj}(S)$. If it is equal, we have identified x; if not, we look for a bit at which they disagree. Having found such a bit i, we know that $x_i \neq \textsc{maj}(S)_i$, and we may delete all strings in S that are inconsistent with this. Since at most half the strings in S disagree with $\textsc{maj}(S)$ at any position, we have at least halved the number of potential strings.

To convert this into a quantum algorithm, we need a subroutine that tests if a given string $\textsc{maj}(S)$ is equal to the oracle string x and finds a disagreement otherwise. This can be done by running Grover's algorithm on the bitwise $\textsc{xor}$ of x and $\textsc{maj}(S)$.

Algorithm 1 Basic halving algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Search for a disagreement between x and $\textsc{maj}(S)$. If we find a disagreement, delete all inconsistent strings from S. If not, let $S \leftarrow \{\textsc{maj}(S)\}$.
4: **until** $|S| = 1$

This algorithm always finds the unknown string x, since S always contains x. The loop can run at most $\log M$ times, since each iteration cuts down the size of S by a factor of 2. Grover's algorithm needs $O(\sqrt{N})$ queries, but it is a bounded-error algorithm. For this section, let us assume that bounded-error algorithms can be treated like exact algorithms and need no error reduction. Assuming this, Algorithm 1 makes $O(\sqrt{N}\log M)$ queries.

2.2 Improved halving algorithm

Even assuming free error reduction, Algorithm 1 is not optimal. Primarily, this is because Grover's algorithm can find an index i such that $x_i \neq \textsc{maj}(S)_i$ faster if there are many such

indices to be found, and Algorithm 1 does not exploit this fact. Given an N-bit binary string, we can find a 1 with $O(\sqrt{N/K})$ queries in expectation, where $K > 0$ is the number of 1s in the string. Alternately, there is a variant of Grover's algorithm that finds the first 1 (from left to right, say) in the string in $O(\sqrt{p})$ queries in expectation where p is the position of the first 1. This follows from the known $O(\sqrt{N})$ algorithm for finding the first 1 in a string of size N [10], by running that algorithm on the first 2^k bits, for $k = 1, 2, \ldots, \log N$. We can now modify the previous algorithm to look for the first disagreement between x and $\textsc{maj}(S)$ instead of any disagreement.

Algorithm 2 Improved halving algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Search for the first disagreement between x and $\textsc{maj}(S)$. If we find a disagreement, delete all inconsistent strings from S. If not, let $S \leftarrow \{\textsc{maj}(S)\}$.
4: **until** $|S| = 1$

As before, the algorithm always finds the unknown string. Let r be the number of times the loop repeats and $p_1, p_2, \ldots, p_r$ be the positions of disagreement found. After the first run of the loop, since a disagreement is found at position p_1, we have learned the first p_1 bits of x; the first $p_1 - 1$ bits agree with $\textsc{maj}(S)$, while bit p_1 disagrees with $\textsc{maj}(S)$. Thus we are left with a set S in which all strings agree on these p_1 bits. For convenience, we can treat S as a set of strings of length $N - p_1$ (instead of length N). Each iteration reduces the effective length of strings in S by p_i, which gives $\sum_i p_i \leq N$, since there are at most N bits to be learned. As before, the loop can run at most $\log M$ times, thus $r \leq \log M$. Finally, if we assume again that these bounded-error search subroutines are exact, this algorithm requires $O(\sum_i \sqrt{p_i})$ queries, which is $O(\sqrt{N \log M})$, by the Cauchy–Schwarz inequality.

2.3 Final algorithm

While Algorithm 2 is an improvement over Algorithm 1, it is still not optimal. One reason is that sometimes a disagreement between the majority string and x may eliminate more than half the possible strings. This observation can be exploited by finding disagreements in such a way as to maximize the reduction in size when a disagreement is found. This idea is due to Hegedűs [12].

To understand the basic idea, consider searching for a disagreement between x and $\textsc{maj}(S)$ classically. The most obvious strategy is to check if $x_1 = \textsc{maj}(S)_1$, $x_2 = \textsc{maj}(S)_2$, and so on until a disagreement is found. This strategy makes more queries if the disagreement is found at a later position. However, we could have chosen to examine the bits in any order. We would like the order to be such that if a disagreement is found at a later position, it cuts down the size of S by a larger factor. Such an ordering would ensure that either we spend very few queries and achieve a factor-2 reduction right away, or we spend more queries but the size of S goes down significantly. Hegedűs shows that there is always a reordering of the bits that achieves this. The following lemma is similar to [12, Lemma 3.2], but we provide a proof for completeness.

▶ **Lemma 3.** *For any $S \subseteq \{0,1\}^N$, there exists a string $s \in \{0,1\}^N$ and a permutation σ on N, such that for any $p \in [N]$, $|S_p| \leq \frac{|S|}{\max\{2,p\}}$, where $S_p = \{y \in S : y_{\sigma(i)} = s_{\sigma(i)}$ for $1 \leq i \leq p-1$ and $y_{\sigma(p)} \neq s_{\sigma(p)}\}$, the set of strings in S that agree with s at $\sigma(1), \ldots, \sigma(p-1)$ and disagree with it at $\sigma(p)$.*

Proof. We will construct the permutation σ and string s greedily, starting with the first position, $\sigma(1)$. We choose this bit to be one that intuitively contains the most information, i.e., a bit for which the fraction of strings that agree with the majority is closest to $1/2$. This choice will make $|S_1|$ as large as possible. More precisely, we choose $\sigma(1)$ to be any j that maximizes $|\{y \in S : y_j \neq \textsc{maj}(S)_j\}|$. Then let $s_{\sigma(1)}$ be $\textsc{maj}(S)_{\sigma(1)}$.

In general, after having chosen $\sigma(1), \ldots, \sigma(k-1)$ and having defined s on those bits, we choose $\sigma(k)$ to be the most informative bit assuming all previous bits have agreed with string s on positions $\sigma(1), \ldots, \sigma(k-1)$. This choice makes $|S_k|$ as large as possible. More precisely, define $\bar{S}_p = \{y \in S : y_{\sigma(i)} = s_{\sigma(i)} \text{ for all } 1 \leq i \leq p\}$. We choose $\sigma(k)$ to be any bit j that maximizes $|\{y \in \bar{S}_{k-1} : y_j \neq \textsc{maj}(\bar{S}_{k-1})_j\}|$. Then let $s_{\sigma(k)}$ be $\textsc{maj}(\bar{S}_{k-1})_{\sigma(k)}$.

This construction ensures that $|S_1| \geq |S_2| \geq \ldots \geq |S_N|$. Since $\sigma(k)$ was chosen to maximize $|\{y \in \bar{S}_{k-1} : y_j \neq \textsc{maj}(\bar{S}_{k-1})_j\}|$, we have $|S_k| = |\{y \in \bar{S}_{k-1} : y_{\sigma(k)} \neq \textsc{maj}(\bar{S}_{k-1})_{\sigma(k)}\}| \geq |\{y \in \bar{S}_{k-1} : y_{\sigma(k+1)} \neq \textsc{maj}(\bar{S}_{k-1})_{\sigma(k+1)}\}|$. The size of this set is at least $|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq \textsc{maj}(\bar{S}_{k-1})_{\sigma(k+1)}\}|$, since $\bar{S}_k \subseteq \bar{S}_{k-1}$. We do not know the value of $\textsc{maj}(\bar{S}_{k-1})_{\sigma(k+1)}$ (e.g., it need not be equal to $s_{\sigma(k+1)}$), but we do know that it is either 0 or 1. So this term is at least $\min\{|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq 0\}|, |\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq 1\}|\} = \min\{|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq s_{\sigma(k+1)}\}|, |\{y \in \bar{S}_k : y_{\sigma(k+1)} = s_{\sigma(k+1)}\}|\} = \min\{|S_{k+1}|, |\bar{S}_{k+1}|\} = |S_{k+1}|$, where the last equality uses $|S_k| \leq |\bar{S}_k|$ for all k. Finally, combining $|S_1| + \ldots + |S_p| \leq |S|$ with $|S_1| \geq |S_2| \geq \ldots \geq |S_p|$ gives $|S_p| \leq |S|/p$. Combining this with $|S_1| \leq |S|/2$, which follows from the definition of S_1, yields the result. ◀

We can now state our final oracle identification algorithm.

Algorithm 3 Final algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Let σ and s be as in Lemma 3. Search for the first (according to σ) disagreement between x and s. If we find a disagreement, delete all inconsistent strings from S. If not, let $S \leftarrow \{s\}$.
4: **until** $|S| = 1$

As before, it is clear that this algorithm solves the problem. Let us analyze the query complexity. To compute the query complexity, let r be the number of times the loop repeats. Let $p_1, p_2, \ldots, p_r$ be the positions of disagreement. We have $\sum_{i=1}^{r} p_i \leq N$, as in Algorithm 2.

Unlike the previous analysis, the bound $r \leq \log M$ can be loose, since the size of S may reduce by a larger factor due to Lemma 3. Instead, we know that each iteration reduces the set S by a factor of $\max\{2, p_i\}$, which gives us $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$. As before, we will assume the search subroutine is exact, which gives us a query upper bound of $O(\sum_{i=1}^{r} \sqrt{p_i})$, subject to the constraints $\sum_{i=1}^{r} p_i \leq N$ and $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$. We solve this optimization problem in the full version [15] to obtain the following lemma.

▶ **Lemma 4.** *Let $C(M, N)$ be the maximum value attained by $\sum_{i=1}^{r} \sqrt{p_i}$, subject to the constraints $\sum_{i=1}^{r} p_i \leq N$, $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$, $r \in [N]$ and $p_i \in [N]$ for all $i \in [r]$. Then $C(M, N) = O\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$ and $C(M, N) = O(\sqrt{M})$.*

Thus Algorithm 3 achieves the upper bound claimed in Theorem 2, under our assumption that the search subroutine is exact. Since it is not exact, we could reduce the error with logarithmic overhead, but it is usually unnecessary to incur this loss in quantum query algorithms. In the next section we prove this and establish the complexity of Algorithm 3.

3 Composition theorem for input-dependent query complexity

The primary aim of this section is to rigorously establish the query complexity of Algorithm 3. Along the way, we will develop techniques that can be used more generally. Let us begin by describing what we would like to prove. Algorithm 3 essentially consists of a loop repeated $r(x)$ times. We write $r(x)$ to make explicit its dependence on the input x. The loop itself consists of running a variant of Grover's algorithm on x, based on information we have collected thus far about x. Call these algorithms $A_1, A_2, \ldots, A_{r(x)}$. To be clear, A_1 is the algorithm that is run the first time the loop is executed, i.e., it looks for a disagreement under the assumption that $S = \mathcal{C}$. It produces an output $p_1(x)$, which is then used by A_2. A_2 looks for a disagreement assuming a modified set S, which is smaller than $\mathcal{C}$. Let us say that in addition to $p_2(x)$, A_2 also outputs $p_1(x)$. This ensures that the output of A_i completely describes all the information we have collected about x. Thus algorithm A_{i+1} now only needs the output of A_i to work correctly.

We can now view Algorithm 3 as a composition of $r(x)$ algorithms, $A_1, A_2, \ldots, A_{r(x)}$. It is a composition in the sense that the output of one is required as the input of the next algorithm. We know that the expected query complexity of A_i is $O(\sqrt{p_i(x)})$. If these algorithms were exact, then running them one after the other would yield an algorithm with expected query complexity $O(\sum_i \sqrt{p_i(x)})$. But since they are bounded error, this does not work. However, if we consider their worst-case complexities, we can achieve this complexity. If we have r algorithms $A_1, A_2, \ldots, A_r$ with worst-case query complexities Q_i, then there is a quantum algorithm that solves the composed problem with $O(\sum_i Q_i)$ queries. This is a remarkable property of quantum algorithms, which follows from the work of Lee et al. [16]. We first discuss this simpler result before moving on to input-dependent complexities.

3.1 Composition theorem for worst-case query complexity

We now show a composition theorem for solutions of the filtered γ_2-norm SDP, which implies a similar result for worst-case quantum query complexity. This follows from the work of Lee et al. [16], which we generalize in the next section. As discussed in the introduction, let $D \subseteq \{0,1\}^N$, and consider functions that map D to E. For any matrix A indexed by D, we define a quantity $\gamma(A)$. (To readers familiar with the notation of [16], this is their $\gamma_2(A|\Delta)$.)

▶ **Definition 5.** Let A be a square matrix indexed by D. We define $\gamma(A)$ as the following:

$$\gamma(A) := \min_{\{|u_{xj}\rangle, |v_{yj}\rangle\}} \max_{x \in D} \quad c(x) \tag{1}$$

$$\text{subject to:} \qquad \forall x \in D, \quad c(x) = \max\left\{ \sum_j \||u_{xj}\rangle\|^2, \sum_j \||v_{xj}\rangle\|^2 \right\} \tag{2}$$

$$\forall x, y \in D, \quad \sum_{j : x_j \neq y_j} \langle u_{xj} | v_{yj} \rangle = A_{xy} \tag{3}$$

We use $\gamma(A)$ to refer to both the SDP above and its optimum value. For a function $f : D \to E$, let F be its Gram matrix, defined as $F_{xy} = 1$ if $f(x) \neq f(y)$ and $F_{xy} = 0$ otherwise. Lee et al. showed that $Q(f) = \Theta(\gamma(J - F))$, where J is the all-ones matrix.

More generally, they showed that this SDP also upper bounds the quantum query complexity of state conversion. In the state conversion problem, we have to convert a given state $|s_x\rangle$ to $|t_x\rangle$. An explicit description of the states $|s_x\rangle$ and $|t_x\rangle$ is known for all $x \in D$, but we do not know the value of x. Since the query complexity of this task depends only on the Gram matrices of the starting and target states, define S and T by $S_{xy} = \langle s_x | s_y \rangle$ and $T_{xy} = \langle t_x | t_y \rangle$ for all $x, y \in D$. Let $S \mapsto T$ denote the problem of converting states with

Gram matrix S to those with Gram matrix T. If F is the Gram matrix of a function f, then $J \mapsto F$ is the function evaluation problem. Lee et al. showed that $Q(S \mapsto T) = O(\gamma(S - T))$, which generalizes $Q(f) = O(\gamma(J - F))$.

We now have the tools to prove the composition theorem for the filtered γ_2-norm SDP.

▶ **Theorem 6** ([16]). *Let $f_0, f_1, \ldots, f_k$ be functions with Gram matrices $F_0, F_1, \ldots, F_k$. Let $C_1, C_2, \ldots, C_k$ be the optimum value of the SDPs for the state conversion problems $F_0 \mapsto F_1, \ldots, F_{k-1} \mapsto F_k$, i.e., for $i \in [k]$, $C_i = \gamma(F_{i-1} - F_i)$. Then, $\gamma(F_0 - F_k) \leq \sum_{i=1}^{k} C_i$.*

This does not appear explicitly in [16], but simply follows from the triangle inequality $\gamma(A+B) \leq \gamma(A)+\gamma(B)$ [16, Lemma A.2]. From this we can also show an analogous theorem for quantum query complexity, which states $Q(F_0 \mapsto F_k) = O(\sum_{i=1}^{k} Q(F_{i-1} \mapsto F_i))$. We do not prove this claim as we do not need it in this paper.

For our application, we require a composition theorem similar to Theorem 6, but for input-dependent query complexity. However, it is not even clear what this means a priori, since the value $\gamma(J - F)$ does not contain information about input-dependent complexities. Indeed, the value is a single number and cannot contain such information. However, the SDP does contain this information and we modify this framework to be able to access this.

For example, let f be the find-first-one function, which outputs the smallest i such that $x_i = 1$ and outputs $N + 1$ if $x = 0^N$. There is a quantum algorithm that solves this with $O(\sqrt{f(x)})$ queries in expectation. Furthermore, there is a feasible solution for the $\gamma(J - F)$ SDP with $c(x) = O(\sqrt{f(x)})$, where $c(x)$ is the function that appears in (2). This suggests that $c(x)$ gives us information about the x-dependent query complexity. The same situation occurs when we consider the search problem with multiple marked items. There is a feasible solution with $c(x) = O(\sqrt{N/K})$ for inputs with K ones. This function $c(x)$ will serve as our input-dependent cost measure.

3.2 Cost functions

▶ **Definition 7** (Cost function). Let A be a square matrix indexed by D. We say $c : D \to \mathbb{R}$ is a feasible cost function for $\gamma(A)$ if there is a feasible solution of $\gamma(A)$ with values $c(x)$ in eq. (2). Let the set of all feasible cost functions for $\gamma(A)$ be denoted $\Gamma(A)$.

Note that if c is a feasible cost function for $\gamma(J - F)$, then $\max_x c(x)$ is an upper bound on the worst-case cost, $\gamma(J - F)$, which is exactly what we expect from an input-dependent cost. We can now prove an input-dependent analogue of Theorem 6 with $c(x)$ playing the role of $\gamma(J - F)$.

▶ **Theorem 8.** *Let $f_0, f_1, \ldots, f_k$ be functions with Gram matrices $F_0, F_1, \ldots, F_k$. Let $c_1, \ldots, c_k$ be feasible cost functions for $\gamma(F_0 - F_1), \ldots, \gamma(F_{k-1} - F_k)$, i.e., for $i \in [k]$, $c_i \in \Gamma(F_{i-1} - F_i)$. Then there is a $c \in \Gamma(F_0 - F_k)$ satisfying $c(x) \leq \sum_i c_i(x)$ for all $x \in D$.*

As in the case of Theorem 6, this follows from an analogous triangle inequality.

▶ **Lemma 9.** *Let A and B be square matrices indexed by D. If $c_A \in \Gamma(A)$ and $c_B \in \Gamma(B)$, there exists a $c \in \Gamma(A + B)$ satisfying $c(x) \leq c_A(x) + c_B(x)$ for all $x \in D$.*

This is shown by constructing a feasible solution for $\gamma(A + B)$ by taking the direct sum of vectors in a solution of $\gamma(A)$ and $\gamma(B)$. A proof appears in the full version [15].

In our applications, we will encounter algorithms that also output their input, i.e., accept as input $f(x)$ and output $(f(x), g(x))$. Note that the Gram matrix of the function $h(x) = (f(x), g(x))$ is merely $H = F \circ G$, defined as $H_{xy} = F_{xy}G_{xy}$.

Such an algorithm can either be thought of as a single quantum algorithm that accepts $f(x) \in E$ as input and outputs $(f(x), g(x))$ or as a collection of algorithms A_e for each $e \in E$, such that algorithm $A_{f(x)}$ requires no input and outputs $(f(x), g(x))$ on oracle input x. These are equivalent viewpoints, since in one direction you can construct the algorithms A_e from A by hardcoding the value of e and in the other direction, we can read the input e and call the appropriate A_e as a subroutine and output $(e, A_e(x))$. Additionally, if the algorithm $A_{f(x)}$ makes $q(x)$ queries on oracle input x, the algorithm A we constructed accepts $f(x)$ as input, outputs $(f(x), g(x))$, and makes $q(x)$ queries on oracle input x. While intuitive for quantum algorithms, we establish this rigorously for cost functions in the full version [15]:

▶ **Theorem 10.** *Let $f, g : D \to E$ be functions with Gram matrices F and G. For any $e \in E$, let $f^{-1}(e) = \{x : f(x) = e\}$. For every $e \in E$, let $c_e : f^{-1}(e) \to \mathbb{R}$ be a feasible cost function for $\gamma(J - G_e)$, where G_e denotes the matrix G restricted to those x that satisfy $f(x) = e$. Then there exists a $c \in \Gamma(F - F \circ G)$, such that $c(x) = c_{f(x)}(x)$.*

3.3 Algorithm analysis

We can now return to computing the query complexity of Algorithm 3. Using the same notation as in the beginning of this section, for any $x \in \mathcal{C}$, we define $r(x)$ to be the number of times the repeat loop is run in Algorithm 3 for oracle input x assuming all subroutines have no error. Similarly, let $p_1(x), p_2(x), \ldots p_{r(x)}(x)$ be the first positions of disagreement found in each run of the loop. Note that $p_1(x), p_2(x), \ldots p_{r(x)}(x)$ together uniquely specify x. Let $r = \max_x r(x)$.

We now define r functions $f_1, \ldots, f_r$ as $f_1(x) = p_1(x), f_2(x) = (p_1(x), p_2(x)), \ldots, f_r(x) = (p_1(x), \ldots, p_r(x))$, where $p_k(x) = 0$ if $k > r(x)$. Thus if P_i are the Gram matrices of the functions p_i, then $F_1 = P_1, F_2 = P_1 \circ P_2, \ldots, F_r = P_1 \circ P_2 \circ \cdots \circ P_r$.

We will now construct a solution for $\gamma(J - F_r)$, using solutions for the intermediate functions f_i. From Theorem 8 we know that we only need to construct solutions for $\gamma(J - F_1), \gamma(F_1 - F_2), \ldots, \gamma(F_{r-1} - F_r)$. From Theorem 10 we know that instead of constructing a solution for $\gamma(F_k - F_{k+1})$, which is $\gamma(F_k - F_k \circ P_{k+1})$, we can construct several solutions, one for each value of $f_k(x)$. More precisely, let $f_k : D \to E_k$; then we can construct solutions for $\gamma(J - P^e_{k+1})$ for all $e \in E_k$, where P^e_{k+1} is the matrix P_{k+1} restricted to x that satisfy $f_k(x) = e$.

For any k, the problem corresponding to $\gamma(J - P^e_{k+1})$ is just the problem of finding the first disagreement between x and a known string, which is the essentially the find-first-one function. This has a solution with cost function $O(\sqrt{f(x)})$, which in this case is $O(\sqrt{p_{k+1}(x)})$.

▶ **Theorem 11.** *Let f be the function that outputs the smallest i such that $x_i = 1$ and outputs $N + 1$ if $x = 0^N$ and let F be its Gram matrix. Then there is a $c \in \Gamma(J - F)$ such that $c(x) = O(\sqrt{f(x)})$.*

Proof. Let $a_k = k^{-1/4}$ and $b_k = 1/a_k = k^{1/4}$. Define $|u_{xj}\rangle = |v_{xj}\rangle$ as the following.

$$|u_{xj}\rangle = |v_{xj}\rangle = \begin{cases} a_j, & \text{if } j < f(x) \\ b_{f(x)}, & \text{if } j = f(x) \\ 0, & \text{if } j > f(x). \end{cases}$$

This is a feasible solution for $\gamma(J - F)$. Since the constraints are symmetric in x and y, there are two cases: either $f(x) < f(y)$ or $f(x) = f(y)$. In the first case, $\sum_{j:x_j \neq y_j} \langle u_{xj} | v_{yj} \rangle = \sum_{j=f(x)} \langle u_{xj} | v_{yj} \rangle = a_{f(x)} b_{f(x)} = 1$, since x and y agree on all positions before $f(x)$. In the

second case, $\sum_{j:x_j \neq y_j} \langle u_{xj}|v_{yj}\rangle = 0$, since x and y only disagree after position $f(x) = f(y)$. To compute the cost function, note that $c(0^N) = \sum_{k=1}^{N} a_k^2 = O(\sqrt{N}) = O(\sqrt{f(0^N)})$. For $x \neq 0^N$, $c(x) = \sum_{k=1}^{f(x)-1} a_k^2 + b_{f(x)}^2 = \sum_{k=1}^{f(x)-1} k^{-1/2} + \sqrt{f(x)} = O(\sqrt{f(x)})$. ◀

Our function is different from this one in two ways. First, we wish to find the first disagreement with a fixed string s instead of the first 1. This change does not affect the Gram matrix or the SDP. Second, we are looking for a disagreement according to an order σ, not from left to right. This is easy to fix, since we can replace j with $\sigma(j)$ in the definition of the vectors in the proof above.

This shows that for any k, there is a feasible cost function for $\gamma(J - P_{k+1}^e)$ with cost $c(x) = O(\sqrt{p_{k+1}(x)})$ for any x that satisfies $f_k(x) = e$. Using Theorem 10, we get that for any k there is a $c_k \in \Gamma(F_k - F_k \circ P_{k+1})$ with $c_k(x) = O(\sqrt{p_{k+1}(x)})$ for all $x \in D$. Finally, using Theorem 8, we have a $c \in \Gamma(J - F_r)$ with cost $c(x) = O(\sum_{i=1}^{r} \sqrt{p_i(x)}) = O(\sum_{i=1}^{r(x)} \sqrt{p_i(x)})$.

Since the function $f_r(x)$ uniquely determines x, we have a feasible cost function for oracle identification with cost $O(\sum_{i=1}^{r(x)} \sqrt{p_i(x)})$, subject to the constraints of Lemma 4, which we have already solved. Along with the lower bound, this yields the main result.

▶ **Theorem 2.** *For $N < M \leq 2^N$, $Q(\text{OIP}(M, N)) = \Theta\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$.*

4 Other applications

4.1 Quantum learning theory

The oracle identification problem has also been studied in quantum learning theory with the aim of characterizing $Q(\text{OIP}(\mathcal{C}))$. The algorithms and lower bounds studied apply to arbitrary sets $\mathcal{C}$, not just to the class of sets of a certain size, as in the rest of the paper. We show that Algorithm 3 also performs well for any set $\mathcal{C}$, outperforming the best known algorithm. The known upper and lower bounds for this problem are in terms of a combinatorial parameter $\hat{\gamma}^{\mathcal{C}}$, defined by Servedio and Gortler. They showed that for any $\mathcal{C}$, $Q(\text{OIP}(\mathcal{C})) = \Omega(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} + \frac{\log M}{\log N})$ [18]. Later, Atıcı and Servedio showed that $Q(\text{OIP}(\mathcal{C})) = O(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} \log M \log\log M)$ [5].

While we do not define $\hat{\gamma}^{\mathcal{C}}$, we can informally describe it as follows: $\hat{\gamma}^{\mathcal{C}}$ is the largest $\alpha < 1$, such that for any set $S \subseteq \mathcal{C}$, if we know that x belongs to S, there is a bit of x that can be queried such that size of the set of strings consistent with the answer to this query is at most $(1 - \alpha)|S|$, no matter what the oracle responds. This ensures that if we query the oracle with the permutation of Lemma 3, which was chosen to maximize the number of strings eliminated with a query, each query reduces the size of S by a factor of $(1 - \hat{\gamma}^{\mathcal{C}})$.

This adds an extra constraint to Lemma 4 of the form $M \prod_i^r (1 - \hat{\gamma}^{\mathcal{C}})^{p_i} \geq 1$, since learning p_i bits will reduce the size of the remaining set by a factor of $(1 - \hat{\gamma}^{\mathcal{C}})^{p_i}$. From this constraint we get $(\sum_i p_i) \log(1 - \hat{\gamma}^{\mathcal{C}}) \geq -\log M$. Using $\log(1 - \hat{\gamma}^{\mathcal{C}}) \leq -\hat{\gamma}^{\mathcal{C}}$ gives $\sum_i p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$.

We may now replace the constraint $\sum_i p_i \leq N$ with $\sum_i p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ in the optimization problem of Lemma 4. This inequality also implies $p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ and $r \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$. Thus we may simply replace all occurrences of N by $\frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ in Lemma 4. This yields the following theorem, which resolves a conjecture of Hunziker et al. [13, Conjecture 2].

▶ **Theorem 12.** *Algorithm 3 solves $\text{OIP}(\mathcal{C})$ with $O\left(\sqrt{\frac{1/\hat{\gamma}^{\mathcal{C}}}{\log 1/\hat{\gamma}^{\mathcal{C}}}} \log M\right)$ queries.*

Since $Q(\text{OIP}(\mathcal{C})) = \Omega(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} + \frac{\log M}{\log N})$, we see that Algorithm 3 makes $O(\frac{Q(\text{OIP}(\mathcal{C}))^2}{\sqrt{\log Q(\text{OIP}(\mathcal{C}))}} \log N)$ queries, which means it can be at most about quadratically worse than the optimal algorithm for $\text{OIP}(\mathcal{C})$.

4.2 Boolean matrix multiplication

In this section we show how to improve the upper bound on Boolean matrix multiplication (BMM) from $O(n\sqrt{l}\,\mathrm{poly}(\log n))$ [14] to $O(n\sqrt{l})$, where n is the size of the matrices and l is the output sparsity. Like in the analysis in Section 3, we will break up the BMM algorithm of [14] into a sequence of algorithms A_i such that the output of A_i is the input of A_{i+1}, and convert each algorithm into a feasible solution for the corresponding SDP.

The BMM algorithm is almost of this form: It uses two subroutines for graph collision, one for the decision problem and another to find all collisions. The first subroutine solves the problem on a bipartite graph with $2n$ vertices and m nonedges in $O(\sqrt{n}+\sqrt{m})$ queries. Since this query complexity is not input dependent, there is a feasible SDP solution for this problem with $c(x) = O(\sqrt{n}+\sqrt{m})$ using the known characterization of Lee et al. [16].

The second subroutine finds all graph collisions in an instance with λ collisions using $O(\sqrt{n\lambda}+\sqrt{m})$ queries. This upper bound is input dependent, since λ is a function of the input. In this subroutine, the only input-dependent algorithm is the variant of Grover's algorithm that uses $O(\sqrt{nk})$ queries to find all k ones in an n-bit string with k ones. It is easy to show that there is a feasible cost function for this with $c(x) = O(\sqrt{nk})$. For example, we may compose the SDP solution for the find-first-one function (Theorem 11) with itself repeatedly to find all ones. The cost function of the resultant SDP will satisfy $c(x) = O(\sum_i \sqrt{p_i})$, where p_is are the locations of the ones. By the Cauchy-Schwarz inequality this is $O(\sqrt{nk})$. Thus the second subroutine has a feasible cost function $c(x) = O(\sqrt{n\lambda}+\sqrt{m})$.

The BMM algorithm breaks up the problem into n instances of graph collision. The algorithm repeatedly searches for indices i such that the ith graph collision instance has a collision. Then it finds all graph collisions of this instance and repeats. Instead of searching for any i, we can search for the first i. The problem of searching for the first i that has a graph collision is the composition of the find-first-one function (Theorem 11) and the graph collision function. It is a composition in the sense that each input bit of the first problem is the output bit of another problem. It is known that the optimal value of the γ SDP for $f \circ g^n$ is at most $\gamma(J-F)\gamma(J-G)$. Similarly, it can be shown that there is a feasible cost function for $f \circ g$ that is at most the product of the cost functions. This is similar to [16, Lemma 5.1] or Lemma 9, but we take the tensor product instead of taking the direct sum.

Finally, let $p_1, \ldots, p_t$ be the positions of indices found in the algorithm. The search problem requires $O(\sqrt{p_i}(\sqrt{n}+\sqrt{m}))$ queries for each i, since it is the composition of the two above-mentioned algorithms. The algorithm that finds all graph collisions has a feasible cost function $O(\sqrt{n\lambda_i}+\sqrt{m})$, where λ_i is the number of graph collisions in the ith graph collision instance. This gives a feasible cost function for BMM with cost $O(\sum_i(\sqrt{p_i}(\sqrt{n}+\sqrt{m})+\sqrt{n\lambda_i}+\sqrt{m}))$, which is the same optimization problem solved in [14], without log factors. This is $O(n\sqrt{l})$.

5 Open questions

Our composition theorem only works for solutions of the filtered γ_2-norm SDP, not for quantum query complexity itself. While this is sufficient for our application, it would be interesting to know if bounded-error quantum algorithms with input-dependent query complexities can be composed in general without incurring log factors.

While the query complexity of oracle identification in terms of M and N has been fully characterized, finding an optimal quantum algorithm for $\mathrm{OIP}(\mathcal{C})$ remains open, even classically. It would also be interesting to study time-efficient oracle identification algorithms for specific sets $\mathcal{C}$, since none of the known algorithms is known to be time efficient.

Acknowledgments. I thank Andrew Childs and Ben Reichardt for helpful discussions, Seiichiro Tani for pointing me to Ref. [3], and Andrew Childs and Ansis Rosmanis for comments on a preliminary draft. This work was supported in part by NSERC, the Ontario Ministry of Research and Innovation, and the US ARO.

References

1 Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Hiroyuki Masuda, Raymond H. Putra, and Shigeru Yamashita. Quantum Identification of Boolean Oracles. In *STACS 2004*, volume 2996 of *LNCS*, pages 105–116. Springer, 2004.

2 Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Rudy Raymond, and Shigeru Yamashita. Improved algorithms for quantum identification of Boolean oracles. *Theor. Comput. Sci.*, 378(1):41 – 53, 2007.

3 Andris Ambainis, Kazuo Iwama, Masaki Nakanishi, Harumichi Nishimura, Rudy Raymond, Seiichiro Tani, and Shigeru Yamashita. Average/worst-case gap of quantum query complexities by on-set size. *arXiv preprint arXiv:0908.2468*, 2009.

4 Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

5 Alp Atıcı and Rocco Servedio. Improved Bounds on Quantum Learning Algorithms. *Quantum Information Processing*, 4:355–386, 2005.

6 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

7 Ethan Bernstein and Umesh Vazirani. Quantum Complexity Theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997.

8 Dan Boneh and Mark Zhandry. Quantum-Secure Message Authentication Codes. In *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 592–608. Springer, 2013.

9 Andrew M. Childs, Robin Kothari, Maris Ozols, and Martin Roetteler. Easy and Hard Functions for the Boolean Hidden Shift Problem. In *TQC 2013*, volume 22 of *LIPIcs*, pages 50–79, 2013.

10 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM J. Comput.*, 35(6):1310–1328, 2006.

11 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC 1996*, pages 212–219, 1996.

12 Tibor Hegedűs. Generalized teaching dimensions and the query complexity of learning. In *COLT 1995*, pages 108–117, 1995.

13 Markus Hunziker, David A. Meyer, Jihun Park, James Pommersheim, and Mitch Rothstein. The geometry of quantum learning. *Quantum Information Processing*, 9(3):321–341, 2010.

14 Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Improving Quantum Query Complexity of Boolean Matrix Multiplication Using Graph Collision. In *ICALP 2012*, volume 7391 of *LNCS*, pages 522–532. Springer, 2012.

15 Robin Kothari. An optimal quantum algorithm for the oracle identification problem. *arXiv preprint arXiv:1311.7685*, 2013.

16 Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum Query Complexity of State Conversion. In *FOCS 2011*, pages 344–353, 2011.

17 Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

18 Rocco A. Servedio and Steven J. Gortler. Equivalences and Separations Between Quantum and Classical Learnability. *SIAM J. Comput.*, 33(5):1067–1092, 2004.

19 Wim van Dam. Quantum Oracle Interrogation: Getting All Information for Almost Half the Price. In *FOCS 1998*, page 362, 1998.

20 Wim van Dam, Sean Hallgren, and Lawrence Ip. Quantum Algorithms for Some Hidden Shift Problems. *SIAM J. Comput.*, 36(3):763–778, 2006.