The Structure of Promises in Quantum Speedups*

Shalev Ben-David

Massachusetts Institute of Technology, Cambridge, MA, USA
shalev@mit.edu

Abstract

In 1998, Beals, Buhrman, Cleve, Mosca, and de Wolf showed that no super-polynomial quantum speedup is possible in the query complexity setting unless there is a promise on the input. We examine several types of "unstructured" promises, and show that they also are not compatible with super-polynomial quantum speedups. We conclude that such speedups are only possible when the input is known to have some structure.

Specifically, we show that there is a polynomial relationship of degree 18 between $\mathrm{D}(f)$ and $\mathrm{Q}(f)$ for any Boolean function f defined on permutations (elements of $[n]^n$ in which each alphabet element occurs exactly once). More generally, this holds for all f defined on orbits of the symmetric group action (which acts on an element of $[M]^n$ by permuting its entries). We also show that any Boolean function f defined on a "symmetric" subset of the Boolean hypercube has a polynomial relationship between $\mathrm{R}(f)$ and $\mathrm{Q}(f)$ – although in that setting, $\mathrm{D}(f)$ may be exponentially larger.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Quantum computing, quantum query complexity, decision tree complexity, lower bounds, quantum adversary method

Digital Object Identifier 10.4230/LIPIcs.TQC.2016.7

1 Introduction

When can quantum computers provide super-polynomial speedups over classical computers? This has been one of the central questions of quantum computing research since its inception. On one hand, Shor [10] showed that quantum computers can be used to factor an n-bit integer in $O(n^3)$ time – exponentially faster than the best known classical algorithm (which is only conjectured to achieve $e^{O(n^{1/3}\log^{2/3} n)}$ time [6]). On the other hand, quantum algorithms are not believed be able to solve NP-complete problems efficiently, which heavily restricts the set of problems for which they may offer such a speedup. The intuition, then, is that quantum algorithms help only for certain "structured" problems, but not for unstructured ones.

In the query complexity model, we can hope to formalize this intuition. To this end, in 1998, Beals, Buhrman, Cleve, Mosca, and de Wolf [5] showed that the classical and quantum query complexities of any total Boolean function are polynomially related. On the other hand, *partial* functions – functions that assume the input satisfies some promise – can exhibit exponential quantum speedups [11, 8, 2]. However, we still do not have an understanding of *which* partial functions should be expected to provide such speedups.

▶ **Open Problem 1.** Can we characterize the partial functions f for which $\mathrm{Q}(f) = \mathrm{R}(f)^{o(1)}$?

* This work is partially supported by the National Science Foundation and by the Natural Sciences and Engineering Council of Canada. I thank Scott Aaronson for many helpful discussions.

11th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2016).
Editor: Anne Broadbent; Article No. 7; pp. 7:1–7:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Although we are currently far from such a characterization, a natural first step would be to find *any* type of promise for which we can show a polynomial relationship between $\mathrm{R}(f)$ and $\mathrm{Q}(f)$ (similar to the Beals et al. result for total functions). In this work, we give the first such relationship. We show that when the promise is "the input is a permutation of $\{1, 2, \ldots, n\}$," there is a power 18 relationship between quantum and deterministic query complexities. We also show that when the promise has the form "the input has Hamming weight in the set S" (with $S \subseteq \mathbb{N}$), there is a power 18 relationship between quantum and randomized query complexities (though it's possible for the deterministic query complexity to be exponentially larger). We generalize these results to other classes of promises.

1.1 Previous Work

In 1998, Beals, Buhrman, Cleve, Mosca, and de Wolf [5] proved the following theorem.

▶ **Theorem 1** ([5]). *Let $f : \{0,1\}^n \to \{0,1\}$ be a total function. Then $\mathrm{Q}(f) = \Omega(\mathrm{D}(f)^{1/6})$.*

Their result easily extends to larger alphabets:

▶ **Theorem 2.** *Let $f : [M]^n \to \{0,1\}$ be a total function. Then $\mathrm{Q}(f) = \Omega(\mathrm{D}(f)^{1/6})$.*

This tells us that there is never a super-polynomial quantum speedup for total functions. Note that these results compare quantum query complexity to deterministic query complexity, which is stronger than comparing to randomized query complexity. However, no better relationship is known, even between $\mathrm{Q}(f)$ and $\mathrm{R}(f)$. For more information, see [7].

Another interesting result was proved by Aaronson and Ambainis [1]. They defined a function f to be *permutation-invariant* if

$$f(x_1, x_2, \ldots, x_n) = f(\tau(x_{\sigma(1)}), \tau(x_{\sigma(2)}), \ldots, \tau(x_{\sigma(n)})) \tag{1}$$

for all inputs x and all permutations $\sigma \in S_n$ and $\tau \in S_M$. Here f may be a partial function, but the domain of f must itself be invariant under these permutations. As an example, if $M = 2$, the domain of f might contain all binary strings of Hamming weight in $\{1, 2, n-2, n-1\}$, and $f(x)$ will depend only on the Hamming weight of x (with the value of f being equal on Hamming weights k and $n-k$). Note in particular that the Collision problem – in which we're promised that the input either contains no repeated alphabet symbol, or else repeats each symbol exactly twice, and must discern which is the case – is a permutation-invariant function.

Aaronson and Ambainis [1] proved the following theorem.

▶ **Theorem 3.** *Let f be permutation-invariant. Then $\mathrm{Q}(f) = \tilde{\Omega}(\mathrm{R}(f)^{1/7})$.*

This theorem means that if f is unstructured in a way that looks like the Collision problem, $\mathrm{Q}(f)$ and $\mathrm{R}(f)$ are polynomially related. However, the property of "looking like Collision" places strong constraints on both the function and its promise. In this work, we will show a relationship that holds for *all functions defined defined over a fixed promise P*: we will not assume anything about the structure of f. However, our results will not generalize Theorem 3 (we will provide a generalization of Theorem 1 instead).

Recently, Aaronson and Ben-David [3] characterized the total Boolean functions f that can be "sculpted" to give an exponential quantum speedup; that is, the functions f that can be restricted to a promise P on which quantum algorithms provide an exponential advantage. They showed that the sculptable total functions are those with a large number of large certificates. In particular, this means most total functions are scultable. One interpretation of this is that *quantum speedups are all about the promise* – if we can carefully chose the promise, we can make almost any function exhibit an enormous quantum speedup over classical algorithms.

The question we study here flips the quantifiers: on which promises does there exist a function that exhibits a large quantum advantage? Plausibly, there are very few such promises, which means that characterizing them is a useful way to approach Open Problem 1. Unfortunately, proving the non-existence of quantum-friendly functions can be difficult.

1.2 Our Results

Our first result is a polynomial relationship between $\mathrm{Q}(f)$ and $\mathrm{D}(f)$ for all functions whose domain is the set of permutations.

▶ **Theorem 4.** *Let $M = n$, and let $P \subseteq [M]^n$ be the set of permutations. Then for all $f : P \to \{0,1\}$, $\mathrm{Q}(f) = \Omega(\mathrm{D}(f)^{1/18})$.*

We prove this result as a special case of a more general theorem. To state the general version, we need a few definitions.

Given $x \in [M]^n$, the *orbit* $\mathrm{orb}(x)$ of x is the set of all strings in $[M]^n$ that can be reached by permuting the characters of x (in other words, the orbit under the symmetric group action acting on the entries of the input). Note that each orbit is uniquely identified by a the multiset $\{x_1, x_2, \ldots, x_n\}$ of characters that appear in the strings of that orbit. We will use $\tau(x)$ to refer to this multiset. If $T \subseteq [M]^n$ is an orbit, we will also use $\tau(T)$ to refer to $\tau(x)$ for any $x \in T$. For example, the orbit of $x = (1,1,2)$ is $\mathrm{orb}(x) = \{(1,1,2),(1,2,1),(2,1,1)\}$, and corresponds to the multiset $\tau(x) = \{1,1,2\}$.

We prove the following generalization of Theorem 4.

▶ **Theorem 5.** *Let $T \subseteq [M]^n$ be an orbit, and let $f : T \to \{0,1\}$. Then $\mathrm{Q}(f) = \Omega(\mathrm{D}(f)^{1/18})$.*

Note that this is a relationship between quantum query complexity and deterministic (not randomized) query complexity. In this sense, the result is similar to Theorem 2, and indeed we use some similar tools in its proof. However, unlike for total functions, a function defined on an orbit might have certificate complexity exponentially smaller than $\mathrm{D}(f)$, which prevents the techniques of [5] from directly applying. We develop some new tools to get around this.

Our second result extends the previous theorem from promises that are orbits to promises that are unions of orbits; that is, the promise may be any "symmetric" set. Here we are only able to prove a polynomial relationship when M is constant.

▶ **Theorem 6.** *Let M be constant. If $f : X \to \{0,1\}$ is a function on any symmetric promise $X \subseteq [M]^n$ (that is, a set X satisfying $x \in X \Rightarrow \tau(x) \subseteq X$), then $\mathrm{Q}(f) = \Omega\left(\mathrm{R}(f)^{1/(18(M-1))}\right)$. In particular, when $M = 2$, we have $\mathrm{Q}(f) = \Omega\left(\mathrm{R}(f)^{1/18}\right)$, so any function defined on a symmetric subset of the Boolean hypercube does not exhibit a super-polynomial quantum speedup.*

Unlike the previous theorem, this one only relates quantum query complexity to randomized (rather than deterministic) query complexity. This is necessary; indeed, if X is the set of binary strings of Hamming weight 0 or $\lfloor n/2 \rfloor$ and f is defined to be 0 on 0^n and 1 elsewhere, then $\mathrm{D}(f) = \lfloor n/2 \rfloor + 1$ but $\mathrm{R}(f)$ is constant.

Notice that this last theorem applies even to the promise $X = [M]^n$ (for constant M), so it can be viewed as a generalization of Theorem 1 (although our polynomial relationship has higher degree, and our generalization replaces $\mathrm{D}(f)$ with $\mathrm{R}(f)$).

As a final note, we remark that our results are incomparable with the Aaronson-Ambainis result (Theorem 3). When M is constant, our Theorem 6 is *much* more general (since it doesn't place restrictions on the function). However, when M is constant, Theorem 3 is not very difficult in the first place; most of the work in [1] went towards dealing with the fact that M may be large (as it is in the Collision problem).

2 Preliminaries

In query complexity, there is a known (possibly partial) function $f : [M]^n \to \{0,1\}$ and an unknown string x in the domain of f. The goal is to determine the value of $f(x)$ using as few queries to the entries of x as possible. Here $[M] := \{0, 1, \dots, M-1\}$ is the input alphabet; often we set $M = 2$, so the domain is $\{0,1\}^n$.

The query complexity achieved by an algorithm A is defined to be the number of queries used by A over the worst-case choice of x. The query complexity of the function f is then defined to be the minimum query complexity achieved by any algorithm A.

When A is a deterministic algorithm, we denote the query complexity of f by $\mathrm{D}(f)$; when A is a bounded-error randomized algorithm, we denote it by $\mathrm{R}(f)$; and when A is a bounded-error quantum algorithm, we denote it by $\mathrm{Q}(f)$. We also define the zero-error randomized query complexity $\mathrm{R}_0(f)$ to be the expected number of queries used by the best zero-error randomized algorithm (over the worst-case choice of input x). As expected, we have the relationship $\mathrm{D}(f) \geq \mathrm{R}_0(f) \geq \mathrm{R}(f) \geq \mathrm{Q}(f)$ for every function f. We denote the domain of f by $\mathrm{Dom}(f)$. We sometimes refer to the domain as the *promise* of f.

A partial assignment is a string $p \in ([M] \cup \{*\})^n$ that represents partial knowledge of a string in $[M]^n$. An input $x \in [M]^n$ is consistent with a partial assignment p if for all indices i, either $p_i = x_i$ or $p_i = *$. The size $|p|$ of p is the number of non-star entries in p.

A partial assignment is called a 0-certificate for f if the only strings in $\mathrm{Dom}(f)$ it is consistent with are 0-inputs to f. 1-certificates are defined similarly. A partial assignment is a certificate if it is a 0- or 1-certificate. The certificate complexity $\mathrm{C}_x(f)$ of an input x is the minimum size of a certificate for f consistent with x. The maximum of $\mathrm{C}_x(f)$ over all $x \in \mathrm{Dom}(f)$ is the certificate complexity $\mathrm{C}(f)$ of f.

A block is a set of indices in $\{1, 2, \dots n\}$. We say that a block B is sensitive for a string $x \in \mathrm{Dom}(f)$ if there is a string $y \in \mathrm{Dom}(f)$ that agrees with x outside of B, and satisfies $f(y) \neq f(x)$. The maximum number of disjoint sensitive blocks of x is the block sensitivity of x, denoted by $\mathrm{bs}_x(f)$. The maximum block sensitivity of x over all $x \in \mathrm{Dom}(f)$ is called the block sensitivity of f, denoted by $\mathrm{bs}(f)$.

If $f : [M]^n \to \{0,1\}$ is a total function, we can also define the sensitivity $\mathrm{s}_x(f)$ of a string x as the maximum number of disjoint sensitive blocks of size 1, and the sensitivity $\mathrm{s}(f)$ of f as the maximum value of $\mathrm{s}_x(f)$ over all $x \in [M]^n$. However, since we will be dealing with non-total functions, we will define sensitivity slightly differently in the next section.

It is not hard to see that $\mathrm{s}_x(f) \leq \mathrm{bs}_x(f) \leq \mathrm{C}_x(f)$ for all $x \in \mathrm{Dom}(f)$. Also, since a zero-error algorithm always finds a certificate, we have $\mathrm{s}(f) \leq \mathrm{bs}(f) \leq \mathrm{C}(f) \leq \mathrm{R}_0(f) \leq \mathrm{D}(f)$. The lower bound on Grover search implies $\mathrm{Q}(f) = \Omega(\sqrt{\mathrm{bs}(f)})$ and $\mathrm{R}(f) = \Omega(\mathrm{bs}(f))$. For total functions, we have $\mathrm{D}(f) \leq \mathrm{C}(f)\,\mathrm{bs}(f)$ and $\mathrm{C}(f) \leq \mathrm{s}(f)\,\mathrm{bs}(f)$ [5], so

$$\mathrm{D}(f) \leq \mathrm{C}(f)\,\mathrm{bs}(f) \leq \mathrm{s}(f)\,\mathrm{bs}(f)^2 \leq \mathrm{bs}(f)^3 = O(\mathrm{Q}(f)^6). \tag{2}$$

For a nice survey of query complexity, see [7].

3 Orbit Promises

In this section, we show that the deterministic and quantum query complexity measures are polynomially related when the promise is exactly an orbit, proving Theorem 5.

One particular case which will motivate a lot of our analysis is the case where $M = n$ and T is the orbit corresponding to the multiset $\{0, 1, \ldots, n-1\}$ (i.e. the case where the inputs are all permutations), together with the function f satisfying $f(x) = 0$ if and only if 0 occurs in the first $\lfloor \frac{n}{2} \rfloor$ entries of x. This is sometimes called the permutation inversion problem.

Informally, in this problem we are promised that the input x is a permutation of the elements $0, 1, \ldots, n-1$, and the task is to find the 0 element using as few queries as possible (to turn this into a decision problem, we only ask whether the 0 occurs in the first half of the entries of x). The permutation inversion problem has been shown to require $\Omega(\sqrt{n})$ quantum queries using a variety of methods [9]; our approach uses Ambainis's adversary method [4].

3.1 Sensitivity on Orbit Promises

We start by attempting to mimic the proof that $\mathrm{D}(f) = O(\mathrm{Q}(f)^6)$ for total functions. There are two missing pieces that don't immediately work for partial functions. One is the relationship $\mathrm{C}(f) \le \mathrm{bs}(f)\,\mathrm{s}(f)$; as defined, $\mathrm{s}(f) = 0$ for permutation inversion, since it's impossible to change only one bit and stay in the promise. The other is the relationship $\mathrm{D}(f) \le \mathrm{C}(f)\,\mathrm{bs}(f)$; for permutation inversion, we have $\mathrm{D}(f) = \lfloor n/2 \rfloor$ but $\mathrm{C}(f) = \mathrm{bs}(f) = 1$.

We fix the former by changing the definition of $\mathrm{s}(f)$ for orbit promises. The latter problem is harder to handle, and does not have an elementary solution. In the next section, we will attack it by showing that the permutation inversion problem – in which we are looking for a hidden marked item that's promised to be unique – is essentially the only difficult case.

▶ **Definition 7.** Let $T \subseteq [M]^n$ be an orbit, let $f : T \to \{0, 1\}$, and let $x \in T$. We define the sensitivity $\mathrm{s}_{2,x}(f)$ of x is the maximum number of disjoint sensitive blocks of size 2 (instead of size 1). The sensitivity $\mathrm{s}_2(f)$ of f is the maximum value of $\mathrm{s}_{2,x}(f)$ out of all $x \in T$.

Note that letting blocks have size 2 allows two entries to be swapped, maintaining the promise. It is also clear that we still have $\mathrm{s}_{2,x}(f) \le \mathrm{bs}_x(f)$ for all $x \in T$.

▶ **Theorem 8.** *For all $f : T \to \{0, 1\}$ with $T \subseteq [M]^n$ an orbit, we have $\mathrm{C}(f) \le 3\,\mathrm{bs}(f)\,\mathrm{s}_2(f)$.*

Proof. Let $x \in T$. Then x has $\mathrm{bs}_x(f)$ disjoint sensitive blocks; let them be $b_1, b_2, \ldots, b_{\mathrm{bs}_x(f)}$, and assume each b_i is minimal (under subsets). Then $\bigcup b_i$ is a certificate consistent with x (for otherwise, x would have more than $\mathrm{bs}_x(f)$ disjoint sensitive blocks). We claim that the size of a sensitive block b_i is at most $3\,\mathrm{s}_2(f)$. This gives us the desired result, because we then have a certificate of size at most $3\,\mathrm{bs}_x(f)\,\mathrm{s}_2(f)$.

Let $y \in T$ agree with x outside b_i with $f(y) \ne f(x)$. Since x and y have the same orbit, the difference between them must be a permutation on the entries of b_i. In other words, there is some permutation σ on b_i such that for $j \in b_i$, we have $y_j = x_{\sigma(j)}$.

Consider the cycle decomposition $c_1 c_2 \ldots c_k$ of σ. Let $c_j = (a_1, a_2, \ldots, a_m)$ be any cycle in it. We claim that switching a_s and a_{s+1} for $s \in \{1, 2, \ldots, m-1\}$ gives a sensitive block for y of size 2. Indeed, if this was not a sensitive block, then block b_i would not be minimal, since $(a_s, a_{s+1})\sigma$ would be a permutation corresponding to a smaller sensitive block (with a_s removed). Note that the number of disjoint sensitive blocks of size 2 we can form this way is at least $\frac{|b_i|}{3}$, since for each cycle c_j we can form $\lfloor \frac{|c_j|}{2} \rfloor \ge \frac{|c_j|}{3}$ of them. Thus $\mathrm{s}_2(f) \ge \frac{1}{3}|b_i|$, as desired. ◀

▶ **Corollary 9.** *Let $f : T \to \{0,1\}$ with $T \subseteq [M]^n$ an orbit. Then* $\mathrm{R}(f) = \Omega(\mathrm{C}(f)^{1/2})$ *and* $\mathrm{Q}(f) = \Omega(\mathrm{C}(f)^{1/4})$.

Proof. We have $\mathrm{C}(f) \leq 3\,\mathrm{bs}(f)\,\mathrm{s}_2(f) \leq 3\,\mathrm{bs}(f)^2$, so $\mathrm{bs}(f) = \Omega(\sqrt{\mathrm{C}(f)})$. Combined with $\mathrm{Q}(f) = \Omega(\sqrt{\mathrm{bs}(f)})$ and $\mathrm{R}(f) = \Omega(\mathrm{bs}(f))$, this gives the desired result. ◀

3.2 The Structure of Small Certificates

The previous section showed a lower bound on $\mathrm{Q}(f)$ in terms of $\mathrm{C}(f)$ on orbit promises. However, this result by itself cannot be used to relate $\mathrm{Q}(f)$ to $\mathrm{D}(f)$ or $\mathrm{R}(f)$, because the certificate complexity of a function on an orbit promise may be much smaller than the query complexities (an example of this is given by permutation inversion, in which $\mathrm{C}(f) = 1$).

In this section, we prove the following technical lemma, which will be the main tool for handling functions for which the certificate complexity is much smaller than the deterministic query complexity.

▶ **Lemma 10.** *Let $f : T \to \{0,1\}$ with $T \subseteq [M]^n$ an orbit. Fix any $k \leq \frac{1}{2}\sqrt{\mathrm{D}(f)}$. If $k \geq \mathrm{C}(f)$, then there is*

- *a partial assignment p, consistent with some input in T, of size at most $4k^2$, and*
- *a set of alphabet elements $S \subseteq [M]$, of size at most $4k^2$, whose elements each occur less than $2k$ times in $\tau(T) - \tau(p)$*

such that for any $x \in T$ consistent with p and any certificate c consistent with x of size at most k, at least one of the alphabet elements of $c - p$ is in S.

Some clarifications are in order. By $\tau(T)-\tau(p)$, we mean multiset subtraction between the alphabet elements in T and those occurring in p (multiset subtraction is defined analogously to set subtraction; the frequency count of an element in $\tau(T) - \tau(p)$ is the difference of frequency counts in $\tau(T)$ and $\tau(p)$, or 0 if this difference is negative). By $c - p$, we mean the string d with $d_i = c_i$ when $p_i = *$ and $d_i = *$ otherwise.

Intuitively, this lemma is saying that if we fix a few input coordinates p and restrict to inputs consistent with p, then there is a small set $S \subseteq [M]$ of alphabet elements such that an element of S must exist in any small certificate. For example, for the problem of inverting a permutation, we can choose $p = *^n$, $S = \{0\}$, and $k = \lfloor n/2 \rfloor - 1$; then any certificate of size less than k must include the alphabet element 0. The intuition, then, is that solving the function quickly requires searching for an alphabet symbol in S, which will be a difficult task since there are few of them and each occurs a small number of times.

The proof of this lemma is motivated by the proof that $\mathrm{D}(f) \leq \mathrm{C}(f)^2$ for total boolean functions. That proof describes a deterministic algorithm for computing $f(x)$: repeatedly pick 0-certificates consistent with the entries of x seen so far, and query the entries of x corresponding to the non-$*$ entries of that certificate. Since each 0-certificate conflicts with all 1-certificates, each time we do this we reveal a new entry of every 1-certificate. Therefore, after $\mathrm{C}(f)$ iterations, a certificate has been revealed and the value of $f(x)$ has been determined.

Our proof works similarly, except that it is no longer true that each 0-certificate must contradict every 1-certificate on some entry. Instead, it might be possible that a 0-certificate and a 1-certificate disagree on the location of an alphabet element. However, in that case we can conclude that there are a few alphabet elements that are included in all small certificates.

Proof. Fix such T, f, and k. The proof is based on the following algorithm, which either generates the desired p and S or else computes $f(x)$ for a given input x. We will proceed by arguing that the algorithm always generates p and S after at most $4k^2$ queries, which must happen before it computes $f(x)$ when x is the worst-case input (as guaranteed by the requirement that $k \le \frac{1}{2}\sqrt{\mathrm{D}(f)}$). The algorithm is as follows.

```
Get input $x$
Set $p = *^n$, $S = \emptyset$, $R = \emptyset$
loop
    Find any certificate $c$ (consistent with a legal input) that
      - has size at most $k$
      - is consistent with $p$
      - has the property that $c - p$ has no alphabet elements in $S$.
    If there are no such certificates, output $p$ and $S$ and halt.
    Add all the alphabet elements of $c$ to $R$.
    Set $S$ to be the set of elements $i$ of $R$ that occur less than $2k$ times in $\tau(T) - \tau(p)$.
    Query $x$ on all domain elements of $c$ and add the results to $p$.
    If $p$ is a 0-certificate, output "$f(x) = 0$" and halt; if it's a 1-certificate, output
      "$f(x) = 1$" and halt.
```

We claim that this algorithm will go through the loop at most $4k$ times. Indeed, each iteration through the loop selects a certificate. A 0-certificate must conflict with all 1-certificates, and vice versa. There are two ways for certificates to conflict: either they disagree on the value of an entry, or else there is some alphabet element i that they claim to find in different places (and in addition, there must be few unrevealed instances of i in x).

This motivates the following definition: for a certificate c, let $h_{p,S}(c)$ be $|c - p| + |\,\mathrm{alphabet}(c) - S|$ if c is consistent with p, and zero otherwise (here $\mathrm{alphabet}(c)$ denotes the set of alphabet elements occurring in c). Note that at the beginning of the algorithm, $h_{p,S}(c) \le 2|c| \le 2k$ for all certificates c of size at most k. Now, whenever the algorithm considers a 0-certificate c_0, the value of $h_{p,S}(c_1)$ decreases for all 1-certificates c_1 of size at most k (unless it is already 0). This is because either c_0 and c_1 conflict on an input, in which case an entry of c_1 is revealed and included in p, decreasing $|c_1 - p|$ (or contradicting c_1), or else c_0 and c_1 both include an alphabet element i which has less than $2k$ occurrences left to be revealed (if it had at least $2k$ unrevealed occurrences, it wouldn't be the source of a conflict between c_0 and c_1, since they each have size at most k). In the latter case, i is added to S, which decreases $|\,\mathrm{alphabet}(c_1) - S|$.

We have shown that each iteration of the algorithm decreases $h_{p,S}(c)$ either for all 0-certificates or for all 1-certificates (of size at most k). This means that unless the loop is terminated, one of the two values will reach 0 in less than $4k$ iterations. We claim this cannot happen, implying the loop terminates in less than $4k$ iterations.

Suppose by contradiction that $h_{p,S}(c)$ reaches 0 for all 0-certificates. This means p is either a certificate – in which case the value of $f(x)$ was determined, which is a contradiction – or else p is not a certificate, and conflicts with all 0-certificates of size at most k. In the latter case, there is some input y consistent with p such that $f(y) = 0$, and there are no certificates consistent with y of size at most k. Thus $C(f) > k$, contradicting the assumption in the lemma.

This shows the loop always terminates in less than $4k$ iterations, which means it cannot calculate $f(x)$, and must instead output p and S. This gives the desired result, since all certificates of size at most k that are consistent with p have the property that $c - p$ has an alphabet element in S. ◀

Note that if we restrict to inputs consistent with p, then the lemma asserts that finding a small certificate requires finding an element of S. This gives us the following corollary.

▶ **Corollary 11.** *If $f : T \to \{0,1\}$ with $T \subseteq [M]^n$ an orbit, then we have*

$$\mathrm{R}_0(f) = \Omega(\min(\mathrm{D}(f)^{1/2}, n^{1/4})) = \Omega(\mathrm{D}(f)^{1/4}). \tag{3}$$

When $M = n$ and T is the set of permutations, $\mathrm{R}_0(f) = \Omega(\min(\mathrm{D}(f)^{1/2}, n^{1/3})) = \Omega(\mathrm{D}(f)^{1/3})$.

Proof. Fix T and f, and let $k = \lfloor\min(\frac{1}{2}\sqrt{\mathrm{D}(f)}, \frac{1}{4}n^{1/4})\rfloor - 1$ (in the case of permutations, let $k = \lfloor\min(\frac{1}{2}\sqrt{\mathrm{D}(f)}, \frac{1}{4}n^{1/3})\rfloor - 1$). Since a zero-error randomized algorithm must find a certificate, if $k < \mathrm{C}(f)$, the result follows. It remains to treat the case where $k \geq \mathrm{C}(f)$.

In this case, let p and S be as in the lemma. We restrict to inputs consistent with p. Any zero-error randomized algorithm A must find a certificate on such inputs. If A uses $\mathrm{R}_0(f)$ expected queries, then it finds a certificate with probability at least $1/2$ after $2\,\mathrm{R}_0(f)$ queries.

If $2\,\mathrm{R}_0(f) \leq k$, then A must find a certificate of size at most k with probability $1/2$. But this means that on all inputs x, A finds an element of S in x outside p with probability at least $\frac{1}{2}$. However, there are at most $2k|S| = 8k^3$ such elements in the entries of x outside p (in the case of permutations, at most $|S| = 4k^2$ such elements), and the size of the domain is $n - |p| \geq n - 4k^2 \geq \frac{n}{2}$. If x is generated by fixing p and permuting the remaining entries randomly, the chance of a query finding an element of S is at most $\frac{16k^3}{n}$, so by the union bound, the chance of finding such an element after k queries is at most $\frac{16k^4}{n}$ (in the case of permutations, this becomes $\frac{8k^3}{n}$). Since $k < \frac{1}{2}n^{1/4}$ (or $k < \frac{1}{2}n^{1/3}$ in the case of permutations) gives the desired contradiction. ◀

3.3 Lower bounds on R(f) and Q(f)

We now put everything together to prove lower bounds on $\mathrm{R}(f)$ and $\mathrm{Q}(f)$ in terms of $\mathrm{D}(f)$, proving Theorem 5.

▶ **Theorem 12.** *Let $f : T \to \{0,1\}$ with $T \subseteq [M]^n$ an orbit. Then:*

$$\mathrm{R}(f) = \Omega(\min\{\mathrm{D}(f)^{1/6}, n^{1/9}\}) = \Omega(\mathrm{D}(f)^{1/9})\,,$$
$$\mathrm{Q}(f) = \Omega(\min\{\mathrm{D}(f)^{1/12}, n^{1/18}\}) = \Omega(\mathrm{D}(f)^{1/18}).$$

Throughout this proof, we will identify a partial assignment $q \in ([M] \cup \{*\})^n$ with the set $\{(i, q_i) : i \in \mathbb{N}, q_i \neq *\}$. This will allow us to use set theoretic notation to manipulate partial assignments; for example, if q and q' are consistent partial assignments, then $q \cup q'$, $q \cap q'$, and $q - q'$ are all also a partial assignments.

Proof of Theorem 12. Apply Lemma 10 with $k = \min\{\frac{1}{4}\sqrt{\mathrm{D}(f)}, \frac{1}{3}n^{1/3}\}$. If $\mathrm{C}(f) > k$, then we're done by Corollary 9. Otherwise, we get p and S from the lemma, with all certificates of size at most k that are consistent with p having alphabet elements in S.

We use Ambainis's adversary method (see Appendix A) to get a lower bound for $\mathrm{Q}(f)$, which will look very similar to the lower bound for permutation inversion found in [4]. In order to apply the adversary method, we use p and S to find some specific inputs to our function f.

First, let Y be the multiset of alphabet symbols in $\tau(T) - \tau(p)$, except for the alphabet symbols found in S. Note that there are at most $4k^2$ alphabet symbols in S, and each occurs at most $2k$ times outside of p. Since $|\tau(T)| = n$ and $|\tau(p)| \leq 4k^2$, we have $|Y| \geq n - 4k^2 - 8k^3 \geq n - 12k^3 \geq n/2$.

We now run the following procedure.

We now describe the modification in step 5. We make c consistent with p in two sub-steps: in step A, we expand c by setting $c_i = \gamma$ for various choices of $i \in [n]$ and $\gamma \in [M]$; and in

```
1: Initialize the partial assignment $r = *^n$.
2: Initialize the multiset $Z = Y$.
3: while $|r| \leq k$ do
4:     Pick any 0-certificate $c$ consistent with $r$ of size at most $\mathrm{C}(f)$.
5:     Modify $c$ to get $c'$ consistent with $p$ and $r$, as described below.
6:     Replace any alphabet symbols of $c' - p$ that are in $S$ by symbols from $Z$ to get $c''$.
7:     Update $Z$ by removing the used symbols from it.
8:     Add the entries of $c''$ to $r$.
9:     If $|r| > k$, stop. Otherwise, repeat steps 4-8 for a 1-certificate.
```

step B, we permute the non-$*$ entries of $c - r$. The choices of i in step A will always be entries with $c_i = r_i = *$, and the choices of γ will always be alphabet symbols from either Z or $\tau(p - r)$. When we use symbols from Z, we also update Z to remove those symbols.

Explicitly, we do the following. First, note that c is consistent with r and r is consistent with p. Let $d = c - r$ and let $q = p - r$. We will modify d to make it consistent with q. First, for each symbol γ in the multiset $\tau(d) \cap \tau(q)$, we ensure there is a distinct entry i such that $d_i \neq *$ and $q_i = \gamma$. If there isn't one, we pick i with $q_i = \gamma$ and $d_i = *$ and set c_i to an element of Z (and remove that element from Z). This step ensures that all alphabet elements of d that "must be" part of q can be placed inside q by permuting the non-$*$ entries of d. Next, for each i such that $d_i \neq *$ and $q_i \neq *$, we ensure there is a distinct j such that $d_j = q_i$. If there isn't one, we pick j such that $r_j = p_j = q_j = d_j = c_j = *$ and set $c_j = q_i$.

It is not hard to see that after these additions to c, we can permute the non-$*$ entries of $c - r$ to make it consistent with $p - r$: we can ensure the intersection of non-$*$ entries of $p - r$ and $c - r$ get filled with the correct alphabet symbols, and doing this also ensures that the only alphabet symbols used in the remainder of the partial assignment are not necessary for p. Hence we get c' consistent with p and r. c' was formed by increasing the size of c' by at most $2|c|$, so $|c'| \leq 3|c| \leq 3\,\mathrm{C}(f)$.

We note a few invariants of this algorithm. The first invariant is that any alphabet symbols of $c' - p$ that are in S do not occur in $\tau(p - c')$. This means that after the current iteration, these symbols will not occur in $\tau(p - r)$, so they will be swapped for elements of Z whenever they occur outside of r.

The second invariant is that r is consistent with p and that $r - p$ uses no alphabet elements in S. By Lemma 10, r is not a certificate as long as $|r| \leq k$. Hence step 4 of the algorithm (where a certificate is chosen) never fails. Moreover, each iteration of the algorithm increases $|r|$ by at most $6\,\mathrm{C}(f)$. Thus the loop repeats at least $\frac{k}{6\,\mathrm{C}(f)}$ times.

Consider the entries of r that were added by the selection of 0-certificates. Let the first α of them be the partial assignments $a_1^{(0)}, a_2^{(0)}, \ldots, a_\alpha^{(0)}$, with $\alpha = \lfloor \frac{k}{6\,\mathrm{C}(f)} \rfloor$. Each $a_i^{(0)}$ is equal to $c'' - r$ at the i-th round of the algorithm. Similarly, let the subsets of r that were added by 1-certificates be $a_1^{(1)}, a_2^{(1)}, \ldots, a_\alpha^{(1)}$.

Let W be the multiset $\tau(T) - \tau(p)$ restricted to S; that is, the collection of alphabet symbols in S outside of p. Note that if some of the non-$*$ alphabet symbols in $a_i^{(0)}$ were replaced by some symbols from S and the non-star entries of $a_i^{(0)}$ were permuted, we would get a 0-certificate, and similarly for $a_i^{(1)}$. By the first invariant, it is actually sufficient to replace the alphabet symbols of $a_i^{(0)}$ or $a_i^{(1)}$ by those from the multiset W. We use this fact to construct the sets for the adversary method.

Let A be the sub-multiset of W consisting of the symbols in W that actually need to be swapped in for some $a_i^{(0)}$ or $a_i^{(1)}$. Since the total size of the $a_i^{(0)}$ and $a_i^{(1)}$ sets is $|r| \leq k$, we have $|A| \leq k$.

To each $a_i^{(j)}$, we add an arbitrary block of $|A|$ entries outside r with alphabet symbols from Y. To be able to do this, we require that $2|A|\alpha \leq n - |r| - 2k|A|$ (the $2k|A|$ term appears because each alphabet element in A may occur up to $2k$ times), and also that $|Y| - |r| \geq 2|A|\alpha$. Since $|r| \leq k$, $|A| \leq k$, $\alpha \leq k/6$, $|p| \leq 4k^2$, and $|Y| \geq n/2$, and since $k \leq n^{1/3}/3$, it is easy to check that these conditions hold.

Now, for each $j \in \{0,1\}$ and each $i = 1, 2, \ldots, \alpha$, we can place all the alphabet elements of A inside $a_i^{(j)}$ and permute its non-$*$ entries in a way that restores the j-certificate. We can thus generate 2α inputs, α of which have value 0 and α of which have value 1, such that the only difference between the inputs is which of the 2α disjoint bins has the alphabet elements of A (and has been shuffled). This is essentially a version of permutation inversion.

It's clear that a classical randomized algorithm must make $\Omega(\alpha)$ queries, since it must find the special bin containing the alphabet elements of A. For the quantum lower bound, we use Theorem 17. Let X be the set of inputs in which the elements of A were placed for a 0-certificate bin, and let Z be the set of inputs in which the elements of A were placed for a 1-certificate bin. Our relation R will simply be $X \times Z$. Then each element of X has α neighbors in Z, and vice versa. However, for each entry t and $(x,y) \in R$, we have $l_{x,t} = 1$ or $l_{y,t} = 1$, so $l_{x,t} l_{y,t} \leq \alpha$. Thus we get a quantum lower bound of $\Omega(\sqrt{\alpha})$.

Finally, to complete the proof, we note that $\alpha = \Omega(\min(\frac{n}{k}, \frac{k}{\mathrm{C}(f)})) = \Omega(\frac{k}{\mathrm{C}(f)})$ (since $n \geq k^2$), so that, combining with Corollary 9, $\mathrm{R}(f) = \Omega(\beta)$ and $\mathrm{Q}(f) = \Omega(\sqrt{\beta})$ with $\beta = \max(\sqrt{\mathrm{C}(f)}, \frac{k}{\mathrm{C}(f)})$. Note that this satisfies $\beta = \Omega(k^{1/3})$. This gives:

$$\mathrm{R}(f) = \Omega(\min\{\mathrm{D}(f)^{1/6}, n^{1/9}\}),$$
$$\mathrm{Q}(f) = \Omega(\min\{\mathrm{D}(f)^{1/12}, n^{1/18}\}). \quad ◀$$

4 Symmetric Promises with Small Alphabets

In this section, we show a polynomial relationship between $\mathrm{Q}(f)$ and $\mathrm{R}(f)$ for any function on a symmetric promise whose alphabet size is constant, proving Theorem 6. We will use the term symmetric to refer to invariance under permutation of the indices of the inputs. That is, a promise X is symmetric if $x_\sigma \in X$ for all $x \in X$ and $\sigma \in S_n$, where $x_\sigma = (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$, and a function $f : X \to \{0,1\}$ is symmetric if X is symmetric and $f(x) = f(x_\sigma)$ for all $x \in X$ and $\sigma \in S_n$.

4.1 The case of Symmetric Functions

We start by dealing with the case where the function f is itself symmetric.

▶ **Theorem 13.** *Let $f : X \to \{0,1\}$ be a symmetric function. Then $\mathrm{Q}(f) = \Omega\left(\frac{\mathrm{R}(f)^{1/8}}{M \log^{1/8} M}\right)$.*

To prove this theorem, we relate $\mathrm{Q}(f)$ and $\mathrm{R}(f)$ to $g(f)$, which we now define.

▶ **Definition 14.** If T_1, T_2 are orbits on alphabet $[M]$, the distance $d(T_1, T_2)$ between T_1 and T_2 is the maximum over all $i \in [M]$ of the difference between the multiplicity of i in T_1 and the multiplicity of i in T_2. If $f : X \to \{0,1\}$ is a symmetric function, let $d(f)$ be the minimum of $d(T_1, T_2)$ for orbits $T_1, T_2 \subseteq X$ with different values under f. Define $g(f) := \frac{n}{d(f)}$.

We proceed to prove lemmas relating $g(f)$ to $\mathrm{R}(f)$ and $\mathrm{Q}(f)$ to $g(f)$.

▶ **Lemma 15.** *For any $x \in [M]^n$, $O(\frac{n^2 \log M}{d^2})$ queries suffice to find an orbit T such that $d(T, \tau(x)) < d$ with probability at least $\frac{2}{3}$ (where $\tau(x)$ denotes the orbit of x). Hence, if $f : X \to \{0,1\}$ is symmetric, $\mathrm{R}(f) = O(g(f)^2 \log M)$.*

Proof. We describe a classical randomized algorithm for estimating the orbit of input x. The algorithm is simply the basic sampling procedure that queries random entries of x and keeps track of the number r_i of times each alphabet element i was observed. The orbit T is then formed by $T(i) = \text{round}\left(\frac{r_i}{\sum_{i\in[M]} r_i} n\right)$, where the rounding operation sometimes rounds up and sometimes down, in order to preserve the sum of $T(i)$ being n. Note that the ratios $T(i)/n$ are within $1/n$ of the observed frequency of i.

Let the orbit of x be $\tau(x) = (t_1, t_2, \dots, t_M)$, so that the multiplicity of alphabet element i in $\tau(x)$ is t_i. A version of the Chernoff bound states that if we have $k \ge \frac{3}{\epsilon^2} \ln \frac{2}{\delta}$ samples estimating the proportion p of the population with some property, the proportion of the sample with that property is in $(p-\epsilon, p+\epsilon)$ with probability at least $1-\delta$.

Suppose $d \ge 2$. Setting $\epsilon = \frac{d-1}{n}$ and $\delta = \frac{1}{3M}$, we see that $O(\frac{n^2 \log(M)}{d^2})$ samples suffice for $\frac{T(i)}{n}$ to be within $\frac{d}{n}$ of $\frac{t_i}{n}$ with probability at least $1 - \frac{1}{3M}$. In other words, we have $|T(i) - t_i| < d$ with probability $1 - \frac{1}{3M}$ for each i. The union bound then gives us $|T(i) - t_i| < d$ for all i with probability at least $\frac{2}{3}$. This shows that $d(T, \tau(x)) < d$, as desired.

When $d = 1$, we set $\epsilon = \frac{1}{2n}$ and $\delta = \frac{1}{3M}$ to get frequency ratios within $\frac{1}{2n}$ of the true values $\frac{t_i}{n}$ with probability at least $2/3$. The closest integer orbit to the observed frequency ratios will then be exactly correct with probability at least $2/3$.

To compute $f(x)$ for symmetric f, a randomized algorithm can estimate the orbit of x to within $\frac{d(f)}{2}$, and then just output the value of f on any input of the orbit within $\frac{d(f)}{2}$ of the estimated orbit T. Since $g(f) = \frac{n}{d(f)}$, we get $\mathrm{R}(f) = O(g(f)^2 \log M)$. ◀

▶ **Lemma 16.** *If $f : X \to \{0,1\}$ is symmetric, then $\mathrm{Q}(f) = \Omega(g(f)^{1/4}/M^2)$.*

Proof. Let S and T be orbits with distance $d(f)$ such that if x has orbit S and y has orbit T then $f(x) \ne f(y)$. We claim that a quantum algorithm cannot distinguish between these orbits in less than the desired number of queries.

We proceed by a hybrid argument. We form a sequence of orbits $\{S_i\}_{i=0}^k$ with $k \le M$ such that $S_0 = S$, $S_k = T$, and for all $i = 0, 1, \dots, k-1$, the orbits S_i and S_{i+1} differ in the multiplicity of at most 2 alphabet elements and have distance at most $d(f)$.

We do this as follows. Set $S_0 = S$. Let A be the set of alphabet elements on whose multiplicities the current S_i agrees with T; at the beginning, A is the set of alphabet elements on which S and T have the same multiplicity, which may be empty. To construct S_{i+1} given S_i, we simply pick an alphabet element r for which S_i has a larger multiplicity than T and an alphabet element r' for which S_i has a smaller multiplicity than T. We then set S_{i+1} to have the same multiplicities as S_i, except that the multiplicity of r is reduced to that in T and the multiplicity of r' is increased to make up the difference. Note that the multiplicity of r is then equal in S_i and T, so r gets added to A. Moreover, note that $d(S_i, S_{i+1}) \le d(S_i, T)$, and also $d(S_{i+1}, T) \le d(S_i, T)$. Since this is true for all i, it follows that $d(S_i, S_{i+1}) \le d(S, T) = d(f)$.

Since an alphabet element gets added to A each time and the elements are never removed, this procedure is terminated with $S_k = T$ after at most M steps. Thus $k \le M$. Also, consecutive orbits differ in the multiplicities of 2 elements and have distance at most $d(f)$.

We now give a lower bound on the quantum query complexity of distinguishing S_i from S_{i+1}. Without loss of generality, let the alphabet elements for which S_i and S_{i+1} differ be 0 and 1, with 0 having a smaller multiplicity in S_i. Let a be the multiplicity of 0 in S_i, and let b be the multiplicity of 1 in S_i, with $0 < b - a \le d(f)$. Let c and d be the multiplicities of 0 and 1 in S_{i+1}, respectively. Then $c + d = a + b$. Let $e = a + b = c + d$.

We prove two lower bounds using Ambainis's adversary method, corresponding to e being either large or small. For the small case, consider an input x of orbit S_i split into $2\alpha = \lfloor \frac{n}{e} \rfloor$

blocks $B_1, B_2, \ldots, B_{2\alpha}$ of size e each, such that all the 0 and 1 elements lie in block B_1. To change the input from orbit S_i to S_{i+1}, we must simply change the first block. Also, note that rearranging the blocks does not change the orbit. Let X be the set of inputs given by rearranging the blocks of x so that the block B_1 ends up in the first α blocks, and let Y be the set of inputs given by replacing B_1 to get orbit S_{i+1} and then rearranging the blocks so that B_1 ends up in the last α blocks. We now have a reduction from permutation inversion, so using Ambainis's adversary method, we get a lower bound of $\Omega(\sqrt{\alpha}) = \Omega(\sqrt{\frac{n}{e}})$.

For the case when e is big, we restrict to inputs in which all elements are fixed except for those with value 0 or 1. A lower bound of $\Omega(\sqrt{e/d(f)})$ then follows from the appendix of [1].

If $e \leq \sqrt{nd(f)}$, the former bound gives a lower bound of $\Omega((\frac{n}{d(f)})^{1/4})$ for distinguishing S_i from S_{i+1} by quantum queries. If $e \geq \sqrt{nd(f)}$, the latter bound gives the same. Thus we have a lower bound of $\Omega(g(f)^{1/4})$ in all cases.

Finally, note that if a quantum algorithm could compute $f(x)$ in $\mathrm{Q}(f)$ queries, then for some i it could distinguish S_i from S_{i+1} with probability $\Omega(\frac{1}{M})$. Thus we could use $M^2\,\mathrm{Q}(f)$ queries to distinguish S_i from S_{i+1} with constant probability, so $\mathrm{Q}(f) = \Omega(g(f)^{1/4}/M^2)$. ◀

These two lemmas combine to prove Theorem 13.

4.2 The General Case

We now prove Theorem 6. The proof proceeds by describing a classical algorithm that doesn't use too many more queries than the best quantum algorithm. An interesting observation is that this classical algorithm is mostly deterministic, and uses only $O(\mathrm{Q}(f)^8 M^{16} \log M)$ randomized queries at the beginning (to estimate the orbit of the input).

For this proof, we will often deal with certificates c for f that only work on inputs of some specific orbit S; that is, all inputs $x \in X$ of orbit S that are consistent with c have the same value under f. We will say c is a certificate for the orbit S.

Proof. Let f be a function. We describe a classical algorithm for computing f on an input x, and argue that a quantum algorithm cannot do much better.

As a first step, the algorithm will estimate the orbit of x using $O(\mathrm{Q}(f)^8 M^{16} \log M)$ queries. By Lemma 15, this will provide an orbit T such that $d(T, \tau(x)) < \frac{n}{CM^8\,\mathrm{Q}(f)^4}$ with high probability, where we choose the constant C to be larger than twice the asymptotic constant in Lemma 16. We restrict our attention to orbits that are within $\frac{n}{CM^8\,\mathrm{Q}(f)^4}$ of T.

Now, notice that if we fix an orbit S and assume that x has this orbit, then there is a deterministic algorithm that determines the value of $f(x)$ in at most α steps, where $\alpha = O(\mathrm{Q}(f)^{18})$. Since this is a deterministic algorithm, it must find a certificate of size at most α for the orbit S. The only other possibility is that the deterministic algorithm finds a partial assignment that contradicts the orbit S, in which case it cannot proceed. Running this deterministic algorithm on orbit S will be called *examining* S.

Note further that we can assume we never find a 0-certificate c_0 for some orbit S_0 and a 1-certificate c_1 for some other orbit S_1 without the certificates contradicting either orbit. This is because if we found such certificates, then we can lower bound $\mathrm{Q}(f)$ restricting the function to inputs that agree with c_0 and c_1 and have orbit equal to either S_0 or S_1. The quantum algorithm for f would then have to distinguish between these orbits, which is equivalent to distinguishing between orbits with multisets $S_0 - (\tau(c_0) \cup \tau(c_1))$ and $S_1 - (\tau(c_0) \cup \tau(c_1))$ on inputs of size $n - |c_0 \cup c_1|$. These orbits have distance at most $\frac{n}{CM^8\,\mathrm{Q}(f)^4}$. Since $n > 4\alpha$ (or else $\mathrm{R}(f) = O(n) = O(\mathrm{Q}(f)^{18})$), we have $n - |c_0 \cup c_1| > \frac{n}{2}$, and $(\frac{n}{2})/(\frac{n}{CM^8\,\mathrm{Q}(f)^4}) = \frac{CM^8\,\mathrm{Q}(f)^4}{2}$;

then Lemma 16 together with the choice of c imply that a quantum algorithm takes more than $Q(f)$ queries to distinguish these orbits, a contradiction.

For an orbit S, we now define $v(S) \in [2\alpha+1]^M$ to be the vector with $v(S)_i = \min(S(i), 2\alpha)$ for all i, where $S(i)$ is the multiplicity of i in the orbit S. If an input has orbit S, we call $v(S)$ the simplified orbit of the input. We consider the partial order on simplified orbits given by $v(S) \geq v(R)$ if and only if $v(S)_i \geq v(R)_i$ for all $i = 1, 2, \dots M$. We say a simplified orbit $v(S)$ is maximal if it is maximal in this partial order.

The algorithm proceeds by finding the set of maximal simplified orbits, and selecting a representative orbit S for each maximal simplified orbit v so that $v(S) = v$. Let the orbits selected this way be $S_1, S_2, \dots, S_\beta$. For each S_i, we then run the deterministic algorithm that uses α queries assuming orbit S_i. Let c_i be the set of queries made by this algorithm for orbit S_i. Note that the total number of queries made this way is at most $\alpha\beta$.

For each S_i, the partial assignment c_i is either a certificate for S_i or a disproof of the orbit S_i. Consider the pairwise unions $c_i \cup c_j$. We restrict our attention to the orbits S_i that are consistent with $c_i \cup c_j$ for all j. We claim that there is at least one such orbit. Indeed, if T is the true orbit of the input, then $v \geq v(T)$ for some maximal simplified orbit v, and $v(S_k) = v$ for some k. Then S_k cannot be disproven in 2α queries, as that would disprove v and therefore $v(T)$ as well.

Now, let S_i and S_j be any two orbits remaining. Then they are both consistent with $c_i \cup c_j$. As we saw earlier, we cannot have c_i be a 0-certificate for S_i and c_j be a 1-certificate for S_j (or vice versa); the certificates c_i and c_j must agree. We conclude that the certificates c_i for the remaining orbits are either all 0-certificates (for their respective orbits) or all 1-certificates. Our algorithm will then output 0 in the former case and 1 in the latter.

To see that the algorithm is correct, recall that S_k is one of the remaining orbits, with $v(S_k) = v \geq v(T)$. Without loss of generality, suppose the algorithm output 0, so that c_k is a 0-certificate. Suppose by contradiction that $f(x) = 1$ for the our input. Let c be a 1-certificate consistent with x of size at most α. Then c is a 1-certificate for the orbit T. Now, $c \cup c_k$ cannot disprove $v(T)$ (since it has size at most 2α), so $c \cup c_k$ cannot disprove T. Since $c \cup c_k$ cannot disprove $v(T)$, it also cannot disprove v, so it cannot disprove S_k. This means T and S_k are not disproven by their 0- and 1-certificates, which we've shown is a contradiction. Thus if the algorithm outputs 0, we must have $f(x) = 0$ as well.

The total number of queries required is $O(\mathrm{Q}(f)^8 M^8 \log M) + \alpha\beta$, where $\alpha = O(\mathrm{Q}(f)^{18})$. We must estimate β, the number of maximal simplified orbits. This is at most the number of maximal elements in $[2\alpha + 1]^M$ in our partial order. We can show by induction that this is at most $(2\alpha + 1)^{M-1}$: in the base case of $M = 1$, the value is 1, and when M increases by 1 the number of maximal elements can increase by at most a factor of $(2\alpha + 1)$. This gives a final bound of $O(\mathrm{Q}(f)^{18M})$ on the number of queries when M is constant.

To reduce this to $O(\mathrm{Q}(f)^{18(M-1)})$, we note that some alphabet element a must occur at least n/M times in T, by the pigeonhole principle. We could then use $O(M\alpha)$ queries to find 2α instances of a with high probability. Then each simplified orbit v will have $v_a = 2\alpha$, so the simplified orbits are effectively elements of $[2\alpha + 1]^{M-1}$ instead of $[2\alpha + 1]^M$. This decreases β to $(2\alpha+1)^{M-2}$, so the total number of queries decreases to $O(\mathrm{Q}(f)^{18(M-1)})$. ◀

References

1 Scott Aaronson and Andris Ambainis. The need for structure in quantum speedups. *arXiv preprint* arXiv:0911.0996, 2009.

2 Scott Aaronson and Andris Ambainis. Forrelation: A problem that optimally separates quantum from classical computing. In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC 2015)*, pages 307–316, 2015. doi:10.1145/2746539.2746547.

3 Scott Aaronson and Shalev Ben-David. Sculpting quantum speedups. *arXiv preprint* arXiv:1512.04016, 2015.

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

5 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. arXiv:arXiv:quant-ph/9802049, doi:10.1145/502090.502097.

6 Joe P Buhler, Hendrik W Lenstra Jr, and Carl Pomerance. Factoring integers with the number field sieve. In *The development of the number field sieve*, pages 50–94. Springer, 1993.

7 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

8 Richard Cleve. The query complexity of order-finding. *Information and Computation*, 192(2):162–171, 2004.

9 Ashwin Nayak. Inverting a permutation is as hard as unordered search. *arXiv preprint* arXiv:1007.2899, 2010.

10 Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. arXiv:quant-ph/9508027.

11 Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. doi:10.1137/S0097539796298637.

A The Quantum Adversary Method

▶ **Theorem 17.** *Let $f : X \to \{0,1\}$ with $X \subseteq [M]^n$. Let $A, B \subseteq X$ be such that $f(a) = 0$ for all $a \in A$ and $f(b) = 1$ for all $b \in B$. Let $R \subseteq A \times B$ be such that*

1. *For each $a \in A$, there exist at least m different $b \in B$ such that $(a,b) \in R$.*
2. *For each $b \in B$, there exist at least m' different $a \in A$ such that $(a,b) \in R$.*

Let $l_{a,i}$ be the number of $b \in B$ such that $(a,b) \in R$ and $a_i \neq b_i$. Let $l_{b,i}$ be the number of $a \in A$ such that $(a,b) \in R$ and $a_i \neq b_i$. Let l_{max} be the maximum of $l_{a,i}l_{b,i}$ over all $(a,b) \in R$ and $i \in \{1,2,\ldots,n\}$ such that $a_i \neq b_i$. Then $Q(f) = \Omega\left(\sqrt{\frac{mm'}{l_{max}}}\right)$.