Graph Properties and Circular Functions: How Low Can Quantum Query Complexity Go?*

Xiaoming Sun
Computer Science Department,
Tsinghua University,
Beijing, 100084, P. R. China.
sun_xm97@mails.tsinghua.edu.cn

Andrew C. Yao
Computer Science Department,
Princeton University,
NJ 08544, USA.
yao@cs.princeton.edu

Shengyu Zhang
Computer Science Department,
Princeton University,
NJ 08544, USA.
szhang@cs.princeton.edu

Abstract

In decision tree models, considerable attention has been paid on the effect of symmetry on computational complexity. That is, for a permutation group Γ, how low can the complexity be for any boolean function invariant under Γ? In this paper we investigate this question for quantum decision trees for graph properties, directed graph properties, and circular functions. In particular, we prove that the n-vertex Scorpion graph property has quantum query complexity $\tilde{\Theta}(n^{1/2})$, which implies that the minimum quantum complexity for graph properties is strictly less than that for monotone graph properties (known to be $\Omega(n^{2/3})$). A directed graph property, SINK*, is also shown to have the $\tilde{\Theta}(n^{1/2})$ quantum query complexity. Furthermore, we give an N-ary circular function which has the quantum query complexity $\tilde{\Theta}(N^{1/4})$. Finally, we show that for any permutation group Γ, as long as Γ is transitive, the quantum query complexity of any function invariant to Γ is at least $\Omega(N^{1/4})$, which implies that our examples are (almost) the best ones in the sense of pinning down the complexity for the corresponding permutation group.*

1. Introduction

In classical decision trees, considerable attention has been paid on the effect of symmetry on computational complexity. The most famous example is the confirmation of the Aanderra-Rosenberg Conjecture that all non-constant monotone graph properties on n vertices have (deterministic) decision tree complexity $\Theta(n^2)$ [11]. There are however many intriguing questions that remain open, in classical as well as quantum decision trees. The purpose of this paper is to fill some of these gaps in the area of quantum complexity.

Let Γ be a group of permutations of $1, 2, \cdots, N$. Let $f : \{0,1\}^N \to \{0,1\}$ be any boolean function *invariant* under Γ, i.e., $f(x_1, x_2, \cdots, x_N) = f(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(N)})$ for all $\sigma \in \Gamma$. We are interested in how low the complexity can be for such functions. In the case of classical decision trees, let D_Γ be the minimum deterministic complexity for any non-constant boolean function invariant under Γ, and let $D_\Gamma^{(M)}$ be the minimum deterministic complexity for any non-constant *monotone* boolean function invariant under Γ. For quantum decision trees, let Q_Γ and $Q_\Gamma^{(M)}$ be the corresponding minimum complexities (allowing ϵ two-sided errors).

Much is known about D_Γ and $D_\Gamma^{(M)}$ when Γ is a transitive group. See Section 4 for the definition of transitive groups. In particular, the Aanderra-Rosenberg Conjecture mentioned above can be phrased as $D_\Gamma^{(M)} = \Theta(N)$, where $N = \binom{n}{2}$ and Γ is the permutation group on the N edges of an n-vertex graph induced by the relabelling of the n vertices. It also has been proved that, for the same Γ, $D_\Gamma = \Theta(N^{1/2})$. We remark that it was not easy to construct a graph property with $O(N^{1/2})$ complexity, and the first non-trivial such example found was the property of being Scorpion graphs (see Section 2 for definition).

Less is known about Q_Γ and $Q_\Gamma^{(M)}$. For the case when Γ

* This research was supported in part by NSF grant CCR-0310466.

is the set of all permutations on $1, 2, \cdots, N$, it is known that $Q_\Gamma, Q_\Gamma^{(M)} = \Theta(\sqrt{N})$ [3]. It is generally believed that, for the Γ corresponding to graph properties, $Q_\Gamma^{(M)} = \Theta(N^{1/2})$. Clearly, $O(N^{1/2})$ is an upper bound, and so far the best lower bound (by Santha and Yao (unpublished)) is $\Omega(N^{1/3})$. There does not seem to be published results on Q_Γ, and at first glance, it appears that Q_Γ may even be $\Theta(N^{1/2})$. The main result of this paper is to pin down Q_Γ.

Theorem 1 *For the Γ corresponding to graph properties, $Q_\Gamma = \tilde{\Theta}(N^{1/4})$.*

It is of interest to note that this is strictly below the quantum complexity of any non-constant *monotone* graph properties. The graph property used to establish the upper bound in Theorem 1 is the scorpion graph property. It is perhaps a natural choice, since scorpion graph property has low classical complexity, but the proof is nontrivial.

We have also obtained results for several other Γ. Let Γ_{directed} be the group of permutations of $1, 2, \cdots, N$ corresponding to the directed graph properties on n vertices (where $N = n(n-1)$). Let Γ_{circular} be the group of all cyclic shifts of $1, 2, \cdots, N$.

Theorem 2 $Q_{\Gamma_{directed}} = \tilde{\Theta}(N^{1/4})$.

Theorem 3 $Q_{\Gamma_{circular}} = \tilde{\Theta}(N^{1/4})$.

The upper bound for Theorem 3 is established by a specially constructed boolean function. In contrast, the upper bounds for Theorems 1 and 2 are achieved by some well-known boolean functions.

The lower bound part of all the above 3 theorems can actually be generalized to the general transitive permutation group case.

Theorem 4 *For any transitive group Γ, $Q_\Gamma = \Omega(N^{1/4})$.*

We prove the upper bound parts of Theorems 1 and 2 in Section 2. The upper bound part of Theorem 3 is proved in Section 3, and Theorem 4, which implies the lower bound parts of the first three theorems, is shown in Section 4.

2. Proof of the upper bound parts of Theorems 1 and 2

To prove the upper bound part of Theorems 1 and 2, we give $\tilde{O}(N^{1/4})$ algorithms for a graph property SCORPION and a directed graph property SINK, two problems classically interesting because their classical decision tree complexity is $O(n)$ [4, 5]. Note that $N = \binom{n}{2}$ for graph properties and $N = n(n-1)$ for directed graph properties, where n is the number of vertices. (We remark that instead of SINK, one can also use a directed graph version of SCORPION.) The definitions of Scorpion graphs and Sink graphs are as follows.

Definition 1 *An n-vertex undirected graph G is a Scorpion if there are three special vertices called Body, Tail and Sting, whose degrees are $n-2$, 2 and 1, respectively. Furthermore, the only vertex that Body does not connect to is Sting, and the only neighbor of Sting is Tail. (See Figure 1) We call any vertex other than these three special ones a Foot. Between any pair of Feet, there may or may not be an edge.*

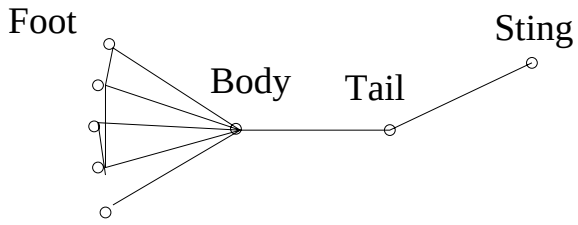

Figure 1. A Scorpion graph

An n-vertex directed graph G is a Sink if there is a special vertex with out-degree 0 and in-degree $n-1$. (See Figure 2)

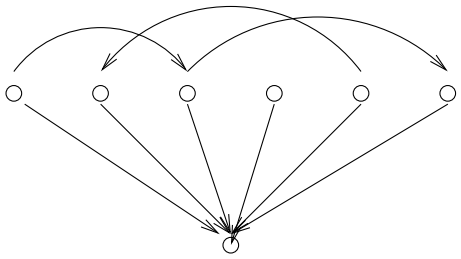

Figure 2. A Sink graph

Both the properties have classical decision tree complexity of $\Theta(n)$. The algorithm for SINK is as follows. From an arbitrary vertex v, do a depth first search. The first time we cannot find a new vertex, we get a candidate sink vertex. Check whether this candidate has out-degree 0 and in-degree $n-1$. Output "Sink" if yes and "not Sink" otherwise. The algorithm for SCORPION is a little more complicated, and we refer to [4, 5] for details of algorithms for these two properties.

We give quantum algorithms for SCORPION. Similar algorithms work for SINK too, and are omitted in this extended abstract. We first make some basic observations. The first one is about Grover search [8] of bounded error input by Hoyer, Mosca and de Wolf [9].

Lemma 5 *(Hoyer, Mosca, de Wolf [9]) Given n elements, some of which may be marked, we wish to decide whether*

there is a marked element and locate one if any exists. Suppose that for any given element, one can decide with error probability less than $1/3$ whether it is marked, by a q-quantum-query subroutine. Then we can decide whether there exists a marked element (and output one if there is any) using an quantum algorithm with expected time and query complexity $E = O(\sqrt{n/mq})$, where m is the unknown number of marked elements.

The second lemma considers to find multiple marked elements rather than just one. Suppose we do not know the number m of marked elements, and are now required to find k marked ones. We use the following algorithm.

1. **for** $i = 0$ to $k-1$
2. **for** $j = 0$ to $\log(k/\epsilon)$
3. run the algorithm in Lemma 5 for $2E_i$ long time,
4. **if** we find a marked element during the time
5. go to line 1 with i incremented
6. output $m = i$ and halt.

In line 3, the E_i is the expected time of the algorithm in Lemma 5 in the i-th iteration, and $E_i = O(q\sqrt{\frac{n}{m-i}})$.

We give a brief analysis of this algorithm. If $k \leq m$, then in the i-th outer iteration, the expected time to find a marked element is $\sum_{t=1}^{log(k/\epsilon)} 2tE_i/2^t < 4E_i = O(kq\sqrt{\frac{n}{m-i}})$ and the error probability is $2^{-t} = \epsilon/k$. So the expected value of the whole running time is

$$\begin{aligned} & O(q(\sqrt{n/m} + \sqrt{n/(m-1)} + ... + \sqrt{n/(m-k+1)})) \\ = \; & O(q\sqrt{n} \cdot 2(\sqrt{m} - \sqrt{m-k})) \\ = \; & O(q\sqrt{n}k/\sqrt{m}) = O(kq\sqrt{n/m}) \\ = \; & O(q\sqrt{kn}) \end{aligned}$$

and the whole error probability is less than ϵ. If $k > m$, then for the first m marked elements the analysis is the same as above and the expected time complexity is $O(q\sqrt{mn})$. After we find m marked elements in the $i = 0, ..., m-1$ iterations, we shall not find any marked element in the $i = m$ iteration, thus output m and halt by executing line 6. This iteration need $O(q\sqrt{n}\log(k/\epsilon))$ time and the error probability is ϵ/k. Thus the whole time complexity in $k > m$ case is $O(q\sqrt{mn} + q\sqrt{n}\log(k/\epsilon))$ and the error probability is less than ϵ. In summary, we have the following lemma.

Lemma 6 *The above algorithm outputs k marked elements if $k \leq m$, and outputs m and the m marked elements when $k > m$. In both cases the algorithm is correct with probability $1-\epsilon$, and the time/query complexity is $O(q\sqrt{\min\{k,m\}n} + q\sqrt{n}\log(k/\epsilon))$.*

The third lemma is a basic observation that if we find a candidate for Body, Tail or Sting, then we can easily decide whether the given graph is a Scorpion or not.

Lemma 7 *Given a graph G and a vertex v, We can check whether G is a Scorpion graph with v being Body, Tail or Sting with error probability ϵ by $O(\sqrt{n}\log(1/\epsilon))$ queries.*

Proof We first check whether the degree of v is 1, 2, or n-2 by $O(\sqrt{n}log(2/\epsilon))$ queries.

- Case 1: $deg(v) = 1$. We use Grover search to find the only neighbor u of v. Check that $deg(u) = 2$ and then find the other neighbor w of u. Finally, check that $deg(w) = n-2$. Output YES and w, u, v as Body, Tail and Sting and then halt if and only if G passes all these checks, and NO otherwise.
- Case 2: $deg(v) = n-2$. We use Grover search to find the only vertex u which is not adjacent with v. Check $deg(u) = 1$ and then the rest part is similar with Case 1.
- Case 3: $deg(v) = 2$. Find the neighbor u, w of v. Check that $deg(u) = 1$ or $deg(u) = n-2$. Then similar as Case 1 or Case 2.

In all the three cases we should use Grover search for $\Theta(\log(1/\epsilon))$ times to let the error probability decrease to $\epsilon/2$, and thus the whole error probability is less than ϵ. □

The last lemma is the key idea of the algorithms in this paper. Basically, it uses random sampling to find a vertex with low degree.

Definition 2 *For any vertex v in an undirected graph $G = (V, E)$ and any set $U \subseteq V$, define the set of neighbors of v in U as $N_U(v) = \{u : (u, v) \in E, u \in U\}$. We write $N(v)$ for $N_V(v)$.*

Lemma 8 *In a graph $G = (V, E)$, if we pick a random set T of $2\frac{u}{d}\log n$ vertices from a set U of u vertices, then we have*

$$\mathbf{Pr}[\forall v \in V,\ if\ N(v) \cap T = \phi,\ then\ |N_U(v)| \leq d] > 1 - 1/n.$$

This lemma says that, intuitively, if we pick a random subset T of U and T does not hit any neighbor of v (in U), then the degree of v in U is small with high probability.

Proof

$$\begin{aligned} & \mathbf{Pr}[\forall v \in V,\ if\ N(v) \cap T = \phi,\ then\ |N_U(v)| \leq d] \\ = \; & 1 - \mathbf{Pr}[\exists v \in V,\ s.t.\ N(v) \cap T = \phi,\ and\ |N_U(v)| > d] \\ \geq \; & 1 - n \cdot \mathbf{Pr}[N(v) \cap T = \phi,\ and\ |N_U(v)| > d] \\ \geq \; & 1 - n \cdot \mathbf{Pr}[N(v) \cap T = \phi \mid |N_U(v)| > d] \\ > \; & 1 - n \cdot (\tfrac{u-d}{u})^{|T|} \\ > \; & 1 - n \cdot e^{-\frac{u}{2}\log n} \\ = \; & 1 - 1/n \end{aligned}$$

as claimed. □

2.1. An $\tilde{O}(n^{3/4})$ algorithm

In this section, we give an $O(n^{3/4}\log^{1/4} n)$ algorithm. The basic idea is to find a low degree vertex by random sampling, and then search among the neighbors of this vertex for the body. The algorithm is as in **Algorithm 1** box. The quantity following each step is the query complexity of that step.

The correctness and complexity of Algorithm 1 is given by the following theorem.

Theorem 9 *Given a graph G,* **Algorithm 1** *can decide whether G is a Scorpion graph and, if yes, output the Body, Tail and Sting. The time and query complexity of* **Algorithm 1** *is $O(n^{3/4}\log^{1/4} n)$.*

Proof If G is a Scorpion, then there are at least two low degree vertices: Tail and Sting. So we can find a vertex v in Step 2 with probability at least 9/10, and we know by Lemma 8 that $deg(v) \le d$ with probability at least $1-1/n$. Then if v is Tail or Sting we will find that G is a Scorpion by Lemma 7 in Step 3. Otherwise, v is a low degree Foot. So Body must be one of v's neighbors, all of which have already been found in Step 4. Thus with probability 9/10, Body is found in Step 5 and finally it uses Lemma 7 to output the correct answer. The total error probability is less than $3/10 + 1/n \le 1/3$.

If G is not a Scorpion, then error can only be made in Steps 3 or Step 5, where we use Lemma 7 twice each of which has error probability less than 1/10.

Finally, The query complexity of the algorithm is $O(n^{3/4}\log^{1/4} n)$ by letting $d = \sqrt{n\log n}$. □

2.2. Improvement

Now we give an improved algorithm, which is of $\tilde{O}(\sqrt{n})$ complexity. The key idea is that in Step 4 in **Algorithm 1**, instead of getting all neighbors of v, we again pick some random samples from these neighbors. We then find another vertex u which connects to none of these samples, then by Lemma 8, we know $N_{N(v)}(u)$ is small — in other words, v and u share few common neighbors. But note that Body still connects to both v and u if they are feet. So we can search among these common neighbors for Body. Directly following this idea gives an $O(n^{2/3})$ algorithm. To make full use of it, we apply it for about $\log n$ rounds as in **Algorithm 2**.

The following theorem actually shows **Theorem 1** in Section 1.

Theorem 10 *Given a graph G,* **Algorithm 2** *decides whether G is a Scorpion with error probability less than $1/3$ and, if yes, outputs the body, tail and sting. The time and query complexity of* **Algorithm 2** *is $O(\sqrt{n}\log^2 n)$.*

Proof The proof is similar to the one for **Algorithm 1**. Note that by Lemma 8, after the i^{th} iteration of 1(b), with high probability (at least $1-1/n$) the number of common neighbors of $v_1, ..., v_{i+1}$ is no more than d_{i+1}. And after the total Step 1, with high probability (at least $7/10 - m/n$) the number of common neighbors of $v_1, v_2, ..., v_m$ (or $v_1, ..., v_i$ if we get Step 2 by jumping out of the i^{th} iteration of Step 1) is no more than d_m (or d_i, respectively). So if G is a Scorpion, then Body will be found finally in Step 3 with probability at least 6/10, if Tail or Sting is not found in Step 1(c) luckily (which makes the algorithm succeed even earlier). On the other hand, if G is not a Scorpion, then similar arguments as in the proof of **Theorem 5** yield the small constant error probability result.

For the complexity, let us first calculate the cost of Step 1. Without loss of generality, suppose that all m iterations of Step 1 are done before we get to Step 2. The first iteration of Step 1 is of cost $O(\sqrt{k_0 n}\log m) = O(\frac{n}{\sqrt{d_1}}\sqrt{\log n}\log m)$ and the cost of the i^{th} iteration is $O(\sqrt{k_i i n} + \sqrt{k_i n}\log m)$. Note that

$$\begin{aligned} k_i &= 2(d_i/d_{i+1})\log n \\ &= 2((i-1)!n/m^i)/(i!n/m^{i+1})\log n \\ &= \tfrac{2m}{i}\log n, \end{aligned}$$

thus by noting that $\sum_{i=1}^{m-1}\sqrt{k_i i n} \ge \sum_{i=1}^{m-1}\sqrt{k_i n}\log m)$, we have the cost of Step 1 as

$$\begin{aligned} & O\left(\sqrt{k_0 n\log m} + \textstyle\sum_{i=1}^{m-1}(\sqrt{k_i i n} + \sqrt{k_i n}\log m)\right) \\ = \; & O\left(\sqrt{k_0 n\log m} + \textstyle\sum_{i=1}^{m-1}(\sqrt{k_i i n})\right) \\ = \; & O\left(\sqrt{2mn\log n\log m} + \textstyle\sum_{i=1}^{m-1}\sqrt{2mn\log n}\right) \\ = \; & O\left(m\sqrt{mn\log n}\right) \\ = \; & O\left(\sqrt{n}\log^2 n\right) \end{aligned}$$

since $m = \frac{1}{2}(\log n - \log\log n) - 1$.

Now we consider the cost of Step 2 and 3, which is $O(\sqrt{k_i n i} + \sqrt{d_m m n}) = O(\sqrt{n}\log n)$ since

$$\begin{aligned} d_m &= \tfrac{(m-1)!n}{m^m} \\ &= O\left(\tfrac{\sqrt{m}(m/e)^{m-1}n}{m^m}\right) \\ &= O\left(\tfrac{n}{\sqrt{m}}(1/e)^{m-1}\right) \\ &= O\left(\tfrac{n}{\sqrt{\log n}}\tfrac{1}{n/\log n}\right) \\ &= O\left(\sqrt{\log n}\right) \end{aligned}$$

As a result, the time and query complexity of Algorithm 2 is $O(\sqrt{n}\log^2 n)$. □

We add some remarks about the SINK problem to end this section. The same algorithm can be applied to solve SINK. Now instead of find a vertex with low degree, we are trying to find one with low out-degree. Here the sink point plays both the Sting role (by having no out-degree) and the Body role (by having $n-1$ in-degree).

Input: $G = (V, E)$.
Output: Tell whether G is a Scorpion, and if yes, output the Body, Tail and Sting.[a]

1. Let $d = \sqrt{n \log n}$. Pick a set U of $2\frac{n}{d} \log n$ random vertices. — 0
2. Use the algorithm in Lemma 5 to find a vertex v such that $N_U(v) = \emptyset$, and the error probability is less than 1/10. If we do not find one, then output "not Scorpion" and halt. — $O(\sqrt{n}\sqrt{\frac{n}{d} \log n}) = O(\frac{n}{\sqrt{d}}\sqrt{\log n})$
3. Use Lemma 7 to check whether G is a Scorpion with v being Body, Tail or Sting (and halt if it is). Make the error probability less than 1/10. — $O(\sqrt{n})$
4. Find all but up to d neighbors of v by Lemma 6 with error probability less than 1/10. If we find more than d neighbors of v, output FAILURE and halt. — $O(\sqrt{nd})$
5. Search among these neighbors of v for a vertex u with degree $n - 2$ by Lemma 5. If we do not find u, then output "not Scorpion"; otherwise, use Lemma 7 to check whether G is a Scorpion with u being Body and output the result. — $O(\sqrt{nd})$

a There appear "FAILURE" outputs in the algorithms. That is to make the algorithms more clear: "FAILURE" is something unexpected and the overall probability of "FAILURE" is no more than a small constant.

Figure 3. Algorithm 1

Input: $G = (V, E)$.
Output: tell whether the G is a Scorpion, and if yes, output the Body, Tail and Sting.
Parameters: $m = \frac{1}{2}(\log n - \log \log n) - 1$, $d_i = (i-1)!n/m^i$ for $i = 1, ..., m$.

1. **for** $i = 0$ **to** $m - 1$ **do**
   (a) If $i = 0$, randomly pick a set T_0 of $k_0 = 2\frac{n}{d_1} \log n$ vertices from V. — 0
   Else, randomly pick a set T_i of $k_i = 2\frac{d_i}{d_{i+1}} \log n$ vertices from $\cap_{j=1}^{i} N(v_j)$ using Lemma 6 and making the error probability less than $\frac{1}{10m}$. If we cannot find so many vertices, jump out of this **for** loop and go to Step 2. — $O(\sqrt{k_i in} + \sqrt{in} \log(k_i \cdot 10m)) = O(\sqrt{k_i in})$
   (b) Find a point v_{i+1} such that v_{i+1} connects to none of points in T_i using Lemma 5 and making the error probability less than $\frac{1}{10m}$. If no v_{i+1} is found, output "not Scorpion" and halt. —$O(\sqrt{k_i n} \log m)$
   (c) Use Lemma 7 to check whether G is a Scorpion with v_{i+1} being Body, Tail or Sting (and halt if yes), making the error probability less than $\frac{1}{10m}$. — $O(\sqrt{n} \log m)$
2. If we get here by jumping out of the **for** loop at the i^{th} iteration, then find all common neighbors of $v_1, ..., v_i$ using Lemma 6 with error probability less than 1/20. — $O(\sqrt{k_i ni})$
   Otherwise, find all but no more than d_m common neighbors of $v_1, ..., v_m$ using Lemma 6 with error probability less than 1/20. — $O(\sqrt{d_m nm})$
3. Use Lemma 5 to search among these common neighbors for a vertex u with degree $n - 2$. If we do not find u, then output "not Scorpion"; otherwise, use Lemma 7 to check whether G is a Scorpion and output the result. — $\max\{O(\sqrt{d_m n}), O(\sqrt{k_i n})\}$

Figure 4. Algorithm 2

3. Proof of the upper bound part of Theorem 3

To show the upper bound part of Theorem 3, it is sufficient to construct a circular function f whose quantum query complexity is $\tilde{O}(N^{1/4})$. A circular function is one whose value is invariant to any circular shift of the input indices.

Definition 3 *A function $f : \{0,1\}^N \to \{0,1\}$ is a circular function if any circular permutation of input indices does not change the function value. In other words, $f(x') = f(x)$ for any x and x' with $x' = x_{k+1}x_{k+2}...x_nx_1...x_k$ for some $1 \leq k < n$.*

Now we give a particular circular function f. Basically, it is a variant of SINK. Here we only give the definition when $N = n^2$ for some integer n. The general case can be shown using the similar algorithm and theorems.

Definition 4 *Let the function $f : \{0,1\}^N \to \{0,1\}$ be a Boolean function of N Boolean variables where $N = n^2$. For each input $x = x_1x_2...x_N$ we write it as an $n \times n$ matrix with row and column indices ranging over $\{0,1,...,n-1\}$, and the (i,j)-entry being x_{in+j+1}. We denote this matrix by M_x, or M if x is clear from the context. We use $M(i,j)$ to denote the (i,j) entry of the matrix M.*

We denote by $+_m$ and $-_m$ the addition and substraction mod n, respectively. For example, $(n-1) +_m 1 = 0$ and $0 -_m 1 = n-1$.

Let $f(x) = 1$ if and only if M_x is of the following (i,j)-form for some $i,j \in \{0,1,...,n-1\}$: row i contains all 0 entries; in row $(i -_m 1)$, all the entries with column index greater than j are 0; in row $(i+_m 1)$, all the entries with column index less than j are 0; at last, all the entries in column j except $M_x(i,j)$ are 1.

	0	...	$j-1$	j	$j+1$	...	$n-1$
0				1			
$\vdots$				$\vdots$			
$i-1$				1	0	...	0
i	0	...	0	0	0	...	0
$i+1$	0	...	0	1			
$\vdots$				$\vdots$			
$n-1$				1			

(1)

Sometimes we say M is of $(i,*)$-form if M is of (i,j)-form for some j; we also say M is of $(*,j)$-form if M is of (i,j)-form for some i. The following facts are obvious, where the $D(f) = O(\sqrt{N})$ part can be shown by using an algorithm similar to the one for SINK described in Section 2.

Lemma 11 *f is a circular function, and $D(f) = O(\sqrt{N})$.*

Another key property about f is that if $f(x) = 1$, then any row except row i has at least one entry being 1, and exactly one column — column j — intersects all rows but row i with a 1-entry.

Before describing the algorithm, we construct subroutines analogous to those in Lemma 7.

Lemma 12 *Give x and row index i , we can decide whether M_x is in $(i,*)$-form with high probability by using $O(\sqrt{n})$ queries; symmetrically, given x and column index j , we can decide whether M_x is in $(*,j)$-form with high probability by using $O(\sqrt{n})$ queries.*

Proof An algorithm for the row case:

1. Check whether $M(i,1) = ... = M(i,n) = 0$ and if not, return NO.
2. Find j *s.t.* $M(i+_m 1,1) = ... = M(i+_m 1, j-1) = 0$ and $M(i +_m 1, j) = 1$. If no j is found, return NO.
3. Check whether $M(i -_m 1, j+1) = ... = M(i -_m 1, n) = 0$ and $M(1,j) = ... = M(i-1,j) = M(i+1,j) = ... = M(n,j) = 1$. Return YES if so and NO otherwise.

An algorithm for the column case:

1. Check that only one entry is 0 in column j and assume $M(i,j) = 0$ if so. If not, return NO.

2. Check whether $M(i -_m 1, j+1) = ... = M(i -_m 1, n) = M(i +_m 1, 1) = ... = M(i +_m 1, j-1) = 0$ and all entries in row i are also 0's. Return YES if so and NO otherwise. □

Now we give an $O(\sqrt{n}\log^2 n)$ algorithm for f as in **Algorithm 3** box.

The following theorem validates **Theorem 3** in Section 1. We omit the proof because it is almost the same as that for **Algorithm 2** .

Theorem 13 **Algorithm 3** *decides f with high probability and the time and query complexity is $O(\sqrt{n}\log^2 n)$.*

We give some brief remarks on the case of N not being a square number to end this section. Let n to be the maximal integer such that $n^2 \leq N$. Again we write $x_1...x_N$ in the matrix form similar as in **Definition 4**, now with n columns and $n+1$ or $n+2$ rows, but the last row may be not complete. Suppose the last entry in the last row is in the j_0-th column. We define $f(x) = 1$ if, for some circular permutation σ, $M_{\sigma(x)}$ is in the form of (1) with $j \leq j_0$. It can be shown that all the lemmas and theorem hold for this case. We leave the details in the complete version of the paper.

Input: x
Output: $f(x)$
Parameters: $m = \frac{1}{2}(\log n - \log\log n) - 1$, $d_i = (i-1)!n/m^i$ for $i = 1, ..., m$.

1. **for** $i = 0$ **to** $m-1$ **do**
   (a) If $i = 0$, randomly pick a set T_0 of $k_0 = 2\frac{n}{d_1}\log n$ column indices $\{c_1, ..., c_{k_0}\}$. — 0
   Else, randomly pick a set T_i of $k_i = 2\frac{d_i}{d_{i+1}}\log n$ column indices $\{c_1, ..., c_{k_i}\}$ *s.t.* $\forall c \in T_i$, $M(r_s, c) = 1$ for all $s \in [i]$ using Lemma 6 and making the error probability less than $\frac{1}{10m}$. If we cannot find so many columns, then jump out from this **for** loop and go to Step 2. — $O(\sqrt{k_i i n})$
   (b) Find a row index r_{i+1} *s.t.* $M(r_{i+1}, c_1) = ... = M(r_{i+1}, c_{k_i}) = 0$ using 5 and making the error probability less than $\frac{1}{10m}$. If no r_{i+1} is found, output $f(x) = 0$ and halt. — $O(\sqrt{k_i n})$
   (c) Check whether M_x is in $(r_{i+1}, *)$-form by Lemma 12 with error probability less than $\frac{1}{10m}$. If YES, output $f(x) = 1$ and halt. — $O(\sqrt{n})$
2. If we get here by jumping out of the **for** loop at the i^{th} iteration, then use Lemma 6 to get all the columns c *s.t.* $M(r_1, c) = ... = M(r_{i+1}, c) = 1$ with error probability less than $\frac{1}{20}$. — $O(\sqrt{k_i n i})$
   Otherwise, get all but no more than d_m columns c *s.t.* $M(r_1, c) = ... = M(r_m, c) = 1$ using Lemma 6 with error probability less than $\frac{1}{20}$. — $O(\sqrt{d_m n m})$
3. Search among these columns for a column c *s.t.* M_x is of $(*, c)$-form by the algorithm in Lemma 12 with error probability less than $\frac{1}{20}$. Output $f(x) = 1$ if we succeed and $f(x) = 0$ otherwise. — $\max\{O(\sqrt{d_m n}), O(\sqrt{k_i n})\}$

Figure 5. Algorithm 3

4. Proof of Theorem 4

Before we prove Theorem 4, we remark that the lower bound part of Theorem 1 is easy to obtain by existing results. Let us first review some complexity measures (see [7] for an excellent survey).

Definition 5 *The sensitivity of a Boolean function f on input $x = x_1x_2...x_n \in \{0,1\}^n$ is*

$$s(f, x) = |\{i \in \{1, ..., n\} : f(x) \neq f(x^{(i)})\}|,$$

where $x^{(i)}$ is the n-bit string obtained from x by flipping x_i. The sensitivity of f is $s(f) = \max_{x\in\{0,1\}^n} s(f, x)$.

The block sensitivity of f on x is the maximum number b such that there are b disjoint $B_1, ..., B_b$ for which $f(x) \neq f(x^{B_i})$. The block sensitivity of f is $bs(f) = max_x bs(f, x)$.

A certificate set CS_x of f on x is a set of indices such that $f(x) = f(y)$ whenever $y_i = x_i$ for all $i \in CS_x$. The certificate complexity $C(f, x)$ of f on x is the size of a smallest certificate set of f on x. The b-certificate complexity of f is $C_b(f) = \max_{x:f(x)=b} C(f, x)$. The certificate complexity of f is $C(f) = \max\{C_0(f), C_1(f)\}$.

Turan showed that, for any graph property f, $s(f) \geq n/4$ [12] and Beals *et al* showed $Q_2(f) \geq \sqrt{bs(f)}/4$ [3], which together imply $Q_2(f) \geq \sqrt{n}/8$ because of the trivial fact $bs(f) \geq s(f)$.

In this section we prove Theorem 4 using the Ambainis's lower bound technique [2].

A permutation group Γ on the set $\{1, 2, ..., N\}$ is transitive if, for any $i, j \in \{1, 2, ..., N\}$, there is a permutation $\sigma \in \Gamma$ such that $\sigma(i) = j$. A simple but useful fact about any transitive group Γ is the following lemma in [11]. We denote by $w(x)$ the number of 1's in x, and by $\sigma(x)$ the string $x_{\sigma(1)}x_{\sigma(2)}...x_{\sigma(N)}$.

Lemma 14 *(Rivest and Vuillemin) If Γ is transitive, then for any $x \in \{0,1\}^N$ and any $i \in \{1, 2, ..., N\}$*

$$w(x) \cdot |\{\sigma(x) : \sigma \in \Gamma\}| = N \cdot |\{\sigma(x) : \sigma \in \Gamma, \sigma(x)_i = 1\}|.$$

For completeness, we first show the following classical folklore result.

Proposition 15 *For any N-ary function f invariant to a transitive group Γ, we have $C_0(f)C_1(f) \geq N$ and thus $D(f) \geq C(f) \geq \sqrt{N}$.*

Proof Let A and B be a 1 and 0-certificate with size $C_1(f)$ and $C_0(f)$, respectively. Let $\Gamma(B) = \{\sigma(B) : \sigma \in \Gamma\}$. Then for any $B' \in \Gamma(B)$, B' is also a 0-certificate assign-

ment. So $A \cap B' \neq \emptyset$, and thus

$$\sum_{B' \in \Gamma(B)} |A \cap B'| \geq |\Gamma(B)| \quad (2)$$

By Lemma 14 we know that

$$|B| \cdot |\Gamma(B)| = N \cdot |\sigma(B) : \sigma \in \Gamma, i \in \sigma(B)|.$$

Therefore,

$$\begin{aligned} \textstyle\sum_{B' \in \Gamma(B)} |A \cap B'| &= |A| \cdot |\sigma(B) : \sigma \in \Gamma, i \in \sigma(B)| \\ &= |A| \cdot \tfrac{|B| \cdot |\Gamma(B)|}{N} \end{aligned} \quad (3)$$

Combine (2) and (3) we have $|A| \cdot |B| \geq N$, i.e. $C_1(f)C_0(f) \geq N$. So $D(f) \geq C(f) \geq \sqrt{N}$. □

Now the proof of Theorem 4 is as follows. We denote $\mathbf{0} = 00...0$. For any x, let $x^{(S)}$ be the string that obtained from x by flipping all the x_i that $i \in S$.

Theorem 4 $Q_{\Gamma_{\text{transitive}}} = \Omega(N^{1/4})$.

Proof Let f be a nontrivial function invariant under a transitive permutation group Γ. Without loss of generality, we assume that $f(\mathbf{0}) = 0$. Let B be a minimal subset such that $f(\mathbf{0}^{(B)}) = 1$, *i.e.* for any $B' \subseteq B$, we have $f(\mathbf{0}^{(B')}) = 0$. Thus flipping any x_i where $i \in B$ changes the value of $f(\mathbf{0}^{(B)})$, therefore $bs(f) \geq |B|$.

Now we use the Ambainis's lower bound technique [2] to show that $Q_2(f) = \Omega(\sqrt{n/|B|})$. Let $X = \{\mathbf{0}\}$, $Y = \{\sigma(\mathbf{0}^{(B)}) : \sigma \in \Gamma\}$ and $R = X \times Y$. Then

$$m = \max_x |\{y : (x, y) \in R\}| = |Y|,$$

$$m' = \max_y |\{x : (x, y) \in R\}| = 1.$$

And

$$\begin{aligned} l &= \max_{x,i} |\{y : (x, y) \in R, x_i \neq y_i\}| \\ &= |\{\sigma(\mathbf{0}^{(B)}) : \sigma \in \Gamma, \sigma(\mathbf{0}^{(B)})_i = 1\}|, \end{aligned}$$

$$l' = \max_{y,i} |\{x : (x, y) \in R, x_i \neq y_i\}| = 1.$$

Thus by Theorem 5.1 in [2] and the above lemma, we have

$$\begin{aligned} Q_2(f) = \Omega(\sqrt{\tfrac{mm'}{ll'}}) &= \Omega\left(\sqrt{\tfrac{|Y| \cdot 1}{\frac{|B||Y|}{N} \cdot 1}}\right) \\ &= \Omega(\sqrt{N/|B|}) = \Omega(\sqrt{N/bs(f)}). \end{aligned}$$

On the other side, we know $Q_2(f) = \Omega(\sqrt{bs(f)})$, so $Q_2(f) = \Omega(N^{1/4})$. □

References

[1] S. Aaronson, Quantum certificate complexity, IEEE Conference on Computational Complexity 2003. Earlier version in quant-ph/0210020.

[2] A. Ambainis. Quantum lower bounds by quantum arguments, J. Comput. Sys. Sci. 64:750-767, 2002. Earlier version at STOC'00

[3] R. Beals, H. Buhrman, R. Cleve, M.Mosca, R. deWolf. Quantum lower bounds by polynomials. Journal of ACM, 48: 778-797, 2001. Earlier versions in FOCS'98 and quant-ph/9802049.

[4] M. R. Best, P. van Emde Boas and H. W. Lenstra, Jr. A sharpened version of the Aanderaa-Rosenberg conjecture, Report ZW 30/74, Mathematish Centrum, Amsterdam, 1974.

[5] B. Bollobás and S. E. Eldridge, Packing of graphs and applications to computational complexity, J. Of Combinatorial Theory Ser. B 25, 105-124.

[6] Quantum Counting, G. Brassard, P. Hoyer, A. Tapp, ICALP'98, LNCS 1443, 820-831, 1998, also quant-ph/9805082

[7] H. Buhrman, R. de Wolf. Complexity measures and decision tree complexity: a survey. Theoretical Computer Science, Volume 288, Issue 1, 9 October 2002, Pages 21-43

[8] L. Grover. A fast quantum mechaqnical algorithm for database search, STOC'96, 212-219.

[9] Peter Hoyer, Michele Mosca, Ronald de Wolf: Quantum Search on Bounded-Error Inputs. ICALP 2003: 291-299

[10] N. Nisan, CREW PRAMS and decision trees, SIAM Journal on computing, 20, 6, 999-1007, 1991. Earlier version in STOC'89.

[11] R. L. Rivest and J. Vuillemin, On recognizing graph properties from adjacency matrices, Theoretical Computer Science 3: 371 - 384, 1976.

[12] G. Turan, The critical complexity of graph properties, Information Processing Letters, 18, 151-153, 1984