Quantum Speedup Based on Classical Decision Trees

Salman Beigi[1] and Leila Taghavi[2]

[1]*School of Mathematics, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran*

[2]*School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran*

Lin and Lin [LL16] have recently shown how starting with a classical query algorithm (decision tree) for a function, we may find upper bounds on its quantum query complexity. More precisely, they have shown that given a decision tree for a function $f : \{0,1\}^n \to [m]$ whose input can be accessed via queries to its bits, and a *guessing algorithm* that predicts answers to the queries, there is a quantum query algorithm for f which makes at most $O(\sqrt{GT})$ quantum queries where T is the depth of the decision tree and G is the maximum number of mistakes of the guessing algorithm. In this paper we give a simple proof of and generalize this result for functions $f : [\ell]^n \to [m]$ with non-binary input as well as output alphabets. Our main tool for this generalization is non-binary span program which has recently been developed for non-binary functions, and the dual adversary bound. As applications of our main result we present several quantum query upper bounds, some of which are new. In particular, we show that topological sorting of vertices of a directed graph $\mathcal{G}$ can be done with $O(n^{3/2})$ quantum queries in the adjacency matrix model. Also, we show that the quantum query complexity of the maximum bipartite matching is upper bounded by $O(n^{3/4}\sqrt{m+n})$ in the adjacency list model.

1 Introduction

Query complexity of a function $f : [\ell]^n \to [m]$ is the minimum number of adaptive queries to its input bits required to compute the output of the function. In a quantum query algorithm we allow to make queries in superposition, which sometimes improves the query complexity, e.g., in Grover's search algorithm [Gro].

Lin and Lin [LL16] have recently shown that surprisingly sometimes classical query algorithms may result in improved quantum query algorithms. They showed that having a classical query algorithm with query complexity T for some function $f : \{0,1\}^n \to [m]$, together with a *guessing algorithm* that at each step predicts the

value of the queried bit and makes no more than G mistakes, the quantum query complexity of f is at most $Q(f) = O(\sqrt{GT})$. For instance, the trivial classical algorithm for the search problem which queries the input bits one by one have query complexity $T = n$, and the guessing algorithm which always predicts the output 0 makes at most $G = 1$ mistakes (because making a mistake is equivalent to finding an input bit 1 which solves the search problem). Thus the quantum query complexity of the search problem is $O(\sqrt{GT}) = O(\sqrt{n})$ recovering Grover's result.

There are two proofs of the above result in [LL16]. One of the proofs is based on the notion of *bomb query complexity* $B(f)$. Lin and Lin show that there exists a bomb query algorithm that computes f using $O(GT)$ queries, and that the bomb query complexity equals the square of the quantum query complexity, i.e., $B(f) = \Theta(Q(f)^2)$, which together give $Q(f) = O(\sqrt{GT})$. In the second proof, they design a quantum query algorithm with query complexity $O(\sqrt{TG})$ for f using Grover's search; in computing the function they use the values of predicted queries instead of the real values and use a modified version of Grover's search to find mistakes of the guessing algorithm.

Our results: In this paper we give a simple proof of the above result based on the method of *non-binary span program* that has recently been development by the authors [BT19]. Then inspired by this proof, we generalize Lin and Lin's result for functions $f : [\ell]^n \to [m]$ with non-binary input as well as non-binary output alphabets. Our proof of this generalization is based on the dual adversary bound which is another equivalent characterization of the quantum query complexity [LMR+11].

As an application of our main result we show that given query access to edges of a directed and acyclic graph $\mathcal{G}$ in the *adjacency matrix model*, the vertices of $\mathcal{G}$ can be sorted with $O(n^{3/2})$ quantum queries to its edges. We also show that given a directed graph $\mathcal{G}$ and a vertex $v \in V(\mathcal{G})$, the quantum query complexity of determining the length of the smallest directed cycle in $\mathcal{G}$ containing v is $\Theta(n^{3/2})$. Moreover, we show that given an undirected graph $\mathcal{G}$, a vertex v and some constant $k > 0$, the quantum query complexity of deciding whether $\mathcal{G}$ has a cycle of length k containing the vertex v is $O(n^{3/2})$. Furthermore, we show that some existing results on the quantum query complexity of graph theoretic problems such as directed *st*-connectivity, detecting bipartite graphs, finding strongly connected components, and deciding forests can easily be derived from our results.

Our main result is also useful when dealing with graph problems in the *adjacency list model*. In this regard, we show that given query access to the adjacency list of an unweighted bipartite graph $\mathcal{G}$, the quantum query complexity of finding a maximum bipartite matching in $\mathcal{G}$ is $O(n^{3/4}\sqrt{m+n})$, where m is the number of edges of the graph. To the authors' knowledge this is the first non-trivial upper bound for this problem.

2 Preliminaries

In this section we review the notions of the dual adversary bound and the non-binary span program that will be used for the proof of our main result. In this paper we use Dirac's ket-bra notation, so $|v\rangle$ is a complex (column) vector whose conjugate

transpose is denotes by $\langle v|$. Then, $\langle v|w\rangle$ is the inner product of vectors $|v\rangle, |w\rangle$. For a matrix A, we denote by $\|A\|$ the operator norm of A, i.e., the maximum singular value of A. We use $[\ell]$ to denote the ℓ-element set $\{0, \ldots, \ell-1\}$. We also use the Kronecker delta symbol $\delta_{a,b}$ which equals 1 if $a = b$ and equals 0 otherwise.

2.1 Query algorithms

In the query model we deal with the problem of computing a function $f : D_f \to [m]$ with domain $D_f \subseteq [\ell]^n$ by quering coordinates of the input $x = (x_1, \ldots, x_n) \in D_f \subseteq [\ell]^n$. In the classical setting a query algorithm asks the value of some coordinate of the input and based on the answer to that query decides what to do next: either asks another query or outputs the result. Such an algorithm can be modeled by a *decision tree* whose internal vertices are associated with queries, i.e., indices $1 \leq j \leq n$, and whose edges correspond to answers to queries, i.e., elements of $[\ell]$. At each vertex the algorithm queries the associated index, and then moves to the next vertex via the edge whose label equals the answer to that query. The algorithm ends once we reach the leaves of the tree that are labeled by elements of $[m]$, the output set of the function. The query complexity of the algorithm is the maximum number of queries in the algorithm over all $x \in D_f$, which is equal to the height of the decision tree. A randomized classical query algorithm can similarly be modeled by a collection of decision trees where one of them is chosen at random.

In contrast in quantum query algorithms, a query can be made in superposition. Such a query to an input x can be modeled by the unitary operator O_x:

$$O_x|j, p\rangle = |j, (x_j + p) \mod \ell\rangle,$$

where the first register contains the query index $1 \leq j \leq n$, and the second register saves the value of x_j in a reversible manner. Therefore, a quantum query algorithm for computing $f(x)$ is an alternation of unitaries O_x and some U_i's that are independent of x (but depend on f itself). Indeed, a quantum query algorithm consists of sequence of unitaries

$$U_k O_x \ldots U_2 O_x U_1,$$

followed by a measurement which determines the outcome of the algorithm. We say that an algorithm computes f, if for every $x \in D_f \subseteq [\ell]^n$ the algorithm outputs $f(x)$ with probability at least $2/3$. The query complexity of such an algorithm is the number of queries, i.e., the number of O_x's in the sequence of unitaries. $Q(f)$ denotes the *quantum query complexity* of f, which is the minimum query complexity among all quantum algorithms that compute f.

2.2 Dual adversary bound

The *generalized adversary bound* [HLŠ07] gives a lower bound on the quantum query complexity of any function $f : D_f \to [m]$ with $D_f \subseteq [\ell]^n$. This bound can be obtained via a semi-definite program (SDP) whose optimal value, based on the duality of SDPs, has been shown to be equal to that of the following SDP up to a

factor of at most 2 [LMRŠ].

$$\min \quad \max_{x\in D_f} \max\left\{\sum_{j=1}^{n}\left\||u_{xj}\rangle\right\|^2, \sum_{j=1}^{n}\left\||w_{xj}\rangle\right\|^2\right\} \tag{1a}$$

$$\text{subject to} \quad \sum_{j:x_j\neq y_j}\langle u_{xj}|w_{yj}\rangle = 1-\delta_{f(x),f(y)} \qquad \forall x,y\in D_f. \tag{1b}$$

Here the optimization is over vectors $|u_{xj}\rangle, |w_{xj}\rangle$. This SDP is called *the dual adversary bound* and is proved by Lee et al. [LMR+11] to be an upper bound on quantum query complexity of the function f as well. Thus, the above SDP characterizes the quantum query complexity of f up to a constant factor. Moreover, in order to design quantum query algorithms and quantum query complexity upper bounds, it is enough to find a feasible solution of the SDP (1).

Function evaluation is a special case of a more general problem called *state generation* [Shi02, AMRR11]. In the state generation problem, the goal is to generate a state $|\psi_x\rangle$ (which depends on x) up to a constant error, given query access to $x \in D$. That is, the quantum query algorithm is required to output some state ρ_x such that $\|\rho_x - |\psi_x\rangle\langle\psi_x|\|_{\mathrm{tr}} \leq 0.1$ where $\|\cdot\|_{\mathrm{tr}}$ denotes the trace distance. Of course, the function evaluation problem is a special case of the state generation problem in which $|\psi_x\rangle = |f(x)\rangle$. It has been shown in [LMR+11] that a generalization of the SDP (1) characterizes the quantum query complexity of the state generation problem up to a constant factor. This generalized SDP, again called the dual adversary bound, is as follows:

$$\min \quad \max_{x\in D} \max\left\{\sum_{j=1}^{n}\left\||u_{xj}\rangle\right\|^2, \sum_{j=1}^{n}\left\||w_{xj}\rangle\right\|^2\right\} \tag{2a}$$

$$\text{subject to} \quad \sum_{j:x_j\neq y_j}\langle u_{xj}|w_{yj}\rangle = 1-\langle\psi_x|\psi_y\rangle \qquad \forall x,y\in D, \tag{2b}$$

where again the optimization is over vectors $|u_{xj}\rangle, |w_{xj}\rangle$. Observe that this SDP depends only on the gram matrix $\left(\langle\psi_x|\psi_y\rangle\right)_{x,y}$ of the target vectors. Moreover, letting $|\psi_x\rangle = |f(x)\rangle$, for some function f, we recover (1).

2.3 Non-binary span program

Span program introduced by [Rei09] is another algebraic tool that similar to the dual adversary bound, characterizes the quantum query complexity of binary functions up to a constant factor. This model has been used for designing quantum query algorithms of binary decision functions by Špalek and Reichardt [RŠ12]. The notion of span program was generalized for functions with non-binary inputs in [IJ15]. Later, it was further generalized for arbitrary non-binary functions with non-binary input/output alphabets [BT19]. In this paper we use a special form of non-binary span program of [BT19] called non-binary span program *with orthogonal inputs*, which characterizes the quantum query complexity of any functions $f : [l]^n \to [m]$ up to a factor of $\sqrt{\ell-1}$. Here since we will use non-binary span programs only for functions with binary inputs ($\ell = 2$), we may focus on this special form.

A *non-binary span program with orthogonal inputs* (NBSPwOI) P evaluating a function $f : D_f \to [m]$ with $D_f \subseteq [\ell]^n$ consists of[1]

- a finite-dimensional inner product space V,
- m target vectors $|t_0\rangle, |t_2\rangle, \ldots, |t_{m-1}\rangle \in V$,
- and an input set $I_{j,q} \subseteq V$ for every $1 \leq j \leq n$ and $q \in [\ell]$.

Then $I \subseteq V$ is defined by

$$I = \bigcup_{j=1}^{n} \bigcup_{q \in [\ell]} I_{j,q},$$

and for every $x \in D_f$ the set of *available vectors* $I(x)$ is defined by

$$I(x) = \bigcup_{j=1}^{n} I_{j,x_j}.$$

Indeed, when the j-th coordinate of x is equal to q (i.e., $x_j = q$) then the vectors in $I_{j,q}$ become available. We also let A be the $d \times |I|$ matrix consisting of all input vectors as its columns where $d = \dim V$.

We say that P evaluates the function f if for every $x \in D_f$, $|t_\alpha\rangle$ belongs to the span of the available vectors $I(x)$ if and only if $\alpha = f(x)$. Even more, there should be two witnesses indicating this. Namely, a positive witness $|w_x\rangle \in \mathbb{C}^{|I|}$ and a negative witness $|\bar{w}_x\rangle \in V$ satisfying the following conditions:

- The coordinates of $|w_x\rangle$ associated to unavailable vectors are zero.
- $A\,|w_x\rangle = |t_\alpha\rangle$.
- $\forall\, |v\rangle \in I(x)$ we have $\langle v|\bar{w}_x\rangle = 0$.
- $\forall \beta \neq \alpha$ we have $\langle t_\beta|\bar{w}_x\rangle = 1$.

Let positive and negative complexities of P together with the collections w and $\bar{w}$ of positive and negative witnesses $(P, w, \bar{w})$ be

$$\begin{aligned} \mathrm{wsize}^+(P, w, \bar{w}) &:= \max_{x \in D_f} \; \| \, |w_x\rangle \, \|^2, \\ \mathrm{wsize}^-(P, w, \bar{w}) &:= \max_{x \in D_f} \; \|A^\dagger \, |\bar{w}_x\rangle \, \|^2. \end{aligned}$$

Then the complexity of $(P, w, \bar{w})$ is equal to

$$\mathrm{wsize}(P, w, \bar{w}) = \sqrt{\mathrm{wsize}^-(P, w, \bar{w}) \cdot \mathrm{wsize}^+(P, w, \bar{w})}. \tag{3}$$

It is shown in [BT19] that for any NBSPwOI evaluating the function f, its complexity $\mathrm{wsize}(P, w, \bar{w})$ is an upper bound on $Q(f)$. Furthermore, there always exists an associated NBSPwOI whose complexity is bounded by $O(\sqrt{\ell - 1}Q(f))$. Thus, NBSPwOIs characterize the quantum query complexity of all functions up to a factor of $\sqrt{\ell - 1}$.

[1]Non-binary span programs may also have *free input vectors* that will not be used here in this paper.

3 From decision trees to span programs

In this section we first give a simple proof of the main result of [LL16] based on span programs. Later, getting intuition from this proof, we generalize this result for non-binary functions.

Recall that a classical query algorithm for a function $f : D_f \to [m]$ with $D_f \subseteq \{0,1\}^n$ can be modeled by a binary decision tree $\mathcal{T}$ with internal vertices being indexed by elements of $\{1,\ldots,n\}$, edges being indexed by $\{0,1\}$, and leaves being index by elements of $[m]$. The depth of the decision tree, which we denote by T, is the classical query complexity of this decision tree. In [LL16] it is assumed that there is a further algorithm that predicts the values of the queried bits. That is, at each internal vertex of $\mathcal{T}$ it makes a guess for the answer of the associated query. This guess, of course, may depend on the answers to the previous queries. Then it is proven that if for every $x \in D_f$ the number of mistakes of the guessing algorithm is at most G, then the quantum query complexity of f is $O(\sqrt{TG})$.

We can visualize the guessing algorithm in the decision tree by coloring its edges. For each internal vertex of the decision tree, there are two outgoing edges indexed by 0 and 1, one of which is chosen by the guessing algorithm. We *color* the chosen one black, and the other one red. We call such a coloring of the edges of the decision tree a *guessing-coloring (hereafter, G-coloring)*. Now once we make a query at an internal vertex, its answer tells us which edge we should take, the black one or the red one. If it was black it means that the guessing algorithm made a correct guess, and if it was red it means that it made a mistake. Therefore, the number of mistakes of the guessing algorithm for every $x \in D_f$ equals the number of red edges in the path from the root to the leaf of the tree associated to x.

Here we summarize the notion of G-coloring.

Definition 1 (G-coloring)**.** *A G-coloring of a decision tree $\mathcal{T}$ is a coloring of its edges by two colors black and red, in such a way that any vertex of $\mathcal{T}$ has at most one outgoing edge with black color.*

We can now state the result of [LL16] based on decision trees and the notion of G-coloring.

Theorem 2 (Lin and Lin [LL16])**.** *Assume that we have a decision tree $\mathcal{T}$ for a function $f : D_f \to [m]$ with $D_f \subseteq \{0,1\}^n$ whose depth is T. Furthermore, assume that for a G-coloring of the edges of $\mathcal{T}$, the number of red edges in each path from the root to the leaves of $\mathcal{T}$ is at most G. Then there exists a quantum query algorithm computing the function f with query complexity $O(\sqrt{GT})$.*

We remark that the result of [LL16] also works for randomized algorithms. Nevertheless, here to present our main ideas we first consider deterministic decision trees. Later, randomized query algorithms will be considered as well.

To prove this theorem we design an NBSPwOI for f with complexity $O(\sqrt{GT})$. To present this span program first we need to develop some notations. Let $V(\mathcal{T})$ be the vertex set of $\mathcal{T}$. Then for every internal vertex $v \in V(\mathcal{T})$, its associated index is denoted by $J(v)$, i.e., $J(v)$ is the index $1 \leq j \leq n$ that is queried by the classical algorithm at node v. The two outgoing edges of v are indexed by elements

of $\{0,1\}$ and connect v to two other vertices. We denote these vertices by $N(v,0)$ and $N(v,1)$. That is, $N(v,q)$, for $q \in \{0,1\}$, is the next vertex that is reached from v after following the outgoing edge with label q. We also represent the G-coloring of edges of $\mathcal{T}$ by a function $C(v,q) \in \{\text{black}, \text{red}\}$ where v is an internal vertex, $q \in \{0,1\}$ and $C(v,q)$ is the color of the outgoing edge of v with label q.

Proof. For every $x \in D_f$ there is an associate leaf of the tree $\mathcal{T}$ that is reached once we follow edges of the tree with labels x_j starting from the root. In order to find $f(x)$ it suffices to find this associated leaf because this is what the classical query algorithm does; once we find the leaf associated to x, we find the path that the classical query algorithm would take and then find $f(x)$. Thus in order to compute f, we may compute another function $\tilde{f}$ which given x outputs its associated leaf of $\mathcal{T}$, and to prove the upper bound of $O(\sqrt{GT})$ on the quantum query complexity it suffices to design an NBSPwOI for $\tilde{f}$ with this complexity.

The NBSPwOI is the following:

- the vector space V is determined by the orthonormal basis indexed by vertices of $\mathcal{T}$:
$$\{|v\rangle \mid v \in V(\mathcal{T})\},$$
- the input vectors are
$$I_{j,q} = \left\{ \sqrt{W_{C(v,q)}} \Big(|v\rangle - |N(v,q)\rangle \Big) \,\middle|\, \forall v \in V(\mathcal{T}) \text{ s.t. } J(v) = j \right\},$$
where W_{black} and W_{red} are positive real numbers to be determined,
- the target vectors are indexed by leaves u of the tree:
$$|t_u\rangle = |r\rangle - |u\rangle\,,$$
where $r \in V(\mathcal{T})$ is the root of the tree.

For every vertex v of $\mathcal{T}$ we denote by P_v the (unique) path from the root r to vertex v. Then for every $x \in D_f$ there exists a path $P_x = P_{\tilde{f}(x)}$ from the root of the decision tree to the leaf $\tilde{f}(x)$. Thus the target vector $\left|t_{\tilde{f}(x)}\right\rangle$ equals

$$\left|t_{\tilde{f}(x)}\right\rangle = |r\rangle - \left|\tilde{f}(x)\right\rangle = \sum_{v \in P_x} \frac{1}{\sqrt{W_{C\left(v, x_{J(v)}\right)}}} \left\{ W_{C\left(v, x_{J(v)}\right)} \left(|v\rangle - \left| N(v, x_{J(v)}) \right\rangle \right) \right\},$$

where the vectors in the braces are all available for x. Then since by assumptions the number of red edges along the path P_x is at most G and the number of all edges is at most T, the positive complexity is bounded by

$$\text{wsize}^+ \leq \frac{1}{W_{\text{red}}} G + \frac{1}{W_{\text{black}}} T.$$

We let the negative witness for x to be

$$|\bar{w}_x\rangle = \sum_{v \in P_x} |v\rangle\,.$$

It is easy to verify that $|\bar{w}_x\rangle$ is orthogonal to all available vectors, and that $\langle \bar{w}_x | t_u \rangle = \langle \bar{w}_x | r \rangle = 1$ for all $u \neq \tilde{f}(x)$. Thus $|\bar{w}_x\rangle$ is a valid negative witness. Moreover, an input vector of the form

$$\sqrt{W_{C(v,q)}}\big(\,|v\rangle - |N(v,q)\rangle\,\big),$$

contributes in the negative witness size only if its corresponding edge $\{v, N(v,q)\}$ leaves the path P_x, i.e., they have only the vertex v in common. In this case the contribution would be equal to $W_{C(v,q)}$, the weight of that edge. The number of such red (black) edges equals the number of black (red) edges in P_x, which is bounded by T (G). Therefore, the negative witness size is

$$\text{wsize}^- \leq W_{\text{black}} G + W_{\text{red}} T$$

Now letting $W_{\text{black}} = \frac{1}{W_{\text{red}}} = \sqrt{\frac{T}{G}}$, both the positive and negative witnesses are bounded by $2\sqrt{GT}$. Therefore, the quantum query complexity of $\tilde{f}$, and then f are bounded by $O(\sqrt{GT})$. □

4 Main result: generalization to the non-binary case

This section contains our main result which is a generalization of Theorem 2 for functions $f : D_f \to [m]$ with non-binary input alphabet $D_f \subseteq [\ell]^n$. In this case, a classical query algorithm corresponds to a decision tree whose internal vertices have out-degree ℓ (instead of 2). Moreover, a G-coloring can be defined similarly based on a guessing algorithm. Yet, we are interested in a further generalization of the notion of decision tree which we explain by an example.

Consider the following trivial algorithm for finding the minimum of a list of numbers in $[\ell]$: we keep a candidate minimum, and as we query the numbers in the list one by one, we update it once we reach a smaller number. In this algorithm, the possible numbers as answers to a query are of two types: numbers that are greater than or equal to the current candidate minimum, and those that are smaller. Now assuming that the answer to that query is of the first type, what we do next is independent of its exact value (since we simply ignore it and query the next index). Considering the associated decision tree $\mathcal{T}$, for each vertex v we have a candidate minimum, and the outgoing edges of v are labeled by different numbers in $[\ell]$. Then by the above discussion, the subtrees of $\mathcal{T}$ hanging below the outgoing edges whose labels are greater than or equal to the current candidate minimum are identical. Thus we can identify those edges and their associated subtrees. In this case the outgoing edges of v are not labeled by elements of $[\ell]$, but by its certain subsets that form a partition. Indeed, there is an outgoing edge whose label is the *subset* of numbers greater than or equal to the current candidate minimum, and an outgoing edge for any smaller number.

Motivated by the above example of minimum finding, we generalize the notion of decision tree $\mathcal{T}$ for a function $f : D_f \to [m]$ with non-binary input alphabet ($D_f \subseteq [\ell]^n$). As before each internal vertex v of $\mathcal{T}$ corresponds to a query index

$1 \le J(v) \le n$. Each outgoing edge of this vertex is labeled by a subset of $[\ell]$, and we assume that these subsets form a partition of $[\ell]$. We denote this partition by

$$\bigcup_{q=0}^{\ell-1} Q_v(q) = [\ell], \tag{4}$$

where here $Q_v(q)$ is the subset in the partition that contains $q \in [\ell]$. Thus $Q_v(q) \subseteq [\ell]$ contains q, and for $q, q' \in [\ell]$ either $Q_v(q), Q_v(q')$ are disjoint or are equal. Moreover, the out-degree of v equals $|\{Q_v(q) : q \in [\ell]\}|$, the number of different $Q_v(q)$'s. We also denote the neighbor vertex of v connected to the edge with label $Q_v(q)$ by $N(v, Q_v(q))$. See Figure 1 for an example of a decision tree.

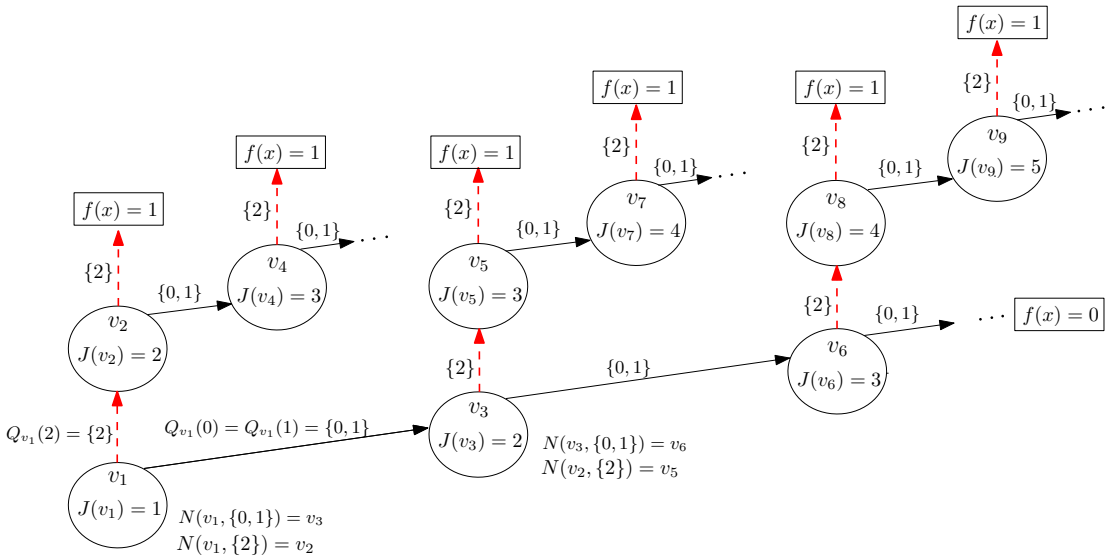

Figure 1: Decision tree for deciding whether a given string $x \in \{0,1,2\}^n$ contains at least two 2's. At any vertex v the queried index is $J(v)$ and the result of the query belongs to one of the two sets appeared in the labels of outgoing edges of v. This tree has a natural G-coloring: edges with label $\{2\}$ are red (dashed edges) and edges with label $\{0,1\}$ are black (solid edges). The depth of the decision tree is $T = n$, and $f(x)$ would be determined once we see two red edges. Thus $G = 2$ and the quantum query complexity of this problem is $O(\sqrt{n})$.

Now given a decision tree $\mathcal{T}$ as above, the corresponding classical algorithm works as follows. We start with the root r of the tree and query $J(r)$. Then $x_{J(r)} \in [\ell]$ corresponds to the outgoing edge of v with label $Q_v(x_{J(r)})$. We take that edge and move to the next vertex $N(v, Q_v(x_{J(r)}))$. We continue until we reach a leaf of the tree which determines the value of $f(x)$.

The notation of G-coloring can also be generalized similarly. Recall that a G-coloring comes from a guessing algorithm that in each step predicts the answer to the queried index. In our generalized decision tree whose edges are labeled by subsets of $[\ell]$, we assume that the guessing algorithm chooses one of these subsets as its guess. Rephrasing this in terms of colors, we assume that for each internal vertex v of $\mathcal{T}$, one of its outgoing edges is colored in black (meaning that its label is the predicted answer) and its other outgoing edges are colored in red. We denote the color of the outgoing edge of vertex v with label $Q_v(q)$ by $C(v, Q_v(q)) \in \{\text{black}, \text{red}\}$.

Here is a summary of the notions of generalized decision tree and G-coloring explained above.

Definition 3 (Generalized decision tree and G-coloring)**.** *A generalized decision tree $\mathcal{T}$ is a rooted directed tree such that each internal vertex v (including the root) of $\mathcal{T}$ corresponds to a query index $1 \le J(v) \le n$. Outgoing edges of v are labeled by subsets of $[\ell]$ that form a partition of $[\ell]$. We denote the subset that contains $q \in [\ell]$ by $Q_v(q)$ so that (4) holds. Leaves of $\mathcal{T}$ are labeled with elements of $[m]$.*

We say that $\mathcal{T}$ decides a function $f : D_f \to [m]$ with $D_f \subseteq [\ell]^n$ if for every $x \in D_f$, by starting from the root of $\mathcal{T}$ and following edges labeled by $Q_v(x_{J(v)})$ we reach a leaf with label $m = f(x)$.

As in Definition 1, a G-coloring of a generalized decision tree $\mathcal{T}$ is a coloring of its edges by two colors black and red, in such a way that any vertex of $\mathcal{T}$ has at most one outgoing edge with black color.

We also consider *randomized classical query algorithms*. In this case, for each value ζ of the outcomes of some coin tosses, we have a (deterministic) generalized decision tree $\mathcal{T}_\zeta$ as above. We also assume that each of these decision trees $\mathcal{T}_\zeta$ is equipped with a guessing algorithm which itself may be randomized. Nevertheless, we may assume with no loss of generality that ζ includes the randomness of the guessing algorithm as well. Therefore, for any ζ we have a generalized decision tree with a G-coloring as before. We assume that the classical randomized query algorithm outputs the correct answer $f(x)$ with high probability:

$$\Pr_\zeta \Big[\text{output of } \mathcal{T}_\zeta \text{ on } x \text{ equals } f(x)\Big] \ge 0.9. \tag{5}$$

The complexity of such a randomized query algorithm is given by the *expectation* of the number of queries over the random choice of ζ.

We can now state our generalization of Theorem 2.

Theorem 4. *In the following let $f : D_f \to [m]$ be a function with $D_f \subseteq [\ell]^n$.*

(i) *Let $\mathcal{T}$ be a generalized decision tree for f equipped with a G-coloring. Let T be the depth of $\mathcal{T}$ and let G be the maximum number of red edges in any path from the root to leaves of $\mathcal{T}$. Then the quantum query complexity of f is upper bounded by $O(\sqrt{TG})$.*

(ii) *Let $\{\mathcal{T}_\zeta : \zeta\}$ be a set of generalized decision trees corresponding to a* randomized *classical query algorithm evaluating f with bounded error as in (5). Moreover, suppose that each $\mathcal{T}_\zeta$ is equipped with a G-coloring. Let P_x^ζ be the path from the root to the leaf of $\mathcal{T}_\zeta$ associated to $x \in D_f$. Let T_x^ζ be the length of the path P_x^ζ, and let G_x^ζ be the number of red edges in this path. Define*

$$T = \max_x \mathbb{E}_\zeta[T_x^\zeta],$$
$$G = \max_x \mathbb{E}_\zeta[G_x^\zeta],$$

where the expectation is over the random choice of ζ. Then the quantum query complexity of f is $O(\sqrt{TG})$.

The span program in the proof of Theorem 2 can easily be adapted for a proof of the above theorem, yet in the complexity of the resulting span program we see an extra factor of $\sqrt{\ell-1}$, i.e., we get the upper bound of $O(\sqrt{(\ell-1)GT})$ on the quantum query complexity. To remove this undesirable factor, getting ideas from the span program in the proof of Theorem 2, we directly construct a feasible solution of the dual adversary SDP (1). Indeed, our starting point for proving Theorem 4 is the proof of Theorem 2 based on span programs. Then getting intuition from this proof, we design a feasible solution of the dual adversary SDP with the desired objective value.

Proof. (i) Let $V_j(\mathcal{T})$ be the set of vertices of $\mathcal{T}$ associated with query index j, i.e., $V_j(\mathcal{T}) = J^{-1}(j)$. Also let P_x be the path from the root r to the leaf of $\mathcal{T}$ associated to $x \in D_f$. We can assume with no loss of generality that $V_j(\mathcal{T}) \cap P_x$ contains at most one vertex since otherwise in computing $f(x)$ we are querying index j more than once.

To construct the feasible solution of the dual adversary SDP we will need the set of vectors $\{|\mu_Q\rangle : Q \subseteq [\ell]\}$ and $\{|\nu_Q\rangle : Q \subseteq [\ell]\}$ in $\mathbb{C}^{2^{[\ell]}}$ first appeared in [LMRŠ]:

$$|\mu_Q\rangle = \sqrt{\frac{2(2^\ell-1)}{2^\ell}}\left(-\theta\,|Q\rangle + \frac{\sqrt{1-\theta^2}}{\sqrt{2^\ell-1}}\sum_{P\neq Q}|P\rangle\right), \tag{6}$$

$$|\nu_Q\rangle = \sqrt{\frac{2(2^\ell-1)}{2^\ell}}\left(\sqrt{1-\theta^2}\,|Q\rangle + \frac{\theta}{\sqrt{2^\ell-1}}\sum_{P\neq Q}|P\rangle\right), \tag{7}$$

where $\theta = \sqrt{\frac{1}{2} - \frac{\sqrt{2^\ell-1}}{2^\ell}}$. These vectors have the property that $\| \, |\mu_Q\rangle \, \|^2 = \| \, |\nu_Q\rangle \, \|^2 = \frac{2(2^\ell-1)}{2^\ell} \leq 2$ for all Q and

$$\langle \mu_Q | \, \nu_P \rangle = 1 - \delta_{Q,P}.$$

Also we use the set of vectors $\{|\tilde{\mu}_\alpha\rangle : \alpha \in [m]\}$ and $\{|\tilde{\nu}_\alpha\rangle : \alpha \in [m]\}$ in $\mathbb{C}^m$ defined similarly as above with the property that $\| \, |\tilde{\mu}_\alpha\rangle \, \|^2 = \| \, |\tilde{\nu}_\alpha\rangle \, \|^2 = \frac{2(m-1)}{m} \leq 2$ for all α, and that $\langle \tilde{\mu}_\alpha | \, \tilde{\nu}_\beta \rangle = 1 - \delta_{\alpha,\beta}$.

Now define vectors $|u_{xj}\rangle$ and $|w_{xj}\rangle$ in the vector space $\mathbb{C}^{V(\mathcal{T})} \otimes \mathbb{C}^{\{\text{black,red}\}} \otimes \mathbb{C}^{2^{[\ell]}} \otimes \mathbb{C}^m$ as follows:

$$|u_{xj}\rangle = \begin{cases} \frac{1}{\sqrt{W_{C(v,Q_v(x_j))}}}\Big|v, C(v,Q_v(x_j))\Big\rangle \otimes \Big|\mu_{Q_v(x_j)}\Big\rangle \otimes \Big|\tilde{\mu}_{f(x)}\Big\rangle & \text{if } \exists v \in P_x \cap V_j(\mathcal{T}) \\ 0 & \text{otherwise,} \end{cases}$$

and

$$|w_{xj}\rangle = \begin{cases} \sum_{c\in C_{v,xj}} \sqrt{W_c}\,|v,c\rangle \otimes \Big|\nu_{Q_v(x_j)}\Big\rangle \otimes \Big|\tilde{\nu}_{f(x)}\Big\rangle & \text{if } \exists v \in P_x \cap V_j(\mathcal{T}) \\ 0 & \text{otherwise,} \end{cases}$$

where assuming that $v \in P_x \cap V_j(\mathcal{T})$, $C_{v,xj} \subseteq \{\text{black}, \text{red}\}$ is defined by

$$C_{v,xj} = \Big\{C(v,Q_v(q)) : \;\; Q_v(q) \neq Q_v(x_j)\Big\}. \tag{8}$$

Observe that assuming there is a (unique) vertex $v \in P_x \cap V_j(\mathcal{T})$, $|u_{xj}\rangle$ is defined in terms of the label and color of the outgoing edge of v with label $Q_v(x_j)$. Moreover, $|w_{xj}\rangle$ is equal to either

$$\sqrt{W_{\text{red}}}\,|v, \text{red}\rangle \otimes \left|\nu_{Q_v(x_j)}\right\rangle \otimes \left|\tilde{\nu}_{f(x)}\right\rangle,$$

or

$$\left(\sqrt{W_{\text{red}}}\,|v, \text{red}\rangle + \sqrt{W_{\text{black}}}\,|v, \text{black}\rangle\right) \otimes \left|\nu_{Q_v(x_j)}\right\rangle \otimes \left|\tilde{\nu}_{f(x)}\right\rangle,$$

depending on whether $C(v, Q_v(x_j)) = \text{black}$ or $C(v, Q_v(x_j)) = \text{red}$ respectively.

We claim that these vectors form a solution of the SDP (1). For every $x, y \in D_f$ with $f(x) \neq f(y)$ there exists a unique vertex $v \in V(\mathcal{T})$ such that $v \in P_x \cap P_y$ with $Q_v^{x_{J(v)}} \neq Q_v^{y_{J(v)}}$ and in particular $x_{J(v)} \neq y_{J(v)}$. In this case,

$$\left\langle u_{xJ(v)} \middle| w_{yJ(v)} \right\rangle = 1.$$

Moreover, for any $j \neq J(v)$, we have $\langle u_{xj}|\, w_{yj}\rangle = 0$ since for such j's either one of $|u_{xj}\rangle, |w_{yj}\rangle$ is zero, or these vectors correspond to different vertices, or they correspond to the same vertex $v' \in P_x \cap P_y$ with $Q_{v'}(x_{J(v')}) = Q_{v'}(y_{J(v')})$ in which case $\left|\mu_{Q_{v'}(x_{J(v')})}\right\rangle$ and $\left|\nu_{Q_{v'}(y_{J(v')})}\right\rangle$ are orthogonal. Note that here we use the fact that if $f(x) \neq f(y)$ then $\left\langle \tilde{\mu}_{f(x)} \middle| \tilde{\nu}_{f(y)} \right\rangle = 1$. As a result,

$$\sum_{j: x_j \neq y_j} \langle u_{xj}|\, w_{yj}\rangle = 1.$$

Also if $f(x) = f(y)$ then since $\left|\tilde{\mu}_{f(x)}\right\rangle$ and $\left|\tilde{\nu}_{f(y)}\right\rangle$ are orthogonal we have

$$\sum_{j: x_j \neq y_j} \langle u_{xj}|\, w_{yj}\rangle = 0.$$

Therefore, the vectors $|u_{xj}\rangle$ and $|w_{xj}\rangle$ form a feasible solution of the dual adversary SDP.

Now we compute the objective value. By assumption there are at most T edges in P_x with black color, and at most G red edges in P_x. Also the norm-squared of $|\mu_Q\rangle$'s and $|\tilde{\mu}_\alpha\rangle$'s are bounded by 2. Therefore,

$$\sum_{j=1}^{n} \| \, |u_{xj}\rangle \, \|^2 \leq 4\left(\frac{1}{W_{\text{black}}} T + \frac{1}{W_{\text{red}}} G\right).$$

Also, in computing $\sum_{j=1}^{n} \| \, |w_{xj}\rangle \, \|^2$, for every vertex $v \in P_x$, if $C(v, Q_v(x_{J(v)})) = \text{black}$ we get a term of $4W_{\text{red}}$, and if $C(v, Q_v(x_{J(v)})) = \text{red}$ we get a contribution of $4(W_{\text{black}} + W_{\text{red}})$. Now having a bound on the number of black and red edges in P_x we find that

$$\sum_{j=1}^{n} \| \, |w_{xj}\rangle \, \|^2 = 4\Big(W_{\text{red}} T + (W_{\text{black}} + W_{\text{red}}) G\Big) \leq 4\Big(2W_{\text{red}} T + W_{\text{black}} G\Big).$$

Therefore, if we let $W_{\text{black}} = \frac{1}{W_{\text{red}}} = \sqrt{\frac{T}{G}}$, then the objective value of the SDP (1) will be $O(\sqrt{GT})$.

(ii) Let $f_\zeta : D_f \to [m]$ be the function that is computed by the decision tree $\mathcal{T}_\zeta$. Then by assumption we have

$$\mathbb{E}_\zeta\Big[\delta_{f_\zeta(x),f(x)}\Big] \geq 0.9. \tag{9}$$

On the other hand, by part (i) for every ζ there is a feasible solution $\left|u^\zeta_{xj}\right\rangle$ and $\left|w^\zeta_{xj}\right\rangle$ of the dual adversary SDP for f_ζ with

$$\sum_{j:x_j\neq y_j} \left\langle u^\zeta_{xj} \middle| w^\zeta_{yj} \right\rangle = 1 - \delta_{f_\zeta(x),f_\zeta(y)},$$

such that

$$\sum_{j=1}^{n} \left\| \left| u^\zeta_{xj} \right\rangle \right\|^2 \leq 4\Big(\frac{1}{W_{\text{black}}} T^\zeta_x + \frac{1}{W_{\text{red}}} G^\zeta_x\Big),$$

and

$$\sum_{j=1}^{n} \left\| \left| w^\zeta_{xj} \right\rangle \right\|^2 \leq 4\Big(2W_{\text{red}} T^\zeta_x + W_{\text{black}} G^\zeta_x\Big).$$

Let us define

$$|u_{xj}\rangle = \frac{1}{\sqrt{K}} \sum_\zeta \left|u^\zeta_{xj}\right\rangle \otimes |\zeta\rangle\,, \tag{10}$$

and

$$|w_{xj}\rangle = \frac{1}{\sqrt{K}} \sum_\zeta \left|w^\zeta_{xj}\right\rangle \otimes |\zeta\rangle\,, \tag{11}$$

where K is the number of possible values that ζ takes. Then we have

$$\sum_{j:x_j\neq y_j} \langle u_{xj} | w_{yj}\rangle = 1 - \frac{1}{K} \sum_\zeta \delta_{f_\zeta(x),f_\zeta(y)}. \tag{12}$$

Now define

$$|\psi_x\rangle := \frac{1}{\sqrt{K}} \sum_\zeta |f_\zeta(x)\rangle\,|\zeta\rangle\,, \tag{13}$$

and consider the state generation problem for these vectors. Observe that

$$\langle \psi_x | \psi_y \rangle = \frac{1}{K} \sum_\zeta \delta_{f_\zeta(x),f_\zeta(y)}.$$

Therefore, by (12) the vectors $|u_{xj}\rangle$ and $|w_{xj}\rangle$ form a feasible solution of the dual adversary SDP (2) for this state generation problem. Letting M be the objective value of this SDP for these vectors, we conclude that with $O(M)$ quantum queries to x we can generate a state ρ_x such that $\|\rho_x - |\psi_x\rangle\langle\psi_x|\,\|_{\text{tr}} \leq 0.1$. Then measuring the first register of ρ_x in the computational basis $\Big\{\,|\alpha\rangle\ :\ \alpha\in[m],\Big\}$ we have

$$\begin{aligned}
\Pr[\text{measurement outcome equals } f(x)] &= \text{tr}\Big[\rho_x \cdot |f(x)\rangle\langle f(x)| \otimes I\Big] \\
&\geq \text{tr}\Big[\,|\psi_x\rangle\langle\psi_x| \cdot |f(x)\rangle\langle f(x)| \otimes I\Big] - 0.1 \\
&= \mathbb{E}_\zeta\Big[\delta_{f_\zeta(x),f(x)}\Big] - 0.1 \\
&\geq 0.9 - 0.1,
\end{aligned}$$

where in the last inequality we use (9). We conclude that there is a quantum query algorithm which makes $O(M)$ quantum queries and outputs $f(x)$ with probability at least 0.8. Thus we only need to bound M, the objective value of the dual adversary bound.

We compute

$$\begin{aligned}\sum_{j=1}^{n} \| \, |u_{xj}\rangle \, \|^2 &= \frac{1}{K} \sum_{\zeta} \sum_{j=1}^{n} \left\| \left| u_{xj}^{\zeta} \right\rangle \right\|^2 \\ &\leq 4 \frac{1}{K} \sum_{\zeta} \left(\frac{1}{W_{\text{black}}} T_x^{\zeta} + \frac{1}{W_{\text{red}}} G_x^{\zeta} \right) \\ &= 4 \left(\frac{1}{W_{\text{black}}} \mathbb{E}_{\zeta} \left[T_x^{\zeta} \right] + \frac{1}{W_{\text{red}}} \mathbb{E}_{\zeta} \left[G_x^{\zeta} \right] \right) \\ &\leq 4 \left(\frac{1}{W_{\text{black}}} T + \frac{1}{W_{\text{red}}} G \right)\end{aligned}$$

and similarly

$$\sum_{j=1}^{n} \| \, |u_{xj}\rangle \, \|^2 \leq 4 \Big(2 W_{\text{red}} T + W_{\text{black}} G \Big).$$

Then as before letting $W_{\text{black}} = \frac{1}{W_{\text{red}}} = \sqrt{\frac{T}{G}}$, we find that the objective value of this feasible solution is bounded by $M = O(\sqrt{GT})$. We are done. □

In the proof of Theorem 4 we assigned two different weights to edges of a decision tree based on their colors; the weight of any red edge is W_{red} and the weight of any black edge is W_{black}. One may suggest that by assigning different wights to edges of $\mathcal{T}$ we may get better bounds. That is, for any internal vertex v of $\mathcal{T}$, we may choose two weights $W_{v,\text{black}}, W_{v,\text{red}}$ and assign them to the outgoing edges of v with the corresponding colors. Then the proof of Theorem 4 can be adopted to get a bound of the form $O(\max_{x,y} \sqrt{M_x^+ M_y^-})$ on the quantum query complexity where

$$\begin{aligned}M_x^+ &= \sum_{\substack{v \in P_x: \\ C(v, Q_v(x_{J(v)}))=\text{black}}} \frac{1}{W_{v,\text{black}}} + \sum_{\substack{v \in P_x: \\ C(v, Q_v(x_{J(v)}))=\text{red}}} \frac{1}{W_{v,\text{red}}}, \\ M_x^- &= \sum_{\substack{v \in P_x: \\ C(v, Q_v(x_{J(v)}))=\text{black}}} W_{v,\text{red}} + \sum_{\substack{v \in P_x: \\ C(v, Q_v(x_{J(v)}))=\text{red}}} W_{v,\text{black}}.\end{aligned}$$

Then a simple application of the Cauchy-Schwartz inequality and $\max_{x,y} \sqrt{M_x^+ M_y^-} \geq \max_x \sqrt{M_x^+ M_x^-}$ would show that updating the weights by

$$W'_{v,\text{black}} = \frac{1}{W'_{v,\text{red}}} = \sqrt{\frac{W_{v,\text{black}}}{W_{v,\text{red}}}},$$

would improve the upper bound $O(\max_{x,y} \sqrt{M_x^+ M_y^-})$. As a result, with no loss of generality we may assume that

$$W_{v,\text{black}} = \frac{1}{W_{v,\text{red}}}.$$

Nevertheless, we still have the freedom to choose different weights for vertices of the decision tree $\mathcal{T}$. These weights could depend on some parameter of the state of algorithm (decision tree) that is updated as we proceed. Moreover, it could depend on the guessing algorithm, e.g., on the number of red edges we have seen so far. In the following theorem, we analyze the latter option, and leave further investigation of this idea for future works.

Theorem 5. *Let $\{\mathcal{T}_\zeta : \zeta\}$ be a set of generalized decision trees corresponding to a* randomized *classical query algorithm evaluating f with bounded error as in (5). Moreover, suppose that each $\mathcal{T}_\zeta$ is equipped with a G-coloring. Let P_x^ζ be the path from the root to the leaf of $\mathcal{T}_\zeta$ associated to $x \in D_f$. Let G_x^ζ be the number of red edges in P_x^ζ, and for $1 \leq g \leq G_x^\zeta$, let $T_{g,x}^\zeta$ be the number of black edges in P_x^ζ after the g-th red edge and before the next red one. Also let $T_{0,x}^\zeta$ be the number of black edges before the first red edge in P_x^ζ, and let $T_{g,x}^\zeta = 0$ for $g > G_x^\zeta$. Let $G = \max_{x,\zeta} G_x^\zeta$ and define*

$$T_g = \max_x \mathbb{E}_\zeta[T_{g,x}^\zeta], \qquad 0 \leq g \leq G.$$

where the expectation is over the random choice of ζ. Then the quantum query complexity of f is

$$O\left(\sum_{g=1}^{G} \sqrt{T_g}\right).$$

Proof. The proof is similar to the proof of Theorem 4 except that we pick different weights for edges of the decision trees. Using the notations we used before, for any choice of ζ and its associated decision tree $\mathcal{T}_\zeta$ define

$$\left|u_{xj}^\zeta\right\rangle = \begin{cases} \frac{1}{\sqrt{W_{g(v),C(v,Q_v(x_j))}}} \left|v, C(v, Q_v(x_j))\right\rangle \otimes \left|\mu_{Q_v(x_j)}\right\rangle \otimes \left|\tilde{\mu}_{f_\zeta(x)}\right\rangle & \text{if } \exists v \in P_x^\zeta \cap V_j(\mathcal{T}_\zeta) \\ 0 & \text{otherwise,} \end{cases} \tag{14}$$

and

$$\left|w_{xj}^\zeta\right\rangle = \begin{cases} \sum_{c \in C_{v,xj}} \sqrt{W_{g(v),c}} \left|v, c\right\rangle \otimes \left|\nu_{Q_v(x_j)}\right\rangle \otimes \left|\tilde{\nu}_{f_\zeta(x)}\right\rangle & \text{if } \exists v \in P_x^\zeta \cap V_j(\mathcal{T}_\zeta) \\ 0 & \text{otherwise,} \end{cases} \tag{15}$$

where as before $C_{v,xj}$ is given by (8), and $g(v)$ is the number of red edges in the path from the root of $\mathcal{T}_\zeta$ to v. Moreover, $W_{g,\text{black}}, W_{g,\text{red}}$, for any $g \geq 0$, are positive weights to be determined. As before, these vectors form a feasible solution of the SDP(1) for the function f_ζ. Then we define vectors $|u_{xj}\rangle$, $|w_{xj}\rangle$ and $|\psi_x\rangle$ as in (10), (11) and (13). As before, we obtain a feasible solution to the SDP (2) whose objective value is an upper bound on the quantum query complexity of f. We estimate the objective value as follows.

Let $W_g = W_{g,\text{black}} = \frac{1}{W_{g,\text{red}}}$, then

$$\begin{aligned}\sum_{j=1}^{n} \| \, |u_{xj}\rangle \, \|^2 &= \frac{1}{K} \sum_{\zeta} \sum_{j=1}^{n} \left\| \left| u_{xj}^{\zeta} \right\rangle \right\|^2 \\ &\leq 4 \frac{1}{K} \sum_{\zeta} \left(\frac{1}{W_0} T_{0,x}^{\zeta} + \sum_{g=1}^{G} \left(\frac{1}{W_g} T_{g,x}^{\zeta} + W_g \right) \right) \\ &= 4 \left(\frac{1}{W_0} T_{0,x} + \sum_{g=1}^{G} \left(\frac{1}{W_g} T_{g,x} + W_g \right) \right) \\ &\leq 4 \left(\frac{1}{W_0} T_0 + \sum_{g=1}^{G} \left(\frac{1}{W_g} T_g + W_g \right) \right).\end{aligned}$$

Then letting $W_0 = T_0$ and $W_g = \sqrt{T_g}$ for $g \geq 1$ we obtain[2]

$$\sum_{j=1}^{n} \| \, |u_{xj}\rangle \, \|^2 = O\left(\sum_{g=1}^{G} \sqrt{T_g} \right).$$

We similarly obtain the same upper bound on $\sum_{j=1}^{n} \| \, |w_{xj}\rangle \, \|^2$. Then the quantum query complexity of f is bounded by $O\left(\sum_{g=1}^{G} \sqrt{T_g}\right)$. □

5 Applications

We can use our main result, Theorem 4, to simplify the proof of some known quantum query complexity bounds as well as to derive new bounds. We start with some simple examples.

Proposition 6. *Suppose that we have query access to a list $x = (x_1, x_2, \ldots, x_n) \in [\ell]^n$. Also let $q \in [\ell]$ and $1 \leq k < n$ be fixed.*

(i) [COUNTING] *The quantum query complexity of finding* all *input indices with values equal to q is $O(\sqrt{rn})$, where $\left|\left\{j : x_j = q\right\}\right| \leq r$.*

(ii) [K-THRESHOLD] *The quantum query complexity of deciding whether $\left|\left\{j : x_j = q\right\}\right| \leq k$ or not is $O(\sqrt{kn})$.*

It is shown that the quantum query complexity of counting equals $\Theta(\sqrt{rn})$ [BHT]. Also it is well-known that the k-threshold problem has quantum query complexity $O(\sqrt{kn})$.

Proof. (i) In order to use Theorem 4 we first need a classical query algorithm. Suppose that we start from the first index and query all the indices one by one. We then output the set of indices j with $x_j = q$. Next we need a G-coloring. To this end, observe that the algorithm is ignorant of the exact value of some index x_j once

[2]Note that $T_g \neq 0$ since every internal vertex of a decision tree has an outgoing black edge.

it makes sure that $x_j \neq q$. Thus is the associated decision tree $\mathcal{T}$ we can unify all outgoing edges of a vertex with label $q' \neq q$. That is, in $\mathcal{T}$ there are two outgoing edges for any vertex that are labeled by $\{q\}$ and $[\ell] \setminus \{q\}$. Now we color all edges with label $\{q\}$ red and color the edges with label $[\ell] \setminus \{q\}$ black. In this coloring there are at most r red edges in any path from the root to leaves: $G = r$. The depth of the decision tree is $T = n$. As a result the quantum query complexity of quantum counting is $O(\sqrt{rn})$.

(ii) The proof is similar to that of part (i). In the classical algorithm we query indices one by one until we find k indices j with $x_j = q$. Then in $\mathcal{T}$ we unify edges with label $q' \neq q$ and color them black, and color edges with label $\{q\}$ red. As the algorithm stops once it faces k indices with value q, the number of red edges in any path in $\mathcal{T}$ from the root to leaves is at most $G = k$. Also the depth of the tree is $T = n$. Therefore the quantum query complexity of the threshold problem is $O(\sqrt{kn})$. □

Proposition 7. *Let $x = (x_1, \ldots, x_n)$ be a list of n numbers.*

(i) [MIN] *The quantum query complexity of finding $\min_j x_j$ is bounded by $O(\sqrt{n \log n})$.*

(ii) [K-MIN] *The problem of finding a subset $S \subseteq \{1, \ldots, n\}$ of size $|S| = k$ such that for all $j \notin S$ we have $x_j \geq \max_{i \in S} x_i$ has quantum query complexity $O(\sqrt{kn \log n})$.*

Two remarks are in line regarding the examples of minimum finding. First, our bounds in these examples are tight only up to a factor of $\sqrt{\log n}$ [DH, DHHM06]. Yet, we would like to present these results since they show how randomization (part (ii) of Theorem 4) may help to improve upper bounds on the quantum query complexity.

Second, observe that a list of numbers may have several minimums, so the problems in this proposition are not really function problems. To turn them into functions we may assume that our goal is to find the minimum number in the list whose index is also minimum. In other words, we consider a new order " $\prec$ " such that $x_i \prec x_j$ if $x_i < x_j$, or if $x_i = x_j$ and $i < j$. Now the minimum in this order is unique and we may ask for finding it.

Proof. (i) Consider the randomized classical algorithm that queries all indices one by one in a random order. The algorithm keeps a candidate for minimum at each step, and updates it once it reaches a smaller number. Observe that this algorithm is ignorant of the exact answer to a query once it makes sure that it is not smaller than the current candidate for minimum. Thus in the associated decision tree (for any choice of random order ζ), at any internal vertex v we can unify outgoing edges with label in $\{q : \ q \geq m_v\}$ where m_v is the candidate for minimum at node v. Thus in $\mathcal{T}_\zeta$ any internal vertex v has an outgoing edge with label $\{q : \ q \geq m_v\}$ and an outgoing edge for any other $q < m_v$. The former edge is colored black and the latter edges are colored red. The depth of $\mathcal{T}_\zeta$ equals $T = n$ for any ζ. However, for a given x, G_x^ζ depends on ζ, so we should compute

$$G = \max_x \mathbb{E}_\zeta[G_x^\zeta].$$

We claim that $G = O(\log n)$. Intuitively speaking, the expected number of x_j's that are smaller than the first queried element is $n/2$, and the guessing algorithm does not make mistakes once we query such x_j's. Thus, after the first query, in expectation, half of the x_j's would become irrelevant in computing G. Repeating this argument, we obtain $G = O(\log n)$. Below we present a more precise argument for this claim.

We can assume with no loss of generality that $x_1 < \cdots < x_n$, since in the beginning of the algorithm we apply a random permutation. If in the random permutation $\zeta = (\zeta(1), \ldots, \zeta(n))$ the first element is n, i.e., $\zeta(1) = n$, then $G_n^\zeta = G_{n-1}^{\zeta'} + 1$ where $\zeta' = (\zeta(2), \ldots, \zeta(n))$. Otherwise, if $\zeta(1) \neq n$ then $G_n^\zeta = G_{n-1}^{\zeta''}$ where ζ'' is the same order as ζ from which n is removed. We conclude that

$$\mathbb{E}[G_n^\zeta] = \frac{1}{n}\Big(\mathbb{E}\Big[G_{n-1}^{\zeta'}\Big] + 1\Big) + \frac{n-1}{n}\mathbb{E}\Big[G_{n-1}^{\zeta''}\Big].$$

Therefore, letting $G_n = \mathbb{E}[G_n^\zeta]$ we have

$$G_n = G_{n-1} + \frac{1}{n}.$$

Using $G_1 = 1$ we obtain

$$G_n = \sum_{t=1}^{n} \frac{1}{t} = O(\log n).$$

As a result, $G = O(\log n)$ and by Theorem 4 the quantum query complexity of finding the minimum is bounded by $O(\sqrt{n \log n})$.

(ii) The proof is similar to that of part (i). Again we read the numbers in a random order and update a k-list as our candidate for S as we reach a number that is smaller than all the number in the list. The associated decision tree and its G-coloring is as before. Again we would have $T = n$. Also by similar ideas as in the proof of part (i) it can be shown that $G_n = G_{n-1} + k/n$ because with probability k/n the largest x_j appears in the first k numbers in a random permutation. Therefore, $G = \max_x \mathbb{E}_\zeta[G_x^\zeta] = O(k \log n)$. We conclude that the quantum query complexity of finding the k smallest numbers is bounded by $O(\sqrt{kn \log n})$.

□

Motivated by Proposition 6 we can state the following general upper bound on the quantum query complexity of functions.

Corollary 8. *For any* partial *function* $f : D_f \to [m]$ *where* $D_f \subseteq [\ell]^n$ *and* $\forall q \in [\ell]$, *let*

$$r_q(x) := \Big|\{j : x_j \neq q\}\Big| \qquad \text{and} \qquad g = \min_{q \in [\ell]} \max_{x \in D_f} r_q(x).$$

Then if the classical query complexity of f *is* T, *the quantum query complexity of* f *is* $O(\sqrt{gT})$. *In particular, the quantum query complexity of* f *is* $O(\sqrt{gn})$.

Proof. We prove this corollary using Theorem 4. Given the classical algorithm for f, for a G-coloring of the edges of the associated decision tree, color every edge of the decision tree with label q_0 black and the rest of the edges red, where q_0 is

such that $g = \max_{x \in D_f} r_{q_0}(x)$. Then since each $x \in D_f$ contains at most g indices with values q_0, in every path from the root to leaves of the decision tree we see at most $G = g$ red edges. Then the quantum quantum query complexity of f is $O(\sqrt{GT}) = O(\sqrt{gT})$. □

5.1 Graph properties in the adjacency matrix model

In this subsection and the following one we use Theorem 4 to prove quantum query complexity upper bounds on some graph theoretic problems. In this subsection, we assume that the graph is given in the *adjacency matrix model*, by which we mean that the queries are from the entries of the adjacency matrix of the graph. That is, given vertices u, v of the graph, we may ask whether there is an edge between u and v or not. Sometimes we assume that the underlying graph is directed in which case we ask whether there is a *directed* edge from u to v.

Inspired by the ideas in [LL16], we make use of the well-known Breadth First Search algorithm (BFS, see Algorithm 1) as our starting point for designing classical algorithms for some graph theoretic problems. The point of the BFS algorithm is that it returns a spanning tree (forest), with at most $n-1$ edges, of the underlying graph. Thus if we always guess that there is no edge between two queried vertices, we make at most $n-1$ mistakes.

Algorithm 1 BFS($\mathcal{G}$): breadth first search algorithm on graph $\mathcal{G}$

Let L be a list of unprocessed vertices and Q be a first in first out queue.
$L \leftarrow V(\mathcal{G})$, $Q = \emptyset$, $E_{\mathcal{S}} = \emptyset$ ▷ $E_{\mathcal{S}}$ stores the edge set of the BFS tree.
while there exists a $v' \in L$ **do**
    add v' to Q
    $L \leftarrow L - v'$
    while $Q \neq \emptyset$ **do**
        $u \leftarrow$ dequeue(Q)
        while there exists a $v \in L$ **do**
            Query (u, v)
            if $(u, v) \in E(\mathcal{G})$ **then**
                add (u, v) to $E_{\mathcal{S}}$
                add v to Q
                $L \leftarrow L - v$
            end if
        end while
    end while
end while
return the BFS forest $\mathcal{S} = \left(V(\mathcal{G}), E_{\mathcal{S}}\right)$

Proposition 9. *Suppose that we have query access to the adjacency matrix of a simple*[3] *(possibly directed) graph $\mathcal{G}$ on n vertices. Then the followings hold.*

[3]We can derive the same results for non-simple graphs by making minor modifications in the proofs.

(i) [BIPARTITENESS] *The quantum query complexity of deciding whether $\mathcal{G}$ is bipartite or not is $O(n^{3/2})$.*

(ii) [CYCLE DETECTION] *The quantum query complexity of deciding whether $\mathcal{G}$ is a forest or has a cycle is $O(n^{3/2})$.*

(iii) [DIRECTED ST-CONNECTIVITY] *The quantum query complexity of finding a shortest path (the path that consists of the least number of edges) between two vertices s and t in $\mathcal{G}$ is $O(n^{3/2})$. This holds for either directed or undirected graphs.*

(iv) [SMALLEST CYCLES CONTAINING A VERTEX] *The quantum query complexity of finding the length of the smallest* directed *cycle containing a given vertex v in a directed graph $\mathcal{G}$ is $\Theta(n^{3/2})$.*

(v) [k-CYCLE CONTAINING A VERTEX] *The quantum query complexity of deciding whether $\mathcal{G}$ has a cycle of length k, for a fixed k, containing a given vertex v is $O((2k)^{(k-1)}n^{3/2})$.*

The problem of bipartiteness has been first shown in [Āri15] to have quantum query complexity $O(n^{3/2})$, which is shown to be tight in [Zha05]. An algorithm for the problem of cycle detection with $O(n^{3/2})$ queries is proposed in [CMB] that works by reducing the problem to the *st-connectivity* problem. This upper bound is known to be tight [CK]. For the directed st-connectivity problem, it has been first shown to have query complexity $\Theta(n^{3/2})$ in [DHHM06]. There exists a quantum query algorithm for deciding whether $\mathcal{G}$ contains a cycle of length less than k containing a given vertex v with query complexity $O(n\sqrt{k})$ [CMB]. For a list of related algorithms on cycle detection consult [Cir06].

We would like to remark that the space complexity of all BFS/DFS-based quantum query algorithms in this subsection and the next one are linear in the size of the input graph. This is because our algorithms are based on feasible solutions of the dual adversary SDP that are obtained from a generalized decision tree. Now the point is that the space complexity of such an algorithm equals the *logarithm* of the dimension of the vectors in the feasible solution of the dual adversary SDP, that itself equals the size of the decision tree which is exponential.

Proof. (i) A graph $\mathcal{G}$ is bipartite iff its vertices can be properly colored with two colors blue and green (such that no two adjacent vertices have the same color). Here is a classical algorithm to solve bipartiteness. We run the BFS algorithm (Algorithm 1) that outputs a spanning forest $\mathcal{S}$ of $\mathcal{G}$. Then we color every vertex of $\mathcal{G}$ with odd depth in $\mathcal{S}$ blue, and every vertex of $\mathcal{G}$ with even depth in $\mathcal{S}$ green. After this coloring, we search for an edge between two vertices with the same color in $\mathcal{G}$. If no such edge exists, then $\mathcal{G}$ is bipartite.

In order to use Theorem 4, in the associated decision tree $\mathcal{T}$ of the above algorithm, color every outgoing edge of $\mathcal{T}$ with label 1 red, and the rest of edges black. The depth of the decision tree is $T \leq n^2$ as the total number of possible queries (possible edges) for $\mathcal{G}$ is $n(n-1)/2$. Also, by the above coloring of edges of $\mathcal{T}$, we see at most n red edges in every path from the root to leaves of $\mathcal{T}$. Indeed,

we see at most $n-1$ red edges once we build the spanning forest $\mathcal{S}$, and at most 1 red edge once we search for an edge in $\mathcal{G}$ between vertices with the same parity depths. Thus $G \leq n$ and the quantum query complexity of bipartiteness is at most $O(\sqrt{GT}) = O(n^{3/2})$.

(ii) In a classical algorithm for this problem we first build a BFS forest and then search for an edge in the whole graph that does not belong to the BFS forest. If such an edge exists it should belong to a cycle in $\mathcal{G}$. In order to use Theorem 4, in the associated decision tree $\mathcal{T}$, as before, we color every edge of $\mathcal{T}$ with label 0 black, and edges with label 1 by red. The depth of the decision tree is $T \leq n^2$, and using this coloring in every path from the root to leaves of the decision tree there are at most $G = n$ red edges. Therefore, the quantum query complexity of the cycle detection problem is $O(n^{3/2})$.

(iii) Again we run the BFS algorithm on $\mathcal{G}$ starting from vertex s to build a subtree $\mathcal{S}$ of $\mathcal{G}$ with root s. Then a shortest path from s to t, if exists, belongs to $\mathcal{S}$, and can be found once we have $\mathcal{S}$. The depth of the associated decision tree is $T = n^2$. For the G-coloring, as before, we color every edge with label 0 black and other edges red to get $G = n$. Then the quantum query complexity of directed st-connectivity is $O(\sqrt{GT}) = O(n^{3/2})$.

(iv) In a classical algorithm for this problem we may run the BFS algorithm starting from vertex v. In parallel, whenever we reach a new vertex u we query if there is an edge from u to v. Finding such an edge corresponds to a smallest cycle containing v. As previous examples for a G-coloring of the associated decision tree, we color every edge with label 0 black and other edges red, then we have $G = n$ and $T = n^2$. Therefore, the quantum query complexity of deciding whether $\mathcal{G}$ has a cycle containing v is $O(n^{3/2})$.

To prove the optimality of this bound we reduce the problem of directed st-connectivity which has query complexity $\Omega(n^{3/2})$ to this problem. Assume that we are given a graph $\mathcal{G}$ and two distinguished vertices $s, t \in V(\mathcal{G})$, and we want to decide whether s is connected to t by a directed path or not. To solve this problem we build an auxiliary graph $\mathcal{H}$ form $\mathcal{G}$ as follows.

$$V(\mathcal{H}) = V(\mathcal{G}) \cup \{w\}, \qquad E(\mathcal{H}) = E(\mathcal{G}) \cup \{(w, s), (t, w)\}$$

Now s is connected to t in $\mathcal{G}$ if and only if there is a directed cycle in $\mathcal{H}$ containing the vertex w. Moreover, if such a cycle exists, its length equals the distance of s, t in $\mathcal{G}$ plus two.

(v) In a classical algorithm for this problem, we first define an auxiliary directed graph $\mathcal{H}$ out of $\mathcal{G}$ with $V(\mathcal{G}) = V(\mathcal{H})$. To define the edge set of $\mathcal{H}$ we use two random functions $C : V(\mathcal{G}) \to [k]$ and $D : E(\mathcal{G}) \to \{-1, +1\}$, and let

$$E(\mathcal{H}) = \Big\{(u, w) \in E(\mathcal{G}) \,:\, C(u) = C(w) + 1(\mod k),\, D(u, w) = +1\Big\}.$$

Observe that if $\mathcal{G}$ has a cycle of length k containing v, then with probability at least $\frac{1}{(2k)^{k-1}}$, which is a constant, $\mathcal{H}$ has a directed cycle of length k containing v. Otherwise, $\mathcal{H}$ does not have any cycle of length k containing v. Moreover, the length

of all cycles of $\mathcal{H}$ are multiples of k. Thus, the aforementioned cycle of $\mathcal{H}$, if exists, is the smallest possible cycle. Then we can decide the existence of such a cycle using the algorithm of part (iv). We can decide the existence of such a cycle with high probability by repeating the above algorithm $O\left((2k)^{(k-1)}\right)$ times.

□

For the next set of examples we use the well-known classical algorithm Depth First Search (DFS). This algorithm builds a spanning forest of a given graph $\mathcal{G}$. It is similar to the BFS algorithm but instead of using a queue which is a first in first out list, it uses a stack which is a first in last out list. This algorithm can also be implemented recursively (see Algorithm 2).

Algorithm 2 DFS($\mathcal{G}$): depth first search algorithm on graph $\mathcal{G}$

let L be a list of undiscovered vertices
let ft be an array of size $|V(\mathcal{G})|$ ▷ ft stores the finishing time of vertices.
function DFS($\mathcal{G}$)
    $L \leftarrow V(\mathcal{G})$
    $time = 1$
    while there exists a $v \in L$ **do**
        DFS($\mathcal{G}, v$)
    end while
    Return the DFS tree
end function
procedure DFS($\mathcal{G}, s$)
    $L \leftarrow L - s$
    while there exists a $v \in L$ **do**
        Query (s, v)
        if $(s, v) \in E(\mathcal{G})$ **then**
            DFS($\mathcal{G}, v$)
        end if
    end while
    $ft[s] \leftarrow time$
    $time \leftarrow time + 1$
end procedure

Proposition 10. *Suppose that we have query access to the adjacency matrix of a directed graph $\mathcal{G} = (V, E)$ on n vertices. Then the followings hold.*

(i) [TOPOLOGICAL SORT] *Suppose that $\mathcal{G}$ is acyclic. Then the quantum query complexity of finding a vertex ordering of $\mathcal{G}$ such that for all $(u, v) \in E$, u appears before v is $O(n^{3/2})$.*

(ii) [CONNECTED COMPONENTS] *The quantum query complexity of determining connected components of $\mathcal{G}$ is $O(n^{3/2})$.*

(iii) [STRONGLY CONNECTED COMPONENTS] *The quantum query complexity of finding strongly connected components of $\mathcal{G}$ is $O(n^{3/2})$. Note that two vertices $u, v \in V$ belong to the same strongly connected component iff there exists a directed path from u to v and a directed path from v to u in $\mathcal{G}$.*

The problem of topological sort is an important problem in large networks and job scheduling. There are several classical algorithms for this problem. The first algorithm is by Kahn [Kah62]. In this algorithm at each step we add all vertices that do not have any incoming edges to the sorted list, and then eliminate them from the original graph. We continue this process until we add all vertices to the sorted list. Another algorithm for this problem, which we use in this proposition, is based on the DFS algorithm, first stated by Tarjan [Tar76]. Note that in these classical algorithms ones needs to read the entire input to discover the structure of the graph, so their query complexity is $O(n^2)$. To the author's knowledge this proposition gives the first non-trivial quantum query complexity upper bound for the topological sort problem. The problem of finding (strongly) connected components of a (directed) graph has been first shown to have query complexity $\Theta(n^{3/2})$ in [DHHM06].

Proof. (i) For a classical algorithm for this problem, run DFS and return vertices in their reverse of finishing time. For a G-coloring of the associated decision tree $\mathcal{T}$, color every edge with label 0 black and every other edge red. Then as before there are at most $G = n$ red edges in every path from root to leaves of $\mathcal{T}$. Also the depth of the decision tree is $T = n^2$. Thus we obtain the bound of $O(\sqrt{GT}) = O(n^{3/2})$ on quantum query complexity of topological sort.

(ii) We again use the DFS algorithm on $\mathcal{G}$ and whenever the stack becomes empty a new connected component has been found. The G-coloring of the associated decision tree is as in part (i), and the bound of $O(n^{3/2})$ is derived similarly.

(iii) As a classical algorithm for this problem we use two DFS calls. In the first one we run the DFS algorithm on a reverse graph $\mathcal{G}^R$ whose adjacency matrix is the transpose of the adjacency matrix of $\mathcal{G}$, i.e., $(u, v) \in E(\mathcal{G}^R)$ iff $(v, u) \in E(\mathcal{G})$. Observe that every query to $\mathcal{G}^R$ is equivalent to a query to $\mathcal{G}$. In the second one, the DFS will be run on the graph $\mathcal{G}$ in the reverse finishing time ordering [4] of vertices from the first DFS run. Here we use the fact that if we start the DFS somewhere in a sink component[5] then we exactly traverse that component. In the resulted DFS forest, vertices in every tree are in the same strongly connected component. For a G-coloring of the decision tree, we color every edge with label 0 black and every other edges red, so that $G \leq 2n$. The depth of the decision tree is $T = n^2$. Therefore, the quantum query complexity of this problem is $O(n^{3/2})$. □

The following corollary is a simple consequence of Corollary 8.

[4]This is a reverse topological order of vertices of $\mathcal{G}$. Therefore, a vertex at the end of this list is in a sink component.

[5]A sink component is a set of vertices $I \subseteq V(\mathcal{G})$ such that $\forall u \in I, v \in V(\mathcal{G}) \setminus I$ we have $(u, v) \notin E(\mathcal{G})$.

Corollary 11. *The quantum query complexity of every graph property of a general graph*[6] *in the adjacency matrix model, is* $O(n\sqrt{|E(\mathcal{G})|})$ *which is faster than the trivial algorithm when* $|E(\mathcal{G})| = o(n^2)$*. In particular, every sparse graph property in the adjacency matrix model has quantum query complexity* $O(n^{3/2})$*.*

The fact that any sparse graph property (particularly minor-closed graph properties) have quantum query complexity $O(n^{3/2})$ has been proven in [CK].

5.2 Graph properties in the adjacency list model

In this subsection we present some bounds on the quantum query complexity of some graph properties when the underlying graph is given in the *adjacency list model*. Let us first describe what we mean by this model.

In the adjacency list model we assume that the graph is given by an array of size $n(n-1)$ which for simplicity we think of it as a matrix of size $n \times (n-1)$. The j-th row of this matrix is a list of neighbors of the j-th vertex v_j of the graph. Assume that v_j has degree d_{v_j}. Then the first d_{v_j} coordinates of the j-row contain the indices of the neighbors of v_j (in some order), and the last $n-1-d_{v_j}$ coordinates are filled with a *nil* symbol. See Figure 2 for an example. Any query in the adjacency list model corresponds to a pair (v_j, i) with $i \leq n-1$. If $i \leq d_{v_j}$, then the output of this query is the i-th adjacent vertex of v_j in $\mathcal{G}$. If $i > d_{v_j}$, the output of this query is nil. This model can also be defined for directed graphs similarly. The only difference is that the j-th row of the matrix contains vertices that can be reached from v_j by a *directed* edge.

In the following we will use the BFS algorithm in the adjacency list model (see Algorithm 3) as a primitive to use Theorem 4. In the decision tree $\mathcal{T}$ associated to this BFS algorithm, each node (query) corresponds to a pair (v, i). The set of possible answers to such a query is the vertex set of $\mathcal{G}$ which we partition as follows. We let $W(v, i)$ be the set of vertices that has been added to the BFS tree before querying (v, i). The point is that the BFS algorithm is ignorant of the exact answer of the query (v, i) once it makes sure that it belongs to $W(v, i)$ (see Figure 2 for an example). Thus in the decision tree $\mathcal{T}$ we identify the outgoing edges of (v, i) with labels in $W(v, i)$. All the other outgoing edges remain untouched. Now the G-coloring of $\mathcal{T}$ is as follows: we color the outgoing edge of (v, i) with label $W(v, i)$ black, and the rest of outgoing edges red. We note that there are n vertices to be added to the BFS tree one-by-one, and we face a red edge once we add a new vertex or a nil. Then in total we see at most $G = O(n)$ red edges in every path from the root to leaves of $\mathcal{T}$. Also the total number of queries in the BFS algorithm equals the number of edges of $\mathcal{G}$ denoted by $m = |E(\mathcal{G})|$ plus n. This is because as we do not know the degrees of vertices, we would stop querying neighbors of a vertex after seeing a nil symbol. This adds an extra query for every vertex. Thus $T = m + n$, and the quantum query complexity of finding the BFS tree in the adjacency list model is $O(\sqrt{GT}) = O(\sqrt{(m+n)n})$.

[6] This applies to weighted, unweighted, directed or undirected graphs.

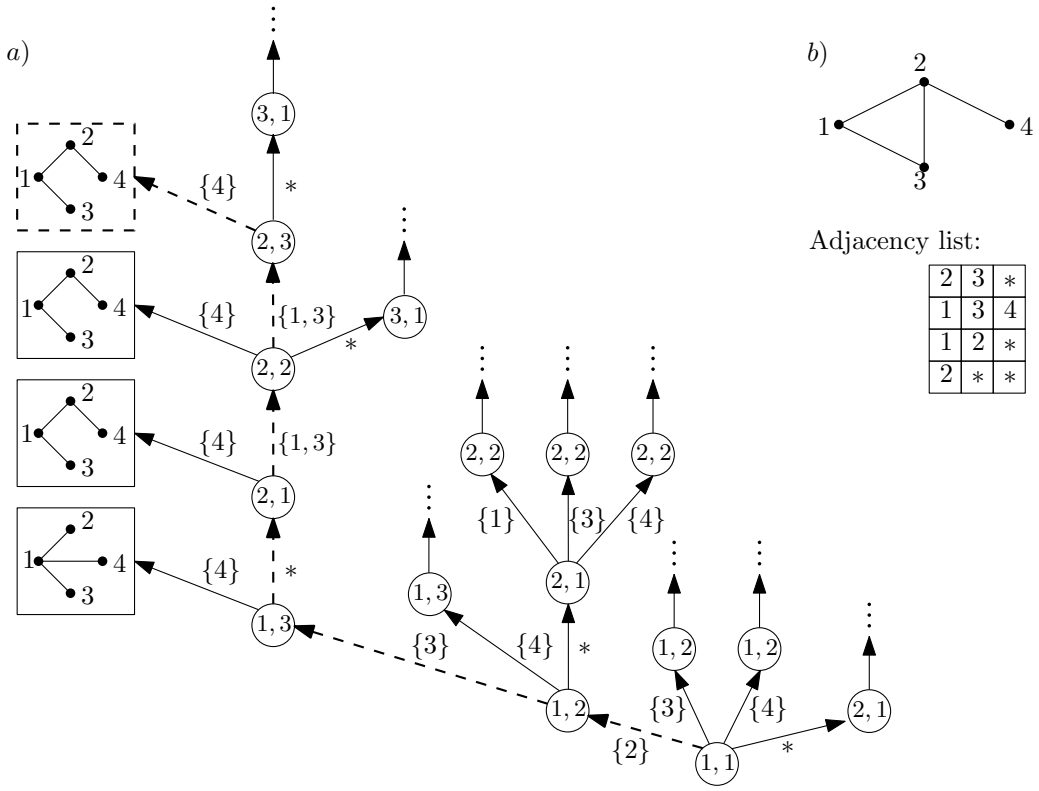

Figure 2: a) The decision tree for finding a BFS tree in a graph with 4 vertices. Each vertex of this decision tree is labeled by a pair which points to an entry of the matrix of adjacency list model. The nil symbol is represented by $*$. The dashed path, is the path we take if the input to the BFS algorithm is the graph depicted on the right hand side. b) A graph and its adjacency list representation.

Proposition 12. *Suppose that the graph $\mathcal{G}$ with n vertices and m edges is given via the adjacency list model. Then the following hold.*

(i) [DIRECTED ST-CONNECTIVITY] *Finding a shortest (directed or undirected) path between two vertices s, t in $\mathcal{G}$ has quantum query complexity $O\left(\sqrt{(m+n)n}\right)$.*

(ii) [BIPARTITNESS] *The quantum query complexity of deciding whether $\mathcal{G}$ is bipartite or not is $O\left(\sqrt{(m+n)n}\right)$.*

(iii) [MAXIMUM BIPARTITE MATCHING] *Assuming that $\mathcal{G}$ is unweighted and bipartite, the quantum query complexity of finding a maximum bipartite matching in $\mathcal{G}$ is $O\left(n^{3/4}\sqrt{m+n}\right)$.*

(iv) [TOPOLOGICAL SORT] *Suppose that $\mathcal{G}$ is acyclic. Then the quantum query complexity of finding a vertex ordering of $\mathcal{G}$ such that for all $(u, v) \in E$, u appears before v is $O\left(\sqrt{(m+n)n}\right)$.*

(v) [CONNECTED COMPONENTS] *The quantum query complexity of determining connected components of $\mathcal{G}$ is $O\left(\sqrt{(m+n)n}\right)$.*

Algorithm 3 BFS($\mathcal{G}$): breadth first search algorithm on graph $\mathcal{G}$ in adjacency list model

Let W be a list of discovered vertices and Q be a first in first out queue.
$W \leftarrow \emptyset$, $Q \leftarrow \emptyset$, $E_{\mathcal{S}} \leftarrow \emptyset$
while there exists a $v' \in V(\mathcal{G}) \setminus L$ **do**
    add v' to Q
    $W \leftarrow W \cup \{v'\}$
    while $Q \neq \emptyset$ **do**
        $u \leftarrow$ dequeue(Q)
        $v \leftarrow$Query $(u, 1)$
        $i \leftarrow 2$
        while $v \neq$ nil **do**
            $v \leftarrow$ Query (u, i) ▷ returns the i-th neighbor of vertex u
            $i \leftarrow i + 1$
            if $v \in V(\mathcal{G}) \setminus W$ **then**
                add (u, v) to $E_{\mathcal{S}}$
                add v to Q
                $P \leftarrow W \cup \{v\}$
            end if
        end while
    end while
end while
return the BFS forest $\mathcal{S}\big(V(\mathcal{G}), E_{\mathcal{S}}\big)$

Having query access to the adjacency list of a directed graph $\mathcal{G}$, it has been proved in [DHHM06] that finding a minimum spanning tree of $\mathcal{G}$ has quantum query complexity $O(\sqrt{mn})$. Using minimum spanning tree one can prove that checking directed st-connectivity and graph bipartiteness have quantum query complexity $O(\sqrt{mn})$ in the adjacency list model. Lin and Lin [LL16] proved the upper bound of $O(n^{7/4})$ for the problem of maximum bipartite matching in the adjacency matrix model. Here using their ideas we prove the first non-trivial upper bound for this problem in the adjacency list model.

Proof. (i) To find a shortest path we run the BFS algorithm in the adjacency list model starting from the vertex s. Then s and t will be connected in the resulting spanning forest with their shortest path. As discussed before, the quantum query complexity of finding this BFS spanning forest is $O\left(\sqrt{(m+n)n}\right)$. Thus a shortest path between s, t can be found with $O\left(\sqrt{(m+n)n}\right)$ quantum queries.

(ii) In the classical algorithm for this problem we start by finding a spanning tree on $\mathcal{G}$ by running the BFS Algorithm 3. We then color vertices of $\mathcal{G}$ using the resulting spanning forest $\mathcal{S}$ with two colors blue and green. We color every vertex of $\mathcal{G}$ with even depth in $\mathcal{S}$ blue, and every vertex with odd depth in $\mathcal{S}$ green. Then we search for two adjacent vertices in $\mathcal{G}$ with the same color. If we find such an edge, the graph is not bipartite, and is bipartite otherwise. The G-coloring of the associated

decision tree $\mathcal{T}$ is as follows. In the first part that we run the BFS algorithm the G-coloring is as before. In the second part that we search for an edge between two vertices of the same color, we partition the set of possible answers (vertices of $\mathcal{G}$) to in two parts: the set of blue vertices and the set of green vertices. As we query (v, i), i.e., the i-th neighbor of v in $\mathcal{G}$, the color of the two outgoing edges associated to this query labeled by sets of blue and green vertices would be colored as follows: if v is blue, the outgoing edge of blue vertices is colored red and the other one is colored black; if v is green the outgoing edge of green vertices is colored red and the other one is colored black. Observe that in the second part of the algorithm, once we see a red edge of $\mathcal{T}$ the algorithm halts (and $\mathcal{G}$ would not be bipartite). Thus in total we see at most $G = n$ red edges in any path from the root to leaves of $\mathcal{T}$. On the other hand, the depth of the decision tree is $T = m+n$. Therefore, the quantum query complexity of this problem is $O\left(\sqrt{(m+n)n}\right)$.

Algorithm 4 Hopcroft-Karp algorithm for maximum bipartite matching on graph $\mathcal{G} = (X \cup Y, E)$

$\mathcal{M} = \emptyset$ ▷ $\mathcal{M}$ is an empty matching and will be updated until becoming a maximum matching
while $\mathcal{M}$ is not a maximum matching **do**
    define an auxiliary directed graph $\mathcal{G}' = (V', E')$ as follows

$$E' = \{(s,x)|x \in X, \forall y \in Y : (x,y) \notin \mathcal{M}\} \cup \{(y,t)|y \in Y, \forall x \in X : (x,y) \notin \mathcal{M}\}$$
$$\cup \{(x,y)|x \in X, y \in Y, (x,y) \notin \mathcal{M}\} \cup \{(y,x)|x \in X, y \in Y, (x,y) \in \mathcal{M}\}$$
$$V' = X \cup Y \cup \{s,t\}$$

    ▷ any query to the adjacency list of $\mathcal{G}'$ can be simulated using a query to the adjacency list of $\mathcal{G}$
    S= a maximal set of vertex disjoint shortest paths from s to t in $\mathcal{G}'$ ▷ this can be found using one call to the algorithm 3 in $\mathcal{G}'$
    if $S = \emptyset$ **then return** $\mathcal{M}$
    else
        for every path $(s, x_1, y_1, x_2, y_2, \ldots, x_p, y_p, t) \in S$ **do**
            for $i = 1$ to $p - 1$ **do**
                $\mathcal{M} = \mathcal{M} - (x_{i+1}, y_i)$
            end for
            for $i = 1$ to p **do**
                $\mathcal{M} = \mathcal{M} + (x_i, y_i)$ ▷ the size of $\mathcal{M}$ has been increased by 1
            end for
        end for
    end if
end while

(iii) We use Algorithm 4 by Hopcroft and Karp for maximum bipartite matching [HK70]. In this algorithm we repeatedly increase the size of a partial matching $\mathcal{M}$ by finding augmenting paths in the graph. An augmenting path is a path with

two end edges not in $\mathcal{M}$ and alternates between edges of the graph that belong to $\mathcal{M}$ and edges that do not. Swapping these edges from being in $\mathcal{M}$ to not being in $\mathcal{M}$ would increase the size of matching by one. However, instead of finding just an augmenting path in each iteration of the algorithm, it finds a maximal set of shortest vertex disjoint augmenting paths. After only $O(\sqrt{n})$ iterations, the maximum matching would be found. Since all queries to the input are made inside calls to the BFS Algorithm, the G-coloring of the associated decision tree, is as for BFS algorithm. There are $O(\sqrt{n})$ calls to BFS algorithm (Line 2 in Algorithm 4 repeats $O(\sqrt{n})$ times), so we have $G = n\sqrt{n}$ and the depth of the decision tree is $T = m + n$, where those n extra queries are for the nils. Therefore, the quantum query complexity of this problem is $O\left(n^{3/4}\sqrt{(m+n)}\right)$.

(iv), (v) The algorithms are similar to those of Proposition 10 and the G-coloring is as above, so we skip the details. □

6 Concluding remarks

In this paper we generalized a result of [LL16] that is a method for designing quantum query algorithms based on classical ones. Our generalization of [LL16] is twofold: first, we assume that the input alphabet of the function may be non-binary; second, we assume that in a decision tree the outgoing edges connected to a vertex may be indexed by subsets in a partition of the input alphabet set. These two enabled the possibility of using this method, in particular, for graph properties in the adjacency list model. Our proof of this generalization is based on span programs in the non-binary case as well as the dual adversary bound.

Let us at this stage review different approaches we have in proving Lin and Lin's results in [LL16] as well as Theorems 4 and 5:

- The first idea in [LL16] is to use the notion of bomb query complexity, which we did not mention here. It is an interesting question that whether this idea can be extended to prove our generalized results (Theorems 4 and 5).

- The second idea in [LL16] is to use a modified version of Grover's search to find mistakes of the guessing algorithm. However, a naive application of Grover's search here results in an extra logarithmic factor for error reduction. It is shown in [LL16] that for functions with binary inputs this undesired factor can be eliminated using properties of the so called γ_2 norm. It seems plausible that the first part of Theorem 4 is provable by the same technique. However, it is not clear to us whether the second part of Theorem 4 or Theorem 5 are achievable taking the same path.

- The third idea is to use the notion of non-binary span program as we did for a proof of Theorem 2. The idea is to use a "st-connectivity type span program" (taken from [BR]) in order to reach from the root of a decision tree to some leaf. However, to not end up with the trivial upper bound of T (the depth of the decision tree) on the quantum query complexity, we equipped

edges of the decision tree with some weights that are chosen based on a G-coloring. Incorporating these weights in the span program the desired result was obtained.

- The last idea is to use the dual adversary bound. This approach is essentially the same as the approach of span programs, but with the advantage that it does not give an undesirable extra factor of $\sqrt{\ell-1}$ as explained before. Comparing to the first two methods, we believe that the ideas of using span programs and dual adversary bound are more advantageous since the choice of *weights* in these approaches is arbitrary. For proving Theorem 4 the weights that we chose were among two possible choices. We then in Theorem 5 showed how using a larger set of weights we may further improve the upper on the quantum query complexity. Thus, a possible direction to extend our results is to further investigate other possible choices for the weights.

One may suggest that our generalized non-binary version of the result of [LL16] can be obtained simply by representing non-binary inputs of $f : [\ell]^n \to [m]$ by binary strings, simulating a single non-binary query by $\log(\ell)$ binary ones, and then using the result of [LL16] in the binary case. Even ignoring the extra $\log(\ell)$ factor we obtain in this method, we argue that this approach does not work. First, in our notion of generalized decision tree we allow to *identify* some edges in the decision tree and label its edges with subsets of $[\ell]$. This is missing in the notion of decision tree in [LL16]. Identification of edges is a necessary part of our results especially in the examples of graph properties in the adjacency list model. To elaborate the second limitation of this approach, let us think of the example of minimum finding explained in Proposition 7. Suppose that $\ell = 8$ and at some stage of the algorithm our candidate for minimum is 6 that is equal to $(1, 1, 0)$ in the binary representation. Then we read the first bit of the next number in the list and find it to be equal to 1. This means that the next number in the list is one of the numbers $4, 5, 6$ or 7. In the algorithm and its associated G-coloring, there is a difference between 4, 5 and 6, 7 since the first two are smaller than 6. Indeed, in our proposed G-coloring edges 6, 7 are merged to a single edge with black color, while the edges 4 and 5 are colored in red. Therefore, to convert this coloring to a G-coloring in the binary decision tree whose edges are labeled by binary inputs, we have no choice but coloring the edge with label 1 by red. Then the parameter G of the new G-coloring not only scales by a factor of $\log \ell$, but also is increased by something like $T - G$ because of such extra red edges. In summary, in order to use the result of [LL16] in the binary case to prove our generalized result in the non-binary case, we need to convert a G-coloring of a generalized non-binary decision tree to a G-coloring of a binary decision tree. It is now clear how this can be done in general without drastically weakening our bound on the parameter G.

Our results give bounds on the space complexity of our algorithms as well. The point is that the space complexity of a quantum algorithm based dual adversary bound, is bounded by the logarithm of the dimension of the vectors in the feasible solution of the dual adversary SDP. In our proofs the dimension of such feasible solutions is of the order of the size of the decision tree. Thus the space complexity of our algorithms equals the logarithm of the size of the corresponding decision tree.

In particular, since in our examples (especially those for graph properties) the sizes of decision trees is exponential, the space complexity of our quantum algorithms is linear.[7]

Prior to our work designing a span program based quantum query algorithm for directed graphs was not an easy task. We eased the process of designing such algorithms by relating them to classical decision trees. Comparing to span programs for undirected graphs, however, the size of these span programs for directed graphs is exponential. It would be interesting to see if we can decrease the space complexity of such quantum algorithms to logarithmic size.

References

[AMRR11] Andris Ambainis, Loïck Magnin, Martin Roetteler, and Jérémie Roland. Symmetry-assisted adversaries for quantum state generation. In *2011 IEEE 26th Annual Conference on Computational Complexity*, pages 167–177. IEEE, 2011. arXiv:1012.2112, doi:10.1109/CCC.2011.24.

[Āri15] Agnis Āriņš. Span-program-based quantum algorithms for graph bipartiteness and connectivity. In *International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 35–41. Springer, 2015. arXiv:1510.07825v1, doi:10.1007/978-3-319-29817-7_4.

[BHT] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming*, pages 820–831, Berlin, Heidelberg. Springer Berlin Heidelberg. arXiv:9805082, doi:10.1007/BFb0055105.

[BR] Aleksandrs Belovs and Ben W Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Proceedings of the 20th Annual European conference on Algorithms (ESA 12)*, pages 193–204. Springer Berlin Heidelberg. arXiv:1203.2603, doi:10.1007/978-3-642-33090-2_18.

[BT19] Salman Beigi and Leila Taghavi. Span program for non-binary functions. *Quantum Information and Computation*, 19(9-10):760–792, 2019. arXiv:1805.02714, doi:10.26421/QIC19.9-10.

[Cir06] Jill Cirasella. Classical and Quantum Algorithms for Finding Cycles. Master's thesis, University of Amsterdam, 2006. URL: http://www.illc.uva.nl/Research/Publications/Reports/MoL-2006-06.text.pdf.

[7] Note that although the size of the decision tree can be exponential, as in the examples in this paper, we do not need to explicitly build it. We usually have a classical algorithm which directly gives a decision tree. To use our results, we then only need to give a G-coloring.

[CK] Andrew M. Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. *SIAM Journal On Computing*. arXiv:1011.1443, doi:10.4230/LIPIcs.STACS.2011.661.

[CMB] Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and Space Efficient Quantum Algorithms for Detecting Cycles and Testing Bipartiteness. oct. arXiv:1610.00581.

[DH] Christoph Dürr and Peter Høyer. A Quantum Algorithm for Finding the Minimum. arXiv:9607014.

[DHHM06] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. arXiv:0401091, doi:10.1137/050644719.

[Gro] Low K Grover. A fast quantum mechanical algorithm for database search. *Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 212–219. arXiv:9605043, doi:10.1145/237814.237866.

[HK70] John E Hopcroft and Richard M Karp. An $O(n^{2.5})$ algorithm for maximum matching in bipartite graphs. *SIAM Journal On Computing*, pages 122–125, 1970. doi:10.1137/0202019.

[HLŠ07] Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 526–535. ACM, 2007. arXiv:0611054, doi:10.1145/1250790.1250867.

[IJ15] Tsuyoshi Ito and Stacey Jeffery. Approximate Span Programs. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, 2015. URL: http://arxiv.org/abs/1507.00432, arXiv:1507.00432, doi:10.4230/LIPIcs.ICALP.2016.12.

[Kah62] Arthur B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5:558–562, 1962. doi:10.1145/368996.369025.

[LL16] Cedric Yen-Yu Lin and Han-Hsuan Lin. Upper bounds on quantum query complexity inspired by the Elitzur-Vaidman bomb tester. *Theory of Computing*, 12:1–35, 2016. URL: https://theoryofcomputing.org/articles/v012a018/, arXiv:1410.0932, doi:10.4086/toc.2016.v012a018.

[LMR+11] Troy Lee, Rajat Mittal, Ben W Reichardt, Robert Špalek, and Mario Szegedy. Quantum query complexity of state conversion. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 344–353. IEEE, 2011. arXiv:1011.3020, doi:10.1109/FOCS.2011.75.

[LMRŠ] Troy Lee, Rajat Mittal, Ben W Reichardt, and Robert Špalek. An adversary for algorithms. arXiv:1011.3020.

[Rei09] Ben W Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 544–551. IEEE, 2009. arXiv:0904.2759, doi:10.1109/FOCS.2009.55.

[RŠ12] Ben W Reichardt and Robert Špalek. Span-program-based quantum algorithm for evaluating formulas. volume 8, pages 291–319. Theory of Computing, 2012. URL: http://www.theoryofcomputing.org/articles/v008a013, arXiv:0710.2630, doi:10.4086/toc.2012.v008a013.

[Shi02] Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. In *Proceedings 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 513–519, 2002. arXiv:0112086, doi:10.1109/SFCS.2002.1181975.

[Tar76] Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–185, 1976. doi:10.1007/BF00268499.

[Zha05] Shengyu Zhang. On the power of Ambainis lower bounds. *Theoretical Computer Science*, 339(2-3):241–256, 2005. arXiv:0311060, doi:10.1016/j.tcs.2005.01.019.