Theory Comput Syst (2010) 47: 786–807
DOI 10.1007/s00224-009-9219-1

Quantum Search with Variable Times

Andris Ambainis

Published online: 3 June 2009

Abstract Since Grover's seminal work, quantum search has been studied in great detail. In the usual search problem, we have a collection of n items $x_1, \ldots, x_n$ and we would like to find $i : x_i = 1$. We consider a new variant of this problem in which evaluating x_i for different i may take a different number of time steps.

Let t_i be the number of time steps required to evaluate x_i. If the numbers t_i are known in advance, we give an algorithm that solves the problem in $O(\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2})$ steps. This is optimal, as we also show a matching lower bound. The case, when t_i are not known in advance, can be solved with a poly-logarithmic overhead. We also give an application of our new search algorithm to computing read-once functions.

Keywords Quantum search · Quantum algorithms

1 Introduction

Grover's quantum search algorithm [12] is one of two most important quantum algorithms. It allows to search a collection of n items in $O(\sqrt{n})$ quantum steps. This gives a quadratic speedup over the exhaustive search for a variety of search problems [2].

An implicit assumption is that any two items can be examined in the same number of time steps. This is not necessarily true when Grover's algorithm is applied to a specific search problem. It might be the case that some possible solutions to the search problem can be checked faster than others.

Supported by University of Latvia research project Y2-ZP01-100 and Marie Curie International Reintegration Grant (IRG). This work done at University of Waterloo, supported by NSERC, CIAR, MITACS, ARO and IQC University Professorship.

A. Ambainis (✉)
Faculty of Computing, University of Latvia, Raina bulv. 19, Riga 1586, Latvia
e-mail: andris.ambainis@lu.lv

Let t_i be the number of time steps required to check the i^{th} solution. Classically, searching for an item $i : x_i = 1$ requires time $\Theta(t_1 + \ldots + t_n)$. A naive application of Grover's search would use $O(\sqrt{n})$ steps, with the maximum possible query time $t_{max} = \max_i t_i$ in each step. This gives an $O(\sqrt{n}t_{max})$ time quantum algorithm.

In this paper, we give a better quantum algorithm. We consider two settings:

1. The times t_i are known in advance and can be used to design the algorithm;
2. The times t_i are not known in advance. The algorithm learns t_i only if it runs the computation for checking the i^{th} item for t_i (or more) steps.

For the first setting, we give a quantum algorithm that searches in time $O(\sqrt{T})$ where $T = t_1^2 + \ldots + t_n^2$. For the second, more general setting, we give an $O(\sqrt{T}\log^2 T \log^2 \log T)$ time quantum algorithm. We show a lower bound of $\Omega(\sqrt{T})$ for the first and, hence, also the second setting.

To illustrate the usefulness of our search algorithm, we show an application to computing read-once Boolean functions. A Boolean formula (consisting of AND, OR and NOT operations) $f(x_1, \ldots, x_N)$ is read-once if each of the variables $x_1, \ldots, x_N$ appears at most once in f. We show that any read-once Boolean formula of depth d can be computed using $O(\sqrt{N}\log^{d-1} N)$ queries. The resulting algorithm is weaker than the recent breakthrough work of [4, 11, 17] but is also much simpler than the algorithms in [4, 11, 17].

This is the first paper to construct quantum algorithms for a model in which queries to different x_i take different time. A similar model, however, has been studied in the context of quantum lower bounds by Høyer et al. [14].

2 Model

Our model is a generalization of the usual quantum query model. We model a situation when the variable x_i is computed by a query algorithm $\mathcal{A}_i$ which is initialized in the state $|0\rangle$ and, after t_i steps, outputs the final state $|x_i\rangle|\psi_i\rangle$ for some unknown $|\psi_i\rangle$. (For simplicity, we restrict ourselves to the case when $\mathcal{A}_i$ always outputs the correct x_i.) In the first $t_i - 1$ steps, $\mathcal{A}_i$ can be in arbitrary intermediate states.

Our goal is to find $i : x_i = 1$. (We sometimes refer to $i : x_i = 1$ as *marked items* and $i : x_i = 0$ as *unmarked*.) Our search algorithm $\mathcal{A}$ can run the query algorithms $\mathcal{A}_i$ for some number of steps t, with $\mathcal{A}_i$ outputting x_i if $t_i \le t$ or "the computation is not complete" if $t_i > t$. The computational cost is the amount of time that is spent running the query algorithms $\mathcal{A}_i$. Any transformation that does not involve $\mathcal{A}_i$ is free.

For completeness, we include a more formal definition of our model in the Appendix A. Our search algorithms, however, can be understood with just the informal description in the previous two paragraphs.

Known vs. Unknown Times We consider two variants of this model. In the "known times" model, the times $t_1, \ldots, t_n$ are known in advance and can be used to design the search algorithm. In the "unknown times" model, $t_1, \ldots, t_n$ are unknown to the designer of the search algorithm.

Terminology We use the following terms:

- *Query algorithm:* the algorithm $\mathcal{A}_i$ that computes x_i.
- *Search algorithm:* the algorithm that searches for $i : x_i = 1$, using calls to query algorithms.

3 Methods and Subroutines

We use the well-known methods of *amplitude amplification* and *amplitude estimation*.

3.1 Amplitude Amplification

Amplitude amplification [7] is a generalization of Grover's quantum search algorithm. Let

$$\sin\alpha|1\rangle|\psi_1\rangle + \cos\alpha|0\rangle|\psi_0\rangle \tag{1}$$

be the final state of a quantum algorithm $\mathcal{A}$ that outputs 1 with probability $\sin^2\alpha = \delta$. We would like to increase the probability of the algorithm outputting 1. Brassard et al. [7] showed that, by repeating $\mathcal{A}$ and $\mathcal{A}^{-1}$ $2m+1$ times, it is possible to generate the final state

$$\sin(2m+1)\alpha|1\rangle|\psi_1\rangle + \cos(2m+1)\alpha|0\rangle|\psi_0\rangle. \tag{2}$$

In particular, taking $m = O(\frac{1}{\sqrt{\delta}})$ achieves a constant probability of answer 1.

We use a result by Aaronson and Ambainis [1] who gave a tighter analysis of the same algorithm:

Lemma 1 [1] *Let $\mathcal{A}$ be a quantum algorithm that outputs a correct answer and a witness with probability*[1] *$\delta \le \epsilon$ where ϵ is known. Furthermore, let*

$$m \le \frac{\pi}{4\arcsin\sqrt{\epsilon}} - \frac{1}{2}. \tag{3}$$

Then, there is an algorithm $\mathcal{A}'$ which uses $2m+1$ calls to $\mathcal{A}$ and $\mathcal{A}^{-1}$ and outputs a correct answer and a witness with probability

$$\delta_{new} \ge \left(1 - \frac{(2m+1)^2}{3}\delta\right)(2m+1)^2\delta. \tag{4}$$

The distinction between this lemma and the standard amplitude amplification is as follows. The standard amplitude amplification increases the probability from δ to $\Omega(1)$ in $2m+1 = O(\frac{1}{\sqrt{\delta}})$ repetitions. In other words, $2m+1$ repetitions increase the success probability $\Omega((2m+1)^2)$ times. Lemma 1 achieves an increase of almost

[1] Result in [1] requires the probability to be exactly ϵ but the proof works without changes if the probability is less than the given ϵ.

$(2m+1)^2$ times, without the big-Ω factor. This is useful if we have an algorithm with k levels of amplitude amplification nested one inside another. Then, with the usual amplitude amplification, a big-Ω constant of c would result in a c^k factor in the running time. Using Lemma 1 avoids that.

We also need another fact about amplitude amplification.

Claim 1 *Let δ and δ' be such that $\delta \leq \epsilon$ and $\delta' \leq \epsilon$ and let m satisfy the constraint (3). Let $p(\delta)$ be the success probability obtained by applying the procedure of Lemma 1 to an algorithm with success probability δ. If $\delta' \leq \delta \leq c\delta'$ for $c \geq 1$, then*

$$p(\delta') \leq p(\delta) \leq cp(\delta').$$

Proof In Appendix B. □

3.2 Amplitude Estimation

The second result that we use is a version of quantum amplitude estimation.

Theorem 1 [7] *There is a procedure* **Est-Amp**$(\mathcal{A}, M)$ *which, given a quantum algorithm $\mathcal{A}$ and a number M, outputs an estimate $\tilde{\epsilon}$ of the probability ϵ that $\mathcal{A}$ outputs 1 and, with probability at least $\frac{8}{\pi^2}$, we have*

$$|\epsilon - \tilde{\epsilon}| \leq 2\pi \frac{\sqrt{\min(\epsilon(1-\epsilon), \tilde{\epsilon}(1-\tilde{\epsilon}))}}{M} + \frac{\pi^2}{M^2}.$$

The algorithm uses M evaluations of $\mathcal{A}$.

We are interested in a slightly different type of error bound. We would like to have $|\epsilon - \tilde{\epsilon}| \leq c\tilde{\epsilon}$ for some small constant $c > 0$.

Theorem 2 *There is a procedure* **Estimate**$(\mathcal{A}, c, p, k)$ *which, given a constant c, $0 < c \leq 1$ and a quantum algorithm $\mathcal{A}$ (with the promise that the probability ϵ that the algorithm $\mathcal{A}$ outputs 1 is either 0 or at least a given value p) outputs an estimate $\tilde{\epsilon}$ of the probability ϵ such that, with probability at least $1 - \frac{1}{2^k}$, we have*

(i) $|\epsilon - \tilde{\epsilon}| < c\tilde{\epsilon}$ *if* $\epsilon \geq p$;
(ii) $\tilde{\epsilon} = 0$ *if* $\epsilon = 0$.

The procedure **Estimate**$(\mathcal{A}, c, p, k)$ *uses the expected number of*

$$\Theta\left(k\left(1 + \log\log\frac{1}{p}\right)\sqrt{\frac{1}{\max(\epsilon, p)}}\right)$$

evaluations of $\mathcal{A}$.

Proof In Appendix B. □

4 Search Algorithm: Known Running Times

For this case, we have the following result, with a matching lower bound (Theorem 8 in Sect. 7).

Theorem 3 *A collection of n items with times $t_1, \ldots, t_n$ can be searched in time*

$$O\left(\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}\right).$$

4.1 Slightly Weaker Result, with a Simpler Proof

We first show a weaker version of Theorem 3.

Theorem 4 *A collection of n items with times $t_1, \ldots, t_n$ can be searched in time*

$$O\left(c^{\log^* n}\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}\right)$$

for some constant c.

Proof Let $t_{max} = \max_i t_i$. First, our algorithm sequentially queries all items i with small t_i (that is, $t_i \le \frac{t_{max}}{n}$). The total query time for that is at most

$$n\frac{t_{max}}{n} = t_{max} \le \sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}.$$

Then, the algorithm subdivides the remaining items ($t_i > \frac{t}{n}$) into groups so that all items in one group have similar times t_i (e.g. $\frac{t_{max}}{2} \le t_i \le t_{max}$ for some t_{max}). We can perform the standard Grover search in a group in time $s = O(\sqrt{l}t_{max})$ where l is the size of the group. We then observe that

$$s^2 = O(lt_{max}^2) = O\left(\sum_i t_i^2\right),$$

with the summation over all items i in the same group. By summing over all groups, we get

$$\sum_j s_j^2 = O\left(\sum_{i=1}^n t_i^2\right),$$

where j on the left ranges over all groups.

Let k be the number of the groups that we have. If we have a search algorithm that searches k items in time

$$O\left(\sqrt{s_1^2 + \ldots + s_k^2}\right),$$

we can then substitute the search algorithms for searching the k groups instead of the query algorithms for k items and obtain a search algorithm for n items that runs in time

$$O\left(\sqrt{t_1^2 + \ldots + t_n^2}\right).$$

Since we have $\frac{t_{max}}{n} \leq t_i \leq t_{max}$ for the items that were not queried at the beginning of the algorithm, $k \leq \log n + 1$. Thus, we have reduced search on n items to search on $k \leq \log n + 1$ items. We then apply the same strategy recursively to reduce search on k items to search on $\log k + 1$ items and so on, until the number of items becomes less than some constant C. Then, we query all items sequentially.

Reducing the number of items to less than C takes $\log^* n$ levels of recursion. The total running time is

$$O\left(c^{\log^* n}\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}\right),$$

because each reduction from n items to $k = \log n + 1$ items increases the big-O constant by a constant factor. □

The $c^{\log^* n}$ factor can be avoided, by a more sophisticated implementation of the same idea, which we describe in the next subsections.

4.2 General Case: Preliminaries and Overview

We first restrict to the case when there is exactly one marked item. The general case can be reduced to this case with a constant factor overhead, by running the algorithm on all n elements, a random set of $\frac{n}{2}$, a random set of $\frac{n}{4}$, etc. As shown in [1], there is a constant probability that at least one of those sets contains exactly one marked item. The expected running time increases by at most a constant factor, because of the following lemma.

Lemma 2 *Let S be a uniformly random set of $\frac{n}{2^j}$ elements of $\{1, 2, \ldots, n\}$. Then,*

$$E\left[\sqrt{\sum_{i\in S} t_i^2}\right] \leq \frac{1}{2^{j/2}}\sqrt{\sum_{i\in\{1,\ldots,n\}} t_i^2}.$$

Proof By concavity of the square root function,

$$E\left[\sqrt{\sum_{i\in S} t_i^2}\right] \leq \sqrt{E\left[\sum_{i\in S} t_i^2\right]} = \frac{1}{2^{j/2}}\sqrt{\sum_{i\in\{1,\ldots,n\}} t_i^2}.$$ □

Therefore, the reduction from the general case to one marked item case increases the bound on the query time by a factor of at most

$$1 + \frac{1}{2^{1/2}} + \frac{1}{2} + \ldots < \frac{1}{1 - \frac{1}{\sqrt{2}}}.$$

Second, we introduce a generalized search problem in which the algorithm $\mathcal{A}_i$ for the marked i returns the correct answer with a probability at least p_i, instead of a certainty. More formally,

- if $x_i = 0$, the final state of $\mathcal{A}_i$ is of the form $|0\rangle|\psi_0\rangle$.
- if $x_i = 1$, the final state of $\mathcal{A}_i$ is of the form $\alpha|1\rangle|\psi_1\rangle + \sqrt{1-\alpha^2}|0\rangle|\psi_0\rangle$, where $p_i \le |\alpha|^2 \le d \cdot p_i$, for some constant $d > 1$.

The probabilities $p_1, \ldots, p_n$ and the constant d are known to us when we design the algorithm, just as the times $t_1, \ldots, t_n$. (Knowing both the success probability and the running time may look quite artificial. However, we only use the "known success probability" model to design an algorithm for the case when all $\mathcal{A}_i$ return the correct answer with certainty.) We prove that

Theorem 5 *The generalized search problem can be solved using $O(\sqrt{T})$ query steps where*

$$T = \frac{t_1^2}{p_1} + \frac{t_2^2}{p_2} + \ldots + \frac{t_n^2}{p_n}. \tag{5}$$

Our main theorem now follows as the particular case $p_1 = \ldots = p_n = 1$.

4.3 Proof of Theorem 5

The algorithm for the generalized search problem is described as Algorithm 1. We now prove that this algorithm achieves the bound of Theorem 5. To improve the readability, proofs of more technical lemmas are postponed to the next subsection.

We start with lemma bounding the amplification step.

Lemma 3 *If $\mathcal{A}'_i$ is obtained from $\mathcal{A}_i$ by amplification in step* 1 *of the pre-processing part of Algorithm* 1, *then*:

(i) *The success probability of $\mathcal{A}'_i$ is between p'_i and $d \cdot p'_i$, where p'_i satisfies* $(1 - \frac{1}{3\log n})\frac{1}{9\log n} \le p'_i$.

(ii)

$$\frac{(t'_i)^2}{p'_i} \le \left(1 + O\left(\frac{1}{\log n}\right)\right)\frac{(t_i)^2}{p_i}.$$

Proof Postponed to Sect. 4.4. □

This lemma essentially means that we can use $\frac{(t'_i)^2}{p'_i}$ instead of $\frac{(t_i)^2}{p_i}$ in expressions such as (5).

Next, we bound the time to run the quick algorithms. To achieve the success probability of $\Omega(1)$, we need to amplify $\mathcal{A}'_i$ with $O(1/\sqrt{p'_i})$ repetitions. Therefore, the

Algorithm 1 Search algorithm for known query times

Input: Query algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$, running times $t_1, \ldots, t_n$ and estimates $p_1, \ldots, p_n, d$.

Pre-processing:

1. Define $\mathcal{A}'_i = \mathcal{A}_i$ if $p_i \geq \frac{1}{9 \log n}$. Otherwise, let m be the smallest number for which $(2m+1)^2 p_i \geq \frac{1}{9 \log n}$. Define $\mathcal{A}'_i$ as the algorithm obtained from $\mathcal{A}_i$ by amplitude amplification (Lemma 1) with $2m+1$ calls to $\mathcal{A}_i$ and $(\mathcal{A}_i)^{-1}$.
2. Let $t'_i = (2m+1)t_i$ be the number of query steps of $\mathcal{A}'_i$ and let p'_i be the success probability that $\mathcal{A}'_i$ would have if the success probability of $\mathcal{A}_i$ is exactly p_i.
3. Let T_0 be the maximum of $\frac{t'_i}{\sqrt{p'_i}}$. We call $\mathcal{A}'_i$ *quick* if $\frac{t'_i}{\sqrt{p'_i}} \leq \frac{T_0}{n \log n}$ and *slow* otherwise.
4. Let T_{min} and p_{min} be the minimums of t'_i and p'_i among all slow algorithms. Partition the intervals $[T_{min}, T_0]$ and $[p_{min}, 1]$ into subintervals of the form $[T', T'']$ and $[P', P'']$ with $T'' \leq (1 + \frac{1}{\log n})T'$ and $P'' \leq (1 + \frac{1}{\log n})P'$.
5. Let $S_{T',T'',P',P''}$ be the set of all i with $T' < t'_i \leq T''$ and $P' < p'_i \leq P''$. Define $\mathcal{B}_{T',T'',P',P''}$ as the algorithm that randomly chooses $i \in S_{T',T'',P',P''}$ and then runs $\mathcal{A}'_i$.

Search algorithm:

1. Search among the quick $\mathcal{A}'_i$ by running each of them, amplified to the success probability $\Omega(1)$.
2. Search among the slow $\mathcal{A}'_i$ in one of the following ways:
   a) If the number of algorithms $\mathcal{B}_{T',T'',P',P''}$ is less than a fixed constant C, run each of them, amplified to the success probability $\Omega(1)$.
   b) Otherwise, call the algorithm recursively with $\mathcal{B}_{T',T'',P',P''}$ as the query algorithms $\mathcal{A}_i$. Use T' as t_i, $\frac{P'}{|S_{T',T'',P',P''}|}$ as p_i and $d(1 + \frac{1}{\log n})$ as d.

number of query steps for checking one such $\mathcal{A}'_i$ is

$$O\left(\frac{t'_i}{\sqrt{p'_i}}\right) = O\left(\frac{T_0}{n \log n}\right).$$

The number of query steps for checking all such $\mathcal{A}'_i$ is at most the number of such $(\mathcal{A}'_i)$'s times $O(\frac{T_0}{n \log n})$ which is of the order at most

$$\frac{T_0}{\log n} \leq \frac{1}{\log n}\sqrt{T_0^2} \leq \frac{1}{\log n}\sqrt{\frac{(t'_1)^2}{p'_1} + \ldots + \frac{(t'_n)^2}{p'_n}}.$$

For slow algorithms, we have

Lemma 4 *There are $k = O(\log^3 n \log\log n)$ pairs of intervals $([P', P''], [T', T''])$.*

Proof Postponed to Sect. 4.4. □

Thus, recursively calling Algorithm reduces the number of items to be searched from n to $k = O(\log^3 n \log\log n)$. After $O(\log^* n)$ levels of recursion, the number of items is reduced to a constant. Next, we bound the effect of the recursion on the running time.

We enumerate the pairs of intervals $([T', T''], [P', P''])$ by numbers $1, \ldots, k$. Let $[T'_j, T''_j]$ and $[P'_j, P''_j]$ be the time and probability intervals from the j^{th} pair and let $B_j = B_{T'_j, T''_j, P'_j, P''_j}$ and $S_j = S_{T'_j, T''_j, P'_j, P''_j}$.

Let $s_j = \max_{i \in S_j} t'_i$ be the number of query steps performed by $\mathcal{B}_j$. Then, $s_j \le T''_j$. If one of $\mathcal{A}'_i$, $i \in S_j$ outputs 1, the success probability of $\mathcal{B}_j$ is $\frac{1}{|S_j|}$ (the probability of choosing the right $i \in S_j$) times the success probability of $\mathcal{A}'_i$. The success probability of $\mathcal{A}'_i$ is in the interval $[p'_i, dp'_i]$ and we have $p'_i \in [P'_j, P''_j] \subseteq [P'_j, (1 + \frac{1}{\log n}) P'_j]$.

Therefore, the success probability of $\mathcal{B}_j$ is in the interval $[q_j, d(1 + \frac{1}{\log n}) q_j]$ where $q_j = \frac{P'_j}{|S_j|}$. This means that the values of p_j and d in step 2b of Algorithm 1 are correct.

We now relate s_j and q_j to t'_j and p'_j:

$$\frac{(s_j)^2}{q_j} \le |S_j| \frac{(T''_j)^2}{P'_j} \le \left(1 + O\left(\frac{1}{\log n}\right)\right) |S_j| \frac{(T'_j)^2}{P'_j}$$
$$\le \left(1 + O\left(\frac{1}{\log n}\right)\right) \sum_{i \in S_j} \frac{(t'_i)^2}{p'_i}.$$

By summing over all j,

$$\frac{s_1^2}{q_1} + \ldots + \frac{s_k^2}{q_k} \le \left(1 + O\left(\frac{1}{\log n}\right)\right) \left(\frac{(t'_1)^2}{p'_1} + \ldots + \frac{(t'_n)^2}{p'_n}\right).$$

Thus, one level of recursion increase the sum (5) by a factor of $1 + O(\frac{1}{\log n})$ and $O(\log^* n)$ levels of recursion increase it by a factor of $(1 + O(\frac{1}{\log n}))^{\log^* n} = 1 + o(1)$.

Let $s_1, \ldots, s_k$ and $q_1, \ldots, q_k$ be the times and success probabilities for the final $k \le C$ algorithms. In step 1, we amplify the success probability of each of them to $\Omega(1)$. This gives us query algorithms with running times $s'_i = O(\frac{s_i}{\sqrt{q_i}})$ and success probabilities $p'_i = \Omega(1)$. We then search them sequentially, in time

$$C \max s'_i = O(\max s'_i) = O\left(\sqrt{(s'_1)^2 + \ldots + (s'_n)^2}\right) = O\left(\sqrt{\frac{s_1^2}{q_1} + \ldots + \frac{s_n^2}{q_n}}\right).$$

4.4 Proofs of Lemmas

Proof of Lemma 3 (i) Let p'_i be the success probability that amplitude amplification gives for $\mathcal{A}'_i$ if the success probability of $\mathcal{A}_i$ is exactly p_i. By Claim 1, if the success

probability of $\mathcal{A}_i$ is in the interval $[p_i, d \cdot p_i]$, then the success probability of $\mathcal{A}'_i$ is in the interval $[p'_i, d \cdot p'_i]$.

It remains to lower-bound p'_i. We have

$$(2m+1)^2 p_i \leq \frac{(2m+1)^2}{(2m-1)^2}(2m-1)^2 p_i < 9\frac{1}{9\log n} = \frac{1}{\log n}, \tag{6}$$

with the last inequality following from $m \geq 2$ and $(2m-1)^2 p_i < \frac{1}{9\log n}$ (which follows from the definition of m). Therefore, by Lemma 1,

$$p'_i \geq \left(1 - \frac{(2m+1)^2}{3} p_i\right)(2m+1)^2 p_i \geq \left(1 - \frac{1}{3\log n}\right)\frac{1}{9\log n}.$$

(ii) As noted above, we have

$$\frac{p'_i}{p_i} \geq \left(1 - \frac{1}{3\log n}\right)(2m+1)^2.$$

Together with $t'_i = (2m+1)t_i$, this implies part (ii). □

Proof of Lemma 4 By Lemma 3, we have $(1 - \frac{1}{3\log n})\frac{1}{9\log n} \leq p'_i$. We also have $p'_i \leq 1$. The interval $[(1 - \frac{1}{3\log n})\frac{1}{9\log n}, 1]$ can be partitioned into $O(\log n \log\log n)$ intervals $[P', P'']$ with $P'' \leq (1 + \frac{1}{\log n})P'$.

Running times t'_i for the amplified algorithms are bounded from above by $T_0\sqrt{p'_i} \leq T_0$ (because $T_0 = \max_i \frac{t'_i}{\sqrt{p'_i}}$). From below, we have

$$t'_i \geq \frac{T_0}{n\log n}\sqrt{p'_i} \geq (1 - o(1))\frac{T_0}{n\log^{3/2} n}.$$

The interval $[(1 - o(1))\frac{T_0}{n\log^{3/2} n}, T_0]$ can be partitioned into $O(\log^2 n)$ intervals $[T', T'']$ with $T'' \leq (1 + \frac{1}{\log n})T'$.

Together, we have $O(\log^3 n \log\log n)$ pairs of intervals $([P', P''], [T', T''])$. □

5 Application: Read-once Functions

A Boolean function $f(x_1, \ldots, x_N)$ that depends on all variables $x_1, \ldots, x_N$ is read-once if it has a Boolean formula (consisting of ANDs, ORs and NOTs) in which every variable appears exactly once. A read-once function can be represented by a tree in which every leaf contains x_i or NOT x_i and every internal vertex contains AND or OR.

Barnum and Saks [5] have shown that, for any read-once f, $\Omega(\sqrt{N})$ queries are necessary to compute f in the quantum query model. Høyer, Mosca and de Wolf [13] have constructed an $O(\sqrt{N})$ query quantum algorithm for balanced AND-OR trees of constant depth (improving over an earlier $O(\sqrt{N}\log^{d-1} N)$ query algorithm

by [9]). In a very recent breakthrough work, [4, 11] showed how to evaluate any AND-OR tree of depth d in $O(\sqrt{Nd})$ queries.

A simple application of our result from the previous section gives a quantum algorithm for evaluating depth-d AND-OR trees. The algorithm is weaker than the one in [4, 11] but is also much simpler.

Theorem 6 *Any read-once function $f(x_1, \ldots, x_N)$ of depth d can be computed by a quantum algorithm that uses $O(\sqrt{N}\log^{d-1} N)$ queries.*

Proof By induction. The base case, $d = 1$ is just the OR (or AND) function which can be computed with $O(\sqrt{N})$ queries using Grover's search to search for $i : x_i = 1$ (or $i : x_i = 0$).

For the inductive case, assume that f is represented by a depth-d tree with OR at the root. (The case when the root contains AND is similar.) Let n be the number of vertices on the level 1 (that is, the number of children of the root vertex) and t_i be the number of vertices in the subtree rooted in the i^{th} level-1 vertex. By re-ordering the variables, we can assume that

$$f(x_1, \ldots, x_N) = \bigvee_{i=1}^{n} f_i(x_{t_1+\ldots+t_{i-1}+1}, \ldots, x_{t_1+\ldots+t_i}).$$

To compute f, we have to determine if there exists $i \in \{1, \ldots, n\}$ for which $f_i = 1$. By the inductive assumption, there is an algorithm that computes f_i using $O(\sqrt{t_i}\log^{d-2} t_i) = O(\sqrt{t_i}\log^{d-2} N)$ queries. We repeat this algorithm $O(\log N)$ times to increase the probability of correct answer to at least $1 - \frac{1}{N^2}$. Let $\mathcal{A}_i$ be the resulting algorithm and $T_i = O(\sqrt{t_i}\log^{d-1} N)$ be the number of queries in $\mathcal{A}_i$.

By Theorem 3, a collection of n items with query times $T_1, \ldots, T_n$ can be searched using

$$\begin{aligned} O\left(\sqrt{T_1^2 + T_2^2 + \ldots + T_n^2}\right) &= O(\log^{d-1} N)\sqrt{t_1 + t_2 + \ldots + t_n} \\ &= O(\sqrt{N}\log^{d-1} N) \end{aligned}$$

query steps. Substituting $\mathcal{A}_1, \ldots, \mathcal{A}_n$ instead of the query algorithms gives an $O(\sqrt{N}\log^{d-1} N)$ algorithm for f.

A minor issue is that Theorem 3 assumes that query algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$ always output the correct answer, while our $\mathcal{A}_1, \ldots, \mathcal{A}_n$ may be incorrect with a small probability.

To resolve this issue, let $\mathcal{A}'_1, \ldots, \mathcal{A}'_n$ be the "ideal versions" of $\mathcal{A}_1, \ldots, \mathcal{A}_n$. If the final state of $\mathcal{A}_i$ is

$$\alpha|a\rangle|\psi_a\rangle + \sqrt{1-\alpha^2}|1-a\rangle|\psi_{1-a}\rangle, \tag{7}$$

where a is the correct answer (the value of f_i), then $\mathcal{A}'_i$ is a unitary mapping $|0\rangle$ to $|a\rangle|\psi_a\rangle$.

While $\mathcal{A}_i'$'s may not be easy to construct, they conform to the definition of a query algorithm in Appendix A. Therefore, if $\mathcal{A}_i'$ were used as query algorithms, the algorithm of Theorem 3 would output the correct answer with a constant probability (e.g., at least 2/3).

Since each $\mathcal{A}_i$ outputs the correct answer with probability at least $1 - \frac{1}{N^2}$, replacing $\mathcal{A}_i$ by $\mathcal{A}_i'$ in one time step changes the state of the search algorithm by at most $O(\frac{1}{N})$ (in the l_2 norm). Replacing $\mathcal{A}_i$ by $\mathcal{A}_i'$ in every time step changes the state by at most

$$O\left(\frac{\sqrt{N}\log^{d-1} N}{N}\right) = O\left(\frac{\log^{d-1} N}{\sqrt{N}}\right)$$

in l_2 norm. Therefore, the success probability of our search algorithm that uses the actual $\mathcal{A}_1, \ldots, \mathcal{A}_n$ as query algorithms will still be $\frac{2}{3} - o(1)$. □

Since this paper appeared as a preprint, a faster algorithm has been found [4], using quantum-walk based breakthrough techniques of Farhi et al. [11].

The construction of Theorem 6 is however, substantially simpler than the quantum walk approach of [4]. Therefore, we have included it, to demonstrate a possible application of our variable-time search algorithm.

6 Search Algorithm: Unknown Running Times

6.1 Overview

In some applications, it may be the case that the times t_i are not known in advance. We can also solve this case, with a polylogarithmic overhead.

Theorem 7 *Let $\epsilon > 0$. There is an algorithm that searches collection of n items with unknown times $t_1, \ldots, t_n$ and, with probability at least $1 - \epsilon$, stops after*

$$O\left(T \log^2 T \log^2 \log T\right)$$

steps, where $T = \sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}$.

Proof Again, we assume that there is exactly one marked item. (The reduction from the general case to the one marked item case is similar to one in the proof of Theorem 3.)

Let S_t be the set of items such that $x_i = 1$ or $t_i \geq 2^t$ and let $n_t = |S_t|$. Our main procedure, Algorithm 2, defines a sequence of algorithms $\mathcal{B}_1, \ldots, \mathcal{B}_l$. The algorithm $\mathcal{B}_j$, with some success probability, outputs a bit 1 and, conditional on output bit 1, it also outputs a uniformly random index $i \in S_j$.

The algorithm $\mathcal{B}_j$ is defined as follows. We first define $\mathcal{B}_j'$ as the algorithm that runs $\mathcal{B}_{j-1}$ and, if the output bit of $\mathcal{B}_{j-1}$ is 1, takes the index i output by $\mathcal{B}_{j-1}$ and tests if $i \in S_j$ by running the query to i for 2^j steps. $\mathcal{B}_j$ is just the algorithm $\mathcal{B}_j'$ amplified to

Algorithm 2 Search algorithm for unknown $t_1, \ldots, t_n$

1. Set $j = 0$. Define $\mathcal{B}_0$ as the algorithm that just outputs 1 and a uniformly random $i \in \{1, \ldots, n\}$.
2. Repeat:
   (a) Use the algorithm $\mathcal{B}_j$ (amplified to a success probability of $1 - o(1)$) to generate a sample of a uniformly random element $i \in S_j$. Run 2^{j+1} steps of the query procedure on i. If $x_i = 1$, output i and stop.
   (b) Define a new search algorithm $\mathcal{B}'_{j+1}$, as follows. $\mathcal{B}'_{j+1}$ runs $\mathcal{B}_j$ once and, if the output bit is 1, takes the output index i and runs 2^{j+1} steps of the checking procedure on i. If the result is $x_i = 0$, $\mathcal{B}'_{j+1}$ outputs 0. Otherwise, it outputs 1 and the same index i.
   (c) Let $p = \textbf{Estimate}(\mathcal{B}'_{j+1}, c, \frac{1}{n}, 2\log(D(j+1)))$. If $p = 0$, output "no $i : x_i = 0$".
   (d) If $p \geq \frac{1}{9\log n}$, let $\mathcal{B}_{j+1}$ be $\mathcal{B}'_{j+1}$.
   (e) If $p < \frac{1}{9\log n}$, let $\mathcal{B}_{j+1}$ be the algorithm obtained by amplifying $\mathcal{B}'_{j+1}$ $2m+1$ times, where m is the smallest number for which $\frac{1}{9\log n} \leq (2m+1)^2 p$.
   (f) Let $j = j + 1$.

a success probability of $\Omega(\frac{1}{\log n})$. (We avoid amplification to a success probability of $\Omega(1)$ because amplitude amplification to $\Omega(1)$ probability is less efficient and would result in a worse overall running time.)

The full algorithm is given as Algorithm 2.

Lemma 5 *Assume that the constant D in steps (a) and (c) satisfies $D \geq \frac{\pi}{\sqrt{3\epsilon}}$. Then, with probability $1 - \epsilon$, the estimates p are accurate within an multiplicative factor of $(1 + c)$.*

Proof The probability of error for **Estimate** is at most $\frac{1}{D^2(j+1)^2}$. By summing over all j, the probability of error for some j is at most

$$\frac{1}{D^2} \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{1}{D^2} \frac{\pi^2}{6},$$

which can be made less than $\frac{\epsilon}{2}$ by choosing $D \geq \frac{\pi}{\sqrt{3\epsilon}}$. □

We now bound the running time of Algorithm 2, assuming that the estimates p are correct. The first step is to bound the running time of the algorithms $\mathcal{B}_j$.

Lemma 6 *The running time of $\mathcal{B}_j$ is*

$$O\left(j\sqrt{\log n}\sqrt{\frac{t_1^2 + t_2^2 + \ldots + t_n^2}{n_j}}\right).$$

Proof Postponed until Sect. 6.2. □

We now bound the overall running time. To generate a sample from S_j, one needs $O(\sqrt{\log n})$ invocations of $\mathcal{B}_j$ (because the success probability of $\mathcal{B}_j$ is of the order $\Omega(\frac{1}{\log n})$). Therefore, we need $O(\sqrt{\log n}\log j)$ invocations to generate $O(\log j)$ samples in step 2a. By Lemma 6, that can be done in time

$$O\left(j\log j\log n\sqrt{\frac{t_1^2+t_2^2+\ldots+t_n^2}{n_j}}\right).$$

For each of those samples, we run the checking procedure with 2^{j+1} steps. That takes at most twice the time required by $\mathcal{B}_j$ (because $\mathcal{B}_j$ includes the checking procedure with 2^j steps). Therefore, the time for the 2^{j+1} checking procedure is of the same order or less than the time to generate the samples.

Second, the success probability estimated in the last step is of order $\frac{p_j n_{j+1}}{n_j} = \Omega(\frac{n_{j+1}}{n_j \log n})$. By Theorem 2, it can be estimated with

$$O\left(\log j\log\log n\sqrt{\frac{n_j\log n}{n_{j+1}}}\right)$$

invocations of $\mathcal{B}_j$, each of which runs in time described by Lemma 6.

Thus, the overall number of steps in one loop of Algorithm 2 is of order at most

$$\sqrt{t_1^2+t_2^2+\ldots+t_n^2}\left(\frac{j\log j\log n}{\sqrt{n_j}}+\frac{j\log j\log n\log\log n}{\sqrt{n_{j+1}}}\right).$$

Since $n_j \geq 1$ and $n_{j+1} \geq 1$, this is of order

$$O\left(\sqrt{t_1^2+t_2^2+\ldots+t_n^2}\,j\log j\log n\log\log n\right).$$

Let t_{max} be the maximum of $t_1,\ldots,t_n$. Then, the maximum value of j is at most $\lceil\log(t_{max}+1)\rceil$. Therefore, the number of steps used by the Algorithm 2 is

$$O\left(\sqrt{t_1^2+t_2^2+\ldots+t_n^2}\log n\log\log n\log t_{max}\log\log t_{max}\right).$$

The theorem now follows from $n \leq T$ and $t_{max} \leq \sqrt{T}$, where $T = t_1^2+t_2^2+\ldots+t_n^2$. □

6.2 Proof of Lemma 6

Let p_j be the success probability of $\mathcal{B}_j$ and p'_j be the success probability of $\mathcal{B}'_j$. Let $r_{k,l}$ be the number of times step (e) is performed, for $j \in \{k, k+1,\ldots,l-1\}$. (When $k = l$, we define $r_{k,k} = 0$.)

Lemma 7 *The number of query steps used by $\mathcal{B}_j$ is at most*

$$\sum_{j'=1}^{j} \left(1 + \frac{C}{\log n}\right)^{r_{j'-1,j}} \sqrt{\frac{p_j n_{j'-1}}{p_{j'-1} n_j}} 2^{j'} \tag{8}$$

for some constant C.

The intuition behind this expression is as follows. The terms of the sum (8) describe the number of query steps coming from the checking procedures of $\mathcal{B}'_{j'}$, for $j' = 1, 2, \ldots, j$. $2^{j'}$ is the number of steps used by the checking procedure of $\mathcal{B}'_{j'}$. The coefficient in the front of $2^{j'}$ is an upper bound on the number of times that $\mathcal{B}'_{j'}$ is repeated during the algorithm $\mathcal{B}_j$.

Proof By induction on j.

Base case: $j = 0$. $\mathcal{B}_0$ uses 0 query steps.

Inductive case: We first consider the running time of $\mathcal{B}'_{j+1}$. It can be decomposed into two parts: the running time of $\mathcal{B}_j$ and the running time of the 2^{j+1}-step checking procedure. The running time of $\mathcal{B}_j$ is described by (8). We have $p'_{j+1} = \frac{p_j n_{j+1}}{n_j}$. Therefore, we can rewrite (8) as

$$\sum_{j'=1}^{j} \left(1 + \frac{C}{\log n}\right)^{r_{j'-1,j}} \sqrt{\frac{p'_{j+1} n_{j'-1}}{p_{j'-1} n_{j+1}}} 2^{j'}.$$

The time for the checking procedure is just 2^{j+1} which is equal to $\frac{p'_{j+1} n_j}{p_j n_{j+1}} 2^{j+1}$ (since $\frac{p'_{j+1} n_j}{p_j n_{j+1}} = 1$). Therefore, the running time of $\mathcal{B}'_{j+1}$ is

$$\sum_{j'=1}^{j+1} \left(1 + \frac{C}{\log n}\right)^{r_{j'-1,j}} \sqrt{\frac{p'_{j+1} n_{j'-1}}{p_{j'-1} n_{j+1}}} 2^{j'} \tag{9}$$

where we have used $r_{j,j} = 0$ to include the checking time as the $j' = j + 1$ term in the sum. If step (d) is performed, then $\mathcal{B}_{j+1} = \mathcal{B}'_{j+1}$, $p_{j+1} = p'_{j+1}$, $r_{j',j} = r_{j',j+1}$ and the expression (9) is the same as (8) with $j + 1$ instead of j.

If the step 2e is performed, the running time of $\mathcal{B}_{j+1}$ is $(2m + 1)$ times the running time of $\mathcal{B}'_{j+1}$. The success probability is

$$p_{j+1} \geq \left(1 - \frac{(2m+1)^2}{3} p'_{j+1}\right) (2m+1)^2 p'_{j+1}.$$

Similarly to (6) in the proof of Lemma 3, we have $(2m + 1)^2 p'_{j+1} \leq \frac{1}{\log n}$, which implies

$$p_{j+1} \geq \left(1 - \frac{1}{3 \log n}\right) (2m+1)^2 p'_{j+1}.$$

Therefore,

$$2m+1 \le \left(1+\frac{C}{\log n}\right)\sqrt{\frac{p_{j+1}}{p'_{j+1}}} \tag{10}$$

for some constant C. Multiplying (9) by $2m+1$ and applying (10) completes the induction step. □

Lemma 8 *For all $j, j'(j<j')$, $r_{j,j'} = O(\log n)$.*

Proof We consider the ratio $q_j = \frac{p_j}{n_j}$. We have $q_1 = \frac{1}{n}$ and $q_j \le 1$ for all j (since $p_j \le 1$ and $n_j \ge 1$).

Next, we relate q_j and q_{j+1}. We have $\frac{p'_{j+1}}{n_{j+1}} = \frac{p_j}{n_j}$. If step 2d is applied, $p_{j+1} = p'_{j+1}$ and $q_{j+1} = \frac{p'_{j+1}}{n_{j+1}} = q_j$. If step 2e is applied,

$$p_{j+1} \ge (2m+1)^2\left(1-\frac{1}{3\log n}\right)p'_{j+1} \ge 9\left(1-\frac{1}{3\log n}\right)p'_{j+1}.$$

Therefore, $q_{j+1} \ge 9(1-\frac{1}{3\log n})q_j$. This means that $q_{j'} \ge (9-\frac{3}{\log n})^{r_{j,j'}}q_j$. Together with $q_{j'} \le 1$ and $q_j \ge q_1 \ge \frac{1}{n}$, this implies $r_{j,j'} = O(\log n)$. □

We can now complete the proof of Lemma 6.

Proof of Lemma 6 We look at each of the components of the sum (8) separately. Consider a term

$$\left(1+\frac{C}{\log n}\right)^{r_{j',j}}\sqrt{\frac{p_j n_{j'-1}}{p_{j'-1}n_j}}2^{j'}. \tag{11}$$

Because of Lemma 8, the first multiplier is bounded from above by a constant. Since $p_{j'-1} \ge \frac{1-o(1)}{9\log n}$ (similarly to Lemma 3), we can upperbound (11) by $O(\sqrt{\log n\frac{n_{j'-1}}{n_j}}2^{j'})$.

By the assumption at the beginning of the proof of Theorem 7, there is at most one marked item. All unmarked $i \in S_{j'-1}$ must have $t_i \ge 2^{j'-1}$. Since $|S_{j'-1}| = n_{j'-1}$, this means that there are at least $n_{j'-1}-1$ indices i with $t_i \ge 2^{j'-1}$, Hence,

$$t_1^2+t_2^2+\ldots+t_n^2 \ge (n_{j'-1}-1)(2^{j'-1})^2 = \Omega(n^{j'-1}2^{2j'}).$$

Therefore, each term in (11) is at most

$$O\left(\sqrt{\log n}\sqrt{\frac{t_1^2+t_2^2+\ldots+t_n^2}{n_j}}\right).$$

The lemma follows by summing over all j terms in (8). □

7 Search Lower Bound

Theorem 8 *For any positive integers $t_1, \ldots, t_n$, searching a collection of n items with query times $t_1, \ldots, t_n$ requires $\Omega(\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2})$ query steps.*

Proof Let t_i' be the maximum integer such that $\lceil \frac{\pi}{4}\sqrt{t_i'} \rceil + 1 \leq t_i$ (with $t_i' = 1$ if the maximum integer is 0). Let $m = t_1' + \ldots + t_n'$. We consider the following problem:

UNIQUE-OR. We are given $x_1, \ldots, x_m \in \{0, 1\}$ as the input, with the promise that there is either 0 or 1 element $j : x_j = 1$. We have to determine whether there are 0 elements $j : x_j = 1$ or 1 element $j : x_j = 1$.

By the standard lower bound on quantum search, solving UNIQUE-OR in the standard query model (where every query takes 1 step) requires $\Omega(\sqrt{m})$ quantum queries. The next lemma shows that variable-item search is at least as hard as UNIQUE-OR:

Lemma 9 *Assume that there is a search algorithm $\mathcal{A}$ for searching a collection of n items with query times $t_1, \ldots, t_n$, in the known-query-time model. Let t be the number of query steps used by $\mathcal{A}$. Then, there is an algorithm $\mathcal{A}'$ that solves UNIQUE-OR in the standard query model with t queries.*

Proof To design $\mathcal{A}'$, we subdivide the inputs $x_1, \ldots, x_m$ into n groups $S_1, \ldots, S_n$, with $t_1', \ldots, t_n'$ elements, respectively. Let $y_i = 1$ if there exists $j \in S_i$ with $x_j = 1$. Since there is either 0 or 1 element $j : x_j = 1$, we know that there is either 0 or 1 element $i : y_i = 1$. We have

Claim 2 *There is an algorithm that implements the transformation $|i\rangle \to |i\rangle|y_i\rangle|\psi_i\rangle$ for some states $|\psi_i\rangle$, using t_i queries.*

Proof To simplify the notation, we assume that the group S_i consists of variables $x_1, \ldots, x_{t_i'}$. If $t_i' = 1$, then $y_i = x_1$ and we can just query x_1. This produces the required transformation $|i\rangle \to |i\rangle|y_i\rangle$.

For the $t_i' > 1$ case, we have to search t_i' items $x_1, \ldots, x_{t_i'}$ for an item $j : x_j = 1$, if we are promised that there is either 0 or 1 such item. There is a modification of Grover's algorithm which succeeds with probability 1, using at most $\lceil \frac{\pi}{4}\sqrt{t_i'} \rceil$ queries [7].

The result of Grover's algorithm is:

- the state $|j\rangle$, where j is the index for which $x_j = 1$, if such j exists;
- the superposition $\frac{1}{\sqrt{t_i'}} \sum_{j=1}^{t_i'} |j\rangle$, otherwise.

With one more query (which queries the index j), we can determine the value $y_i = x_j$ (which is 1 in the first case and 0 in the second case). □

Let $\mathcal{A}$ be the given search algorithm for n items with query times $t_1, \ldots, t_n$. Then, we can substitute the algorithm of Claim 2 instead of the queries y_i. This gives us the required algorithm $\mathcal{A}'$ for UNIQUE-OR. □

To complete the proof, we need to show that

$$\sqrt{m} = \Omega(\sqrt{t_1^2 + \ldots + t_n^2}).$$

By definition of t_i', we have

$$t_i \leq \left\lceil \frac{\pi}{4}\sqrt{t_i'} \right\rceil + 2 \leq \frac{\pi}{4}\sqrt{t_i'} + 3.$$

This means that $t_i' \geq \frac{16}{\pi^2}(t_i - 3)^2$. If $t_i \geq 4$, then $t_i - 3 \geq \frac{t_i}{4}$ and $t_i' \geq \frac{1}{\pi^2}t_i^2$. If $t_i \leq 3$, then $t_i' \geq 1 \geq \frac{1}{9}t_i^2 \geq \frac{1}{\pi^2}t_i^2$. Therefore,

$$\sqrt{m} = \sqrt{t_1' + \ldots + t_n'} \geq \sqrt{\frac{1}{\pi^2}(t_1^2 + \ldots + t_n^2)} = \frac{1}{\pi}\sqrt{t_1^2 + \ldots + t_n^2}. \qquad \square$$

8 Conclusion

In this paper, we gave a quantum algorithm for the generalization of Grover's search in which checking different items requires different times. Our algorithm is optimal for the case when times t_i are known in advance and nearly optimal (within a polylogarithmic factor) for the general case. We also gave an application of our algorithm to computing read-once Boolean functions. It is likely that our algorithms will find other applications.

While we have mostly resolved the complexity of search in this setting, the complexity of other problems has not been studied at all. Of particular interest are problems which are frequently used as subroutines in other quantum algorithms (for such problems, there is a higher chance that the variable-time query version will be useful). Besides the usual quantum search, the two most common quantum subroutines are quantum counting [6] and k-item search (a version of search in which one has to find k different i for which $x_i = 1$). Element distinctness [3, 10] has also been used as a subroutine, to design quantum algorithms for the triangle problem [16] and verifying matrix identities [8, 15].

Acknowledgements I would like to thank Robert Špalek and Ronald de Wolf for the discussion that lead to this paper and several anonymous referees for useful comments.

Appendix A: Formal Definition of Our Model

To define our model formally, let $\mathcal{A}_i^{(j)}$ be the j^{th} step of $\mathcal{A}_i$. Then,

$$\mathcal{A}_i = \mathcal{A}_i^{(t_i)}\mathcal{A}_i^{(t_i-1)} \ \ldots \ \mathcal{A}_i^{(1)}.$$

We define $\mathcal{A}_i^{(t)} = I$ for $t > t_i$. We regard the state space of $\mathcal{A}_i$ as consisting of two registers, one of which stores the answer ($c \in \{0, 1, 2\}$, with 2 representing a computation that has not been completed) and the other register, x, stores any other information.

The state space of a search algorithm is spanned by basis states of the form $|i,t,t_r,c,x,z\rangle$ where $i \in \{1,\ldots,n\}$, $t,t_r \in \{0,1,\ldots,T\}$ (with T being the number of the query steps in the algorithm), $c \in \{0,1,2\}$ and x and z range over arbitrary finite sets. i represents the index being queried, t represents the number of the time step in which the query for x_i started and t_r is the number of time steps for which $\mathcal{A}$ will run the query algorithm $\mathcal{A}_i$. c is the output register of $\mathcal{A}_i$ and x holds intermediate data of $\mathcal{A}_i$. Both of those registers should be initialized to $|0\rangle$ at the beginning of every computation of a new x_i. z contains any data that is not a part of the current query.

We define a quantum query algorithm $\mathcal{A}$ as a tuple $(U_0,\ldots,U_T)$ of unitary transformations that do not depend on $x_1,\ldots,x_n$. The actual sequence of transformations that is applied is

$$U_0, Q_1, U_1, Q_2, \ldots, U_{T-1}, Q_T, U_T,$$

where Q_j are queries which are defined below. This sequence of transformations is applied to a fixed starting state $|\psi_{start}\rangle$, which consists of basis states $|i,0,0,c,x,z\rangle$.

Queries Q_j are defined in a following way. If $j \le t + t_r$, we apply $A_i^{(j-t)}$ to $|c\rangle$ and $|x\rangle$ registers. Otherwise, we apply I. We call the resulting sequence of queries $Q_1, Q_2, \ldots$ *generated* by transformations A_i^j. We call Q_1, Q_2 a *valid* sequence of queries corresponding to $x_1,\ldots,x_n$ if it is generated by A_i^j satisfying the following constraints:

1. For $t < t_i$, $A_i^t A_i^{t-1} \ldots A_i^1 |0\rangle$ is of the form $|2\rangle|\psi\rangle$ for some $|\psi\rangle$.
2. For $t = t_i$, $A_i^t A_i^{t-1} \ldots A_i^1 |0\rangle$ is of the form $|x_i\rangle|\psi\rangle$ for some $|\psi\rangle$.

A search algorithm $(U_0,\ldots,U_T)$ with the starting state $|\psi_{start}\rangle$ computes a function $f(x_1,\ldots,x_n)$ if, for every $x_1,\ldots,x_n \in \{0,1\}$ and every valid query sequence $Q_1,\ldots,Q_T$ corresponding to $x_1,\ldots,x_n$, the probability of obtaining $f(x_1,\ldots,x_n)$ when measuring the first qubit of

$$U_T Q_T U_{T-1} \ldots U_1 Q_T U_0 |\psi_{start}\rangle$$

is at least $2/3$.

Appendix B: Proofs of Claims from Sect. 3

Proof of Claim 1 Because of (1), (2),

$$p(\delta) = \sin^2((2m+1)\arcsin\sqrt{\delta}).$$

Let $\gamma = \arcsin\sqrt{\delta}$ and $\gamma' = \arcsin\sqrt{\delta'}$. Then, we have to prove that $\sin^2\gamma' \le \sin^2\gamma \le c\sin^2\gamma'$ implies $\sin^2(2m+1)\gamma' \le \sin^2(2m+1)\gamma \le c\sin^2(2m+1)\gamma'$.

Because of $\delta \le \epsilon$ and $\delta' \le \epsilon$, we have $\sqrt{\delta} \le \sqrt{\epsilon}$ and $\sqrt{\delta'} \le \sqrt{\epsilon}$. Together with (3), that means that $(2m+1)\arcsin\sqrt{\delta} \le \frac{\pi}{2}$ and $(2m+1)\arcsin\sqrt{\delta'} \le \frac{\pi}{2}$. Since sin is an increasing function on $[0,\frac{\pi}{2}]$, $\sin^2\gamma' \le \sin^2\gamma$ implies $\sin^2(2m+1)\gamma' \le \sin^2(2m+1)\gamma$.

Algorithm 3 Procedure **Estimate**

1. Let $M = 2$;
2. Repeat:
   (a) Let $\tilde{\epsilon}$ be the estimate output by repeated **Est-Amp**$(\mathcal{A}, M)$.
   (b) If $2\pi \frac{\sqrt{\tilde{\epsilon}(1-\tilde{\epsilon})}}{M} + \frac{\pi^2}{M^2} \leq c\tilde{\epsilon}$, stop and output $\tilde{\epsilon}$ as the estimate.
   (c) $M = 2 * M$.
   until $M > M_{max}$ where $M_{max} = \frac{16\pi}{c\sqrt{(1-p)p}}$.

To prove the other inequality, it suffices to show that

$$\frac{\sin(2m+1)\gamma}{\sin(2m+1)\gamma'} \leq \frac{\sin\gamma}{\sin\gamma'}$$

whenever $\gamma' \leq \gamma$. Equivalently, it suffices to prove

$$\sin x \sin y \leq \sin x' \sin y' \tag{12}$$

where $x = (2m+1)\gamma$, $y = \gamma'$, $x' = (2m+1)\gamma'$, $y' = \gamma$. Because of $\gamma > \gamma'$, x is always the largest of x, y, x', y'. For the same reason, y is always the smallest of x, y, x', y'. Also, $xy = (2m+1)\gamma\gamma' = x'y'$.

We make the substitution $x = e^u$, $y = e^t$, $x' = e^{u'}$, $y' = e^{t'}$. Then, (12) becomes

$$\sin e^u \sin e^t \leq \sin e^{u'} \sin e^{t'} \tag{13}$$

if u, t, u', t' satisfy $u + t = u' + t'$ and u, t are the largest and the smallest of u, t, u', t'. Equation (13), in turn, is implied by the concavity of the function $f(x) = \ln \sin e^x$.

The concavity of $f(x)$ can be verified by computing its second derivative,

$$f''(x) = e^x \frac{\cos e^x}{\sin e^x} - \frac{e^{2x}}{\sin^2 e^x} = e^x \sin e^x \cos e^x - e^x \sin^2 e^x.$$

For $f(x)$ to be concave, we need $f''(x) < 0$ which is equivalent to $\sin e^x \cos e^x - e^x < 0$. Replacing $y = e^x$, we see that this is equivalent to $\sin y \cos y = \frac{\sin 2y}{2} \leq \frac{2y}{2} = y$ which is true for all positive y. □

Proof of Theorem 2 We can increase the success probability of **Est-Amp**$(\mathcal{A}, M)$ to at least $1 - \frac{1}{2^k \log M_{max}}$ (where $M_{max} = \frac{8\pi}{c\sqrt{(1-c)p}}$), by repeating the algorithm $t = O((1 + \log\log\frac{1}{p})k)$ times and taking the median of the results.

The procedure **Estimate** calls the repeated **Est-Amp** at most $\log M_{max}$ times. Since each call of **Est-Amp** produces the correct answer with probability at least $1 - \frac{1}{2^k \log M_{max}}$, the probability that all calls to **Est-Amp** produce correct results is at least $1 - \frac{1}{2^k}$. In this case, **Estimate** is always correct, because by Theorem 1, the error $|\tilde{\epsilon} - \epsilon|$ is at most $\frac{2\pi\sqrt{\tilde{\epsilon}(1-\tilde{\epsilon})}}{M} + \frac{\pi^2}{M^2}$ and **Estimate** only stops when this quantity becomes less than $c\tilde{\epsilon}$. It remains to bound the number of times **Estimate** calls $\mathcal{A}$.

If $M \geq \frac{8\pi}{c\sqrt{(1-\epsilon)\epsilon}}$, then, by Theorem 1, $|\tilde{\epsilon} - \epsilon|$ is at most

$$\frac{2\pi\sqrt{(1-\epsilon)\epsilon}}{M} + 2\frac{\pi^2}{M^2} \leq \frac{c\sqrt{\epsilon(1-\epsilon)}}{4} + \frac{c^2\epsilon(1-\epsilon)}{32} \leq 0.3\epsilon, \tag{14}$$

with the last inequality following from $c \leq 1$. Therefore,

$$\begin{aligned}\frac{2\pi\sqrt{(1-\tilde{\epsilon})\tilde{\epsilon}}}{M} + 2\frac{\pi^2}{M^2} &\leq \frac{2\pi\sqrt{1.3(1-\epsilon)\epsilon}}{M} + 2\frac{\pi^2}{M^2} \leq \frac{c\sqrt{1.3\epsilon(1-\epsilon)}}{4} + \frac{c^2\epsilon(1-\epsilon)}{32} \\ &\leq 0.32c\epsilon \leq c\tilde{\epsilon},\end{aligned}$$

with the first inequality following from $\tilde{\epsilon} \leq \epsilon + 0.3\epsilon$ and the last inequality following from $\tilde{\epsilon} \leq \epsilon - 0.3\epsilon$.

Therefore, the quantity of (14) is less than or equal to $c\tilde{\epsilon}$. Hence, if $M \geq \frac{8\pi}{c\sqrt{(1-\epsilon)\epsilon}}$, then the condition in step 2b is satisfied and the algorithm stops. Since M is doubled in every iteration, the final value of M is $M_0 < \frac{16\pi}{c\sqrt{(1-\epsilon)\epsilon}}$. The algorithm $\mathcal{A}$ is repeated

$$M_0 t + \frac{M_0 t}{2} + \frac{M_0 t}{4} + \ldots < 2M_0 t < \frac{32\pi}{c\sqrt{(1-\epsilon)\epsilon}} t$$

times.

If $\epsilon \geq p$, the algorithm must stop with M being at most $\frac{16\pi}{c\sqrt{(1-p)p}}$. If that does not happen, we can conclude that $\epsilon = 0$. The number of repetitions of $\mathcal{A}$ in this case is at most $\frac{32\pi}{c\sqrt{(1-p)p}} t$. □

References

1. Aaronson, S., Ambainis, A.: Quantum search of spatial regions. Theory Comput. **1**, 47–79 (2005). quant-ph/0303041
2. Ambainis, A.: Quantum search algorithms. SIGACT News **35**, 22–35 (2004). quant-ph/0504012
3. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**, 210–239 (2007). Also FOCS'04 and quant-ph/0311001
4. Ambainis, A., Childs, A., Reichardt, B., Spalek, R., Zhang, S.: Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In: Proceedings of FOCS'07, pp. 363–372 (2007)
5. Barnum, H., Saks, M.: A lower bound on the quantum complexity of read once functions. J. Comput. Syst. Sci. **69**, 244–258 (2004)
6. Brassard, G., Høyer, P., Tapp, A.: Quantum counting. In: Proceedings of ICALP'98, pp. 820–831 (1998). quant-ph/9805082
7. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. In: Quantum Computation and Quantum Information Science. AMS Contemporary Mathematics Series, vol. 305, pp. 53–74. AMS, Providence (2002). quant-ph/0005055
8. Buhrman, H., Špalek, R.: Quantum verification of matrix products. In: Proceedings of SODA'06, pp. 880–889 (2006). quant-ph/0409035
9. Buhrman, H., Cleve, R., Wigderson, A.: Quantum vs. classical communication and computation. In: Proceedings of STOC'98, pp. 63–68 (1998). quant-ph/9702040
10. Buhrman, H., Durr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. SIAM J. Comput. **34**(6), 1324–1330 (2005). quant-ph/0007016
11. Farhi, E., Goldstone, J., Gutman, S.: A quantum algorithm for the Hamiltonian NAND tree. quant-ph/0702144

12. Grover, L.: A fast quantum mechanical algorithm for database search. In: Proceedings of STOC'96, pp. 212–219 (1996)
13. Høyer, P., Mosca, M., de Wolf, R.: Quantum search on bounded-error inputs. In: Proceedings of ICALP'03. Lecture Notes in Computer Science, vol. 2719, pp. 291–299. Springer, Berlin (2003). quant-ph/0304052
14. Høyer, P., Lee, T., Špalek, R.: Tight adversary bounds for composite functions. quant-ph/0509067
15. Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. Algorithmica **48**(3), 221–232 (2007). Also ICALP'05 and quant-ph/0506265
16. Magniez, F., Santha, M., Szegedy, M.: Quantum algorithms for the triangle problem. SIAM J. Comput. **37**(2), 413–424 (2007). Also SODA'05 and quant-ph/0310134
17. Reichardt, B., Spalek, R.: Span-program-based quantum algorithm for evaluating formulas. In: Proceedings of STOC'2008, pp. 103–112 (2008)