QUANTUM SEARCH WITH VARIABLE TIMES

ANDRIS AMBAINIS [1]

[1] Department of Computer Science, University of Latvia
Raina bulv. 19, Riga, LV-1586, Latvia
E-mail address: andris.ambainis@lu.lv

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 polylogarithmic overhead. We also give an application of our new search algorithm to computing read-once functions.

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 [3].

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.

Let t_i be the number of time steps required to check the ith 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 a $O(\sqrt{n} t_{max})$ 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 ith 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] but is also much simpler than the algorithms in [4, 11].

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].

Some of the proofs are omitted due to the space constraints. A full version of the paper is available as arXiv preprint `quant-ph/0609168`.

2. Model

We would like to model the situation when the variable x_i is computed by an algorithm 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 assume that A_i always outputs the correct x_i.) In the first $t_i - 1$ steps, 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 procedure A can run the algorithms A_i, for some number of steps t, with A_i outputting x_i if $t_i \leq t$ or "the computation is not complete" if $t_i > t$. The computational cost is the amount of time that is spent running algorithms A_i. Any transformations that do not involve A_i are free. This is a generalization of the usual quantum query model.

For completeness, we include a more formal definition of our model in the appendix A. Our 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 algorithm. In the "unknown times" model, $t_1, \ldots, t_n$ are unknown to the designer of the algorithm.

3. Methods and subroutines

3.1. Amplitude amplification

Amplitude amplification [8] is a generalization of Grover's quantum search algorithm. Let
$$\sin\alpha|1\rangle|\psi_1\rangle + \cos\alpha|0\rangle|\psi_0\rangle \tag{3.1}$$
be the final state of a quantum algorithm 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. [8]

showed that, by repeating A and A^{-1} $2m+1$ times, it is possible to generate the final state

$$\sin(2m+1)\alpha|1\rangle|\psi_1\rangle + \cos(2m+1)\alpha|0\rangle|\psi_0\rangle. \tag{3.2}$$

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 3.1. [1] *Let A be a quantum algorithm that outputs a correct answer and a witness with probability[1] $\delta \leq \epsilon$ where ϵ is known. Furthermore, let*

$$m \leq \frac{\pi}{4\arcsin\sqrt{\epsilon}} - \frac{1}{2}. \tag{3.3}$$

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

$$\delta_{new} \geq \left(1 - \frac{(2m+1)^2}{3}\delta\right)(2m+1)^2\delta. \tag{3.4}$$

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 3.1 achieves an increase of almost $(2m+1)^2$ 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 3.1 avoids that.

We also need another fact about amplitude amplification.

Claim 3.2. *Let δ and δ' be such that $\delta \leq \epsilon$ and $\delta' \leq \epsilon$ and let m satisfy the constraint (3.3). Let $p(\delta)$ be the success probability obtained by applying the procedure of Lemma 3.1 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. Omitted.   ∎

3.2. Amplitude estimation

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

Theorem 3.3. [8] *There is a procedure $\mathbf{Est\text{-}Amp}(A, M)$ which, given a quantum algorithm A and a number M, outputs an estimate $\tilde{\epsilon}$ of the probability ϵ that A outputs 1 and, with probability at least $\frac{8}{\pi^2}$, we have*

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

The algorithm uses M evaluations of 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 $c > 0$.

---

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

Theorem 3.4. *There is a procedure **Estimate**(A, c, p, k) which, given a constant c, $0 <$
$c \le 1$ and a quantum algorithm A (with the promise that the probability ϵ that the algorithm
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* $\epsilon \ge p$;
  (ii) $\tilde{\epsilon} = 0$ *if* $\epsilon = 0$.

*The procedure **Estimate**(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 A.

Proof. Omitted.   ∎

4. Search algorithm: known running times

Theorem 4.1. *A collection of n items with times $t_1, \ldots, t_n$ can be searched in time*

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

Proof. The basic idea is to subdivide the items 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

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

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 algorithms for searching the k groups instead of the k items and
obtain a search algorithm for n items that runs in time

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

We then design a search algorithm for k items in a similar way.

The simplest implementation of this strategy gives an algorithm with $\log^* n$ levels of
recursion and running time

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

due to the reduction from n items to k items losing a constant factor every time it is used. The $c^{\log^* n}$ factor can be avoided, by a more sophisticated implementation of the same idea, which we describe below.

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 4.2. *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 number of queries 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 generalization of the problem in which the algorithm 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 the algorithm A_i is of the form $|0\rangle|\psi_0\rangle$.
- if $x_i = 1$, the final state of the algorithm A_i is of the form $\alpha|1\rangle|\psi_1\rangle + \sqrt{1 - \alpha^2}|0\rangle|\psi_0\rangle$, where $p_i \leq |\alpha|^2 \leq d \cdot p_i$, 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 A_i return the correct answer with certainty.)

We claim that, in this case, we can search in time

$$O\left(\sqrt{\frac{t_1^2}{p_1} + \frac{t_2^2}{p_2} + \ldots + \frac{t_n^2}{p_n}}\right).$$

Our main theorem now follows as the particular case $p_1 = \ldots = p_n = 1$. The main part of our proof is

Lemma 4.3. *There exists $k = O(\log^3 n \log \log n)$ with the following property. Assume that there is a search algorithm for k items with some fixed $d > 1$ that works in time at most*

$$C\sqrt{\frac{s_1^2}{q_1} + \frac{s_2^2}{q_2} + \ldots + \frac{s_k^2}{q_k}}.$$

for any given times $s_1, \ldots, s_k$ and probabilities $q_1, \ldots, q_k$. Then, there exists a search algorithm for n items with $d' = \left(1 - O\left(\frac{1}{\log n}\right)\right) d$ instead of d that works in time at most

$$C\left(1 + O\left(\frac{1}{\log n}\right)\right) \sqrt{\frac{t_1^2}{p_1} + \frac{t_2^2}{p_2} + \ldots + \frac{t_n^2}{p_n}}$$

for any given times $t_1, \ldots, t_n$ and probabilities $p_1, \ldots, p_n$.

Proof. Omitted.

∎

To obtain Theorem 4.1, we repeatedly apply Lemma 4.3 until the number of items becomes less than some constant n_0. That happens after $O(\log^* n)$ applications of Lemma 4.3.

Let $t_1, \ldots, t_n$ and $p_1, \ldots, p_n$ be the times and probabilities for the final $n \leq n_0$ items. After that, we just amplify the success probability of every item to $\Omega(1)$ (which increases each $\frac{t_i^2}{p_i}$ by at most a constant factor, as discussed in the proof of Lemma 4.3). We then search n items in time $O(\sqrt{n} \max_i t_i)$, using the amplitude amplification, with $\max_i t_i$ steps for evaluating any of the items i. Since $p_i = \Omega(1)$ and $n \leq n_0$ where n_0 is a constant, we have

$$\sqrt{n} \max t_i = O(\max t_i) = O\left(\sqrt{t_1^2 + \ldots + t_n^2}\right) = O\left(\sqrt{\frac{t_1^2}{p_1} + \ldots + \frac{t_n^2}{p_n}}\right).$$

$O(\log^* n)$ applications of Lemma 4.3 increase the time by a factor of at most $(1 + O(\frac{1}{\log n}))^{\log^* n} = 1 + o(1)$.

∎

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. Hoyer, Mosca and de Wolf [13] have constructed a $O(\sqrt{N})$ 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 [10]). In a very recent breakthrough work, [11, 4] 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 [11, 4] but is also much simpler.

Theorem 5.1. *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. We use induction. If f is represented by a depth-d tree with OR at the root, we express

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

---

(1) Set $j = 1$. Define B_1 as the algorithm that just outputs 1 and a uniformly random $i \in \{1, \ldots, n\}$.

(2) Repeat:

    (a) Use the algorithm B_j to generate $k = 2\log(D(j + 1))$ samples $i_1, \ldots, i_k$ of uniformly random elements $i \in S_j$. Run 2^{j+1} steps of the query procedure on each of $i_1, \ldots, i_k$. If $x_i = 1$ for one of samples, output i and stop.

    (b) Let B'_{j+1} be an algorithm that runs 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 $x_i = 0$, B'_j outputs 0. Otherwise, it outputs 1 and the same index i.

    (c) Let $p = \textbf{Estimate}(B'_{j+1}, c, \frac{1}{N}, 2\log(D(j+1)))$. If $p = 0$, output "no $i : x_i = 0$".

    (d) If $p \geq \frac{1}{9\log n}$, let B_{j+1} be B'_{j+1}.

    (e) If $p < \frac{1}{9\log n}$, let B_{j+1} be the algorithm obtained by amplifying B'_{j+1} $2m + 1$ times, where m is the smallest number for which $\frac{1}{9\log n} \leq (2m + 1)^2 p \leq \frac{1}{\log n}$. (Such choice of m always exists, as described in the proof of Lemma 4.3.)

    (f) Let $j = j + 1$.

Algorithm 1: Search algorithm for unknown $t_1, \ldots, t_n$

By inductive assumption, we construct algorithms computing the functions f_i in $O(\sqrt{t_i}\log^{d-2} N)$ queries. We then combine them into a quantum algorithm computing f by applying Theorem 4.1.

A more detailed proof is given in the arXiv version of the paper. ∎

6. Search algorithm: unknown running times

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 6.1. *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 4.1.)

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 1, defines a sequence of algorithms $B_1, \ldots, B_l$. The algorithm B_j, with some success probability, outputs a bit 1 and, conditional on output bit 1, it also outputs a uniformly random index $i \in S_j$. To avoid the problem with accumulating constant factors (described after Lemma 3.1), we make the success probability of B_j slightly less than 1.

Lemma 6.2. *Assume that the constant D in steps 2a and 2c satisfies $D \leq \frac{\pi}{\sqrt{3\epsilon}}$. Then, with probability $1 - \epsilon$, the following conditions are satisfied:*

    (a) *Estimates p are accurate within an multiplicative factor of $(1 + c)$;*

    (b) *If B_j is defined, then $t_i > 2^{j-1}$ for at least $\frac{n_{j-1}}{2}$ values $i \in \{1, \ldots, n\}$.*

Proof. (a) 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 $D \leq \frac{\pi}{\sqrt{3\epsilon}}$.

(b) By definition, S_{j-1} is the set of all i with the property that either $x_i = 1$ or $t_i > 2^{j-1}$. Let S be the set of i with $x_i = 1$ and $t_i \leq 2^{j-1}$. If $|S| \leq \frac{1}{2}n_{j-1}$, (c) is true. Otherwise, the probability that each i_j generated in step 2a does not belong to S is less than $\frac{1}{2}$. If one of them belongs to S, algorithm 1 stops without defining B_j. The probability that this does not happen (i.e., all i_j do not belong to S) is less than $(\frac{1}{2})^k = \frac{1}{D^2(j+1)^2}$. We can make this probability arbitrarily small similarly to part (a). ∎

For the rest of the proof, we assume that both conditions of Lemma 6.2 are true. Under this assumption, we bound the running time of algorithm 1. The first step is to bound the running time of the algorithms B_j.

Lemma 6.3. *The running time of 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. Omitted. ∎

We now bound the overall running time. To generate a sample from S_j, one needs $O(\sqrt{\log n})$ invocations of B_j (because the success probability of 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 6.3, 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 B_j (because 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 step 2c is of order $\frac{p_j n_{j+1}}{n_j} = \Omega(\frac{n_{j+1}}{n_j \log n})$. By Theorem 3.4, it can be estimated with

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

invocations of B_j, each of which runs in time described by Lemma 6.3.

Thus, the overall number of steps in one loop of algorithm 1 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 1 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 $n \leq \sqrt{T}$ and $t_{max} \leq T$, where $T = t_1^2 + t_2^2 + \ldots + t_n^2$. ∎

7. Search lower bound

Theorem 7.1. *For any positive integers $t_1, \ldots, t_n$, searching a collection of n items that can be checked in times $t_1, \ldots, t_n$ requires time $c\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}$, for some constant $c > 0$.*

Proof. Let t_i' be the minimum positive integer such that $t_i \leq \lceil \frac{\pi}{4}\sqrt{t_i'} \rceil + 1$ (with $t_i' = 1$ if there is no positive integer satisfying this inequality). We consider searching $m = t_1' + \ldots + t_n'$ elements $x_1, \ldots, x_m \in \{0, 1\}$ in the standard model (where every query takes 1 step), with the promise that there is either 0 or 1 element $j : x_j = 1$. By lower bound on quantum search, $c'\sqrt{m}$ queries are required to distinguish between the case when there are 0 elements $j : x_j = 1$ and the case when there is 1 element $j : x_j = 1$, for some constant c'.

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

Lemma 7.2. *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. Omitted. ∎

Let A be a search algorithm for search among n items that require times $t_1, \ldots, t_n$ and let t' be the number of steps used by A. Then, we can substitute the algorithm of Lemma 7.2 instead of the queries y_i. Then, we obtain an algorithm A' that, given $x_1, \ldots, x_n$, asks t' queries and distinguishes whether there is exactly 1 item $i : y_i = 1$ (and, hence, 1 item $j : x_j = 1$) or there is no items $i : y_i = 0$ (and, hence, no items $j : x_j = 1$). Hence,

$$t' \geq c'\sqrt{m} = c'\sqrt{t_1' + \ldots + t_n'}.$$

We now bound t_i' in terms of t_i. By definition of t_i', we have

$$t_i \leq \left\lceil \frac{\pi}{4}\sqrt{t_i'} \right\rceil + 1 \leq \frac{\pi}{4}\sqrt{t_i'} + 2.$$

This means that $t_i' \geq \frac{16}{\pi^2}(t_i - 2)^2$. If $t_i \geq 3$, then $t_i - 2 \geq \frac{t_i}{3}$ and $t_i' \geq \frac{16}{9\pi^2}t_i^2$. If $t_i < 3$, then $t_1' \geq 1 \geq \frac{16}{9\pi^2}t_i^2$. Therefore,

$$t' \geq c'\sqrt{t_1' + \ldots + t_n'} \geq c'\sqrt{\frac{16}{9\pi^2}(t_1^2 + \ldots + t_n^2)} = \frac{4c'}{3\pi}\sqrt{t_1^2 + \ldots + t_n^2}.$$

This means that the theorem is true, with $c = \frac{4c'}{3\pi}$. ∎

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 have not been studied at all. Of particular interest are problems which are frequently used as a 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 [9] and k-item search (a version of search in which one has to find k different i for which $x_i = 1$). Element distinctness [2, 6] has also been used as a subroutine, to design quantum algorithms for the triangle problem [16] and verifying matrix identities [7, 15].

Acknowledgements

References

[1] S. Aaronson, A. Ambainis, Quantum search of spatial regions. *Theory of Computing*, 1:47-79, 2005. Also quant-ph/0303041.

[2] A. Ambainis. Quantum walk algorithm for element distinctness. *Proceedings of FOCS'04*, pp. 22-31. Also quant-ph/0311001.

[3] A. Ambainis. Quantum search algorithms. *SIGACT News*, 35 (2004):22-35. Also quant-ph/0504012.

[4] A. Ambainis, A. Childs, B. Reichardt, R. Spalek, S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *Proceedings of FOCS'07*, to appear.

[5] H. Barnum, M. Saks, A lower bound on the quantum complexity of read once functions. *Journal of Computer and System Sciences*, 69:244-258, 2004.

[6] H. Buhrman, C. Durr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, R. de Wolf. Quantum algorithms for element distinctness. *SIAM Journal on Computing*, 34(6): 1324-1330, 2005. Also quant-ph/0007016.

[7] H. Buhrman, R. Špalek: Quantum verification of matrix products. *Proceedings of SODA'06*, pp. 880-889. Also quant-ph/0409035.

[8] G. Brassard, P. Høyer, M. Mosca, A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information Science*, AMS Contemporary Mathematics Series, 305:53-74, 2002. Also quant-ph/0005055.

[9] G. Brassard, P. Høyer, A. Tapp. Quantum counting. *Proceedings of ICALP'98*, pp. 820-831, quant-ph/9805082.

[10] H. Buhrman, R. Cleve, A. Wigderson, Quantum vs. classical communication and computation. *Proceedings of STOC'98*, pages 63-68, quant-ph/9702040.

[11] E. Farhi, J. Goldstone, S. Gutman, A Quantum Algorithm for the Hamiltonian NAND Tree. quant-ph/0702144.

[12] L. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of STOC'96*, pp. 212-219.

[13] P. Høyer, M. Mosca, and R. de Wolf. Quantum search on bounded-error inputs. *Proceedings of ICALP'03*, Lecture Notes in Computer Science, 2719:291-299. Also quant-ph/0304052

[14] P. Høyer, T. Lee, R. Špalek. Tight adversary bounds for composite functions, quant-ph/0509067.

[15] F. Magniez, A. Nayak. Quantum complexity of testing group commutativity. *Algorithmica*, 48(3): 221-232, 2007. Also ICALP'05 and quant-ph/0506265.

[16] F. Magniez, M. Santha, M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2): 413-424, 2007. Also SODA'05 and quant-ph/0310134.

Appendix A. Formal definition of our model

To define our model formally, let $A_i^{(j)}$ be the jth step of A_i. Then,

$$A_i = A_i^{(t_i)} A_i^{(t_i-1)} \ldots A_i^{(1)}.$$

We define $A_i^{(t)} = I$ for $t > t_i$. We regard the state space of 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 A will run the query algorithm A_i. c is the output register of A_i and x holds intermediate data of 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 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 $j \leq t + t_r$, 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 $|\psi\rangle$.

U_j can be arbitrary transformations that do not depend on $x_1, \ldots, x_n$.

An 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 $2/3$.