Quantum Approximate Counting with Nonadaptive Grover Iterations

Ramgopal Venkateswaran[1]
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Ryan O'Donnell
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Approximate Counting refers to the problem where we are given query access to a function $f : [N] \to \{0,1\}$, and we wish to estimate $K = \#\{x : f(x) = 1\}$ to within a factor of $1 + \epsilon$ (with high probability), while minimizing the number of queries. In the quantum setting, Approximate Counting can be done with $O\big(\min\big(\sqrt{N/\epsilon}, \sqrt{N/K}\,/\,\epsilon\big)\big)$ queries. It has recently been shown that this can be achieved by a simple algorithm that only uses "Grover iterations"; however the algorithm performs these iterations adaptively. Motivated by concerns of computational simplicity, we consider algorithms that use Grover iterations with limited adaptivity. We show that algorithms using only nonadaptive Grover iterations can achieve $O\big(\sqrt{N/\epsilon}\big)$ query complexity, which is tight.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases quantum approximate counting, Grover search

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.59

Related Version *Previous Version*: https://arxiv.org/pdf/2010.04370.pdf

Funding *Ryan O'Donnell*: Supported by NSF grant CCF-1717606. This material is based upon work supported by the National Science Foundation under the grant number listed above. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

Acknowledgements The authors would like to thank Scott Aaronson and Patrick Rall for helpful communications.

1 Introduction

1.1 Grover Search recap

A famous, textbook algorithm in quantum computing is *Grover Search* [6], which solves the following task: Given is a quantum oracle for a function $f : [N] \to \{0,1\}$, where queries for $f(x)$ may be made in quantum superposition. It is promised that $K = \#\{x : f(x) = 1\}$ is exactly 1. The task is to find x^* such that $f(x^*) = 1$. Grover Search solves this problem (with high probability) using $O(\sqrt{N})$ queries.

The algorithm is particularly simple: First, a state $|s\rangle$ equal to the uniform superposition over all $|x\rangle$ is prepared; this state makes an angle of $\arccos\sqrt{1/N}$ with $|x^*\rangle$. We write $|x^*\rangle^\perp$ for the state perpendicular to $|x^*\rangle$ making an angle of $\theta^* = \arcsin\sqrt{1/N}$ with $|s\rangle$. Then the algorithm repeatedly performs *Grover iterations*, each of which consists of one query followed by the simple "Grover diffusion" operation. The effect of a Grover iteration is to rotate $|s\rangle$ by an angle of $2\theta^*$; thus after r rotations the angle of $|s\rangle$ from $|x^*\rangle^\perp$ is

[1] The ordering of the authors was randomized. The authors contributed equally.

$(2r+1)\theta^*$. Setting $r \approx \frac{\pi}{4}\sqrt{N}$, the algorithm makes $O(\sqrt{N})$ queries and ends up with a state at angle approximately $\frac{\pi}{2}$ from $|x^*\rangle^\perp$; measuring then results in $|x^*\rangle$ with probability $\sin^2((2r+1)\theta^*) \approx \sin^2\frac{\pi}{2} = 1$.

In this form the algorithm relies on the assumption $K = 1$. For $K \geq 1$, the only change is that the parameter θ^* becomes $\arcsin\sqrt{K/N}$. Thus if the correct value of K is known to the algorithm (and we assume for simplicity that $K \leq N/2$), it can choose $r \approx \frac{\pi}{4}\sqrt{N/K}$ and solve the search problem using $O(\sqrt{N/K})$ iterations/queries. This algorithm also works if the algorithm knows an estimate K' of K that is correct up to small multiplicative error; say, $K' \overset{1.1}{\approx} K$. Here we are using the following notation:

▶ **Notation 1.** For $a, b, \eta > 0$ we write $a \overset{1+\eta}{\approx} b$ if $\frac{1}{1+\eta} \leq a/b \leq 1+\eta$.

If K is *unknown* to the algorithm, one possibility is try all estimates $K' = 1,\ 1.1,\ 1.1^2,\ 1.1^3,\ \ldots,\ N/2$. The total number of Grover iterations (hence queries) will be $O(\sum_i \sqrt{N/1.1^i}) = O(\sqrt{N})$.[2] A strategy with improved query complexity for large K was given by Boyer, Brassard, Høyer, and Tapp [2]; in brief, for $i = 0, 1, 2, \ldots$ it tries a random number of rotations in the range $[1, 1.1^i]$, stopping if ever an $x \in f^{-1}(1)$ is found. This algorithm solves the search problem using $O(\sqrt{N/K})$ iterations, despite not knowing K in advance.

1.2 Approximate Counting

A natural related problem, called *Approximate Counting* and introduced in [2], is to *estimate* K. More precisely, given as input a parameter $\epsilon \geq 0$, the task is to output a number $\widehat{K}$ such that (with high probability) $\widehat{K} \overset{1+\epsilon}{\approx} K$. To keep the exposition simple, in this paper we will make the following standard assumptions:

- $1/N \leq \epsilon \leq 1$ (setting $\epsilon = 1/N$ just yields the problem of exact counting – any smaller value of ϵ does not change the problem);
- $K \leq N/2$ (otherwise there is generally a dependence on $N - K$, since one can switch the roles of 0 and 1 in f's output);
- $K \neq 0$ (generally, all algorithms can easily be extended to work in the case of $K = 0$, with query complexity being the worst-case query complexity over all $K > 0$).

The quantum Approximate Counting problem was solved with optimal query complexity by Brassard, Høyer, Mosca, and Tapp [3]. Combining quantum Fourier transform ideas from Shor's Algorithm with the ideas behind Grover Search, their algorithm solves Approximate Counting using $O(\sqrt{N/K}\,/\,\epsilon)$ queries.

Let us make some remarks about this query complexity. First, note that the bound takes K into account, even though K is (initially) unknown to the algorithm. Second, although K could be as small as 1, the worst-case query complexity over all K need *not* be $\Omega(\sqrt{N}\,/\,\epsilon)$. (Indeed, this would lead to an illogical query complexity of $\Omega(N^{3/2})$ if one set $\epsilon = 1/N$ to do exact counting.) Instead, note that an algorithm can first run the algorithm from [3] with $\epsilon = 1$, expending $O(\sqrt{N/K}) \leq O(\sqrt{N})$ queries and learning a preliminary estimate $K' \overset{2}{\approx} K$. Now since K is an integer, there is no point in trying to approximate it to a factor better than $1 + 1/K$, hence better than $1 + 1/(2K')$. Thus the algorithm can now raise the initial input ϵ to $1/(2K')$ if necessary, and *then* run [3] to obtain its final estimate. This yields a final query complexity of

$$O(\sqrt{N/K}\,/\max\{\epsilon, 1/K\}) = \min\{O(\sqrt{NK}), O(\sqrt{N/K}\,/\,\epsilon)\} \leq O(\sqrt{N/\epsilon})$$

[2] One must take a small amount of care to bound the overall failure probability without incurring a log factor.

(where in the last inequality we took the geometric mean). This K-independent bound of $O(\sqrt{N/\epsilon})$ is logical: the smallest ϵ one should ever take is $\epsilon = 1/N$, and this leads to a query complexity of $O(N)$ for the general case of exact counting. By similar reasoning one can obtain the more precise fact that exact counting can done with $O(\sqrt{NK})$ queries. Finally, we remark that query complexity obtained by [3] was shown to be optimal by Nayak and Wu [8].

Let us also briefly mention the quantum *Amplitude Estimation* problem, which is essentially the same as the Approximate Counting problem except that the "initial angle" θ^* need not be of the form $\arcsin\sqrt{K/N}$ for some integer K, but can be any value. The solution to the Amplitude Estimation problem in [3] is a widely used tool in quantum algorithm design, and leads to quadratic speedups over classical algorithms for a variety of statistical problems.

1.3 Simpler and nonadaptive?

Although the Approximate Counting algorithm from [3] has optimal query complexity, there has recently been a lot of interest in simplifying it [10, 9, 1]. In particular the latter two of these just-cited works strove to replace it with an algorithm that *only* uses Grover iterations, both for analytic simplicity and practical simplicity (the controlled amplifications of [3] being particularly problematic for NISQ devices). The work of Aaronson and Rall [1] provably succeeds at this challenge, providing an algorithm that solves the Approximate Counting problem using $O(\sqrt{N/K}\,/\,\epsilon)$ Grover iterations (hence queries). Briefly, the Aaronson–Rall algorithm has a first phase (somewhat similar to the algorithm in [2]) that performs a geometrically increasing sequence of Grover iterations until K can be estimated up to a constant factor of 1.1 (see Theorem 9 herein). This requires $O(\sqrt{N/K})$ iterations. In the second phase, their algorithm performs a kind of binary search to improve the approximation factor to $1+\epsilon$; each step of the binary search requires additional Grover iterations, totalling $O(\sqrt{N/K}\,/\,\epsilon)$ in the end.

From the point of view of practicality and simplicity, there is a downside to the Aaronson–Rall algorithm, which is that its Grover iterations are *adaptive* (especially in the second phase of the algorithm). In other words, the steps of the algorithm involve many repetitions of the following: performing some Grover iterations, measuring, and doing some classical computation to decide how many Grover iterations to do in the next step. It has been argued that this repeated switching between quantum and classical computation could be undesirable in practice. Indeed, the final open question in [1] concerned the optimal query complexity of Approximate Counting using *nonadaptive* Grover iterations. This version of the problem was also stressed and studied in [9], but without any provable guarantees being provided.

Other recent developments in the area of approximate counting include [5, 7], which propose variants of the algorithm from [1] but with improved constant factors. As well, the work [4] proves a lower bound for the query complexity of approximate counting in the parallel case.

1.4 Our results

We investigate the problem of Approximate Counting using only nonadaptive Grover iterations. Note that for this version of the problem, there is no hope of obtaining the query complexity $O(\sqrt{N/K}\,/\,\epsilon)$ that improves as a function of K. To see this, suppose even that ϵ is fixed to 1. If the algorithm is to achieve query complexity $O(\sqrt{N/K})$, then it must be able to achieve $O(1)$ query complexity when $K = \Theta(N)$. Since it is nonadaptive, this means it must *always* make only $O(1)$ queries. But this is impossible, as even for *adaptive* algorithms it is known that $\Omega(\sqrt{N})$ queries are required in the case of $K = O(1)$, $\epsilon = 1$.

In other words, with nonadaptive algorithms we can only hope to achieve the optimal query complexity that is independent of K, namely $O(\sqrt{N/\epsilon})$. In this work we indeed show this is achievable. Our main theorem is:

▶ **Theorem 2.** *There is an algorithm for quantum Approximate Counting that uses only* nonadaptive *Grover iterations, and that has a query complexity of* $O(\sqrt{N/\epsilon})$ *(and minimal additional computational overhead).*

We also briefly sketch an extension of our algorithm achieving improved query complexity in the setting where we are allowed multiple rounds of nonadaptive Grover iterations (as opposed to just one).

2 Preliminaries

We will assume throughout that $K \leq 2^{-20}N$.[3] This without loss of generality since we may artificially replace N by $2^{20}N$ and extend f to $f : [2^{20}N] \to \{0,1\}$, with $f(x) = 0$ for $x > N$. We fix

$$\theta^* = \arcsin\sqrt{K/N},$$

sometimes called the "Grover angle". Recall that a query algorithm based on Grover iterations has the following property: At the cost of $q \in \mathbb{N}$ "queries", it can "flip a coin with bias $\sin^2((2q+1)\theta^*)$". By repeating this t times, it can obtain t independent flips of this coin. It is statistically sufficient to retain only the average of the coin flip outcomes, which is a random variable distributed as $\frac{1}{t}\mathrm{Bin}(t, \sin^2((2q+1)\theta^*))$, where "Bin" denotes a binomial distribution. These observations lead to the following:

▶ **Notation 3.** For real $r \geq 1$ we write $\lfloor\!\lfloor r \rfloor\!\rfloor$ for the largest odd integer not exceeding r, and we write $p(r) = \sin^2(\lfloor\!\lfloor r \rfloor\!\rfloor\theta^*)$.

▶ **Definition 4.** *A* Grover schedule *consists of two sequences:* $R = (r_1, \ldots, r_m)$ *(each real* $r_i \geq 1$*) and* $T = (t_1, \ldots, t_m)$ *(each* $t_i \in \mathbb{N}^+$*).* Performing *this Grover schedule refers to obtaining independent random variables* $\widehat{\boldsymbol{p}}_1, \ldots, \widehat{\boldsymbol{p}}_m$*, where* $\widehat{\boldsymbol{p}}_i$ *is distributed as* $\frac{1}{t_i}Bin(t_i, p(r_i))$.

A nonadaptive Grover iteration algorithm for Approximate Counting is simply an algorithm that performs one fixed Grover schedule, and produces its final estimate $\widehat{K}$ by classically post-processing the results. One can more generally study algorithms with "s rounds of nonadaptivity"; this simply means that s Grover schedules are used, but they may be chosen adaptively.

▶ **Fact 5.** Performing the Grover schedule R, T uses at most $\frac{1}{2}\sum_i r_i t_i$ queries.

2.1 On how well we need to approximate θ^*

We will use the following elementary numerical fact:

▶ **Lemma 6.** *Suppose for real* $0 \leq k, k' \leq N$ *and* $\eta \leq 1$ *that* $\arcsin\sqrt{k'/N} \overset{1+\eta}{\approx} \arcsin\sqrt{k/N}$. *Then* $k' \overset{1+3\eta}{\approx} k$.

[3] Our work would be fine with, say, $K \leq N/8$, but we put 2^{-20} so as to be able to cite [1] as a black box.

This lemma helps us show that approximating θ^* well is equivalent to approximating K well:

▶ **Proposition 7.** *Suppose* $\theta \overset{1+\epsilon/6}{\approx} \theta^*$. *If* $\kappa' \in \mathbb{R}$ *satisfies* $\theta = \arcsin\sqrt{\kappa'/N}$, *and* K' *is the nearest integer to* κ', *then* $K' \overset{1+\epsilon}{\approx} K$.

Proof. Since $\theta \overset{1+\epsilon/6}{\approx} \theta^*$, Lemma 6 tells us that $\kappa' \overset{1+\epsilon/2}{\approx} K$, and hence

$$|\kappa' - K| \leq (\epsilon/2)K. \tag{1}$$

We also have $|\kappa' - K'| \leq 1/2$, and hence $|K' - K| \leq (\epsilon/2)K + 1/2$. But we can assume $(\epsilon/2)K \geq 1/2$, as otherwise Inequality (1) implies $|\kappa' - K| < 1/2$ and hence $K' = K$. Thus $|K' - K| \leq (\epsilon/2)K + (\epsilon/2)K = \epsilon K$; i.e., $K' \overset{1+\epsilon}{\approx} K$. ◀

▶ **Lemma 8.** *Given some* $\theta' \overset{1.11}{\approx} \theta^*$, *estimating* θ^* *to a factor of* $1+1/(2N\sin^2\theta')$ *is sufficient to estimate* θ^* *to a factor of* $1+\epsilon/6$.

Proof. From Lemma 6, $1 + 1/(2N\sin^2\theta') \leq 1 + 1.33/(2K) < 1 + 1/K$. The closest possible values to K are $K-1$ and $K+1$; therefore, estimating K within a factor of $1+1/K$ is the same as estimating K exactly. This is at least as good as estimating K to within a factor of $1+\epsilon/6$. ◀

3 The nonadaptive algorithm

Our algorithm can conceptually be thought of as having two stages: the first stage estimates θ^* to a constant factor, and the second stage improves this estimate to the desired factor of $1+\epsilon$. This two-stage approach is similar in flavor to the algorithms in [1, 2]. However we note that, consistent with our nonadaptivity condition, the two stages in our algorithm can be run in parallel.

For the first stage of our algorithm, we require the following result of Aaronson and Rall [1], which estimates θ^* up to a factor of 1.1, using $O(\sqrt{N})$ nonadaptive queries. (In fact, as Aaronson and Rall show, the obvious adaptive version of the algorithm incurs only $O(\sqrt{N/K})$ queries.)

▶ **Theorem 9.** *Let* $R = (1, (12/11), (12/11)^2, \ldots, (12/11)^m)$, *where* $m = \Theta(\log N)$ *is minimal with* $(12/11)^m \geq \sqrt{N}$. *Let* T *consist of* m *copies of* $\lceil 10^5 \ln(120/\delta)\rceil$. *Perform the Grover schedule* R, T. *(By Fact 5 this incurs* $O(\sqrt{N})$ *queries.) Then except with a failure probability of at most* $\delta/2$, *we can obtain* $\widetilde{\theta} \overset{1.1}{\approx} \theta^*$ *by doing the following: take the minimal value of* t *such that* $\widehat{\boldsymbol{p}}_t \geq 1/3$, *and set* $\widetilde{\theta} = (5/8)(11/12)^t$.

The second stage of our algorithm uses the following critical lemma, which we will prove in Section 4.

▶ **Lemma 10.** *Given the parameters* θ', ϵ', *and* δ', *there is an algorithm using only nonadaptive Grover iterations that performs* $O(\log(1/\delta')/(\theta'\epsilon'))$ *queries, and outputs a result* θ_{est} *with the following guarantee: if* $\theta' \overset{1.11}{\approx} \theta^*$, *then* $\theta_{\text{est}} \overset{1+\epsilon'/6}{\approx} \theta^*$ *except with probability at most* $\delta/2$.

In this section, we will show how to use Theorem 9 and Lemma 10 to prove Theorem 2. We will now state our algorithm:

Algorithm 1 Outline of the full algorithm.

1. Run the Aaronson–Rall algorithm from Theorem 9, allowing us to later compute $\widetilde{\theta}$.
2. For $\theta = \arcsin(\sqrt{1/N}), 1.001 \arcsin\sqrt{1/N}, (1.001)^2 \arcsin\sqrt{1/N}, \ldots, 1.1 \arcsin(2^{-20})$:
   Perform the algorithm in Lemma 10 with the parameters $\theta' = \theta$, $\epsilon' = \max(\epsilon, \frac{1}{2N\sin^2\theta'})$, and $\delta' = \delta/2$.
3. Classical Post-processing: Among all iterations in the for-loop, take the iteration with the value of θ that was closest to $\widetilde{\theta}$, and output the result of that iteration.

Each iteration of the for loop in Step 2 can be done in parallel (there are no computational dependencies between the iterations), and Step 1 can also be done in parallel with Step 2. Therefore, the algorithm uses only nonadaptive Grover iterations. Note also that we can write the quantum parts of steps 1 and 2 as one fixed Grover schedule, with the classical parts and step 3 forming the post-processing step; however, it will be more convenient in this section to think about these as individual steps in a logical sequence.

▶ **Proposition 11.** *Algorithm 1 returns a value θ_{est} such that $\theta_{\text{est}} \overset{1+\epsilon}{\approx} \theta^*$, except with probability at most δ.*

Proof. By Theorem 9, Step 1 returns an estimate $\widetilde{\theta}$ such that $\widetilde{\theta} \overset{1.1}{\approx} \theta^*$, with a failure probability of at most $\delta/2$. The value of θ (in Step 2) that is closest to $\widetilde{\theta}$ is at most a factor of 1.001 away from $\widetilde{\theta}$ (note that $\widetilde{\theta}$ is at most $1.1\arcsin(2^{-20})$ by our assumption that $\theta^* \leq 2^{-20}$). If Step 1 succeeded, this is at most a factor of $1.1 \times 1.001 < 1.11$ away from θ^*. By Lemma 10, the algorithm outputs an estimate θ_{est} such that $\theta_{\text{est}} \overset{1+\epsilon'/6}{\approx} \theta^*$, with a failure probability of at most $\delta/2$. Lemma 8 then implies that $\theta_{\text{est}} \overset{1+\epsilon/6}{\approx} \theta^*$. Using Proposition 7 (setting the parameter $\theta = \theta_{\text{est}}$), we get an estimate K_{est} such that $K_{\text{est}} \overset{1+\epsilon}{\approx} K$. By the union bound, the overall failure probability of the algorithm is at most δ. ◀

▶ **Proposition 12.** *Algorithm 1 makes $O(\sqrt{N/\epsilon}\log(1/\delta))$ queries.*

Proof. First, consider all iterations where $2N\sin^2(\theta) \leq 1/\epsilon$. In these cases, the query complexity given by Lemma 10 would be $O(\log(1/\delta)(N\sin^2(\theta))/\theta) = O(N\theta\log(1/\delta))$.

The query complexity associated with these iterations is a geometric series with a constant common ratio of 1.01 where the largest term is $O(\sqrt{N/\epsilon}\log(1/\delta))$. Therefore the overall query complexity due to these iterations is $O(\sqrt{N/\epsilon}\log(1/\delta))$.

Now consider all iterations where $2N\sin^2(\theta) > 1/\epsilon$. In these cases, the query complexity is $O(\log(1/\delta)/(\theta\epsilon))$. This forms a geometrically decreasing series (with a constant common ratio), where the first term is again $O(\sqrt{N/\epsilon}\log(1/\delta))$. The overall query complexity contributed by these schedules is thus also $O(\sqrt{N/\epsilon}\log(1/\delta))$.

Therefore, the query complexity of Algorithm 1 is $O(\sqrt{N/\epsilon}\log(1/\delta))$, as claimed. ◀

Having proven Proposition 11 and Proposition 12, we have established our main result Theorem 2 modulo the proof of Lemma 10, which will appear in the next section. Before giving this, we briefly sketch how our algorithm can also be extended to the setting of being allowed multiple rounds of nonadaptive Grover iterations. If we have two such rounds of nonadaptivity, we can first run step 1 of our algorithm to get a constant-factor approximation, and then based on its result run the algorithm in Lemma 10; this achieves a query complexity of $O(\sqrt{N} + \min(\sqrt{N/\epsilon}, \sqrt{N/K}/\epsilon))$. This nearly matches the query complexity of the fully adaptive case, but for the $\sqrt{N}$ term due to the first step. Given more rounds of nonadaptivity,

we can reduce the cost of this first step by staging it over multiple initial rounds. One can show that with $O(\log N)$ rounds of nonadaptivity, this will yield the optimal query complexity corresponding to the fully adaptive case.

4 Proving Lemma 10

Our goal in this section will be to prove Lemma 10. Assume that we are given some θ', ϵ', and δ'. We will show a nonadaptive Grover iteration algorithm making $O(\log(1/\delta')/(\theta'\epsilon'))$ queries with the property that if $\theta' \overset{1.11}{\approx} \theta^*$, then its output will be a factor-$(1+\epsilon')$ approximation of θ^* (except with failure probability at most δ'). For the remainder of the section, we will assume that we are in the interesting case where $\theta' \overset{1.11}{\approx} \theta^*$ (in the other cases, the algorithm does not need to output a correct answer).

4.1 Proof idea

The algorithm for Lemma 10 is structured exactly as described in Definition 4 and Fact 5; there is an initial nonadaptive quantum part with a fixed Grover schedule (that we will later define), and a classical post-processing step at the end that uses the results of the quantum part to estimate θ^*.

Before stating the key ideas in the quantum part of our algorithm, we mention the "Rotation Lemma" of Aaronson and Rall [1, Lem. 2]. The main idea in that lemma can be roughly stated as follows: given that θ^* lies in some range $[\theta_{\min}, \theta_{\min} + \Delta\theta]$, we can pick an odd integer value of r (where $r = O(1/(\theta \cdot \Delta\theta))$), such that $r\theta_{\min}$ is close to $2\pi k$ and $r(\theta_{\min} + \Delta\theta)$ is close to $2\pi k + \pi/2$. If θ is close to $\theta_{\min}$, $p(r)$ will be nearly 0 (and if it is close to $\theta_{\min} + \Delta\theta$, it will be nearly 1). Aaronson and Rall use this lemma to continually shrink the possible range that θ^* could lie in by a geometric factor at each iteration, until the range is $1 \pm \epsilon$.

We will adopt a similar idea to find an efficient Grover schedule that can distinguish any two candidate angles with high probability; we do this by relaxing the condition of one angle being close to $2\pi k$ and the other being at distance $\pi/2$ from it. Instead, we choose the sequence R in our Grover schedule such that for any pair of values θ_1, and θ_2, there is some $r \in R$ such that $r\theta_1$ and $r\theta_2$ differ by approximately $\pi/8$, and are also "in the same quadrant" (meaning the same interval $[0, \frac{\pi}{2})$, $[\frac{\pi}{2}, \pi)$, $[\pi, \frac{3\pi}{2})$, $[\frac{3\pi}{2}, 2\pi)$ modulo 2π). This relaxation allows us to save on the total number of queries made by reusing the same value of r to distinguish many pairs of candidate angles. Due to this, the nonadaptivity requirement does not make the query complexity grow polynomially larger (whereas, for example, naively simulating the search tree from [1] in a nonadaptive fashion would incur an extra $1/\epsilon'$ factor).

The classical post-processing involves running a "tournament" between all candidate estimates of θ^*, which outputs the winning value as the estimate. This post-processing step can be implemented efficiently in $O(\log(1/\epsilon')/\epsilon')$ classical time.

4.2 Some arithmetic lemmas

We now define some useful sequences and prove a couple of arithmetic lemmas about them.

▶ **Definition 13.** *Define the sequence u by $u_0 = 1$, $u_1 = 1.01$, $\ldots$, $u_L = 1.01^L$ where $L = O(\log(1/(\theta'\epsilon')))$ is minimal with $u_L \geq 1.2\pi/(\theta'\epsilon')$.*

▶ **Lemma 14.** *Suppose we are given θ_0 and θ_1 such that $\theta_0 < \theta_1$, $\theta_0 \overset{1.11}{\approx} \theta'$, $\theta_1 \overset{1.11}{\approx} \theta'$, and $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$. Write $\eta = \theta_1 - \theta_0$. Then there exists some $0 \le i \le L$ such that $u_i\eta \overset{1.01}{\approx} \frac{\pi}{8}$.*

Proof. We know that $u_0\eta = \eta \le \theta_1 \le 1.1 \times 0.0001 < \frac{\pi}{8}$. We also have $u_L\eta \ge 1.2\pi\eta/(\theta'\epsilon') \ge 1.2\pi\eta/(1.1\theta_0\epsilon') \ge 1.2\pi/(1.1 \cdot 6.1) > \frac{\pi}{8}$ where we used $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$ in the second-to-last inequality. The lemma now follows from the geometric growth of the u_i's with ratio 1.01. In particular, $i = \left\lfloor \log_{1.01}(\frac{\pi}{8\eta}) \right\rfloor$ works. ◀

For the u_i given by Lemma 14, we have $u_i\theta_1 - u_i\theta_0 \approx \frac{\pi}{8}$. This seems promising, in that the "coin probabilities" associated to these angles, namely $\sin^2(u_i\theta_1)$ and $\sin^2(u_i\theta_0)$, seem as though they should be far apart. Unfortunately, something annoying could occur; it could be that these angles are, say, $100\pi \pm \frac{\pi}{16}$, in which case the coin probabilities would be identical. As mentioned in Section 4.1, what we would *really* like is to have the two angles be far apart but also in the same quadrant. To achieve this, we will define a new sequence.

▶ **Definition 15.** *For each $0 \le i \le L$, define $a_{i,0} = 0$, $a_{i,1} = \frac{\pi}{4.8\theta'}$, $a_{i,2} = 1.01 \cdot \frac{\pi}{4.8\theta'}$, $a_{i,3} = 1.01^2 \cdot \frac{\pi}{4.8\theta'}$, $\ldots$, $a_{i,C+1} = 1.01^C \cdot \frac{\pi}{4.8\theta'}$, where $C = \lceil 2\log_{1.01}(1.2) \rceil$ is a constant. Also define $s_{i,j} = u_i + a_{i,j}$.*

▶ **Lemma 16.** *In the setting of Lemma 14, there exists some $0 \le j \le C+1$ such that*

$$s_{i,j}\eta \overset{1.5}{\approx} \frac{\pi}{8}$$

and such that $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ are in the same quadrant.

Proof. We first apply Lemma 14 and obtain

$$u_i\eta \overset{1.01}{\approx} \frac{\pi}{8}. \tag{2}$$

Now if $u_i\theta_0$ and $u_i\theta_1$ are already in the same quadrant then we can take $j = 0$ (implying $s_{i,j} = u_i$) and we are done. Otherwise, the plan will be to find $j > 0$ with $a_j\theta_0 \approx \frac{\pi}{4}$, thus shifting them to $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ that still differ by roughly $\frac{\pi}{8}$ but which now must be in the same quadrant.

To find the required j, observe that on one hand, $a_1\theta_0 = (\pi/(4.8\theta')) \cdot \theta_0 \le 1.11\pi/4.8 \le \frac{\pi}{4}$. On the other hand, $a_{i,C+1}\theta_0 \ge 1.2^2 \cdot (\pi/(4.8\theta')) \cdot \theta_0 \ge (1.2/1.1) \cdot \pi/4 \ge \frac{\pi}{4}$. By the geometric growth of the $a_{i,j}$'s with ratio 1.01, we conclude that there exists some $1 \le j \le \ell_i$ achieving

$$a_{i,j}\theta_0 \overset{1.05}{\approx} \frac{\pi}{4}. \tag{3}$$

We may now make several deductions. First,

$$\theta_0 \overset{1.1}{\approx} \theta_t, \theta_1 \overset{1.1}{\approx} \theta_t \implies a_{i,j}\theta_0 \overset{1.22}{\approx} a_{i,j}\theta_1 \implies a_{i,j}\eta \le .22 \cdot a_{i,j}\theta_0 \le .22 \cdot 1.05\frac{\pi}{4} \le .24\frac{\pi}{4}.$$

Combining this with Inequality (2) we conclude

$$s_{i,j}\eta = u_i\eta + a_{i,j}\eta \in [\tfrac{1}{1.01}\tfrac{\pi}{8}, \tfrac{\pi}{8}(1.01) + .24\tfrac{\pi}{4}] \overset{1.5}{\approx} \frac{\pi}{8}. \tag{4}$$

Thus we started with $u_i\theta_0$ and $u_i\theta_1$ differing by $\frac{\pi}{8}$ (up to factor 1.01) but in different quadrants; by passing to $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$, we have offset $u_i\theta_0$ by $\frac{\pi}{4}$ (up to factor 1.05, Inequality (3)) and the two angles still differ by around $\frac{\pi}{8}$ (up to factor 1.5, Inequality (4)). Thus $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ are in the same quadrant and the proof is complete. ◀

4.3 The algorithm

We can now describe our "second stage" algorithm, which simply runs the Grover schedule G defined as follows.

▶ **Definition 17.** *The Grover schedule G comprises the sequence $R = (s_{i,j})_{i=0\ldots L,\ j=0\ldots C+1}$ and $T = (\lceil A \log_2(1/(\delta'\theta'\epsilon' u_i))\rceil)_{i=0\ldots L,\ j=0\ldots C+1}$. Here A is a universal constant to be chosen later.*

Note that the $T_{i,j}$ values we use are exactly the number of coin flips used in the second stage of the algorithm in [1]. Like in their algorithm, this choice of values allows us to avoid stray $\log(1/\epsilon)$ or $\log\log(1/\epsilon)$ factors in the overall query complexity.

▶ **Proposition 18.** *Performing the Grover schedule G takes at most $O(\log(1/\delta')/(\theta'\epsilon'))$ queries.*

Proof. Using Fact 5, the query complexity of performing G is $\sum_{i=0}^{L} \log(1/(\delta'\theta'\epsilon' \cdot 1.01^i))1.01^i$ (up to constant factors). This is $\log(1/\delta)\sum_{i=0}^{L} 1.01^i + \sum_{i=0}^{L} \log(1/(\theta'\epsilon' \cdot 1.01^i))1.01^i$. Noting that $1.01^L = O(1/(\theta'\epsilon'))$, the first term is clearly $O(\log(1/\delta)/(\theta'\epsilon'))$ and the second term, up to a constant factors, is $\sum_{i=0}^{L}(L-i)1.01^i = O(1.01^L) = O(1/(\theta'\epsilon'))$. Therefore, the overall query complexity is $O(\log(1/\delta)/(\theta'\epsilon'))$ as desired. ◀

▶ **Remark.** The above calculation mirrors the one done for stage 2 of the adaptive algorithm in [1]; this is expected because both algorithms use the same number of "coin flips" per coin ($T'_{i,j}$ values), as mentioned above.

It now remains for us to show how to approximate θ^* (with high probability) using the data collected from this Grover schedule.

4.4 Completing the algorithm

We first prove a lemma showing that we can distinguish between any pair of angles (that are not already sufficiently close to each other) by using the ideas developed in Lemma 14 and Lemma 19.

▶ **Lemma 19.** *There is an $O(1)$-time classical deterministic algorithm that, given*

- *$\theta_0 \overset{1.1}{\approx} \theta'$, $\theta_1 \overset{1.1}{\approx} \theta'$, such that $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$*
- *the data collected by the Grover schedule G,*

outputs either "reject θ_0" or "reject θ_1". Except with failure probability at most $c\theta'\delta'\epsilon'/|\theta_1 - \theta_0|$ (where $c > 0$ is a constant to be chosen later), the following is true:

For $b = 0, 1$, if $\theta^* \overset{1+.001\epsilon}{\approx} \theta_b$, then the algorithm does not output "reject θ_b".

Proof. The algorithm computes the (i,j) pair promised by Lemma 16, such that $s_{i,j}\theta_0$ and $s_{i,j}\theta_1$ are in the same quadrant and such that $|s_{i,j}\theta_1 - s_{i,j}\theta_0| \overset{1.5}{\approx} \frac{\pi}{8}$. Letting $q_b = \sin^2(s_{i,j}\theta_b)$ for $b = 0, 1$, it follows from the assumptions in the preceding sentence that $|q_0 - q_1| \geq .04$. The algorithm may now select a threshold $q' \in [.01, .99]$ such that (without loss of generality) $q_0 \leq q' - .01$ and $q_1 \geq q' + .01$.

The algorithm will use just the coin flips from the $s_{i,j}$ part of the schedule; these coin flips have bias $p(s_{i,j}) = \sin^2(\llbracket s_{i,j} \rrbracket \theta^*)$. More precisely, the algorithm will output "reject θ_0" if $\widehat{\boldsymbol{p}}_{i,j} > q'$ and "reject θ_1" if $\widehat{\boldsymbol{p}}_{i,j} \leq q'$. We need to show that if $\theta^* \overset{1+.001\epsilon}{\approx} \theta_0$ then the algorithm outputs "reject θ_0" with probability at most $c\theta'\delta'\epsilon'/|\theta_1 - \theta_0|$. (The case when $\theta^* \overset{1+.001\epsilon}{\approx} \theta_1$ is analogous.)

Now if $\theta^* \overset{1+.001\epsilon}{\approx} \theta_0$, then $\llbracket s_{i,j} \rrbracket \theta^* \overset{1+.001\epsilon}{\approx} \llbracket s_{i,j} \rrbracket \theta_0$. It follows that

$$\left| \llbracket s_{i,j} \rrbracket \theta^* - \llbracket s_{i,j} \rrbracket \theta_0 \right| \leq .001\epsilon \cdot \llbracket s_{i,j} \rrbracket \theta_0 \leq .001 s_{i,j} \cdot \epsilon \theta_0.$$

But we know that

$$\tfrac{\pi}{8} \overset{1.5}{\approx} |s_{i,j}\theta_1 - s_{i,j}\theta_0| = s_{i,j}|\theta_1 - \theta_0| \geq s_{i,j} \cdot (\epsilon/6.1)\theta_0,$$

the last inequality because $\theta_0 \overset{1+\epsilon/6.1}{\not\approx} \theta_1$. Combining the above two deductions yields

$$\left| \llbracket s_{i,j} \rrbracket \theta^* - \llbracket s_{i,j} \rrbracket \theta_0 \right| \leq .001 \cdot 1.5 \cdot 6.1 \cdot \tfrac{\pi}{8} \leq .004.$$

Moreover, $\llbracket s_{i,j} \rrbracket \theta_0$ and $s_{i,j}\theta_0$ differ by at most $2\theta_0 \leq .0002 < .001$. Thus we finally conclude

$$\left| \llbracket s_{i,j} \rrbracket \theta^* - s_{i,j}\theta_0 \right| \leq .005 \quad \Longrightarrow \quad \left| p(s_{i,j}) - q_0 \right| \leq .005 \quad \Longrightarrow \quad p(s_{i,j}) < q' - .005$$

(the first implication using that $\sin^2$ is 1-Lipschitz). Then, using a Chernoff bound, we have that $\widehat{\boldsymbol{p}}_{i,j} > q'$ with probability at most $c\delta'\theta'\epsilon' u_i$, where c is a constant that depends on A (as A increases, c decreases). From Lemma 14, we know that $u_i \overset{1.01}{\approx} \pi/(8|\theta_1 - \theta_0|)$. Then, assuming the constant A is chosen large enough as a function of c, we indeed have that $\widehat{\boldsymbol{p}}_{i,j} > q'$ with probability at most $c\theta'\delta'\epsilon'/|\theta_1 - \theta_0|$. ◀

4.4.1 Description of the post-processing algorithm

We have developed the necessary tools to describe and justify the classical post-processing algorithm.

Fix values $\theta_i = \theta'(1 + .001\epsilon')^i$ for $-V \leq i \leq V$, where $V = O(1/\epsilon')$ is a minimal integer such that $(1 + .001\epsilon')^V \geq 1.11$. We will refer to the θ_i's as "nodes". We may also assume that the number of nodes is a power of 2 for convenience (while there are $2|V| + 1$ nodes, we can always pad these actual nodes with some dummy nodes to reach the nearest power of 2).

The main idea is to repeatedly use Lemma 19 to run a "tournament" amongst all nodes. The tournament is structured as a series of "rounds". In a given round, suppose we start off with n nodes. Sort the nodes in order of the angles they correspond to. Now, pair up node 1 with node $n/2 + 1$, node 2 with node $n/2 + 2$, and so on, until node $n/2$ is paired up with node n. For each pair of nodes, use Lemma 19 to choose a winner to go to the next round. Note that it is possible that two nodes that are matched in the tournament do not satisfy the pre-condition of Lemma 19 because they are within a factor of $(1 + \epsilon/6.1)$ of each other – we call these "void" match-ups. We will call the part of the tournament we have described so far the first phase – when we see a void match-up, we stop this first phase and enter the second phase. (Note that if we never see any void match-ups and there is only one node left, we do not need the second phase and can directly output the remaining node as our estimate.)

When the first phase ends, take all remaining nodes and enter the second phase. In this phase, match up every pair of remaining nodes, and for every pair that does not form a void match-up, eliminate one of the nodes using Lemma 19. At the end of this, output any one of the remaining un-eliminated nodes (if there are none, then the program can output an arbitrary node - this is a failure condition and we will show that, with high probability, such failure conditions will not happen).

In our algorithm, we have arranged for $\theta_{-V} \leq \theta^* \leq \theta_V$, and hence there exists a node θ_{i^*} such that $\theta_{i^*} \overset{1+.001\epsilon}{\approx} \theta^*$. We will proceed to bound the overall failure probability of the algorithm by the probability that this node loses any match-up it is a part of.

▶ **Lemma 20.** *If θ_{i^*} never loses any match-up it is a part of, then the tournament outputs an estimate θ_{est} such that $\theta_{\text{est}} \overset{1+\epsilon/6}{\approx} \theta^*$.*

Proof. Suppose that θ_{i^*} never loses any match-up it is a part of. If the tournament ends in the first phase itself, then the algorithm will output θ_{i^*}, which is correct. If the tournament ends in the second phase, then the only possible other nodes that we could output are the ones that θ_{i^*} did not play, which can be at most a factor of $1+\epsilon/6.1$ away from it. Therefore, these nodes are also at most a factor of $(1+0.001\epsilon)(1+\epsilon/6.1) \leq 1+\epsilon/6$ away from θ^*. ◀

▶ **Lemma 21.** *The probability that θ_{i^*} loses any match-up in the first phase of the tournament can be upper-bounded by $c\delta'$ times a constant factor (where c is the constant from Lemma 19).*

Proof. Consider an arbitrary round in the first phase of the tournament, where we have 2^j nodes remaining for some j. By how we chose the match-ups, we know that every two angles that are matched up in that round must be at least a factor of $(1+0.001\epsilon')^{2^{i-1}}$ apart. This implies that their absolute difference is at least $(\theta'/1.1)\cdot((1+0.001\epsilon')^{2^{i-1}}-1)$. Then, by Lemma 19, the failure probability of any match-up in that round is at most $1.1c\delta'\epsilon'/((1+0.001\epsilon')^{2^{i-1}}-1)$.

Suppose the tournament begins with n nodes. We can use the union bound to upper-bound the probability that θ_{i^*} loses in any of the at most $\log_2 n$ rounds by

$$\sum_{j=1}^{\log_2 n} \frac{1.1c\delta'\epsilon'}{(1+0.001\epsilon')^{2^{i-1}}-1} \leq \sum_{j=1}^{\log_2 n} \frac{1.1c\delta'\epsilon'}{0.001\epsilon 2^{i-1}} \leq 2200c\delta'$$ ◀

▶ **Lemma 22.** *The probability that θ_{i^*} loses any match-up in the second phase of the tournament can be upper-bounded by $c\delta'$ times a constant factor (where c is the constant from Lemma 19).*

Proof. If the algorithm enters the second phase, this means that there is at least one void match-up. Let the number of nodes at this point be n. Then all $n/2$ nodes between the pair of nodes involved in the void match-up must be within a factor of $(1+\epsilon'/6.1)$ of the first node in the void pair. Therefore, there are at most $2\log_{1+0.001\epsilon'}(1+\epsilon'/6.1) \leq 12200$ nodes in total.

Every node that θ_{i^*} plays in a match-up is at least a factor of $1+\epsilon'/6$ away from it, which means that the absolute difference in value between the nodes is at least $(\theta'/1.1)\cdot(\epsilon'/6)$. By Lemma 19, this implies that the failure probability is at most $6.6c\delta'$. Since there are at most 12200 such match-ups, the overall failure probability is at most $81000c\delta'$ by the union bound. ◀

We can now prove Lemma 10, which we restate below.

▶ **Lemma 10.** Given the parameters θ', ϵ', and δ', there is a nonadaptive algorithm that performs $O(\log(1/\delta')/(\theta'\epsilon'))$ queries, and outputs a result θ_{est} with the following guarantee: if $\theta' \overset{1.11}{\approx} \theta^*$, then $\theta_{\text{est}} \overset{1+\epsilon'/6}{\approx} \theta^*$ except with probability at most $\delta/2$.

Proof. The algorithm achieving this lemma involves running the Grover schedule G and then post-processing using the tournament algorithm which we described. By Lemma 20, the failure probability of the algorithm is bounded by the probability that θ_{i^*} loses. This is bounded by the probability that it loses in the first phase or the second phase; by Lemma 21 and Lemma 22, this is at most a constant times $c\delta'$. By choosing c to be sufficiently small, we can successfully make this at most $\delta/2$. ◀

References

1. Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. In *Symposium on Simplicity in Algorithms*, pages 24–32. SIAM, 2020.
2. Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.
3. Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.
4. Paul Burchard. Lower bounds for parallel quantum counting, 2019. `arXiv:1910.04555`.
5. Dmitry Grinko, Julien Gacon, Christa Zoufal, and Stefan Woerner. Iterative quantum amplitude estimation, 2020. `arXiv:1912.05559`.
6. Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.
7. Kouhei Nakaji. Faster amplitude estimation, 2020. `arXiv:2003.02417`.
8. Ashwin Nayak and Felix Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 384–393, 1999.
9. Yohichi Suzuki, Shumpei Uno, Rudy Raymond, Tomoki Tanaka, Tamiya Onodera, and Naoki Yamamoto. Amplitude estimation without phase estimation. *Quantum Information Processing*, 19(2):75, 2020.
10. Chu-Ryang Wei. Simpler quantum counting. *Quantum Information and Computation*, 19(11&12):0967–0983, 2019.