Title

Improved Quantum Algorithms for Solving Abelian Hidden Subgroup Problems for Subgroups of Arbitrary Index

Author

Tran, Austin Nguyen

Publication Date

2022

Peer reviewed|Thesis/dissertation

Improved Quantum Algorithms for Solving Abelian Hidden Subgroup Problems for Subgroups of Arbitrary Index

By

AUSTIN TRAN

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Mathematics

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

Greg Kuperberg, Chair

_____

Elena Fuchs

_____

Ben Morris

Committee in Charge

2022

To my parents.

Abstract: This dissertation continues the progress on using quantum algorithms to efficiently solve the hidden subgroup problem (HSP) for various groups. We first give a necessary foundation on defining hidden subgroup problems and their efficient solutions. We then focus on the case when $G = \mathbb{Z}^k$, a natural continuation of the historical progress on HSP. The current algorithm for HSP on $\mathbb{Z}^k$ as shown by Kitaev is only successful when the hidden subgroup is of finite index [19]. Kuperberg has recently proposed an algorithm that solves HSP on $\mathbb{Z}^k$ for subgroups of arbitrary index [24]. His algorithm follows the quantum component of Kitaev's algorithm closely, but diverges with a novel use of the LLL algorithm in its classical post-processing. When covering the background of quantum algorithms to attack HSP, we emphasize the similarities in their quantum structure. We realize Kuperberg's results by providing explicit bounds on his defined parameters, and show success with computer simulations on his algorithm. We then propose modifications that significantly improve performance and resource usage, and provide computer simulations to demonstrate our results. We end by discussing applications in modern-day cryptography and how our research could assist with further development in the field of quantum algorithms.

Contents

1 Introduction

This dissertation focuses on the hidden subgroup problem (HSP) and the quantum algorithms that were discovered to solve it. The HSP is important since it encapsulates many mathematical problems that lie in the foundation of computing and security. The quantum algorithms that are able to solve it do so exponentially faster than their classical counterparts, which has significant implications for many cryptographic methods. Much of the current public-key cryptographic infrastructure is based on the difficulty of certain hard problems, such as the elliptic curve discrete logarithm problem. A hypothetical quantum computer that is powerful enough to implement these quantum algorithms could quickly solve any of these problems, which would be enough to defeat *all* current, standardized, public-key cryptography. However dramatic the implications, we will not focus on specific post-quantum cryptosystems or their analysis, but rather the historical advancement of the quantum algorithms themselves; see [34, 28] for a good summary of the former.

To formally realize the significance of quantum algorithms, we first disambiguate some of the terms we have been using. We will give the background in the context of problem statements and complexity theory, a perspective on which is rather lacking in the current literature. In particular, we aim to solve two current ambiguities when most papers discuss the use of quantum algorithms:

- What does it mean to *efficiently* solve a problem using a quantum algorithm?
- What is the difference between a problem, an instance, and an algorithm?

Section 2.3 will define these basic terms as well as their application to cryptography. In Section 2.4, we will then discuss the historical context of HSP, and how Kuperberg's paper fits in as the most recent link in a chain of results. It has been observed since the first papers

that most quantum algorithms behave very similarly. In particular, Jozsa [18] has observed that Simon's and Shor's algorithms "are essentially identical in their basic formal structure, differing only in their choice of underlying group". Both algorithms take advantage of the quantum Fourier transform (QFT) to solve HSP, with only the final post-processing steps differing. Indeed, much of the progress in HSP has been adapting and extending algorithms to solve HSP for larger and larger families of groups.

In Section 2.5, we will have seen how these previous algorithms are a natural predecessor to Kuperberg's Algorithm A. Shor's algorithm solves HSP for $\mathbb{Z}$, then Kitaev adapted Shor's work into the Shor-Kitaev algorithm (SKA) which solves the hidden subgroup problem for $\mathbb{Z}^k$ [19]. However, SKA requires that the hidden subgroup H be of finite index inside $\mathbb{Z}^k$. In his 2020 preprint [24], Kuperberg aims to patch this exception, and thus solve HSP for $\mathbb{Z}^k$ regardless of the status of H. Kitaev's original paper differs from the majority of quantum algorithm papers in his treatment of HSP. However, we emphasize that his framing is equivalent to that of his contemporaries, but not as cited. For our discussion, we follow Kuperberg's reframing of Kitaev's work, which employs a more familiar implementation of QFT and in fact shows that the two methods are equivalent. All of the tools used, such as the QFT itself as well as the LLL algorithm, have been analyzed many times previously, and we will not add another explanation here. Rather, we will emphasize the methods that are new, and look specifically at their implementation.

Like most quantum algorithms, Algorithm A consists of two parts. The first part is the quantum part, which we call A.1. Algorithm A.1 is very similar in structure to quantum algorithms that precede it, in particular Shor's algorithm and Kitaev's algorithm.

Because A.1 is so similar to previous algorithms, simulations and analyses of existing algorithms can be applied wholesale to A.1, and we will not focus too much on A.1 in our own analyses.

The second part of Kuperberg's Algorithm A is the classical post-processing part, which we call A.2. Quantum algorithms in general display the most variety in these classical parts, and A.2 follows this pattern as well. In Kuperberg's paper, he shows how A.2 uses the LLL algorithm to cover an exception to Kitaev's algorithm, immediately showing the novelty and power of A.2. Here we deviate from the objective of Kuperberg's paper and take an even closer look at A.2. Kuperberg's discussion of A.2 was focused primarily on its existence, and did not look closely at its practical implementations. There are 3 main areas in which we believe we can improve upon Kuperberg's implementation of A.2.

First, A.2 uses three main parameters; we combine their names and refer to them on the whole as RTQ. However, Kuperberg only bounds RTQ very loosely, and he does not describe the relationship between these parameters explicitly. We aim to describe the relationships explicitly. With computer simulations, we can provide an empirical but concrete range for RTQ that achieves the desired rate of success.

Second, A.2 assumes an extremely noisy input. As we will see, the input to A.2 has quantized noise, which is not easily mathematically modeled. So Kuperberg artificially adds a much larger Gaussian noise to the input, which he is more easily able to model, and bound accordingly. However, this increase in noise makes the bounds on RTQ even looser. We conjecture that A.2 without the Gaussian noise would be equally successful and more efficient; we use our computer simulations to provide evidence of precisely this increase in efficiency.

Third, Kuperberg mentions, but does not explore, the possibility of multiple samples. A.2 depends only on the output for A.1. It is therefore possible to call A.1 multiple times per single instance of A.2, get multiple outputs, and use all of those outputs for a single, modified instance of A.2. We conjecture a significant improvement in resource usage, which is again supported by the results of our computer simulations.

We will conclude by mentioning the uses of HSP and quantum algorithms in modern cryptography, the open questions that remain in our research, and how it could assist with further development in the field of quantum algorithms.

2 Context

In this section, we review some of the framework necessary in order to discuss Kuperberg's algorithm. We first explore the historical background of the hidden subgroup problem and its applications. We then review formal definitions relating to complexity and HSP itself, in order to resolve ambiguity in discussions going forward. Finally, we compare Kuperberg's algorithm to the algorithms that preceded it, placing Kuperberg's algorithm in its appropriate historical context.

2.1 Background

The results of this dissertation are the continuation of an effort to find efficient algorithms for the HSP for various groups. Although the earlier papers in this field were disparate in nature, the history of this research is generally accepted to begin with noteworthy results by Simon, and then Shor, who published their respective papers in 1994. A few years later, around 1998, all related results would begin to be classified under the same umbrella of "hidden subgroup problems".

In his 1994 paper [37], Simon considers the following problem. Suppose we have a function $f : (\mathbb{Z}/2)^n \to X$ where $|X| = 2^n$. It is promised that either: the function f is bijective, or there exists some $s \in (\mathbb{Z}/2)^n$ such that $f(a) = f(b)$ if and only if $a = b + s$.

The problem is to determine whether f is the first, or second type of function. In retrospect,

using the language of HSP, we see that f is hiding some subgroup H, and the problem is to determine whether $H = \langle 0 \rangle$ or $H = \langle s \rangle$. In this sense, Simon was in fact considering a weaker version of the HSP, which would later be known as the hidden subgroup *existence* problem (HSEP). We will the discuss the latter problem in Section 2.3, in the context of decision and function problems.

Simon was able to find a quantum algorithm that solved this problem, requiring only $O(n)$ queries to the hiding function; by comparison, the most efficient classical algorithm requires $O(2^{n/2})$ queries. In fact, in 2005, Koiran [20] showed that a simple generalization of this algorithm could solve the HSP, and not just HSEP, on $(\mathbb{Z}/2)^n$.

Later in 1994, building off of Simon's results, Shor [35] was able to generalize this algorithm and apply it to other problems, most notably including integer factorization and the discrete logarithm problem. His quantum algorithm also featured polynomial complexity when solving either of these problems, again in contrast to the exponential complexity of their classical counterparts. His famous paper was later published in 1997, in which both results are a special case of solving HSP over $\mathbb{Z}$.

The complexity of an algorithm over a given group G is usually parametrized by the number of bits used to canonically encode the output. As a baseline, many naive *classical*, i.e. *non-quantum*, algorithms can solve the HSP for a given G in exponential time, i.e. exponential in the number of bits. In addition to time complexity, it is relevant in some cases to also consider the *query* or *space* complexity. Thus an algorithm is said to be *efficient* at solving a problem - implicitly, with respect to time, space, or query complexity - if it can solve that problem in polynomial time. Similarly, we will say an algorithm *solves* a certain instance of HSP if it efficiently solves such a problem.

It is also worth distinguishing between *algorithms* for HSP and *instances* of said algorithms. For example, Shor's algorithm is a solution for HSP over $\mathbb{Z}$. One *instance* of HSP over $\mathbb{Z}$ is

period finding, which sets $f : \mathbb{Z} \to X$ as $x \mapsto a^x \pmod N$, which is a key reduction in Shor's famous result of integer factorization. When speaking of instances, we colloquially refer to *nontrivial* instances as those for which an efficient quantum algorithm is known, but no efficient classical algorithm is known. One of the results of Shor's paper is that period finding is a nontrivial instance of HSP over $\mathbb{Z}$. Conversely, many algorithms HSP have no known nontrivial instances, including Kuperberg's algorithm and Simon's algorithm.

In 1995, many researchers, including Kitaev, realized that both Simon's and Shor's algorithms were special cases of a more general problem. Kitaev [19] would combine many of these results under the heading of "abelian stabilizer" problems. Later that year, he would generalize Shor's results to solve the HSP over $\mathbb{Z}^k$ for arbitrary but fixed k. His paper takes a different approach to solving abelian stabilizer problems, focusing more on estimating eigenvalues for certain unitary operators, which would include the QFT.

Initially, these results were not thought of as existing as part of some central framework. The term "hidden" would first be used in Boneh and Lipton's 1995 paper [7] in reference to solving problems with "hidden linear forms". Using the language of HSP, Boneh and Lipton propose a method to solve HSP for certain subgroups of $\mathbb{Z}^k$, which would be subsumed as a special case of Kitaev's results. In his 1998 paper, Jozsa compared the results of Simon and Shor by emphasizing their use of the quantum Fourier transform (QFT). He writes in his introduction [18]:

> ...We will see that Simon's and Shor's algorithms are essentially identical in their basic formal structure differing only in the choice of underlying group. Both algorithms amount to the extraction of a periodicity relative to an abelian group G using the Fourier transform of G in a unique way. This general viewpoint may also be useful in developing new quantum algorithms by applying the formalism to other groups.

The phrase "hidden subgroup problem" would first be used in Mosca and Ekert's 1998 paper [25] which, among other observations, summarized the previous results much like we have done here. The phrase "hiding function" would not be coined until 2005, when Childs and van Dam [10] defined it in their paper on generalized hidden shift problems. The problem of HSP for finitely generated abelian groups would remain dormant for some time following this initial interest. There were adjacent developments for HSP for nonabelian groups; however, such results are so distinct that their discussion is outside the scope of this dissertation. We will only pause to mention some applications of nonabelian HSP in cryptography in a later section.

In 2002, Sean Hallgren [16] constructed a quantum algorithm for solving HSP for discrete subgroups of $\mathbb{R}$. In 2014, Eisenträger, Hallgren, Kitaev, and Song [12] found, among other results, an efficient quantum algorithm for HSP in $\mathbb{R}^k$, under the condition that the hidden subgroup as a lattice full rank inside $\mathbb{R}^k$, i.e. it was co-compact. Still, the case of arbitrary hidden subgroups of $\mathbb{Z}^k$ remained as an open problem. This gap from Kitaev's 1995 paper persisted until 2020, where Kuperberg's paper solved the remaining case, and is where our analysis continues from.

2.2 Applications

As previously mentioned, Shor's original paper not only develops a quantum algorithm for period finding, it also emphasizes the reduction of integer factorization to period finding. Shor also mentions an equally important reduction of discrete logarithm to period finding; such a reduction requires a slightly different treatment, as the HSP occurs over $\mathbb{Z}^2$ instead of $\mathbb{Z}$. These two problems provide the hardness assumption to widely used standardized public-key schemes such as RSA, Diffie-Hellman key exchange, and ElGamal encryption [11, 13]. As mentioned in the introduction, the existence of a practical quantum computer

would undermine all currently widely used public-key cryptography.

The existence of these quantum algorithms and their implications created the field of *post-quantum* cryptography (PQC). We do not make any attempt to cover the nascent development of PQC, instead we defer to any number of sources for a broader overview [21, 5]. For PQC proposals being actively considered by the US government, the National Institute of Standards and Technology (NIST) is maintaining such a list [38, 3].

Many quantum algorithms were not initially constructions for the sake of PQC; rather, these contributions came later. Much as in classical cryptography, the discussion of PQC can be split into categories colloquially called "attack" and "defense". In "defense", researchers propose a scheme and demonstrate why it is believed to be secure, usually because a quantum algorithm that solves the underlying hard problem is not yet known. The NIST proposals for future PQC schemes continue to this day [34, 28]. These current proposals all have in common a new proposal for a hard underlying problem. They all invariably depend on certain lattice problems, such as *short vector problems*, creating a subfield known as *lattice-based cryptography*.

To attack these schemes is to develop more quantum algorithms that solve an increasingly wider set of problems. Alternatively, we can attempt to leverage existing algorithms for the same purpose. In his 2004 paper [22] on HSP for dihedral groups, Kuperberg gives a subexponential time quantum algorithm for solving HSP for dihedral groups. In 2006, Rostovtsev and Stolbunov proposed a PQC algorithm that used as its central hard problem the computation of elliptic curve isogenies [33]. Their work would become the basis of many elliptic curve algorithms in PQC, and alongside it, Kuperberg's algorithm would become foundational in attacks against it. In 2014, Childs, Jao, and Soukharev used this algorithm to construct elliptic curve isogenies in quantum subexponential time [9]. In 2019, Peikert used Kuperberg's result to give a subexponential attack on a proposed PQC scheme called CSIDH [8, 27].

In this way, the security of any proposed scheme should become more resistant to both classical and quantum algorithms. While the focus of HSP is often on cryptographic applications, many of the previous results were in fact found in the context of mathematical applications, and were not solely focused on HSP. For instance, in 2002, Hallgren [16] developed an algorithm for HSP over $\mathbb{R}$ in the context of solving Pell's equation. This in turn resulted in more efficient solutions for the principal ideal problem in real quadratic fields, as well as the class group problem. One result of Eisenträger et. al. [12] is the reduction of finding the group of units of an algebraic number field to HSP on $\mathbb{R}^n$. In particular, they take care to place additional restrictions on the hiding function. They require the same condition of a full-rank sublattice as Kitaev, and also impose a Lipschitz condition of the hiding function, since it is defined over a continuous group.

To summarize our discussion, we present the first two tables. The first contains a subjective list of papers relevant to the historical development of HSP. The second contains a subjective list of nontrivial instances, i.e. applications, of HSP. In the next section, we will give an unambiguous and less protracted definition of 'solving' a problem.

Year	Author	Citation	Group and Algorithm
1994	Simon	[37]	Simon solves HSP on $(\mathbb{Z}/2)^k$ in quantum polynomial time.
1994	Shor	[35]	Shor solves HSP on $\mathbb{Z}$ in quantum polynomial time.
1995	Kitaev	[19]	Kitaev solves HSP on $\mathbb{Z}^k$ in quantum polynomial for subgroups of finite index.
2002	Hallgren	[16]	Hallgren solves HSP on $\mathbb{R}$ in quantum polynomial time.
2004	Kuperberg	[22]	Kuperberg solves HSP on D_N in subexponential but superpolynomial time.
2004	Regev	[29]	Regev solves HSP on D_N in subexponential time and logarithmic space.
2011	Kuperberg	[23]	Kuperberg solves HSP on D_N in subexponential time and polynomial quantum space, with a time-space trade-off for classical space usage.
2014	Eisenträger et. al.	[12]	Eisenträger et. al. solve HSP on $\mathbb{R}^k$ with full rank sublattices.
2020	Kuperberg	[24]	Kuperberg solves HSP on $\mathbb{Z}^k$ for subgroups of arbitrary index.

Table 1: An incomplete timeline of results for HSP for various groups G.

Year	Author	Citation	Application
1994	Shor	[35]	Shor reduces integer factorization and the discrete logarithm problem to HSP on $\mathbb{Z}$ and $\mathbb{Z}^2$, respectively.
2002	Hallgren	[16]	Hallgren uses HSP over $\mathbb{R}$ to solve Pell's equation.
2004	Regev	[31]	Regev reduces some short vector problems on lattices to HSP on D_N.
2014	Eisenträger et. al.	[12]	Eisenträger et. al. reduce the problem of finding the group of units of an algebraic number field to HSP on $\mathbb{R}^n$.
2019	Peikert	[27]	Peikert uses Kuperberg's 2011 result [23] to give a subexponential attack on a proposed PQC scheme.

Table 2: An incomplete timeline of various applications of HSP.

2.3 Complexity

When talking of HSP, it is usually not important whether a function $f : G \to X$ is explicitly known. Given an input g, we only want $f(g)$, and the mechanisms of how $f(g)$ is obtained is irrelevant. We contrast this with the definition when f is explicitly known.

Definition 2.1. A function $f : G \to X$ is said to be given by *oracle* if the following is true. For any given input $g \in G$, we can know $f(g)$ in *fiat* time.

Here "fiat" is used to be "arbitrarily given" by whatever black box is providing the oracle. Accessing the oracle once is known as a *query*; now it makes sense to talk of *query complexity*. If the function f is given explicitly as part of an instance, then the time complexity of querying f is assumed to be in $O(\mathrm{poly}(\log g)$. In general, the cost of querying f is irrelevant; it is only assumed to be any function of the bit length of the input. If f is part of an efficient algorithm, as defined below, then of course the cost of f is required to be at most the

polynomial bound given above.

Before even defining HSP, we give a narrow overview of complexity theory, so that we can formally define terms which are often taken for granted. When talking of complexity, we often muddy the distinction between time, space, and query complexity.

Definition 2.2. Unless specified otherwise, we say an algorithm is *polynomially efficient* if it runs in polynomial time, which implies polynomial in all other resources.

Similarly a problem M is *solved* by an algorithm A if A does so efficiently. When there is no confusion, we will write "efficient" in place of "polynomially efficient".

Previously we stated that Kuperberg's algorithm for dihedral HSP resolved in subexponential time. For clarity, we state the explicit definition of subexponential, as well as another classification of running time.

Definition 2.3.

- An algorithm has *subexponential* complexity in terms of n if its complexity is in $O(a^n)$ for any $a > 1$.
- An algorithm has *superpolynomial* complexity in terms of n if its complexity is in $\omega(n^c)$ for any c.
- An algorithm has *stretched exponential* complexity if its complexity is in $O\left(e^{n^a}\right)$ for some $a \in (0, 1)$.

Equivalently, an algorithm is stretched exponential if it is both subexponential and superpolynomial. From these definitions, we see that stretched exponential complexity is a stronger condition than being merely subexponential. While the latter term is used more often in the literature, we will use the former term where it is more accurate. For example,

Table 1 cites Kuperberg's and Regev's algorithms, both from their 2004 papers, that in fact run in stretched exponential time.

A *decision* problem is a function where the input and output are both some strings over any alphabet, but the output is either 0 or 1. Unless stated otherwise, we assume the input and output alphabets are encoded using binary encoding; for certain problems, it is important to explicitly define the encoding in order for the answer to be well-defined [12]. The familiar complexity classes **P** and **NP** are classes of decision problems. With the above remark in mind, the class **P** is also known as **PTIME**.

In cases such as HSP where the function f is given by oracle, the only parametrization of the problem that we know is the length of the output. A priori, there cannot be any relationship between the length of the input and the length of the output. So by default, we say that an algorithm solves M if it does so in polynomial time in the bit length of the output. If other information is known about a specific HSP, we will specify the parameters by which an algorithm is polynomial. For example, if an algorithm A solves HSP over $\mathbb{Z}^k$ for arbitrary k, we might say that A is polynomial in k, as well as the bit length of the output.

However, in the instances where f is given explicitly, there is often a clear relationship between the length of the output, the length of the output, and the length of the circuit - classical or quantum - needed to describe f. In these cases, we will be more specific on the polynomial bounds.

In contrast to **P** and **NP**, there is a distinction to be made with algorithms which are probabilistic rather than deterministic [1].

Definition 2.4. A decision problem M is said to be solvable in *bounded-error probabilistic polynomial time* (***BPP***) if there exists an algorithm to solve M in polynomial time with an error probability of at most $1/3$. Similarly M is said to be in *bounded-error quantum*

polynomial time (***BQP***) if it is solved by a quantum algorithm in such a bound.

Clearly $\mathbf{P} \subseteq \mathbf{BPP}$. It is unknown whether $\mathbf{P} = \mathbf{BPP}$ unconditionally, although it is widely believed to be true due to the belief in cryptographically secure pseudo random number generators [15]. The relationship between **BPP** and **NP** is also unknown [4].

Measurement in quantum computation means the process is inherently probabilistic. In practice, probabilistic algorithms suffice; we can run the experiment multiple times, and take the majority result. This returns the correct answer with probability exponentially close to 1 in the number of runs.

We now distinguish between function problems and decision problems. In contrast to decision problems, function problems can output any possible answer for a given input. Thus function problems must be put into classes distinct from those of decision problems; the analogous counterparts to P and NP are called *function* **P** or *function* **NP**, abbreviated **FP** and **FNP** respectively. So if the decision problems in **P** are thought of as functions from strings to $\{0, 1\}$ that can be efficiently solved, then **FP** consists of functions from strings to *strings* that can be efficiently solved. However, in general the distinction between a class and its functional counterpart is not too important for the following reason [32].

Remark 2.5. Let M be a function problem whose output is in binary encoding b_i. Then M is in **FP** if and only if each bit b_i is, as a decision problem, in **P**.

The analogous theorem is also true for **NP**. By abuse of notation, we can now say that a decision problem can also be in **P**, where we mean that it is actually in **FP**. In this way we similarly define functional **BQP** and **BPP** as well.

As an example, our definition of HSP is as a function problem. So in order to give hardness results about HSP in comparison to the complexity classes of decision problems, a reduction has to be made, which is where the definition of HSEP is useful, as HSEP is a decision

problem instead. HSEP is needed as a comparison to state that Kuperberg's algorithm, as discussed in Section 3, gives an exponential decrease in complexity compared to a purely classical implementation. The algorithm is thus seen to be in functional **BQP**.

2.4 The Hidden Subgroup Problem

Let G be a group. Unless stated otherwise, we assume G is a finitely generated abelian group, and so subject to the classification theorem of finitely generated abelian groups. Furthermore, G must also be what Kuperberg [24, 2] calls an *algorithmic* group. Such a group must have an explicit representation for all its elements. Additionally, the functions of membership, group law, and inversion should all be computable in polynomial time.

Kuperberg uses the example of encoding $\mathrm{PSL}(2,\mathbb{Z}) \simeq \mathbb{Z}/2 * \mathbb{Z}/3$. In the context of $\mathrm{PSL}(2,\mathbb{Z})$, group elements can be represented by certain integer matrices. However, the equivalent elements in $\mathbb{Z}/2 * \mathbb{Z}/3$ can be represented by arbitrary length strings in the alphabet $\{a, b\}$. Some elements of $\mathrm{PSL}(2,\mathbb{Z})$ can be generated by taking Nth powers of the matrix

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

Thus the matrix representation has bit complexity $\Omega(\log N)$, while the string representation has bit complexity $\Omega(N)$, and thus there are two distinct efficient encodings of the same group, up to non-efficient isomorphism. For quantum computation, we assume that a circuit f can be made into a reversible circuit evaluating f, with a negligible increase in complexity. Therefore it is important to specify a canonical efficient encoding for groups. Some common ones include:

- Each element in $\mathbb{Q}$ is written as a/b, where a and b are coprime and written in binary.
- Each element of a finitely generated free group F_k is written as the unique reduced word

in the generators of F_k.

- Each element of $\mathbb{Z}^k$ is written as a list of integers in binary.

With this setup, the *hidden subgroup problem* (HSP) can be stated as follows:

Definition 2.6. Let G be an algorithmic group. Let $H \leq G$ be a subgroup of G, and X be any unstructured set. A function $f : G \to X$ is said to *hide* H if: for any $g_1, g_2 \in G$,

$$f(g_1) = f(g_2)$$

if and only if $g_1 H = g_2 H$.

Definition 2.7. Let G be a group and X a finite unstructured set. Let $f : G \to X$ hide a subgroup $H \leq G$. Then the *hidden subgroup problem* on G is: given G and X, and f by oracle, it finds any generating set of H.

When an algorithm is said to be "solving" HSP on G, the word "solve" is used as an abbreviation for efficiently solving, or polynomially efficient solving. In this case of infinite groups, we can only guarantee that a solution to HSP will solve the problem efficiently, which means in polynomial time complexity in the bit length of the output. Otherwise, the generators for G could be arbitrarily large.

In either the infinite or finite case, an exhaustive search is always exponential in time complexity (again in the bit complexity of the output): we can simply evaluate $f(g)$ for all $g \in G$, and thus determine H with $|G|$ queries of f. In the infinite case, we cannot search the whole group, but we can promise to search up to a certain bound. For example, we can search up to the first S elements, and keep doubling S until a result is found; such a method also requires exponential time. So we would find an algorithm - classical or quantum - nontrivial if it could reduce this bound at all. In the ideal case, such as with Shor's algorithm, it could be reduced to $O(\mathrm{poly}(\log f(g)))$ query complexity. In other cases,

such as Kuperberg's results on dihedral groups [22], a stretched exponential reduction is still nontrivial. When a reduction to polynomial time is desired, we of course must assume at most a polynomial time complexity of evaluating the hiding function f, whether given by oracle or instance.

We also want to distinguish between any hiding function, and nontrivial hiding functions. The latter occur when the function itself does not encode the group structure and elements, i.e. if H cannot trivially be found by looking at the circuit that comprises f. In these cases, the input type must be evaluations of f, and not its circuit or "source code". For classical algorithms, this equivalently means that HSP cannot be solved in less than exponential time without knowing the structure of G, merely by looking at the source code of f.

Following the paper of Mosca and Ekert [25], we list some applications of HSP.

- Simon's problem:

  Definition 2.8. Let $G = (\mathbb{Z}/2)^k$. Let $f : G \to X$ hide a subgroup $H = \langle s \rangle \leq G$, where s is any element of G. Given G and f, *Simon's problem* is to find s, equivalently H.

  As the first well-cited instance of HSP, Simon's problem is a direct statement of HSP, not a reduction. Since any element of G has order at most 2, the function f is either trivial or precisely 2 to 1. It is worth repeating that f is only required to be known by oracle; there are no known nontrivial instances of Simon's algorithm. Simon's algorithm is noted in terms of query complexity. Where any classical algorithm needs at least $\Omega(2^{n/2})$ queries to the oracle, Simon's quantum algorithm needs only $O(n)$ queries. Furthermore, any quantum solution would need at least $\Omega(n)$, thus Simon's algorithm is close to "optimal" [20].

- Integer factorization: Shor's paper defines a period finding problem as follows. For a fixed $N \in \mathbb{Z}$ and a coprime to N, define $f : \mathbb{Z} \to \mathbb{Z}/N$ as $x \mapsto a^x \pmod N$. Then f hides

the subgroup $\mathbb{Z}/r$, where r is the order of a in $\mathbb{Z}/N$. Given f, N, and a, the *period finding* problem is to find r.

Shor's algorithm [36] is a quantum solution to the period finding problem requiring only $O(1)$ queries to f, with a cost of $O(\log N)$. Note that f here is explicit and not given by oracle, but still only takes $O(\log N)$ time to compute. In the same paper, Shor also showed an elementary reduction of integer factorization to the period finding problem in $\mathbb{Z}$. This gives an algorithm for integer factorization in polynomial time (recall polynomial in the bit length of the output, i.e. $\log N$). The current fastest classical algorithm, the number field sieve, still takes subexponential time [6].

- Discrete logarithm problem: The discrete logarithm is generally defined over $(\mathbb{Z}/p)^\times$, and can also be formulated over any finite abelian group, as in the case of elliptic curve cryptography.

  Definition 2.9. Suppose g generates $(\mathbb{Z}/p)^\times$. Given $a \in (\mathbb{Z}/p)^\times$, the *discrete logarithm* problem (DLP) is to find $r \in \mathbb{Z}/(p-1)$ such that $g^r = a$.

  To reduce DLP to HSP, define G as the additive group $(\mathbb{Z}/(p-1))^2$, and let X be the set of $(\mathbb{Z}/p)^\times$. Then define

  $$f : G \to X, \quad (\alpha, \beta) \mapsto g^\alpha a^{-\beta}$$

  Then $(\alpha, \beta) \mapsto 1$ if and only if $\alpha = x\beta$. Thus f hides precisely the subgroup $\langle (x, 1) \rangle$; so solving HSP on f finds the desired x.

  The Diffie-Hellman key exchange protocol [11] is an example of a public-key cryptosystem that can be reduced to a DLP, and which uses elliptic curve methods (since elliptic curves over e.g finite fields are abelian groups). Shor's paper showed a solution to HSP over $\mathbb{Z}^2$ as well as $\mathbb{Z}$, with a similar consequence: the fastest classical algorithms for DLP run in subexponential time, while a quantum algorithm is able to solve this problem in polynomial time.

- Abelian Stabilizer Problem:

  Definition 2.10. Suppose G is a finitely generated abelian group acting on a set X via a function F. Given any $z \in X$, the *abelian stabilizer* problem ASP) is to find a generating set for $\mathrm{Stab}_G(z) = \{g \in G : F(g, z) = z\}$.

  The ASP was first defined by Kitaev, who also showed that the problem can be reduced to HSP. Let f_z denote the function from G to X which maps $g \mapsto g(z)$. Then immediately, the hidden subgroup corresponding to f_z is precisely $\mathrm{Stab}_G(z)$. So ASP is a special case of (abelian) HSP.

- Short vector problem (SVP): Regev's 2004 paper [31] showed a polynomial time reduction of SVP to HSP on the dihedral group. The shortest vector problem has many possible statements, and the two relevant to Regev's paper are:

  Definition 2.11. Let $L \leq \mathbb{Z}^k$ be a full-rank integer lattice. Then the *shortest vector problem* is to find the shortest nonzero vector $v \in L$.

  Definition 2.12. Let $L \leq \mathbb{Z}^k$ be a full-rank integer lattice. If v is the shortest nonzero vector of L, let $|v| = s$. Then the $f(k)$*-approximate-shortest vector problem* is to find any vector u such that $|u| \leq f(k)s$.

  Regev's reduction is significant since SVP was, and continues to be, thought of as difficult to solve. In 2004, Kuperberg [22] gave a stretched exponential time algorithm for this instance of HSP, but this was not sufficient for a subexponential time algorithm for SVP. Regev's reduction was polynomial time, specifically quadratic, and Kuperberg's algorithm had $2^{O(\sqrt{k})}$ time complexity. Kuperberg's algorithm was later improved upon by Regev in a separate paper [29], but the improvement was only on space complexity; the time complexity continued to be exponential after the quadratic reduction.

As mentioned in the previous section, these are a selection of instances where the hiding function f is explicitly known. While an algorithm exists for Simon's problem, no nontrivial

instance of Simon's problem is known. For our algorithm over $\mathbb{Z}^k$, the story is similar. The algorithms we present in the following sections show efficient quantum algorithms for many instances of abelian HSP, however their variety denies a single overarching theorem. The success of solving abelian HSP leads to the natural question of whether nonabelian HSP is equally tractable. It has been shown that a polynomial number of queries suffice [14]. However, in general there is no bound on the overall computational complexity; currently, there is no known efficient algorithm for most nonabelian groups. For example, solving the hidden subgroup problem for the symmetric group would directly solve the graph automorphism problem [17]. We will not discuss the nonabelian HSP cases, and for the remainder of the text, we will emphasize this distinction by referring to the abelian HSP, or AHSP.

2.5 Previous Algorithms

We now briefly review Shor's algorithm and Kitaev's algorithm with our appropriate context. Both algorithms are direct predecessors of Kuperberg's algorithm, and both can be shown to behave very similarly, at least initially. We mostly follow the points given in Kuperberg's paper [24].

At their core, all of these quantum algorithms measure the output of a QFT on a finite quantum state. This must be the case since practically implementing an infinite quantum state, over for example $\mathbb{Z}$ or $\mathbb{Z}^k$, is impossible. To do so, the infinite quantum state is approximated by a large subset, for example $\mathbb{Z}/Q \subset \mathbb{Z}$ for Shor's algorithm, or $(\mathbb{Z}/Q)^k \subset \mathbb{Z}^k$ in Kuperberg's algorithm. This approximation leads to noise or an approximation measurement, in the output of the classical part of the algorithm. Where these algorithms differ is the amount of noise in the measurement, and the classical steps they take to deal with this noise.

Shor's algorithm solves HSP on $\mathbb{Z}$ can be solved, and begins with the constant pure state

$$|\psi\rangle = \sum_{x \in \mathbb{Z}/Q} |x, f(x)\rangle$$

where $\mathbb{Z}/Q$ can be thought of as an embedding into $\mathbb{Z}$. It is important to note that this embedding is not a homomorphism, and should be thought of as a truncation of $\mathbb{Z}$ inside an interval. The hiding function with period r translates exactly into a hidden sublattice in H with index r. Finding this sublattice is thus equivalent to finding the period of the hiding function. More precisely, the Fourier transform of $|\psi\rangle$ produces a dual lattice; however, the dual to a lattice in $\mathbb{Z}/Q$ is also in $\mathbb{Z}/Q$.

The output of Shor's algorithm is a measurement y that is approximately in $H^{\#} \subset \mathbb{R}/\mathbb{Z}$. The dual group $H^{\#}$ is not the same as the Pontryagin dual groups defined in Kuperberg's paper, so we give this dual group another definition.

Definition 2.13. Let H be a lattice in $\mathbb{Z}^k$. The *reciprocal* group $H^{\#}$ with respect to H is a subgroup of $(\mathbb{R}/\mathbb{Z})^k$ given by $H^{\#} = \{y \mid x \cdot y \in \mathbb{Z}\}$ for all $x \in H$.

A general definition of Pontryagin duals is given in Section 7.3 of [24]. We note only what is necessary. In particular, if H is a (closed) subgroup of a (locally compact) abelian group G, then $H^{\#}$ is a subgroup of the Pontryagin dual $\widehat{G}$ of G, and in fact $H^{\#}$ is exactly $\widehat{G/H}$. For example, the Pontryagin dual of $(\mathbb{R}/\mathbb{Z})^k$ is $\mathbb{Z}^k$. Pontryagin dual groups will appear again briefly in Section 3.1 when we discuss the output of the quantum part of Kuperberg's algorithm.

In the case of Shor's algorithm, We distinguish between $\mathbb{R}/\mathbb{Z}$ and $\mathbb{T}$ since in the case of Shor's algorithm, he identifies $\mathbb{R}/\mathbb{Z}$ with $[0, 1)$ instead of $[-1/2, 1/2)$. Furthermore, the reciprocal group $H^{\#}$ has size exactly r, The output of Shor's algorithm is put through its classical post-processing, which is a continued fraction expansion that returns the desired point in $H^{\#}$. This output is then sufficient to determine the entire hidden sublattice.

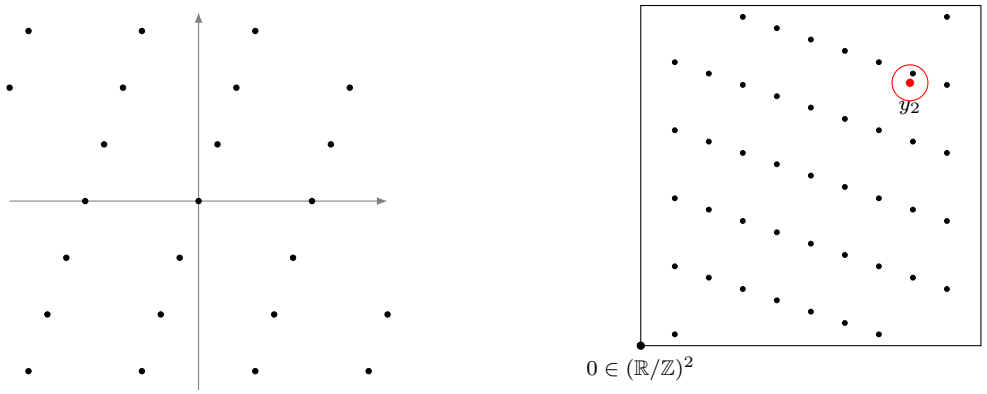
The quantum setup for Kitaev's algorithm is slightly different from Shor's and Kuperberg's algorithm. They both prepare a quantum state

$$|\psi\rangle = \sum_{x \in H \subset \mathbb{Z}^k} E(x) \, |x, f(x)\rangle$$

Here E is a placeholder for the type of state used; in Kitaev's algorithm, it is again a constant pure state, whereas Kuperberg's algorithm uses a Gaussian state. In the latter, the preparation of the quantum state is done over $H = (\mathbb{Z}/Q)^k \subset \mathbb{Z}^k$, which should again be thought of as a truncation of $\mathbb{Z}^k$ inside a k-dimensional hypercube. This extends the reframing of Shor's algorithm above; a 1-dimensional hypercube is simply an interval.

We note that Shor's algorithm requires only 1 sample, and in fact does not improve in performance with the introduction of multiple samples. In contrast, Kitaev's algorithm works with $O(\log n)$ samples needed to generate H. Analogously, Kitaev's algorithm finds an approximate point to a reciprocal subgroup of $(\mathbb{R}/\mathbb{Z})^k$. This subgroup in Kitaev's algorithm must be of full rank since the hiding function $G \to X$ maps to a finite set, and thus the kernel is of finite index. Kitaev's algorithm also uses CFA (Lemma 10) similar to Shor's algorithm, except it is applied coordinate-wise. After this denoising given by CFA, Kitaev is then able to probabilistically generate the reciprocal group $H^\#$ from uniformly random elements.

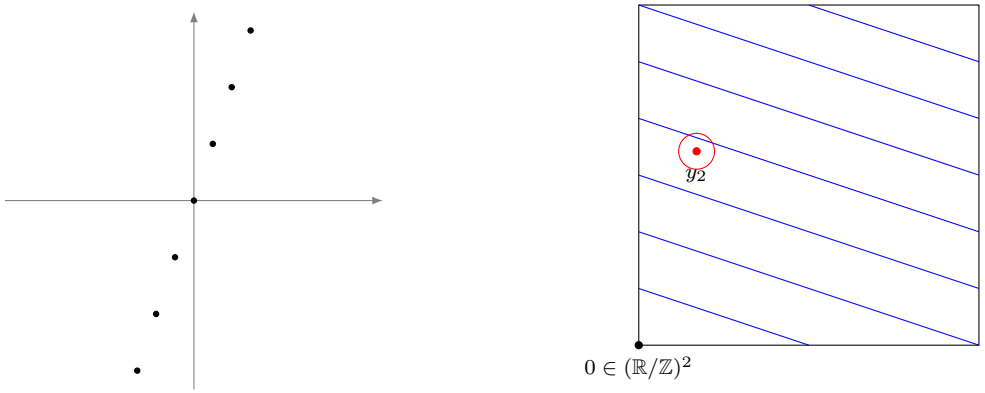
The distinction between Kuperberg's algorithm and previous algorithms is apparent in the following figures. We can see the increased difficulty when the hidden lattice H is not of full rank.



$H \subset \mathbb{Z}^2$ has rank 2. $H^\# \subset (\mathbb{R}/\mathbb{Z})^2$ is finite.

Figure 1: A visualization of H and $H^\#$ when $k = \ell = 2$.

Here, $y_0 \in H^\#$ can be recovered from y_2 using a generalization of continued fraction expansion found in the Shor-Kitaev algorithm.



$H \in \mathbb{Z}^2$ has rank 1. $H^\#$ is one-dimensional, and infinite.

Figure 2: A visualization of H and $H^\#$ when $k = 2$ and $\ell = 1$.

Here, no single coordinate of y_2 can help in recovering y_0; knowing the direction of the stripes would a priori imply that we already know H.

Kuperberg's algorithm is able to work with 1 sample, and then in fact offloads the rest of the work to Kitaev's algorithm, but did not analyze the use of multiple samples, which is one objective of our research.

From section 6.1 of [24], we have the following equivalence. The quantum components of Kuperberg's algorithm and Kitaev's algorithm share the following property: the output is a point approximately on some reciprocal subgroup, and the noise in that approximation has bounded norm with high probability. In the case of Kuperberg's algorithm specifically, the output measurement y_2 of the quantum state can be written as

$$y_2 = y_0 + u$$

where y_0 is in a reciprocal group $H^{\#}$ and $|u|$ is an approximately Gaussian noise term; we give the exact details of this Gaussian noise below, in Theorem 3.3. In particular, the unitary operator applied by Kitaev in Section 2 is identical in distribution to the basis vector Kuperberg produces. Kuperberg's algorithm is shown to work with 1 sample, after which the problem is reduced to Kitaev's algorithm.

In both Shor's and Kitaev's algorithm, the only noise comes from the QFT of a constant state, which has a square wave cutoff outside of $\mathbb{Z}/Q$. In Algorithm A, the initial Gaussian state has a Gaussian wave cutoff outside of $(\mathbb{Z}/Q)^k$.

The result is that the QFT of square wave noise becomes, as in the classical case, a sinc function. Algorithm A has an initial Gaussian state which, as in the classical case, becomes a discretized Gaussian state after the QFT, having support only on the integer multiples of $1/Q$. The important part to note is that, in Kuperberg's algorithm, the Gaussian state is necessary to have decaying tails outside of $(\mathbb{Z}/Q)^k$, instead of the square wave noise that naturally arises as in Shor's algorithm. Thus the Gaussian noise in Kuperberg's results is only added for theoretical completeness, and is not in fact an optimal lower bound of noise.

3 Kuperberg's Algorithm A

In this section, we outline the steps in Kuperberg's Algorithm A, following his treatment in [24]. As discussed above, Algorithm A is cleanly segmented into its classical and quantum components. The discussion of the quantum component is brief, as its setup is almost exactly similar to algorithms that precede it. The novel aspects of Kuperberg's algorithm come in its classical treatment, which we discuss in greater detail. Finally, we discuss how the classical component is implemented practically, and what assumptions we must make about the rest of the algorithm for our implementation to work as intended.

3.1 A.1: The Quantum Part

Let $G = \mathbb{Z}^k$ and let the hidden subgroup $H \leq G$ be of arbitrary rank. Geometrically, H is a sublattice inside the k-dimensional lattice G, not necessarily of full rank. In this context, we state the HSP as follows:

Let $f : \mathbb{Z}^k \to X$ be a function that hides a lattice $H \leq \mathbb{Z}^k$ which has some arbitrary rank $0 \leq \ell \leq k$. We assume that H has a $k \times \ell$ generator matrix M_0 with bit complexity n. The HSP is solved if we can find any generator matrix M of H, not necessarily equal to M_0, in quantum polynomial time in n.

Kuperberg's algorithm, as well as Kitaev's, requires a generalization of the QFT to $(\mathbb{Z}/Q)^k$ as well as $(\mathbb{R}/\mathbb{Z})^k$. The analyses of the algorithms, particularly Kuperberg's, require additional generalizations, which are defined in Section 7.4 of [24]. There is no algorithmic QFT for $(\mathbb{R}/\mathbb{Z})^k$, instead only the theoretical Fourier transform which can be applied to quantum states.

There are a few ways to generalize the QFT to other types of groups. Here we will only

detail the extension that is needed for our own analysis. In Shor's algorithm, the n-qubit input register is usually interpreted as a state on $\mathbb{Z}/(2^n)$, but can also be structured as $(\mathbb{Z}/2)^n$; the converse is usually true for other algorithms such as Simon's, where the latter state is the more common interpretation. In this case, the Fourier transform is the Hadamard transform applied to each of the n qubits in parallel. Our application of QFT over $(\mathbb{Z}/Q)^k$ in Kuperberg's algorithm can be thought of as applying the QFT over each of the k registers containing $\mathbb{Z}/Q$, and then tensoring the results. The resulting QFT can be defined as follows.

Definition 3.1. The QFT over $(\mathbb{Z}/Q)^k$ is given by

$$F\left|\Psi(y)\right\rangle = \frac{1}{Q^{k/2}} \sum_{x\in(\mathbb{Z}/Q)^k} \omega_Q^{x\cdot y} \left|\Psi(x)\right\rangle$$

We are now able to outline the first part of Algorithm A, which we call A.1. Choose positive integers Q, R, S, T. Practically, we want $Q = O(\exp \operatorname{poly} n)$, and it is sufficient, but perhaps not necessary, that $S \ll Q$. However, a central conjecture of our implementation of A.2 is:

Conjecture 3.2. Gaussian noise is not actually necessary in the implementation of A.1. That is, setting $S \to \infty$ yields precisely the same output distribution as in (3.3).

As discussed above, the initial setup is similar to Shor-like algorithms. Initialize the quantum Gaussian state

$$\left|\psi_{GC}\right\rangle \propto \sum_{\substack{x\in\mathbb{Z}^k \\ |x|_\infty < Q/2}} \exp\left(-\pi|x|_2^2/S^2\right)\left|x\right\rangle \tag{1}$$

A two-dimensional version of this quantum state is seen in figure 3, which is taken from [30]. As in Kitaev's algorithm and similar, this construction is done by first constructing

the one-dimensional state

$$|\phi_{GC}\rangle = \sum_{\substack{x\in\mathbb{Z} \\ |x|<Q/2}} \exp\left(-\pi x^2/S^2\right)|x\rangle$$

using the Grover-Rudolph algorithm; then simply

$$|\psi_{GC}\rangle = |\phi_{GC}\rangle^{\otimes k}$$

where $\otimes k$ means that the quantum state $|\phi_{GC}\rangle$ has been tensored with itself k times.

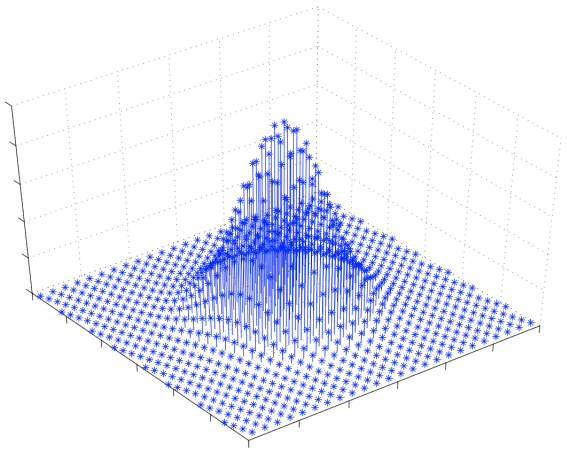


Figure 3: The 2-dimensional Gaussian lattice representing the quantum state $|\psi_{GC}\rangle$.

Then, apply a unitary dilation U_f of the hiding function f to obtain:

$$U_f\,|\psi_{GC}\rangle \propto \sum_{\substack{x\in\mathbb{Z}^k \\ |x|_\infty<Q/2}} \exp\left(-\pi|x|_2^2/S^2\right)|x, f(x)\rangle$$

The output register is thrown away, and we are left with a mixed state $\rho_{H,s}$ in the input register.

Let $F_{(\mathbb{Z}/Q)^k}$ be the quantum Fourier operator for the group $(\mathbb{Z}/Q)^k$ as defined above. Apply $F_{(\mathbb{Z}/Q)^k}$ to the input register, then measure it in the computational basis to obtain $y \in (\mathbb{Z}/Q)^k$.

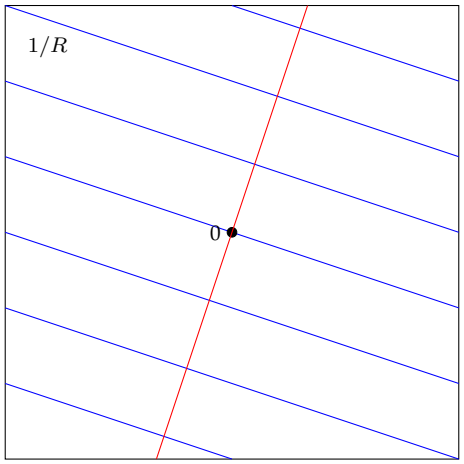


Figure 4: A dual lattice $H^{\#}$ when $k = 2$ and $\ell = 1$. The spacing of the lattice lines and the normal line through 0 are shown in the figure.

Then define

$$y_2 := qy \in q(\mathbb{Z}/Q)^k \leq (\mathbb{R}/\mathbb{Z})^k$$

This is the end of A.1., and the vector y_2 is the output of A.1. If $H \leq \mathbb{Z}^k$, then $H^{\#} \leq (\mathbb{R}/\mathbb{Z})^k$ is defined as the set of all $y \in (\mathbb{R}/\mathbb{Z})^k$ such that $x \cdot y = 0$ for all $x \in H$. Note that $H^{\#}$ here is the reciprocal group of H, as well as the Pontryagin dual of $\mathbb{Z}^k/H$, as defined in X.

We can now formally bound the Gaussian noise mentioned in Section 2.5, which is given in Section 7.2 of [24].

Theorem 3.3. The produced vector y_2 satisfies

$$y_2 = y_0 + u$$

where $y_0 \in H^{\#}$ uniformly randomly, and $u \in (\mathbb{R}/\mathbb{Z})^k$ is an error term such that

$$\Pr\left(|u|_2 \leq \frac{\sqrt{k}}{S}\right) \geq \frac{3}{4}$$

Figure 4 depicts a deficient rank lattice when $k = 2$, as well as the derived normal line through the origin.

Referring back to Kuperberg's paper, we can give an idea of the noise model of the output

of A.1. The quantum state defined in (1) can be approximated by the infinite state

$$|\psi\rangle \propto \sum_{x \in \mathbb{Z}^k} \exp\left(-\pi |x|_2^2 / S^2\right) |x\rangle$$

According to Lemma 7.9 of Kuperberg's paper, the measured state can be approximated as follows. First, we apply the infinite Fourier operator $F_{\mathbb{Z}/k}$, which returns a state in $(\mathbb{R}/\mathbb{Z})^k$. We can then quantize this state by multiplying by Q and then rounding, which results in a state on $(\mathbb{Z}/Q)^k$. This process is in contrast to what the algorithm actually performs, which happens in the other order. In A.1., first $|\psi\rangle$ is interpreted as a state on $(\mathbb{Z}/Q)^k$, and then we apply the Fourier operator $F_{(\mathbb{Z}/Q)^k}$.

The reason for this is that noise resulting from the Fourier operator $F_{\mathbb{Z}^k}$ is easier to bound, but of course the infinite Fourier operator cannot actually be implemented. Applying $F_{(\mathbb{Z}/Q)^k}$ produces a Gaussian noise term, while the rounding produces a quantization noise term. The former noise is distributionally equal to a Gaussian, but in fact is not exactly a Gaussian. This equivalence is shown in Section 7.5 of Kuperberg's paper, which also provides rigorous bounds for both noise terms. We will only implement and experiment with the quantized noise, as discussed in the next section.

Lemma 3.4. If $S \geq k$ and $Q \geq S^2$, then as $S \to \infty$ we have the following bound:

$$\sum_{x \in \mathbb{Z}^k, |x|_\infty \geq Q/2} \exp\left(-\pi |x|_2^2 / S^2\right) = \exp\left(-\Omega(S^2)\right)$$

In particular, this lemma allows us to ignore the tails of multi-dimensional Gaussian without substantially changing the resulting norm or Fourier transform.

3.2 A.2: The Classical Part

Consider appending a small extra coordinate $1/T$ onto y_2. This has the effect of embedding y_2 on a slightly slanted hyperplane in $\mathbb{T}^{k+1}$. Call this augmented vector $\widetilde{y_2}$.

Consider now taking many multiples of $\widetilde{y_2}$ in $\mathbb{R}^{k+1}$. To get this same effect, we can consider the lattice in $\mathbb{T}^{k+1}$ generated by

$$\{e_1, e_2, \ldots, e_k, \widetilde{y_2}\}$$

and apply the LLL algorithm to this basis. This will generate an LLL-ordered basis

$$B = [b_1, b_2, \ldots, b_{k+1}]$$

Let B also represent the $(k+1) \times (k+1)$ matrix whose columns are the above basis vectors. We take a submatrix B_1 of B consisting of all the short basis vectors, i.e. all the basis vectors with 2-norm less than $1/R$. For some ℓ', the matrix B_1 now has size $(k+1) \times (k+1-\ell')$. We have suggestively labeled ℓ', since if we assume success, then in fact $\ell' = \ell$.

Now we need to delete rows until B_1 is square again. If row j is deleted, call the remaining matrix $B_{1,j}$. We first choose j such that

$$|\det(B_{1,j} B_{1,j}^T)|$$

is maximized. After a row is deleted, we replace B_1 with $B_{1,j}$, and iterate until we have a square matrix, which we call B_2. Recall B_1 has size $(k+1) \times (k+1-\ell')$, and now B_2 has size $(k+1-\ell') \times (k+1-\ell')$. So we calculate

$$B_3 = B_1 B_2^{-1}$$

The matrix B_3 now also has size $(k+1) \times (k+1-\ell')$. For each entry in B_3, we calculate its continued fraction approximation (CFA), and replace the entry with its last convergent whose denominator does not exceed R. This gives us a rational matrix $B_0 \approx B_3$.

If B_0 was calculated correctly, it should contain a copy of the identity matrix $I_{k+1-\ell'}$ inside it. So we delete all standard basis vector rows inside B_0, which gives us a $\ell' \times (k+1-\ell')$ matrix we call B_4.

We now create a $(k+1) \times \ell'$ matrix A_0 by concatenating $I_{\ell'}$ with $-B_4^T$ as follows:

$$A_0 = N \begin{pmatrix} I \\ -B_4^T \end{pmatrix}$$

Here A_0 has been turned from a rational matrix into an integer matrix by multiplying it by N, the lcm of its entries. Being an integer matrix, we can now calculate the Smith normal form (SNF) of A_0. This gives us matrices D, V, W such that

$$D = VA_0W$$

where D is a diagonal matrix.

Take $I_{\ell'}$ once again, and append it with rows of zeros until it has $k+1$ rows; call this I'. Finally, we calculate

$$A_1 = V^{-1}I'W^{-1}$$

and remove any null rows to get a matrix H_1. Assuming success up until this point, the original matrix H generates a sublattice of the lattice generated by H_1, but crucially, both lattices have the same rank ℓ. Thus Algorithm A has reduced the problem to exactly the case of SKA, and thus our HSP is solved. The conclusion is Theorem 7.1 of [24]:

Theorem 3.5. The parameters Q, R, S, T can be chosen such that the entirety of Algorithm A runs in polynomial time and succeeds with high probability.

3.3 Implementation

An implementation of the quantum A.1 is outside the scope of this dissertation. We note that it is trivial to simulate a quantum superposition using a classical computer *in exponential time*, but such an implementation is not a part of our analysis. As we will see in our analysis of A.2, the mechanisms of A.1 can be taken as a black box, as A.2 depends only on the actual output of A.1. The success of the implementation is implied by the above Theorem 3.5.

In order to simulate A.2 without having actual measurements from A.1, we need to in turn simulate output from A.1, which is done as follows. As mentioned above, the parameter S is unnecessary when creating the output from A.1, which is a vector that lies very close to the dual lattice $H^{\#}$. So we first need to generate H, which will give $H^{\#}$, which we can use to sample vectors. To generate H, we create a generator matrix M. We fix an integer N which bounds the entries of M between $-N$ and N. Then M is a $k \times \ell$ matrix with integral entries chosen independently and uniformly randomly between $-N$ and N. It is possible that M is not actually of rank ℓ, but we can check the rank of M after it is generated and repeat this generation until M is valid.

The next problem is to simulate the output A.1. By following Conjecture (3.2), we disregard any S term; the problem is thus to generate a uniformly random element of $H^{\#}$, and then simply round to the nearest multiple of $1/Q$. This is done by first putting H into its SNF with $D = UHV$. Then a random vector W is chosen, and the vector WU is rounded to the nearest multiple of $1/Q$ and projected down into $(\mathbb{Z}/Q)^k$.

Unless stated otherwise, the integer k will be the dimension of the ambient space $\mathbb{Z}^k$, and ℓ will be the rank of the hidden sublattice H, with $\ell \leq k$.

We then fix the main parameters of A.2, namely the integers $R, T,$ and Q. As we have seen

in the previous analysis, we only require $R = O(N)$ and $T = O(RN^k)$. Finally, we only require $Q \geq 2RT$. So we choose some integers α and β and set $R = \alpha N$ and $T = \beta RN^2$, then set $Q = 2RT$. Even though R and T will not be used just yet, they are necessary in order to define Q, which is needed to generate a dual lattice sample.

The noisy dual lattice sample will be generated as follows. We randomly select a vector v on $H^{\#}$ with as much floating point precision as the computer will allow. To make it noisy, we then round the coordinates of v to the nearest multiples of $1/Q$. This so-called quantized noise is the substitute for the noisy output of A.1. Later, we will increase the noise of v by adding uniformly Gaussian noise, but for now, merely quantized noise will suffice.

Now we have a noisy sample v from $H^{\#}$. This sample is put through A.2, as detailed in a previous section. The output of A.2 is a predicted generator matrix M_0 for H. So to check if M_0 and M are equivalent, we simply compare their kernels. The experiment is said to succeed only if these kernels match exactly.

This loop of generating a sample and then running A.2 is usually repeated 10-20 times and comprises our core experimentation loop. The objective then is to have a successful loop while simultaneously minimizing the value of Q. Recall that Q represents the resolution of the hypercube space, and thus is directly correlated with the number of qubits needed in a hypothetical implementation. We increase the parameters R, T, Q in that order, in an attempt to improve the success rate of A.2. In line with the definition of the BQP complexity space, we consider a valid set of parameters to occur when a loop has a success rate over 2/3.

3.4 Multiple Samples

One result not explored in Kuperberg's paper is the usage of multiple samples. Since a hypothetical quantum computer makes Algorithm A.1 relatively cheap, it is useful to take advantage of multiple samples $v_1, v_2, \ldots, v_m$. So let m be the number of samples taken from A.1. In our simulation, each sample v_i is chosen independently through the same method above.

Now A.2 needs to be modified to accommodate multiple samples, but only slightly. Appending m samples to the standard basis vectors in A.2 gives a $(k+m) \times (k+m)$ matrix to which LLL is applied, instead of $(k+1) \times (k+1)$. From there, A.2 proceeds as normal, deleting ℓ rows and columns to output a $(k+m-\ell) \times (k+m-\ell)$ matrix, instead of a $(k+1-\ell) \times (k+1-\ell)$ matrix. The full modification to the original implementation of A.2 can be outlined in the following table.

Variable	Dimension	Step
B	$(k+m) \times (k+m)$	Append m samples to basis vectors
B	$(k+m) \times (k+m)$	Apply LLL to B
B_1	$(k+m) \times (k+m-\ell)$	Filter column vectors by norm
B_2	$(k+m-\ell) \times (k+m-\ell)$	Filter row vectors by a greedy determinant algorithm
B_3	$(k+m) \times (k+m-\ell)$	Multiply by $B_1 B_2^{-1}$
B_0	$(k+m) \times (k+m-\ell)$	Perform CFA
B_4	$\ell \times (k+m-\ell)$	Delete basis vectors
A_0	$(k+m) \times \ell$	Perform concatenation
A_1	$k \times \ell$	Put A_0 in SNF

Table 3: A summary of the dimensions of intermediary matrices in A.2.

Using this concept of multiple samples, we can now attempt to bound the algorithm parameters in terms of k and m. Let N be the maximum absolute value of any coordinate,

of any generator of H. If v is any generator of H, then $|v| \leq N\sqrt{k}$.

Now consider b_j, which is any vector produced by the LLL algorithm with norm less than $1/R$. Then by the Cauchy-Schwarz inequality,

$$|v \cdot b_j| \leq \frac{N\sqrt{k}}{R}$$

We want b_j to be on the stripe of $H^{\#}$ which passes through 0. We also know a priori that $|v \cdot b_j| \in \mathbb{Z}$. Combining these two facts, it suffices to have

$$\frac{N\sqrt{k}}{R} < 1 \implies R > N\sqrt{k} + \epsilon$$

where the ϵ term accounts for quantized noise in the measurement.

We now consider the volume of $H^{\#}$ compared to the ball of the radius $1/R$ centered at the origin, which we colloquially call the 'bullseye' B. The number of valid points is equal to the ratio of these two values, which inside

$$\frac{\operatorname{vol} H^{\#}}{\operatorname{vol} B} = \frac{(N\sqrt{k})^{\ell}}{(1/R)^{k-\ell}} = N^k$$

On the other hand, the number of samples we have is $(T/R)^m$, recalling that m is the number of measurements taken. So we want

$$\left(\frac{T}{R}\right)^m \geq N^k \implies T = \Omega(N^{1+k/m})$$

This finally gives the bound

$$Q \geq 2RT = 2kN^{2+k/m} = \Omega(kN^{2+k/m})$$

We will use this heuristic as a baseline in the results that follow.

4 Results

In this section, we review the results of our simulation. The code repository, along with any comments or analyses not covered in this dissertation, can be found at [39].

The results are organized as follows, centered around optimizing for the number of qubits $\log Q$. We first outline all the parameters we need to control for when running the algorithm. We then modify one variable at a time to determine its influence on Q. After introducing m, the number of samples, we focus on the interaction between m and the other variables.

4.1 Variables

As discussed in the bounds above, we set $R = \alpha N$, rounded as necessary, initially with $\alpha = \sqrt{k}$,. We then set $T = \beta R N^{k/m}$. with β initially set to 1. We then set $Q = 2RT$ and simulate A.2 20 times using these values of R, T, Q. As defined above, this procedure is successful if the pass rate is at least 2/3, i.e. at least 14 out of 20 attempts. If it is not successful, the value of β is multiplicatively increased and the process repeats again. The value of β continues to increase until we reach a large threshold, by default 10^{40}. If the search is still not successful, the value of α is increased until the search is successful.

The search loop for our implementation of A.2 is detailed in figure 5.

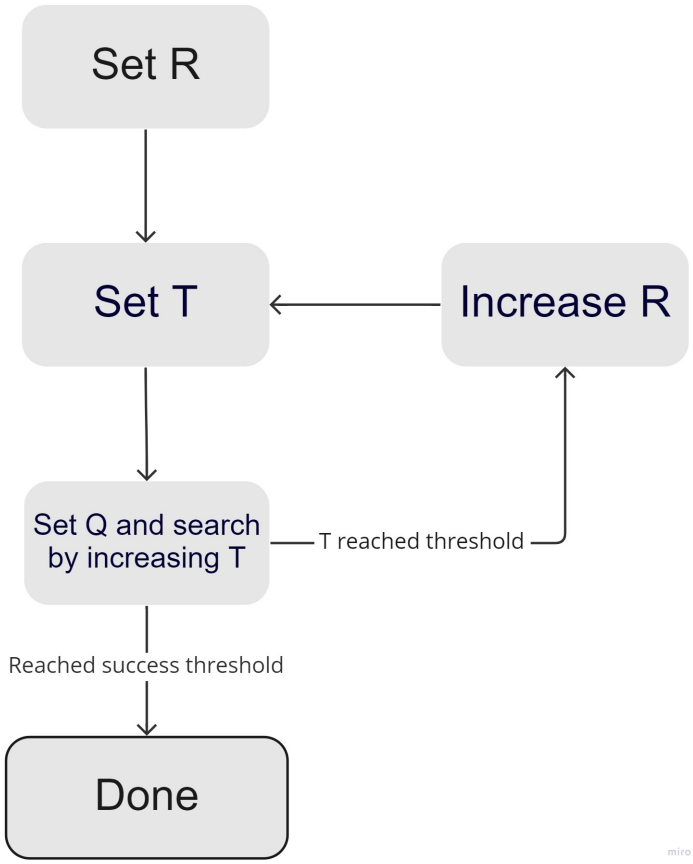


Figure 5: A flowchart describing our implementation of A.2, showing the simple loop over the search space.

The intermediate variables α, β, γ can be confirmed without relevant data being displayed.

We confirm that the bound of $Q \approx 2RT = \Omega(RT)$ is correct through the following experiment. When initially searching through the β space, we instead utilize an overly large value for Q, say $2000RT$ instead of $2RT$. After a valid β is found, we reset Q to $2\gamma RT$ and search over values of γ from 1/500 to 50. We found the values of γ to be around 1 and not much smaller or larger (by orders of magnitude). Moreover, initially modifying with value of γ did not affect the value of Q, within margin of error.

In the data below, $\log Q$ is displayed in base 2, to represent the number of theoretical qubits necessary. $\log N$ is displayed in base 10, since powers of 10 were used to explore different

values of N. All other logarithms are natural logs.

4.2 Varying N

We vary N and fix the other variables. For now, we will fix $m = 1$ and label our experiments (k, ℓ).

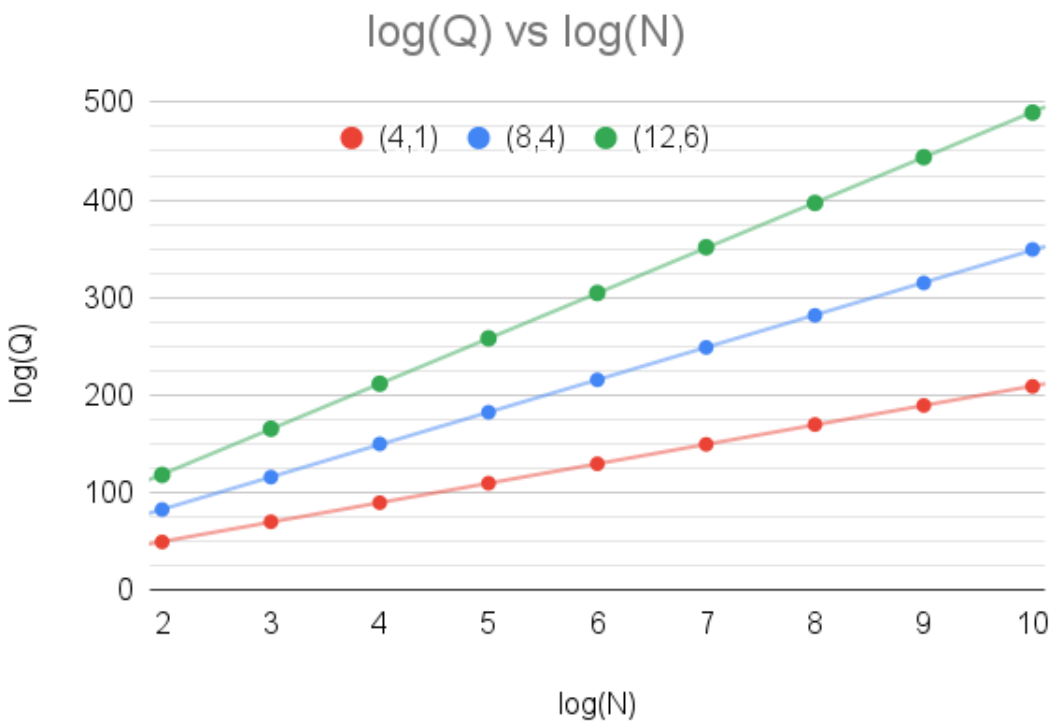


Figure 6: A plot of $\log Q$ against $\log N$. The lines presented are taken when $(k, \ell) = (4, 1), (8, 4)$, and $(12, 6)$.

We see the expected log-linear relationship from the scaling of $Q = O(RT) = O(N^{k+2})$. So we can henceforth fix N without loss of generalization; to speed up computation, we choose $N = 100$. We also note that the slope of each line varies and k and ℓ change; we explore this relationship further.

4.3 Varying k and ℓ

We vary k and fix the other variables. We show three examples with different values of ℓ, again labeling them as (k, ℓ).

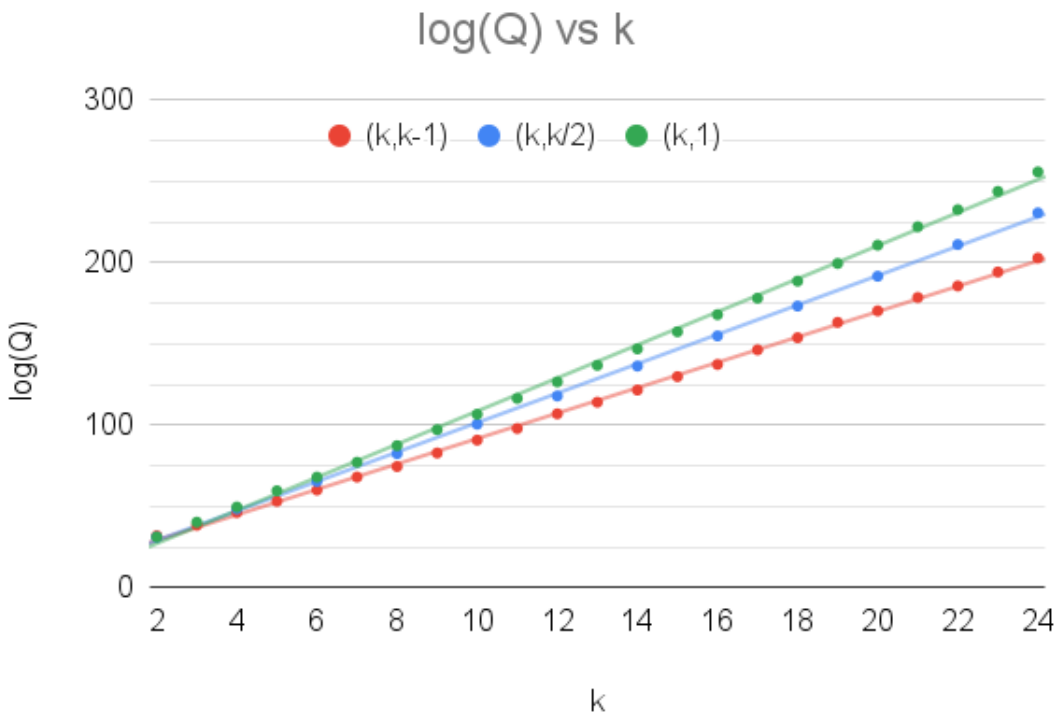


Figure 7: A plot of $\log Q$ against k. For a fixed k, the lines presented are taken when $\ell = 1, k/2$, and $k - 1$.

As k varies, we again see the log-linear relationship between k and Q. We also note that Q changes slightly as ℓ varies. As expected, $\ell = 1$ is the most difficult case, as Algorithm A has the least information to work with, and thus we expect Q to be larger.

We explore this by varying ℓ and fixing the other variables.

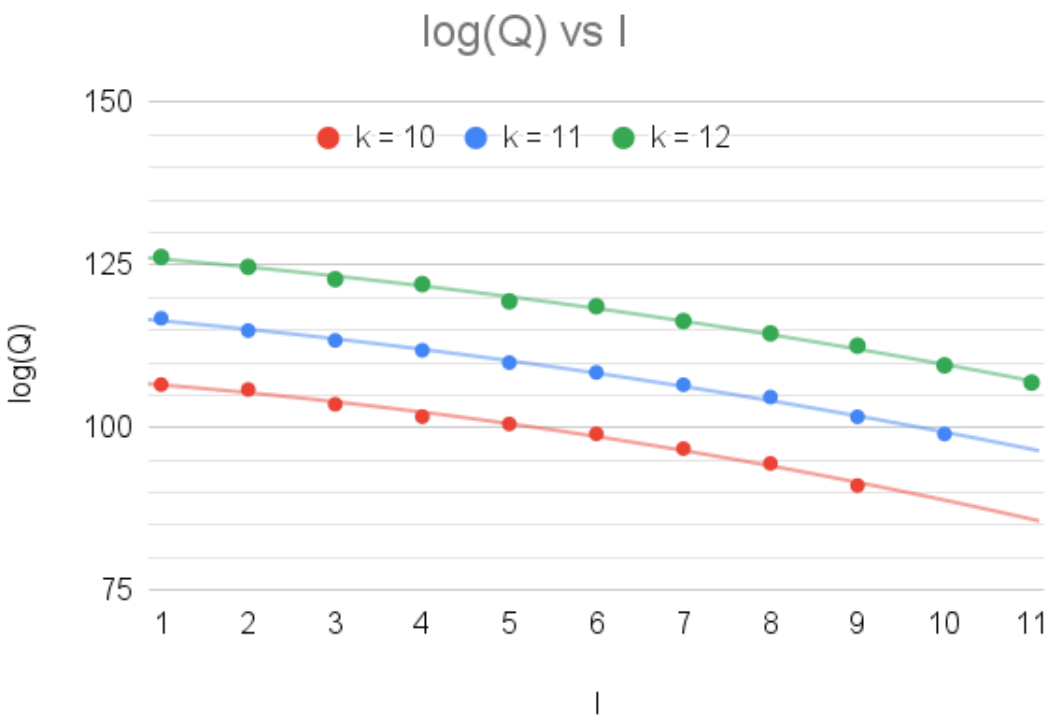


Figure 8: A plot of $\log Q$ against ℓ. The lines presented are taken when $k = 10, 11$, and 12.

We note that although ℓ does not appear explicitly in our estimation of Q, it nevertheless is correlated. By following the trendline, we can make a novel conjecture:

Conjecture 4.1. As ℓ varies from 1 to $k - 1$, the decrease in the number of qubits necessary is approximately $\ell + 0.08\ell^2$.

4.4 Varying m

We now introduce multiple samples by varying m and fixing the other variables. We first take the simplest nontrivial case, where $k = 2, \ell = 1$.

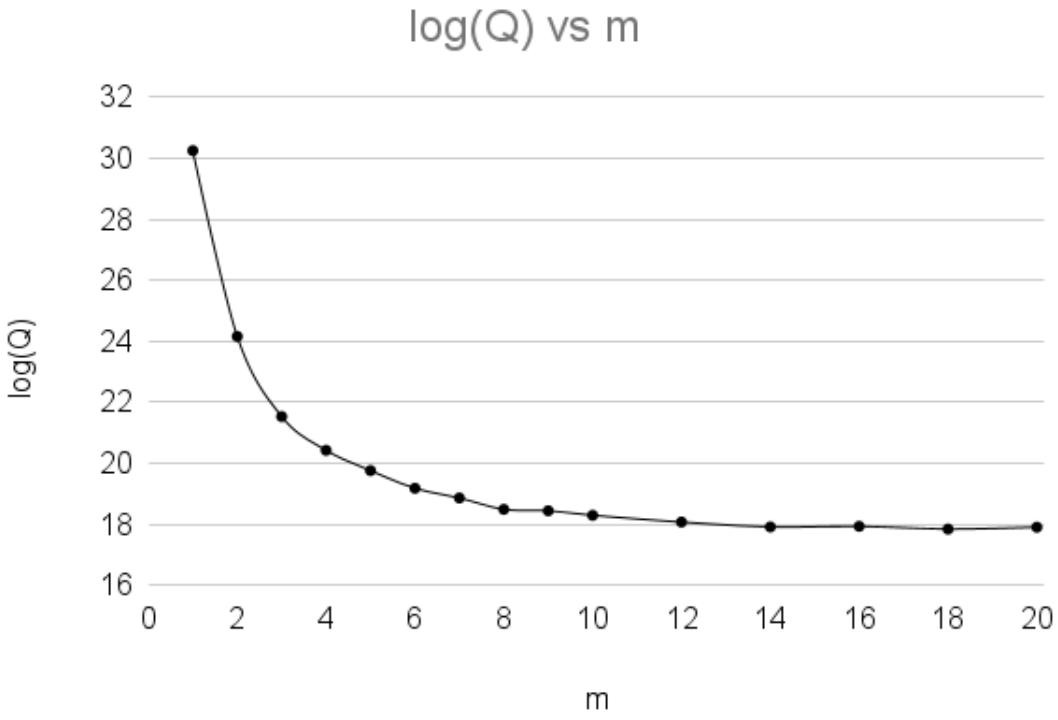


Figure 9: A plot of $\log Q$ against m. Here $k = 2$ and $\ell = 1$.

We can see the expected effect of varying m for different values of N, k, and ℓ. Since $Q = \Omega(kN^{2+k/m})$, as we increase m to infinity, we move towards the asymptotic bound $Q = \Omega(kN^2)$. We also verify this against varying m along with the other variables.

We fix $k = 6, \ell = 1$, and vary N.

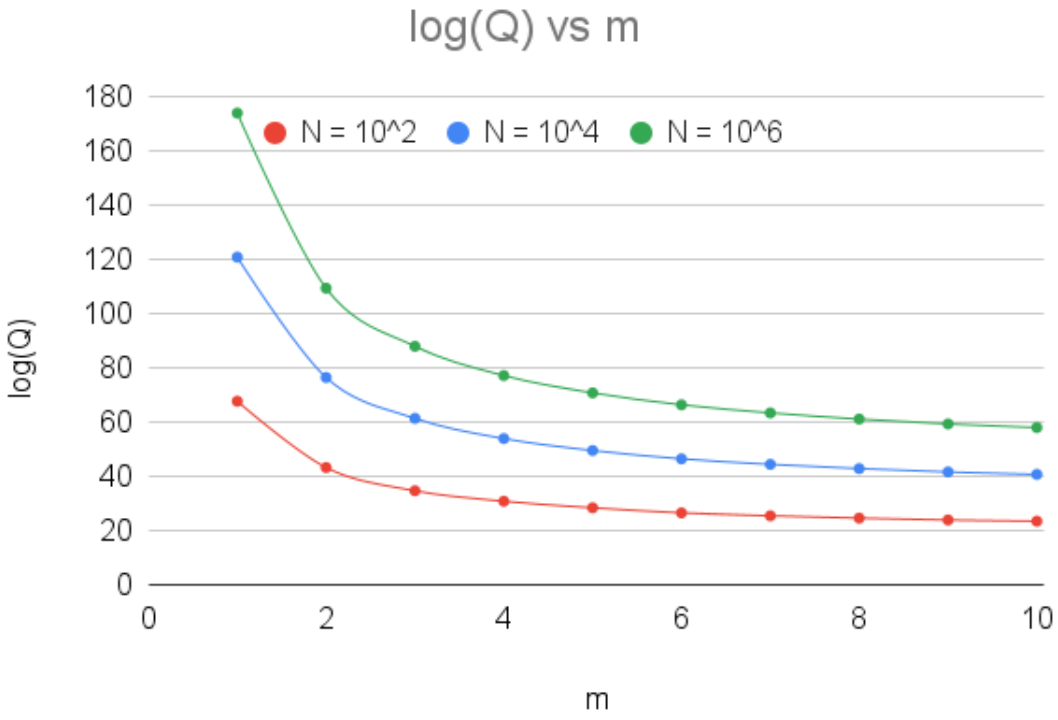


Figure 10: A plot of $\log Q$ against m. Here we fix $(k, \ell) = (6, 1)$, and the lines presented are taken when $N = 10^2, 10^4$, and 10^6.

We set $N = 10^2, \ell = 1$, and vary k.

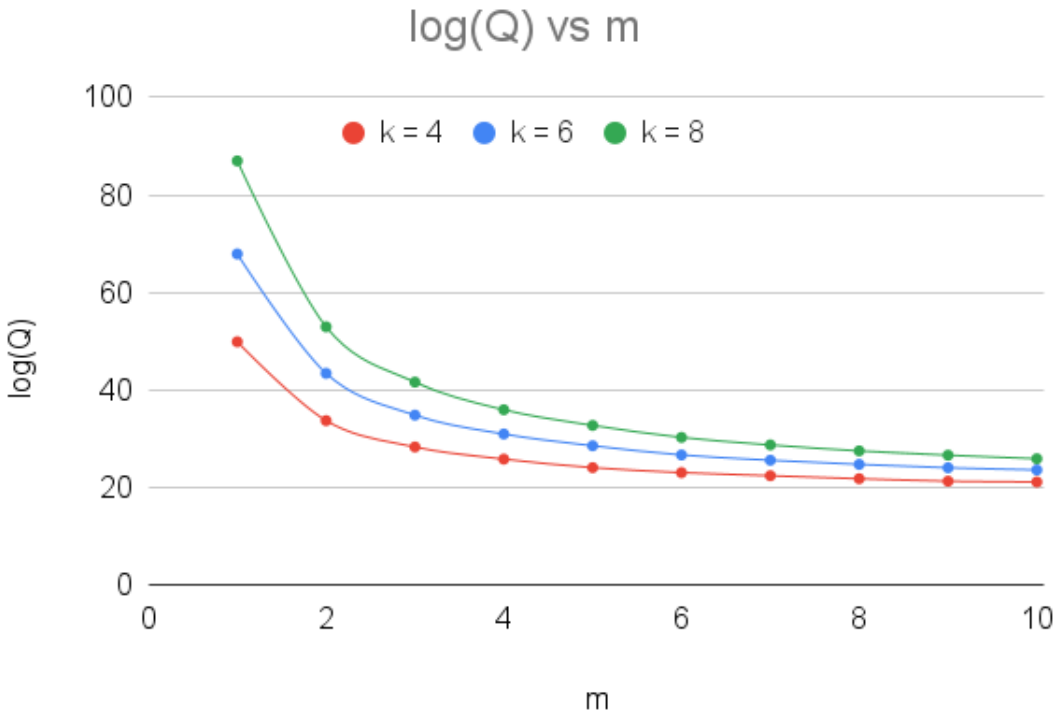


Figure 11: A plot of $\log Q$ against m. Here we fix $N = 100$ and $\ell = 1$, and the lines presented are taken when $k = 4, 6, 8$.

We set $N = 10^2, k = 6$ and vary ℓ in the same way as in figure 7.

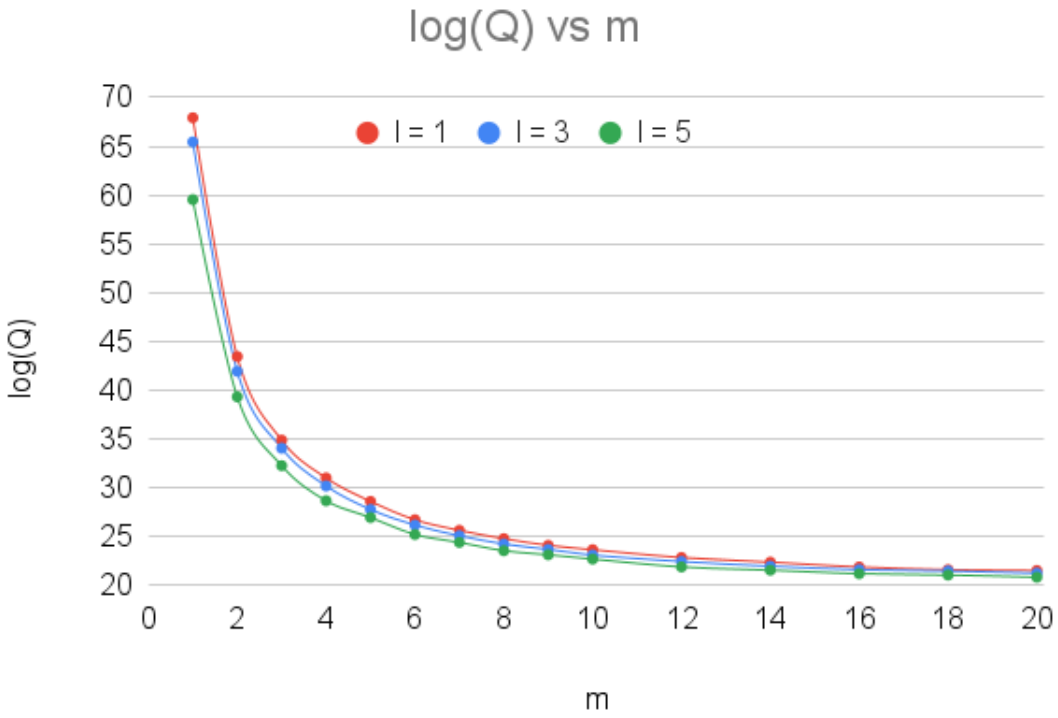


Figure 12: A plot of $\log Q$ against m. Here we fix $N = 100$ and $k = 6$, and the lines presented are taken when $\ell = 1, 3, 5$.

Increasing m is computationally expensive, even classically. Tracking time usage is outside the scope of this dissertation, since there are a number of external variables to control. For instance, calculations using a C++ implementation on an external cluster, both for increased consistency, would be more accurate in documenting the time complexity as well. For now the reader must be satisfied with the empirical claim that increasing m results in the simulation taking noticeably longer.

We have empirically confirmed our main result, which is the expected reduction in Q as m increases. We do note that the diminishing returns on increasing m are not necessarily free, and can become a liability. Since no explicit hiding functions f are yet known, we are assuming that querying f can be done in logarithmic or constant, which is not always the case.

Lastly, the noise modeled as input A.2, while actually smaller than the theoretical minimum, is nevertheless an inaccurate reflection of reality. Using a classical simulation of A.1, we could use the output of A.1 in an attempt to model the noise distribution. Even though such a classical simulation is severely limited in scope and size constraints, this could give us valuable information on how to model that noise for input to A.2. We include our simulation of A.1 in the repository linked in the appendix, but such a model is outside the scope of this dissertation.

4.5 Additional Exploration

We look at two types of open problems: first where exploration has already been done, and further research into Kuperberg's algorithm that is outside the scope of this dissertation, or in general thought to be intractable.

The exact discrepancy between the observed value of T and $RN^{k/m}$ increases, but at a predictable rate. This value was denoted by β, and we can observe its growth below.

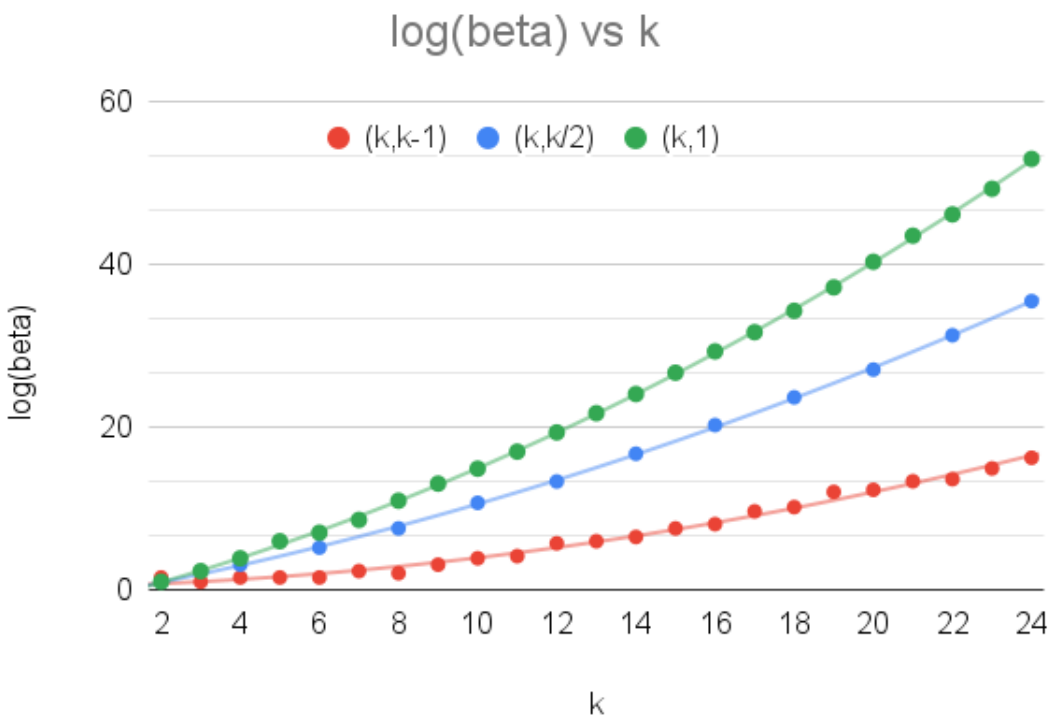


Figure 13: A plot of $\log \beta$ against k. Here we fix $N = 100$, and the lines presented are taken when $\ell = 1, k/2$, and $k - 1$.

This suggests that there is another term hidden in the approximation of T, dependent on both k and ℓ, which remains unknown. Based on the observed data, we estimate it to be exponential in k^2. The following trendlines are approximately, with $R^2 > .99$:

1. When $\ell = 1$, $\log \beta$ is approximately $-1.67 + 1.24k + 0.0432k^2$.
2. When $\ell = k/2$, $\log \beta$ is approximately $-0.929 + 0.892k + 0.0262k^2$.
3. When $\ell = k - 1$, $\log \beta$ is approximately $0.478 + 0.125k + 0.0228k^2$.

The value is that practically, when searching in β space, we can use these trendlines and effective lower bounds for β, instead of starting from 1. In general, we make the following conjecture:

Conjecture 4.2. The value of T in A.2 depends not only on R, N, k, m, but also on ℓ. In

particular, the approximation $T \approx \Omega(RN^{k/m})$ contains an additional factor of $\Omega(k^2)$, which depends on ℓ.

Recall from the previous section that ℓ appeared to be correlated with Q. If we compare ℓ with β instead, we can see that this correlated is most likely inherent to β:

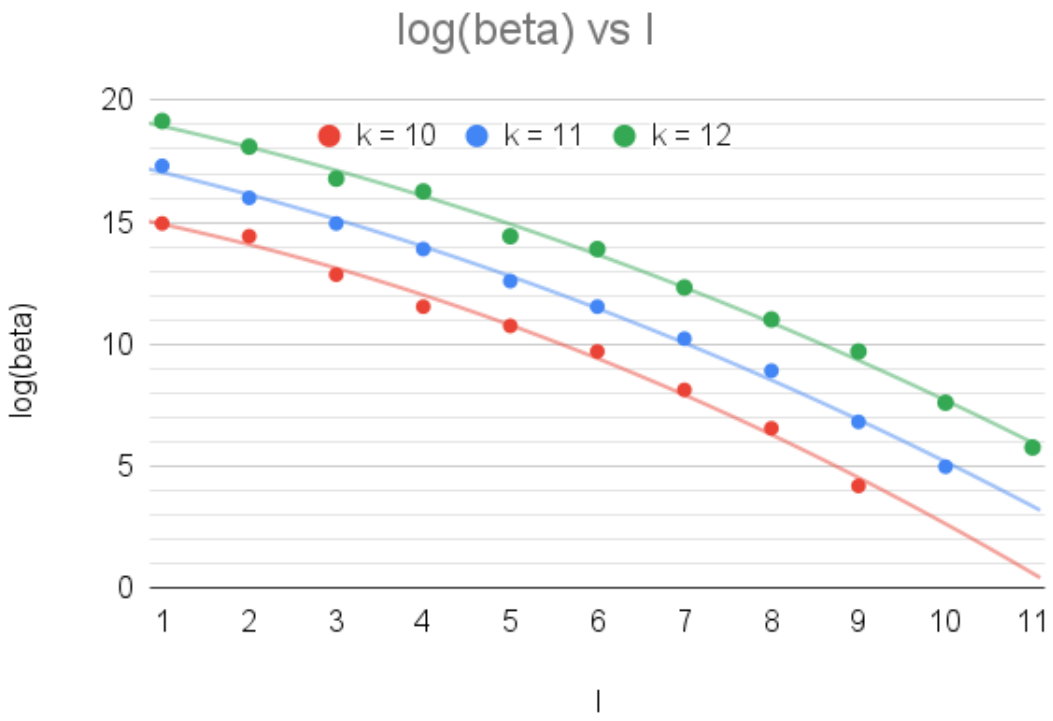


Figure 14: A plot of $\log \beta$ against ℓ. Here we fix $N = 100$, and the lines presented are taken when $k = 10, 11, 12$.

The reduction in β here is exactly proportional to the reduction of Q above.

Another interesting note is that the theoretical value of $R = N\sqrt{k}$ can be improved upon. If we start the search from $R = 0$ instead of $N\sqrt{k}$, we find that smaller values of R work well and substantially reduce $\log Q$.

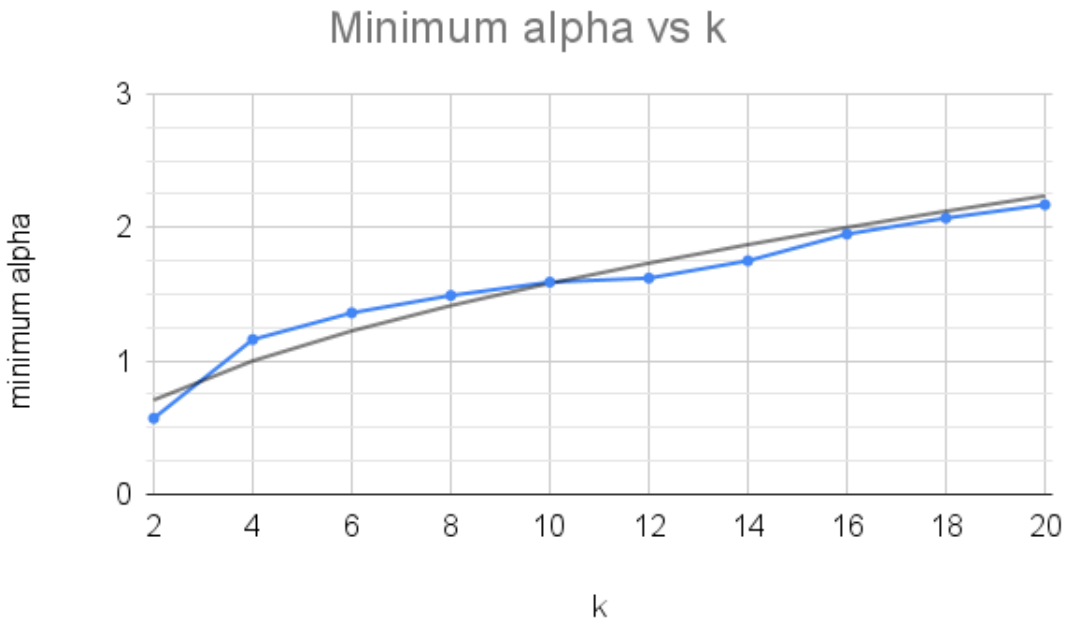


Figure 15: A plot of α against k. Here we fix $N = 100$ and $\ell = 1$, and we plot the line $\alpha = \sqrt{k}/2$ in black.

Remarkably, the linear relationship between $\log Q$ and $\log N$ is maintained. This suggests that the heuristics on R, T, Q are correct, but not necessarily optimal. For larger values of k, this would allow us to significantly optimize the value of Q by reducing the value of R beyond its theoretical minimum, as seen in the chart below.

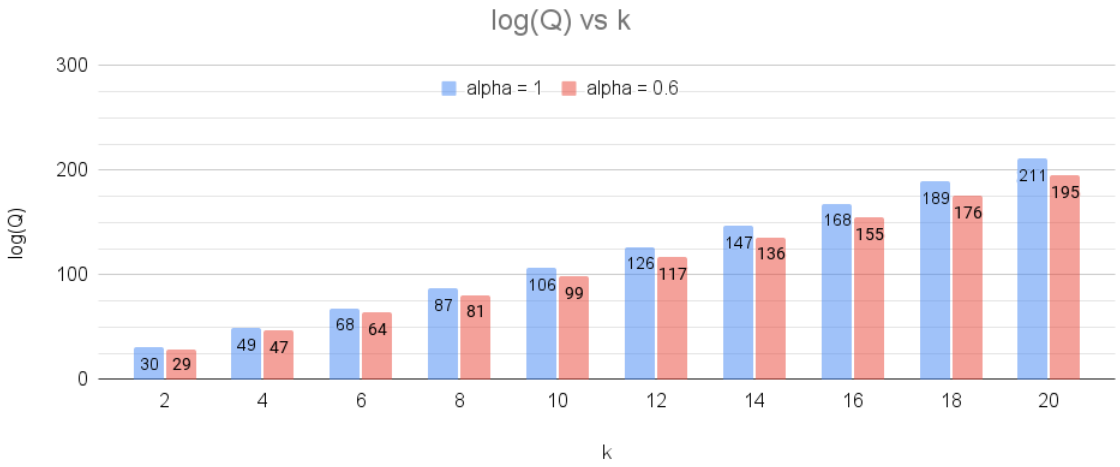


Figure 16: A chart of $\log Q$ against k. Here we fix $N = 100$ and $\ell = 1$. The values for $\log Q$ when $\alpha = 1$ are shown in blue, and the values for $\log Q$ when $\alpha = 0.6$ are shown in red. The labels are rounded to the nearest integer.

Decreasing α from 1 to 0.6 provides an approximate 7% reduction in the number of qubits needed, as shown in figure 16. The value of 0.6 was chosen for simplicity in calculations, but we can conjecture that:

Conjecture 4.3. The minimum value for α is not $\sqrt{k}$ as previously derived, but in fact approximately $\frac{1}{2}\sqrt{k}$. Furthermore, this bound on the minimum value of α is independent of ℓ as well.

4.6 Open Problems

We now transition into discussing broader topics that could serve as lofty goals for further research. The quantum part of Algorithm A, i.e. A.1, was explored but was ultimately outside the scope of this dissertation. A simulation of A.1 can easily be constructed as long as the ambient dimensions, k and ℓ, remain relatively low. The classical bottleneck comes from calculating the truncated finite geometric sums that result from intersecting the infinite quantum state with a hypercube. Although cases are more tractable when $\ell = 1$,

further research is needed for much larger values of Q, k, and ℓ.

A central question unanswered by Kuperberg's preprint is the possible substitution of the LLL algorithm. There are other algorithms, such as the AKS or BKZ algorithm [26], that have a different trade-off between time complexity and accuracy, i.e. the shortness of the output vector. The output LLL is not always sufficiently optimal, so in some circumstances it could be worthwhile to substitute LLL for a slower reduction algorithm, if that algorithm would result in a higher chance of success. Specifically in A.2, LLL acts on a $k + m$-dimensional lattice. So in the case of multiple samples, if m becomes much larger than k, the bounds on the shortness of the output of LLL for a fixed k increases.

As an extreme example, the AKS algorithm solves exact SVP, instead of approximate SVP, with probability much closer to 1, but it requires exponential time and space complexity. This guaranteed shortest vector would undoubtedly give a higher chance of success, if all other variables remained the same. The alternative to sticking with LLL is to simply increase the other variables, R and T, which directly leads to an increase in Q. The tradeoff ratio for the application of other algorithms to Algorithm A remains an open question.

5 Conclusion

The progress on AHSP from Kuperberg's paper and our subsequent simulations continues the process of how AHSP developed historically. The advancement of HSP algorithms from Simon's algorithm to Shor's algorithm, SKA, and finally Kuperberg's algorithm have shown similarities in their quantum components. We used the modern framework of explanation of SKA to restate Shor's algorithm as well as SKA itself, which is presented in a different context in Kitaev's original paper [19]. As we saw in Section 5, it is useful to think of classical post-processing of Shor's algorithm and the SKA as the resolution

of an approximation of a point in a reciprocal group. The difference is that the classical component of SKA assumes a finite index subgroup. Framed as a lattice problem, this makes it easier to correct for noise. However, in the infinite index case, we needed more powerful tools, which in the case of Algorithm A is the LLL algorithm.

Kuperberg's usage of Algorithm A and its reliance on Gaussian shaping assume more noise than is probably necessary, in some optimized version of Algorithm A. In addition to the quantized noise that comes from CFA, Kuperberg also introduces Gaussian noise in the quantum component. With computer simulations, we show that the reliance on Gaussian noise can be substantially reduced, and even eliminated, with no impact on the success rate. In all our experiments, we confirm the heuristics on R, T, Q that are derived from Kuperberg's paper. We also confirm our heuristics on the use of multiple samples. Our computer simulations also showed that multiple samples effortlessly lead to a significant classical speedup, again with no impact on the success rate. These simulations also uncovered variables dependency, specifically on β and ℓ, that were not previously found in Kuperberg's proofs. Further research could include exploration of different types of noise introduced after the quantum algorithm A.1. For instance, we could see whether the "fat tails" of the square wave noise resulting from quantization are actually negligible in impacting the final result.

In our discussion of nontrivial instances, we saw that quantum algorithms could be separated into three sets by usefulness: algorithms that have no known nontrivial instance, those that have a nontrivial instance but no cryptographic application, and those that have such an application. For Kuperberg's Algorithm A to find mainstream attention in the same vein as Shor's algorithm, it must have at least a nontrivial instance, which is yet to be found. Looking at historical context, we saw that preceding algorithms such as Simon's algorithm and SKA did not have known nontrivial instances. However evasive these instances seem to be, we remain cautiously optimistic about the future, again looking at historical context.

There was a decade between Kuperberg's first results on the hidden shift problem [22] and a cryptographic application [27]. Similarly, Eisenträger et. al. [12] found a nontrivial instance for HSP in $\mathbb{R}^k$, but left cryptographic applications as an open question.

The discussion of such quantum algorithms is central to developments in post-quantum and homomorphic cryptography. If a practical fully homomorphic encryption scheme existed, then any nontrivial instance becomes a cryptographic application, accelerating the usefulness of any quantum algorithm. In such a scenario, the structure of the hiding function can be losslessly encrypted and hidden from the computer, as long as such an encryption scheme does not use noisy homomorphic encryption, which for example Regev's scheme uses [30]. So even a trivial hiding function which encodes its hidden subgroup can function as part of a cryptographic scheme. We conjecture that multiple samples will prove an effective tool for many quantum algorithms involving the QFT.

We also see the usefulness of the LLL algorithm as a bootstrapping method to extract additional information from deficient-rank lattices. Many such applications use a nonabelian instance of HSP, for example CSIDH and its attacks [8]. Regev's 2009 paper implements a similar idea to reducing noise from geometric sums in the context of post-quantum cryptographic security. Analyses similar to our own can be performed with other quantum algorithms that align with the following questions: could they benefit from less noise? Are they able to efficiently make use of multiple samples? In general, we hope that our research and methodology presented here is beneficial to the future development of quantum algorithms, as well as cryptographic applications.

References

[1] Scott Aaronson. "BQP and the polynomial hierarchy". In: *Electron. Colloquium Comput. Complex.* 16 (2010), p. 104.

[2] Scott Aaronson and Greg Kuperberg. "Quantum versus Classical Proofs and Advice". In: July 2007, pp. 115–128. ISBN: 0-7695-2780-9. DOI: 10.1109/CCC.2007.27.

[3] Lidong Chen et. al. *Report on Post-Quantum Cryptography*. en. Apr. 2016. DOI: https://doi.org/10.6028/NIST.IR.8105.

[4] Eric Allender and et. al. *Recent Advances Towards Proving P=BPP*. 1998.

[5] Ritik et. al. Bavdekar. *Post Quantum Cryptography: Techniques, Challenges, Standardization, and Directions for Future Research*. 2022. DOI: 10.48550/ARXIV.2202.02826. URL: https://arxiv.org/abs/2202.02826.

[6] Daniel J. Bernstein and A. K. Lenstra. "A general number field sieve implementation". In: *The development of the number field sieve*. Ed. by Arjen K. Lenstra and Hendrik W. Lenstra. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 103–126. ISBN: 978-3-540-47892-8.

[7] Dan Boneh and Richard J. Lipton. "Quantum Cryptanalysis of Hidden Linear Functions". In: *Advances in Cryptology — CRYPT0' 95*. Ed. by Don Coppersmith. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 424–437.

[8] Wouter et. al. Castryck. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *Advances in Cryptology – ASIACRYPT 2018*. Ed. by Thomas Peyrin and Steven Galbraith. Cham: Springer International Publishing, 2018, pp. 395–427.

[9] Andrew Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *Journal of Mathematical Cryptology*

8.1 (Jan. 2014), pp. 1–29. ISSN: 1862-2984. DOI: 10.1515/jmc-2012-0016. URL: http://dx.doi.org/10.1515/jmc-2012-0016.

[10] Andrew M. Childs and Wim van Dam. "Quantum Algorithm for a Generalized Hidden Shift Problem". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2005, pp. 1225–1232. ISBN: 9780898716245.

[11] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Inf. Theory* 22 (1976), pp. 644–654.

[12] Kirsten et. al. Eisenträger. "A Quantum Algorithm for Computing the Unit Group of an Arbitrary Degree Number Field". In: *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '14. New York, New York: Association for Computing Machinery, 2014, pp. 293–302. ISBN: 9781450327107. DOI: 10.1145/2591796.2591860. URL: https://doi.org/10.1145/2591796.2591860.

[13] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology*. Ed. by George Robert Blakley and David Chaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18. ISBN: 978-3-540-39568-3.

[14] Mark Ettinger, Peter Høyer, and Emanuel Knill. "The quantum query complexity of the hidden subgroup problem is polynomial". In: *Inf. Process. Lett.* 91 (2004), pp. 43–48.

[15] Oded Goldreich. "In a World of P=BPP". In: *Electronic Colloquium on Computational Complexity (ECCC)* 17 (Sept. 2010), p. 135. DOI: 10.1007/978-3-642-22670-0_20.

[16] Sean Hallgren. "Polynomial-Time Quantum Algorithms for Pell's Equation and the Principal Ideal Problem". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: Association for Com-

puting Machinery, 2002, pp. 653–658. ISBN: 1581134959. DOI: 10.1145/509907.510001. URL: https://doi.org/10.1145/509907.510001.

[17] R. Jozsa. "Quantum factoring, discrete logarithms, and the hidden subgroup problem". In: *Computing in Science and Engineering* 3.2 (2001), pp. 34–43. ISSN: 1521-9615. DOI: 10.1109/5992.909000. URL: http://dx.doi.org/10.1109/5992.909000.

[18] Richard Jozsa. "Quantum algorithms and the Fourier transform". In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454 (1998), pp. 323–337.

[19] A. Yu. Kitaev. *Quantum measurements and the Abelian Stabilizer Problem*. 1995. arXiv: quant-ph/9511026 [quant-ph].

[20] Pascal Koiran, Vincent Nesme, and Natacha Portier. "A Quantum Lower Bound for the Query Complexity of Simon's Problem". In: *Automata, Languages and Programming*. Ed. by Luís Caires et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1287–1298. ISBN: 978-3-540-31691-6.

[21] Manish Kumar. *Post-Quantum Cryptography Algorithms Standardization and Performance Analysis*. 2022. DOI: 10.48550/ARXIV.2204.02571. URL: https://arxiv.org/abs/2204.02571.

[22] Greg Kuperberg. *A subexponential-time quantum algorithm for the dihedral hidden subgroup problem*. 2004. arXiv: quant-ph/0302112 [quant-ph].

[23] Greg Kuperberg. *Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem*. 2011. arXiv: 1112.3333 [quant-ph].

[24] Greg Kuperberg. *The hidden subgroup problem for infinite groups*. Dec. 2020.

[25] Michele Mosca and Artur Ekert. *The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer*. 1999. DOI: 10.48550/ARXIV.QUANT-PH/9903071. URL: https://arxiv.org/abs/quant-ph/9903071.

[26] Phong Q. Nguyen. "Lattice Reduction Algorithms: Theory and Practice". In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 2–6. ISBN: 978-3-642-20465-4.

[27] Chris Peikert. *He Gives C-Sieves on the CSIDH*. Cryptology ePrint Archive, Report 2019/725. https://ia.cr/2019/725. 2019.

[28] *Post-Quantum Cybersecurity Resources*. URL: https://www.nsa.gov/Cybersecurity/Post-Quantum-Cybersecurity-Resources/.

[29] Oded Regev. *A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space*. 2004. DOI: 10.48550/ARXIV.QUANT-PH/0406151. URL: https://arxiv.org/abs/quant-ph/0406151.

[30] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". English (US). In: *Journal of the ACM* 56.6 (Sept. 2009). ISSN: 0004-5411. DOI: 10.1145/1568318.1568324.

[31] Oded Regev. "Quantum Computation and Lattice Problems". In: *SIAM J. Comput.* 33 (2004), pp. 738–760.

[32] Elaine Rich. "Automata, Computability and Complexity: Theory and Applications". In: 2007. Chap. 28.10, pp. 689–694.

[33] Alexander Rostovtsev and Anton Stolbunov. "Public-Key Cryptosystem Based on Isogenies". In: *IACR Cryptol. ePrint Arch.* 2006 (2006), p. 145.

[34] *Round 3 submissions - post-quantum cryptography: CSRC*. Dec. 2021. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[35] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[36] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.

[37] Daniel R. Simon. "On the power of quantum computation". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 116–123.

[38] Murugiah Souppaya, William Polk, and William Barker. *Getting Ready for Post-Quantum Cryptography: Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms*. en. Apr. 2021. DOI: https://doi.org/10.6028/NIST.CSWP.04282021. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=932330.

[39] Austin Tran. *AlgorithmA2*. 2022. URL: https://github.com/ant314159265/AlgorithmA2.

A Appendix

A.1 Lists

List of Tables

List of Figures

A.2 Notation

- $A \propto B$: A is proportional to B, i.e. their quotient is a constant.
- e_i: the ith standard basis vector, i.e. the vector of k zeros, except the ith coordinate is 1. The ambient space $\mathbb{R}^k, k \geq i$ is assumed.
- $H \leq G$: H is a subgroup, not necessarily proper, of G.
- $\widehat{H}$: the Pontryagin dual of a subgroup $H \leq G$.
- $\Pr(X)$: the probability of X.
- $\mathbb{T}$: the torus $\mathbb{R}/\mathbb{Z}$, which unless specified otherwise is taken to be in $(-1/2, 1/2]$.
- U_f: the quantum unitary dilation of a function f.
- $\mathbb{Z}/p$: used as an abbreviation of $\mathbb{Z}/p\mathbb{Z}$.

A.3 Abbreviations

- AHSP: abelian hidden subgroup problem.
- ASP: abelian stabilizer problem.
- BPP: bounded-error probabilistic polynomial time.
- BQP: bounded-error quantum (probabilistic) polynomial time.
- CFA: continued fraction approximation.
- CSIDH: commutative supersingular isogeny Diffie-Hellman [8]
- DLP: discrete logarithm problem.

- HSEP: hidden subgroup existence problem.
- HSP: hidden subgroup problem.
- LLL: the Lenstra-Lenstra-Lovász algorithm.
- QFT: quantum Fourier transform.
- SKA: Shor-Kitaev algorithm.
- SNF: Smith normal form.
- SVP: short vector problem.