Quantum Lattice Sieving

Nishant Rodrigues * † Brad Lackey ‡

October 27, 2021

Abstract

Lattices are very important objects in the effort to construct cryptographic primitives that are secure against quantum attacks. A central problem in the study of lattices is that of finding the shortest non-zero vector in the lattice. Asymptotically, sieving is the best known technique for solving the shortest vector problem, however, sieving requires memory exponential in the dimension of the lattice. As a consequence, enumeration algorithms are often used in place of sieving due to their linear memory complexity, despite their super-exponential runtime. In this work, we present a heuristic quantum sieving algorithm that has memory complexity polynomial in the size of the length of the sampled vectors at the initial step of the sieve. In other words, unlike most sieving algorithms, the memory complexity of our algorithm does not depend on the number of sampled vectors at the initial step of the sieve.

1 Introduction

Lattices have become attractive tools in the design of post-quantum cryptographic protocols. This is evident from NIST’s recent Post Quantum Cryptography Standardization competition, where three out of the four Round 3 finalists in the Public Key Encryption category, and two out of the three for Digital Signatures are lattice based. Certain lattice problems like that of finding the shortest vector in a given lattice are believed to be hard to solve in the worst case, even using quantum computers. As a result, cryptanalysis of existing lattice-based cryptosystems is necessary as a means of understanding the complexity of solving hard lattice problems, and to set relevant parameters used in these cryptosystems.

In this work we focus on the shortest vector problem (SVP), which is defined as follows:

Definition 1 (Shortest Vector Problem - SVP). Given a lattice $\mathcal{L}$, find a non-zero $s \in \mathcal{L}$ such that $\|s\| = \lambda_1(\mathcal{L})$.

There are two main techniques for solving the SVP, sieving and enumeration. Sieving refers to the class of algorithms that process a list of lattice vectors at each sieve step, and produce shorter vectors for the next sieve step. Sieving gives exponential time algorithms for solving the exact Shortest Vector Problem. This is in contrast to enumeration which gives super-exponential algorithms for SVP. We do not discuss enumeration techniques, but provide references for the interested reader [Poh81, Kan83, FP85, SH95, GNR10, Sch10].

*Joint Center for Quantum Information and Computer Science, College Park, MD
†Department of Computer Science, University of Maryland, College Park, MD
‡Quantum Systems Group, Microsoft Quantum, Redmond, WA
We provide a brief discussion on sieving algorithms starting with the algorithm by Micciancio and Voulgaris in [MV10]. The algorithm starts with an empty list, and adds new vectors to it at each sieve iteration. At each step the algorithm reduces the new vector with the existing vectors in the list. As the sieve proceeds, the list contains smaller and smaller vectors, and eventually outputs the shortest one. By reducing the new vector with the other vectors in the list, they were able to prove a bound on the size of the list of vectors, which in turn, determines the runtime $\tilde{O}(2^{3.199d})$ and memory complexity $\tilde{O}(2^{1.325d})$ of the algorithm. Notice that the constant in the exponent is still very high for this algorithm to be practical for lattices of large dimension. A more recent algorithm by Aggarwal et al. uses Discrete Gaussian Sampling to solve SVP in time and space $2^{d+o(d)}$ [ADRSD15].

In recent years, there has been work on developing practical sieve algorithms that make heuristic assumptions, and show bounds on time and space required for the algorithm based on the heuristic assumption. Nguyen and Vidick gave a heuristic algorithm for solving SVP in [NV08] that runs in time $O(2^{0.415d})$ and space $O(2^{0.2075d})$. This practical variant makes the heuristic assumption that at each level the sieve vectors lie on the surface of the unit sphere, and are distributed uniformly and independently on it. This assumption, which we state in Heuristic (2), helps bound the total number of vectors needed in the initial step of the sieve. The NV-Sieve starts with a large initial set of vectors from the lattice. At each step of the sieve, the algorithm iterates over every vector in this set and checks to see if the vector has norm less than or equal to a specified quantity. If it does, then the vector is promoted to the next step of the sieve. If the norm is greater than the specified quantity, the algorithm searches in a list of centers to see if a center reduces with the current vector. If it does, the reduced vector is promoted to the next step of the sieve. If no center is found that reduces with the current vector, the current vector is added to the list of centers to use for reducing against future vectors. We can see how the structure of this algorithm resembles a real world sieve. At every sieve iteration, only shorter vectors go to the next level, and therefore at the end of the sieve, we are left with very short vectors.

Several other heuristic algorithms have been proposed, including the GaussSieve from [MV10] which is based on ListSieve but without any theoretical guarantees on runtime. Laarhoven proposed algorithms based on locality sensitive hashing in [Laa15] and [LdW15]. The current best known classical heuristic algorithm for SVP is due to [ACKS20] and is based on solving a related problem known as Bounded Distance Decoding. This algorithm runs in time $2^{1.741d+o(d)}$ and space $2^{0.5d+o(d)}$. Concurrently, there has been work on developing quantum speedups for these classical algorithms, usually by replacing the ‘search’ step in these algorithms with quantum Grover search that provides a quadratic speedup.

### 1.1 Algorithm overview

We give an overview of our quantum sieving algorithm that is based on the classical Double Sieve algorithm due to Bai et al. [BLS16]. The Double Sieve is not the best known classical algorithm, however, it is very simple to describe. This algorithm lends itself nicely to a quantum sieving algorithm, where our idea is to start with a superposition over lattice vectors, and search for shorter vectors using amplitude amplification at each step.

First we present the classical Double Sieve along with the heuristic assumptions made in the original paper, and then show how our quantum algorithm works. This algorithm uses memory of the order $O(2^{0.2075d})$ and has time complexity $O(2^{0.415d})$. Technical details of the algorithm can be found in the original paper, but we restate the heuristic and show some computations to derive bounds on the number of vectors needed in the initial step of the sieve, as this analysis will carry over to the quantum case. The algorithm starts by sampling an initial set of vectors $S$ from the given lattice $L$. The heuristic assumption made about this sample of vectors is as follows:
Heuristic 2. The vectors $\frac{v}{||v||}$ for $v \in S$ are distributed independently and uniformly on the surface of the unit sphere.

Using this heuristic, we can compute the number of lattice vectors needed in the initial set $S$. Given a fixed vector $v$ on the surface of the unit sphere, the density of vectors on the sphere that make an angle of $\theta$ with $v$ is given by approximately $\sin \theta$. Given two vectors of nearly equal length, $v_1$ and $v_2$, they reduce with each other, i.e. $||v_2 - v_1|| \leq \min\{||v_1||, ||v_2||\}$ only if the angle $\theta$ between them is less than $\frac{\pi}{3}$. Thus given a fixed vector on the unit sphere, the probability that another vector also on the sphere reduces with it, is given by $p = \sin \frac{\pi}{3} = \frac{\sqrt{3}}{2}$. We can think of this probability as the portion of the unit sphere that one vector covers, which is shown in Figure (1). In order for this probability to be close to 1, we need to cover the whole sphere, and so we need roughly $N \propto \frac{1}{p} = (\frac{4}{\sqrt{3}})^\frac{d}{2} = 2^{0.2075d}$ vectors. This gives us the number of vectors that we need to sample initially when the sieve starts.

The sieve algorithm proceeds by looking at the sum (and difference) of pairs of vectors and promotes the respective sum (or difference) which has norm less than the quantity $\gamma R$ to the next step of the sieve, where $\gamma$ is usually set to $1 - \frac{1}{d}$, and $R$ to the maximum norm of vectors at the current step of the sieve. Each step of the sieve uses the heuristic assumption. After a $\text{poly}(d)$ number of iterations of the sieve, we end up with sufficiently small vectors, which give a good approximation of the shortest vector. The algorithm is presented below:

```
Algorithm 1: Double Sieve [BLS16]

Input: $S, \gamma, R$
1 $S' \leftarrow \{\}$
2 for $u, v \in S$ do
3   if $||u \pm v|| \leq \gamma R$ then
4     $S' \leftarrow S' \cup \{u \pm v\}$
5 return $S'$
```

The main loop iterates over pairs of vectors, and hence the runtime is quadratic in $N$, yielding a running time of $O(2^{0.415d})$. The sieve starts with, and maintains a list of vectors of size $N$ at each iteration, and so the memory is $O(2^{0.2075d})$.

In the quantum sieve, we also sample $N = O(2^{0.2075d})$ vectors, but we do that in superposition instead of storing a classical list. Assume that we have the ability to sample a uniform superposition over $O(2^{0.2075d})$ lattice vectors. We tensor product two of these superpositions to give us a superposition over pairs of vectors.
in the lattice. At the initial stage, we fix \( R \) to be the maximum norm of the vectors in the lattice, as in the classical double sieve. Thus we have a superposition over pairs of vectors in the lattice whose norm is bounded by \( R \).

\[
\frac{1}{\sqrt{N}} \sum_{u_1 \in S, \|u_1\| \leq R} |u_1\rangle \otimes \frac{1}{\sqrt{N}} \sum_{u_2 \in S, \|u_2\| \leq R} |u_2\rangle = \frac{1}{N} \sum_{u_1, u_2 \in S} |u_1, u_2\rangle
\]

This is analogous to iterating over pairs of vectors in the classical double sieve. Next we compute the norm of the difference \( u_2 - u_1 \) in an auxiliary register:

\[
\frac{1}{N} \sum_{u_1, u_2 \in S} |u_1, u_2, \|u_2 - u_1\|\rangle \mapsto \frac{1}{N} \sum_{u_1, u_2 \in S} |u_1, u_2, \|u_2 - u_1\|\rangle
\]

We then perform amplitude amplification (A.A.) to find vectors such that \( \|u_2 - u_1\| \leq \gamma R \). This is analogous to steps 3 and 4 in Algorithm(1) stated above. As a result of the amplitude amplification step, we now have a uniform superposition over vectors in the lattice whose norm is bounded by \( \gamma R \), with high probability.

\[
\frac{1}{N} \sum_{u_1, u_2 \in S} |u_1, u_2, \|u_2 - u_1\|\rangle \xrightarrow{\text{A.A.}} \frac{1}{\sqrt{N}} \sum_{u_1, u_2 \in S} |u_1, u_2, \|u_2 - u_1\|\rangle + \phi_{\text{junk}}
\]

To simplify notation, we only show the first two registers, measure to discard the junk state, compute \( u_2 - u_1 \) in the first register using a unitary \( U_{\text{diff}} \), and rename \( u_2 - u_1 \) to \( v_1 \):

\[
\frac{1}{\sqrt{N}} \sum_{u_1, u_2 \in S, \|u_2 - u_1\| \leq \gamma R} U_{\text{diff}} |u_1, u_2\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{u_1, u_2 \in S, \|u_2 - u_1\| \leq \gamma R} |u_2 - u_1, u_2\rangle = \frac{1}{\sqrt{N}} \sum_{v_1 = u_2 - u_1} |v_1, u_2\rangle
\]

At this point we have successfully run one iteration of the sieve. We started with vectors with norm bounded by \( R \), and ended up with vectors with norm bounded by \( \gamma R \), thus effectively reducing the norm of the vectors in our sieve by a factor of \( \gamma \). In order to keep the sieve going, we need a second superposition over vectors with norm bounded by \( \gamma R \). We combine two such superpositions, run amplitude amplification, and obtain another superposition with vectors of norm bounded by \( \gamma^2 R \). This process produces a binary tree of superpositions, such that to run iteration \( i \) of the sieve, we have to start with \( 2^i \) superpositions at the initial step. Notice that after each level \( i \), we end up with a superposition state that looks like

\[
\frac{1}{\sqrt{N}} \sum_{\|w_1\| \leq \gamma^i R} |w_1, \text{history}_{2^{i-1}}\rangle
\]

where \( |\text{history}_{2^{i-1}}\rangle \) is a state on \( 2^i - 1 \) registers which contains the history of the lattice vectors that produced the vector \( w_1 \). In our presentation we do not need or use the history so we don’t explicitly write the history state, but we note that it exists, and is important to the memory analysis of the algorithm. We run \( t = \text{poly}(d) \) iterations of the sieve, just as in the classical double sieve, which uses \( 2^i \) superpositions. At the end we are left with a uniform superposition over very small vectors in the lattice, and measuring the state, we get a good approximation to the shortest vector in the lattice. The runtime and memory analysis along with the full algorithm is presented in Section (3).
2 Preliminaries

2.1 Notation

The norm of a vector \( \mathbf{v} = (v_1, \ldots, v_n)^\top \) is denoted \( \| \mathbf{v} \| \), and we take this norm to be the Euclidean norm, \( \| \mathbf{v} \| = \sqrt{\sum_{i=1}^{n} v_i^2} \). For a matrix \( \mathbf{B} \), we denote its \( i \)th column by \( \mathbf{b}_i \), and its norm by \( \| \mathbf{B} \| = \max_i \| \mathbf{b}_i \| \). We denote by \( B_n(\mathbf{v}, r) \) the ball of radius \( r \) around the vector \( \mathbf{v} \). Thus \( B_n(\mathbf{v}, r) = \{ \mathbf{y} \in \mathbb{R}^n \mid \| \mathbf{y} - \mathbf{v} \| \leq r \} \). We simplify notation and write \( B_n(0, r) = B_n(r) \), and \( B_n(\mathbf{0}, 1) = B_n \). We denote by \( S \) the \( d \)-dimensional unit sphere.

2.2 Lattices

Given \( d \) linearly independent vectors \( \{\mathbf{b}_1, \ldots, \mathbf{b}_d\} \) with each \( \mathbf{b}_i \in \mathbb{R}^n \), the lattice \( \mathcal{L} \) generated by them is denoted by the set of all integer linear combinations of the basis vectors, and written as \( \mathcal{L} = \{ \sum_{i=1}^{d} x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \} \). We will restrict our discussion to lattices of full rank i.e. \( n = d \), and denote using \( d \) the dimension of the lattice. Alternatively, the basis can be written as a matrix \( \mathbf{B} \) with the vectors forming the columns of \( \mathbf{B} \) which gives us \( \mathcal{L} = \{ \mathbf{B} \mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^d \} \). The volume of the lattice, denoted \( \text{vol}(\mathcal{L}) \), is given by \( |\det(\mathbf{B})| \), and is an invariant of the lattice. The norm of the shortest vector in the lattice is denoted \( \lambda_1(\mathcal{L}) \) and is known as the first minimum of the lattice. One of the central problems in the study of lattices is that of finding the shortest vector which we described in Section (1).

Several cryptographic primitives use other related average-case hard lattice problems like Small Integer Solutions (SIS) [MR07], Learning with Errors (LWE) [Reg09], and their variants as the computational hardness assumption when defining the security of the cryptosystem. The SIS problem is stated as follows:

**Definition 3** (Small Integer Solutions - SIS). Given \( n, m, q, \nu \) with \( \nu < q \), a matrix \( \mathbf{A} \in \mathbb{Z}_q^{m \times n} \), and lattice \( \mathcal{L} = \{ \mathbf{x} \in \mathbb{Z}^m : \mathbf{A} \mathbf{x} \equiv 0 \mod q \} \), find \( \mathbf{y} \in \mathcal{L} \) s.t. \( \| \mathbf{y} \| \leq \nu \)

The Learning with Errors problem is stated as follows:

**Definition 4** (Learning with Errors - LWE). Let \( s \in \mathbb{Z}_q^m \), for a given \( n \) and modulus \( q \). Let \( \chi \) be a probability distribution on \( \mathbb{Z}_q^n \). We define a probability distribution \( A_{s, \chi} \) with sampling as follows: draw \( \mathbf{a} \in \mathbb{Z}_q^n \) at uniform, and \( e \in \mathbb{Z}_q \) according to \( \chi \). Then return \( (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \mod q \). The problem is then defined as: Given \( n, q, \chi \) and any number of independent samples from \( A_{s, \chi} \), determine \( s \).

2.3 Gram-Schmidt orthogonalization

The Gram-Schmidt basis is an orthogonal set of vectors \( \tilde{\mathbf{B}} = \{ \tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_d \} \), where each \( \tilde{\mathbf{b}}_i \) is orthogonal to \( \text{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}) \). Given a basis \( \mathbf{B} \), the Gram-Schmidt basis can be computed as follows: set \( \tilde{\mathbf{b}}_1 = \mathbf{b}_1 \), and compute \( \tilde{\mathbf{b}}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \left( \frac{\langle \mathbf{b}_i, \mathbf{b}_j \rangle}{\langle \mathbf{b}_j, \mathbf{b}_j \rangle} \right) \tilde{\mathbf{b}}_j \), for \( i = 2, \ldots, d \).

2.4 q-ary lattices

Ajtai described a method for generating random lattices that are hard in the average-case [Ajt96, Ajt99]. We summarize Ajtai’s construction and state the worst-case to average-case connection. Given parameters \( q, r, d \) and \( r < d \), such that \( r^{c_1} < q < 2r^{c_1} \) and \( c_2 r \log(r) \leq d < r^{c_3} \) for constants \( c_1, c_2, c_3 \), pick \( \{\mathbf{u}_1, \ldots, \mathbf{u}_{d-1}\} \) uniformly at random from \( \mathbb{Z}_q^r \). Then pick \( \delta = \{\delta_1, \ldots, \delta_{d-1}\} \in \{0,1\}^{d-1} \) uniformly at random. Let \( \mathbf{u}_d = - \sum_{i=0}^{d-1} \delta_i \mathbf{u}_i \). This defines a lattice \( \mathcal{L} = \{ \sum_{i=1}^{d} x_i \mathbf{u}_i \equiv 0 \mod q, x_i \in \mathbb{Z} \} \), with basis
\[ \mathbf{B}_{\mathcal{A}^f} = \{ \mathbf{u}_1, \ldots, \mathbf{u}_d \}. \] Ajtai proved that the existence of a polynomial algorithm that finds the shortest vector in the lattice \( \mathcal{L}(\mathbf{B}_{\mathcal{A}^f}) \) (average-case) implies the existence of a polynomial algorithm that solves SVP in any lattice \( \mathcal{L} \subset \mathbb{R}^d \) (worst-case).

### 2.5 Quantum background

We denote quantum states as \( \sum_x \alpha_x |x\rangle \), where \( \alpha_x \in \mathbb{C} \), and \( \sum_x |\alpha_x|^2 = 1 \). We refer the reader to [NC02] or [KLM+07] for an introduction to quantum information processing. We present some details about the method of amplitude amplification that is relevant to the analysis of our algorithm.

**Definition 5 (Amplitude Amplification [BHMT02, Gro96]).** Given a function \( f \), consider the problem of searching for \( m \) marked elements in a set of \( N \) elements with \( 0 < m < N \), such that \( f(x) = 1 \) iff \( x \) is marked, and \( f(x) = 0 \) otherwise. Let \( \mathcal{A} \) be a quantum algorithm that makes no measurements and produces the superposition \( |\Psi\rangle = \mathcal{A}|0 \cdots 0\rangle = \sqrt{p_{\text{good}}} |\Psi_{\text{good}}\rangle + \sqrt{p_{\text{bad}}} |\Psi_{\text{bad}}\rangle \), where \( |\Psi_{\text{good}}\rangle \) is a superposition over the \( m \) marked 'good' elements. Let \( U_f \) be the unitary that flips the phase of the good states, i.e. \( U_f |\Psi_{\text{good}}\rangle = -|\Psi_{\text{good}}\rangle \), and leaves the bad states unchanged. Let \( U_{0\perp} \) be the unitary that flips the phase of the all-zeros state i.e. \( U_{0\perp}|0 \cdots 0\rangle = -|0 \cdots 0\rangle \) and leaves all other states unchanged. Let the iterate \( Q = \mathcal{A} U_{0\perp} \mathcal{A}^{-1} U_f \). Applying the iterate \( k \) times to the state \( |\Psi\rangle \), i.e. \( Q^k |\Psi\rangle \) results in the state \( |\Psi\rangle = \sin(2k+1)\theta |\Psi_{\text{good}}\rangle + \cos(2k+1)\theta |\Psi_{\text{bad}}\rangle \), where \( \sin^2 \theta = p_{\text{good}} \). Setting \( k = \mathcal{O}\left(\sqrt{\frac{N}{m}}\right) \) and measuring \( |\Psi\rangle \) produces the state \( |\Psi_{\text{good}}\rangle \) with probability close to 1.

We note that often times the goal of amplitude amplification is to search for an element in a given set of elements. However in our algorithm, we use it as a tool to take a uniform superposition over all elements in our set, and produce a uniform superposition over just the marked elements in our set with high probability.

### 2.6 Discrete Gaussians

Let \( s > 0 \) and \( \rho_s(\mathbf{v}) = \frac{1}{\sqrt{s}} e^{-\frac{\|\mathbf{v}\|^2}{2s}} \) for \( \mathbf{v} \in \mathbb{R}^d \), and \( \rho_s(\mathcal{L}) = \sum_{\mathbf{v} \in \mathcal{L}} \rho_s(\mathbf{v}) \) for a given lattice \( \mathcal{L} \). The Discrete Gaussian over the lattice \( \mathcal{L} \), denoted \( D_{\mathcal{L},s} \) is a distribution where the probability of a vector \( \mathbf{x} \in \mathcal{L} \) is given by

\[
\Pr_{\mathbf{X} \sim D_{\mathcal{L},s}} [\mathbf{X} = \mathbf{x}] = \frac{\rho_s(\mathbf{x})}{\rho_s(\mathcal{L})}
\]

The smoothing parameter for a lattice \( \mathcal{L} \), denoted \( \eta(\mathcal{L}) \) is defined as the smallest real \( s > 0 \) such that \( \rho_s(\mathcal{L}^* \setminus \{0\}) \leq \varepsilon \) for any positive real \( \varepsilon > 0 \), and where \( \mathcal{L}^* = \{ \mathbf{v} \in \mathbb{R}^n \mid \langle \mathbf{v}, \mathbf{u} \rangle \in \mathbb{Z}, \forall \mathbf{u} \in \mathcal{L} \} \) is called the dual lattice to \( \mathcal{L} \). In [GPV08], the authors show how to efficiently sample from a Discrete Gaussian distribution, given a width greater than the smoothing parameter of the lattice. Combining the result above with a result from Grover et al. in [GR02], we assume we are able to create a superposition over lattice points of the form \( \frac{1}{\sqrt{N}} \sum_{\mathbf{v} \in \mathcal{L}} |\mathbf{v}\rangle \) where \( N \) is the number of lattice points we want to sample. We call this subroutine \textsc{Sample}(\mathcal{L}) that produces such a superposition for use in our algorithms.

### 3 Quantum Double Sieve

#### 3.1 Binary quantum double sieve

The quantum double sieve is a quantization of the classical double sieve algorithm of Bai et al. [BLS16]. The idea is to reduce pairs of vectors, the difference being that we do this in quantum superposition. We
sample in superposition from a discrete Gaussian distribution of width greater than the smoothing parameter $\eta$ for our lattice. Following the analysis in Section (1.1), we need $O(2^{0.2075d})$ vectors in our initial sample in order for any fixed vector in our set to reduce against any other vector in the set with probability close to 1. We combine two such superpositions in order to get a superposition over pairs of vectors in the lattice. We then use amplitude amplification to search over pairs of vectors where the norm of the difference between the pair is less than $\gamma R$. This is analogous to running one iteration of the classical double sieve. We do this iteratively for $t$ steps of the sieve, starting with an initial set of $2^t$ superpositions as shown in Figure (2). The sieve runs for $t$ iterations, thus immediately yielding the memory requirement for the algorithm. The runtime of this algorithm is determined by the number of amplitude amplification steps that we need to run. Algorithm (2) presents the binary quantum double sieve algorithm.

Algorithm 2: Binary Quantum Sieve

Input: $\mathcal{L}, \gamma, R, C$

function sieve:
1. Let $t$ be such that $\gamma^t R \leq C$
2. $s \leftarrow \text{Measure(build_superposition}(t))$
3. return $s$

function build_superposition(level $\leftarrow i$):
1. if $level = 0$ then
2. return $\frac{1}{\sqrt{N}} \sum_{\|u\| \leq R} |u\rangle \leftarrow \text{Sample} (\mathcal{L})$
3. else
4. $\phi_1 \leftarrow \text{build_superposition}(level \leftarrow i - 1)$
5. $\phi_2 \leftarrow \text{build_superposition}(level \leftarrow i - 1)$
6. $|\psi\rangle \leftarrow \text{Amplitude amplification on } |\phi_1\rangle \otimes |\phi_2\rangle \text{ using } U_f$, where $f_i$ is given in Eq (1)
7. return $|\psi\rangle$

We explain the amplitude amplification step of the algorithm above. At level $i$ of the protocol, $|\phi_1\rangle = \frac{1}{\sqrt{N}} \sum_{\|u\| \leq \gamma^{i-1} R} |u\rangle_1$ and $|\phi_2\rangle = \frac{1}{\sqrt{N}} \sum_{\|u_2\| \leq \gamma^{i-1} R} |u_2\rangle$, each with a history state that we do not write. At this point we tensor those two states to get a superposition over pair of vectors $\frac{1}{N} \sum_{u_1, u_2} |u_1, u_2\rangle$. Let $f_i(x) : \mathbb{R} \mapsto \{0, 1\}$ be defined as follows:

$$f_i(x) = \begin{cases} 1 & \text{if } x \leq \gamma^i R \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

We can define a unitary $U_f$ that maps a state $|x\rangle$ to $(-1)^{f_i(x)}|x\rangle$, where $x$ will be $\|u_2 - u_1\|$ in our case. Then we use standard amplitude amplification:

$$\frac{1}{N} \sum_{u_1, u_2} |u_1, u_2, \|u_2 - u_1\|\rangle \xrightarrow{AA} \frac{1}{\sqrt{N}} \sum_{\|u_2 - u_1\| \leq \gamma R} |u_1, u_2, \|u_2 - u_1\|\rangle + |\text{junk}\rangle$$

We then measure to discard the junk state, compute $u_2 - u_1$ in the first register, and then call this state $|\psi\rangle$.

Theorem 6. Given a lattice $\mathcal{L}$ of dimension $d$, $N = O(2^{0.2075d})$, smoothing parameter $\eta$, norm reduction factor $\gamma$, and $R \geq \eta$, Algorithm (2) solves the SVP for $\mathcal{L}$ in $t$ iterations, taking time $2^t \sqrt{N} = O(R^c \sqrt{N})$ for some constant $c$, and memory $2^t = poly(R)$.
\begin{proof}
Algorithm (2) gives a binary tree of superpositions as shown in Figure (2). The amount of memory used by the algorithm is then given by $2^t$ where $t$ is the height of the tree. At each level of the tree, we use amplitude amplification between pairs of superpositions, so the total number of amplitude amplification steps is also $2^t$. Assume $C$ is a small constant that is a good enough approximation for the length of the shortest vector. In order to compute the quantity $2^t$, we have that:

\[
\gamma^t R \leq C
\]

\[
\therefore \log(\gamma^t R) \leq \log(C)
\]

\[
\therefore t \leq \frac{1}{\log(\gamma)} \left( \log(C) - \log(R) \right)
\]

Plugging in this value of $t$ in $2^t$ we get:

\[
2^t \leq 2^{\frac{1}{\log(\gamma)} \left( \log(C) - \log(R) \right)}
\]

\[
= 2^{\log(C) \log(\gamma) - \log(R) \log(\gamma)}
\]

\[
= C \frac{1}{\log(\gamma)} R^{-\frac{1}{\log(\gamma)}} = poly(R)
\]

Next we analyze the time complexity of the amplitude amplification step. At level $i$ of the algorithm, we perform amplitude amplification on a superposition of vectors $|\phi_1 \otimes \phi_2\rangle = \frac{1}{N} \sum_{u_1, u_2, \|u_2 - u_1\| \leq \gamma^{-1} R} |u_1, u_2, \|u_2 - u_1\|\rangle$ which can be split up as a superposition over good and bad states. We note that we have roughly $N$ marked good elements such that $\|u_2 - u_1\| \leq \gamma^i R$. We rewrite $|\phi_1 \otimes \phi_2\rangle$ as

\[
|\phi_1 \otimes \phi_2\rangle = \frac{1}{\sqrt{N}} |\psi_{\text{good}}\rangle + \sqrt{1 - \frac{1}{N}} |\psi_{\text{bad}}\rangle
\]

where $|\psi_{\text{good}}\rangle = \frac{1}{\|u_2 - u_1\| \leq \gamma^i R} \frac{1}{\sqrt{N}} |u_1, u_2, \|u_2 - u_1\|\rangle$, and

\[
|\psi_{\text{bad}}\rangle = \frac{1}{\|u_2 - u_1\| > \gamma^i R} \frac{1}{\sqrt{N^2 - N}} |u_1, u_2, \|u_2 - u_1\|\rangle
\]

Next we perform standard amplitude amplification and get the state $|\psi_{\text{good}}\rangle$ with probability close to 1 using $O(\sqrt{\frac{N^2}{m}})$ iterations of the search iterate, where $m$ is the number of marked elements. In our case, we have $m = N$ marked elements, so amplitude amplification takes time $O(\sqrt{\frac{N^2}{N}}) = O(\sqrt{N})$. We perform a total of $2^t$ amplitude amplification steps, so the runtime of the algorithm is given by $O(2^t \sqrt{N}) = O(R^{c \sqrt{N}})$ for $c = \frac{1}{\log(\gamma)}$.
\end{proof}

\begin{corollary}
Given a q-ary Ajtai lattice $L$ of dimension $d$, $N = O(2^{0.2075d})$, smoothing parameter $\eta$, norm reduction factor $\gamma$, and $\eta \leq R = poly(d)$, Algorithm (2) solves SVP for $L$ in time $O(2^{0.1038d + o(d)})$ and memory $poly(d)$.
\end{corollary}

We note however that, in practice, the Ajtai basis is used as a trapdoor basis, and the public basis typically has size exponential in the lattice dimension. This means that in practice, the value for $R$ can be
Figure 2: Each level in the tree is one iteration of the sieve. In order to run the sieve for \( t \) iterations we need \( 2^t \) initial superpositions over vectors in our lattice exponential in the lattice dimension. This is to be expected for an exponential time algorithm. We state the following corollary.

**Corollary 8.** Given a lattice \( \mathcal{L} \) of dimension \( d \), \( N = \mathcal{O}(2^{0.2075d}) \), smoothing parameter \( \eta \), norm reduction factor \( \gamma \), and \( \eta \leq R = \mathcal{O}(2^{\alpha d}) \) for \( \alpha > 0 \), Algorithm (2) solves SVP for \( \mathcal{L} \) in time \( \mathcal{O}(2^{(\alpha c + 0.1038)d}) \) and memory \( \mathcal{O}(2^{\alpha cd}) \) for some constant \( c \).

### 3.2 \( r \)-ary Quantum Sieve

We show a variation of the binary quantum double sieve technique to improve the memory bound of the sieve. At each step of the binary quantum sieve, we combined vectors of roughly the same norm, and searched for differences of them that gave us shorter vectors. We can provide a slight improvement in memory by combining two superpositions where the first one is over vectors of similar norm, say norm bounded by a quantity \( R \), while the second is over vectors of norm bounded by a smaller quantity, say \( \gamma R \).

We describe the process step by step.

Let us begin by sampling two superpositions where the norms of the vectors are bounded by \( R \) in both superpositions. Let these superpositions be over \( M \) vectors, for \( M \) that will be determined later.

\[
\frac{1}{\sqrt{M}} \sum_{\|v_1\| \leq R} |v_1\rangle \quad \frac{1}{\sqrt{M}} \sum_{\|v_2\| \leq \gamma R} |v_2\rangle
\]

We compute \( \|v_2 - v_1\| \) in an auxiliary register, and using a unitary \( U_f \) defined in the same way as Eq(1), we search for pairs of vectors where the norm of the difference is less than \( \gamma R \). Amplitude amplification gives us

\[
\frac{1}{M} \sum_{v_1, v_2} |v_1, v_2, \|v_2 - v_1\|\rangle \xrightarrow{A.A.} \frac{1}{\sqrt{M}} \sum_{\|v_2 - v_1\| \leq \gamma R} |v_1, v_2, \|v_2 - v_1\|\rangle \xrightarrow{A.A.} \frac{1}{\sqrt{M}} \sum_{\|w_1\| \leq \gamma R} |w_1\rangle |\text{history}\rangle
\]

Notice that this first step is the same as in the Binary Quantum Sieve. In the next step, however, we combine vectors with slightly larger norm, bounded by \( R \), with the output from our amplitude amplification step. We
run amplitude amplification on this superposition to find even shorter vectors, namely, those whose norm is bounded by $\gamma^2 R$.

$$\frac{1}{\sqrt{M}} \sum_{\|\mathbf{w}_1\| \leq \gamma R} |\mathbf{w}_1\rangle \otimes \frac{1}{\sqrt{M}} \sum_{\|\mathbf{v}_3\| \leq R} |\mathbf{v}_3\rangle \stackrel{\text{SA}}{\longrightarrow} \frac{1}{\sqrt{M}} \sum_{\|\mathbf{w}_2\| \leq \gamma^2 R} |\mathbf{w}_2\rangle$$

We continue combining shorter vectors with slightly longer vectors and search over the superposition for even shorter vectors. This process for $t'$ steps is illustrated in Figure (3).

![Figure 3: Each level of the tree represents one step of the sieve and vectors on that level have norm shorter by a factor of $\gamma$. This tree shows that we get linear growth in the number of superpositions we need for up to $t'$ iterations of the sieve, as opposed to an exponential number of superpositions in the binary sieve.](image)

At the $i^{th}$ (for $i < t'$) iteration, we combine vectors $\mathbf{w}_i$ of length bounded by $\gamma^i R$ with vectors $\mathbf{v}_{i+2}$ of length bounded by $R$. For our analysis, we normalize the lengths of $\mathbf{w}_i$ and $\mathbf{v}_{i+2}$ by $R$, so $\|\mathbf{w}_i\| \approx \gamma^i$ and $\|\mathbf{v}_{i+2}\| \approx 1$. Note that we require the angle $\theta_{\mathbf{w}_i,\mathbf{v}_{i+2}}$ between $\mathbf{w}_i$ and $\mathbf{v}_{i+2}$ to be $\leq \frac{\pi}{4}$, and $\gamma^i > \frac{1}{2}$ since no feasible solutions exist for $\|\mathbf{v}_{i+2} - \mathbf{w}_i\| \leq \|\mathbf{w}_i\|$ for $\|\mathbf{w}_i\| \leq \frac{1}{2}$ and $\|\mathbf{v}_{i+2}\| = 1$. Thus for $\gamma^i > \frac{1}{2}$, in order to get reductions between $\mathbf{w}_i$ and $\mathbf{v}_{i+1}$ we want vectors $\mathbf{w}_i \in B(\mathbf{0}, \gamma^i) \cap B(\mathbf{v}_{i+2}, \gamma \cdot \gamma^i)$.

We restate a lemma from [BLS16] that will help us compute the quantity $B(\mathbf{0}, \gamma^i) \cap B(\mathbf{v}_{i+2}, \gamma \cdot \gamma^i)$

**Lemma 9.** Given $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^d$, and $r_1, r_2 \in \mathbb{R}$, and $\|\mathbf{u}_1 - \mathbf{u}_2\| = \epsilon$, such that $r_1, r_2 < \epsilon < r_1 + r_2$,

$$|B_d(\mathbf{u}_1, r_1) \cap B_d(\mathbf{u}_2, r_2)| \propto \left( \frac{-\epsilon^4 + 2\epsilon^2(r_1^2 + r_2^2) - (r_1^2 - r_2^2)^2}{4\epsilon^2} \right)^{\frac{d}{2}} |B_d|$$
Setting $\gamma^i = s$ for simplicity, and using Lemma (9), we get

$$|B_d(0, s) \cap B_d(v_{i+2}, \gamma s)| \propto \left( \frac{1 + 2(s^2 + s^2 \gamma^2) - (s^2 - s^2 \gamma^2)^2}{4} \right)^{\frac{d}{2}} |B_d|$$

$$= \left( \frac{1 + 2s^2(1 + \gamma^2) - s^4(1 - \gamma^2)^2}{4} \right)^{\frac{d}{2}} |B_d|$$

$$= \left( \frac{1}{4s^2} + \frac{(1 + \gamma^2)}{2} - \frac{s^2(1 - \gamma^2)^2}{4} \right)^{\frac{d}{2}} |B_d(s)|$$

(2)

This volume gives us the portion of the sphere of radius $s$ that one fixed vector $v_i$ covers. In order for any given vector in our set to reduce with a larger vector with probability close to 1, we need to cover the whole sphere. Therefore we need to sample $M \propto \frac{1}{p}$, where $p = \left( \frac{1}{4s^2} + \frac{(1 + \gamma^2)}{2} - \frac{s^2(1 - \gamma^2)^2}{4} \right)^{\frac{d}{2}}$.

Now that we have a bound on the number of samples we need, we can state the complete algorithm for solving SVP, shown in Algorithm (3).

**Theorem 10.** Given a lattice $\mathcal{L}$ of dimension $d$, a norm reduction factor $\gamma$, and a value $M = \mathcal{O}(2^{kd})$ where $\delta = \frac{1}{2} \log(1/( - \frac{1}{4\gamma^2} + \frac{(1 + \gamma^2)}{2} - \frac{s^2(1 - \gamma^2)}{4} ))$ and $t'$ is the number of iterations of the sieve with $\gamma^{t'} > \frac{1}{2}$.

Let $\eta$ be the smoothing parameter and $R$ such that $\eta \leq R = \mathcal{O}(2^{kd})$ for $\alpha > 0$. Then Algorithm (3) solves SVP for $\mathcal{L}$ in memory $\mathcal{O} \left( 2^{(ac + \frac{d}{2})d} \right)$ and time $\mathcal{O}(R^{c'} \sqrt{M})$ for some constant $c'$.

**Proof.** The appropriate value for $M$ given $\gamma$ and the number of iterations is given by the analysis of Eq (2). We can compute the exponent $\delta$ by taking the appropriate logarithm, and we get $\delta = \frac{1}{2} \log(1/( - \frac{1}{4\gamma^2} + \frac{(1 + \gamma^2)}{2} - \frac{s^2(1 - \gamma^2)}{4} ))$ since we have $t'$ iterations. Let $t' + 1 = r$. Let $x$ be such that $\gamma^{t'x} R \leq C$ where $C$ is a bound on the shortest vector (for example $C$ can be taken to be the Minkowski bound). We need $(t' + 1)^x = r^x$ vectors. We have:

$$\gamma^{t'x} R = \gamma^{(r-1)x} R \leq C$$

$$\therefore \log_\gamma (\gamma^{(r-1)x} R) = \log_\gamma (C)$$

$$\therefore x = \log_\gamma (C^{\log_\gamma (\gamma^{(r-1)x})}) - \log_\gamma (R^{\log_\gamma (\gamma^{(r-1)x})})$$

$$\therefore x = C^{\log_\gamma (\gamma^{(r-1)x})} R^{\log_\gamma (\gamma^{(r-1)x})} \mathcal{O}(R^{c'})$$

The time complexity is identical to the analysis in Theorem (6), and hence we get that the runtime is $\mathcal{O}(R^{c'} \sqrt{M})$ where $c' = \frac{\log(t' + 1)}{t' \log(\gamma)}$. \[ \square \]

We note that the parameters $\delta, t'$, and $\gamma$ are tunable, and can be optimized based on the value of $\alpha$ to get the lowest possible value for the constant $c'$.

### 4 Future Work

This work is a first step toward quantizing classical algorithms by doing more than just a replacement of the classical search step with quantum search. In our case, the Double Sieve was a very good candidate
Algorithm 3: r-ary Quantum Sieve

Input: $L$, $\gamma$, $M$, $R$, $C$, $t'$

function sieve:

1. Let $x$ be such that $\gamma^{xt'}R \leq C$. Set $r \leftarrow t' + 1$
2. Sample $r^x$ superpositions $\frac{1}{\sqrt{M}} \sum_{\|u\| \leq R} |u\rangle$, $\cdots$, $\frac{1}{\sqrt{M}} \sum_{\|u^r\| \leq R} |u^r\rangle \leftarrow \text{Sample}(L)$.
3. Set $c \leftarrow x$
4. while $c > 0$ do
5. Group the $r^c$ superpositions in groups of $r$ superpositions each, giving us $r^{c-1}$ groups.
6. Within each group, run combine($\cdots$) and re-index the $u_i$ as follows:
7. for $i \leftarrow 1, \cdots, r^{c-1}$ do
8. $\frac{1}{\sqrt{M}} \sum_{\|u\| \leq \gamma^{(x-c-1)t'}R} |u\rangle \leftarrow \text{combine} \left( \frac{1}{\sqrt{M}} \sum_{\|u_{(i-1)r+1}\| \leq \gamma^{(x-c)t'}R} |u_{(i-1)r+1}\rangle, \cdots, \frac{1}{\sqrt{M}} \sum_{\|u_{tr}\| \leq \gamma^{(x-c+1)t'}R} |u_{tr}\rangle \right)$
9. $c \leftarrow c - 1$
10. end
11. end

12. $s \leftarrow \text{Measure}(\frac{1}{\sqrt{M}} \sum_{\|u\| \leq \gamma^{xt'}R} |u\rangle)$
13. return $s$
14. end

function combine($s_1, \cdots, s_r$):
15. $|\phi_1\rangle \leftarrow s_1$
16. Set $ct \leftarrow 2$
17. while $ct \leq r$ do
18. $|\phi_2\rangle \leftarrow s_{ct}$
19. $|\phi_1\rangle \leftarrow \text{Amplitude Amplification on } |\phi_1\rangle \otimes |\phi_2\rangle \text{ using } U_{f_{\exp}} \text{ where } f_{\exp} \text{ is given by Eq(1)}$
20. for the appropriate value of exp at this iteration.
21. $ct \leftarrow ct + 1$
22. end
23. return $|\phi\rangle$
24. end

for using the ‘sieving-in-superposition’ technique. It might be interesting to see if this technique extends to other classical sieving algorithms like Laarhoven’s SphereSieve[La15] and HashSieve [LdW15] to yield even better exponents in the runtime and memory complexities.
References

[ACKS20] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, and Yixin Shen. Improved (provable) algorithms for the shortest vector problem via bounded distance decoding. *arXiv preprint arXiv:2002.07955*, 2020.

[ADRSD15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in $2^n$ time using discrete gaussian sampling. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 733–742, 2015.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *International Colloquium on Automata, Languages, and Programming*, pages 1–9. Springer, 1999.

[BHMT02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *LMS Journal of Computation and Mathematics*, 19(A):146–162, 2016.

[FP85] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of computation*, 44(170):463–471, 1985.

[GNR10] Nicolas Gama, Phong Q Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 257–278. Springer, 2010.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

[GR02] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint quant-ph/0208112*, 2002.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 193–206, 1983.

[KLM+07] Phillip Kaye, Raymond Laflamme, Michele Mosca, et al. *An introduction to quantum computing*. Oxford University Press on Demand, 2007.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Annual Cryptology Conference*, pages 3–22. Springer, 2015.
[LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *International Conference on Cryptology and Information Security in Latin America*, pages 101–118. Springer, 2015. 2, 12

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007. 5

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1468–1480. SIAM, 2010. 2

[NC02] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002. 6

[NV08] Phong Q Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. 2

[Poh81] Michael Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *ACM Sigsam Bulletin*, 15(1):37–44, 1981. 1

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009. 5

[Sch10] Claus Peter Schnorr. Average time fast SVP and CVP algorithms for low density lattices and the factorization of integers. *Technical Report*, 2010. 1

[SH95] Claus-Peter Schnorr and Horst Helmut Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–12. Springer, 1995. 1