Finding shortest lattice vectors faster using quantum search

Thijs Laarhoven · **Michele Mosca** · **Joop van de Pol**

Abstract By applying a quantum search algorithm to various heuristic and provable sieve algorithms from the literature, we obtain improved asymptotic quantum results for solving the shortest vector problem on lattices. With quantum computers we can provably find a shortest vector in time $2^{1.799n+o(n)}$, improving upon the classical time complexities of $2^{2.465n+o(n)}$ of Pujol and Stehlé and the $2^{2n+o(n)}$ of Micciancio and Voulgaris, while heuristically we expect to find a shortest vector in time $2^{0.268n+o(n)}$, improving upon the classical time complexity of $2^{0.298n+o(n)}$ of Laarhoven and De Weger. These quantum complexities will be an important guide for the selection of parameters for post-quantum cryptosystems based on the hardness of the shortest vector problem.

Keywords lattices · shortest vector problem · sieving · quantum search

1 Introduction

Large-scale quantum computers will redefine the landscape of computationally secure cryptography, including breaking public-key cryptography based on integer factorization or the discrete logarithm problem [87] or the principle ideal problem in real quadratic number fields [40], providing sub-exponential attacks for some systems based on elliptic curve isogenies [24], speeding up exhaustive searching [38, 16], counting [19] and (with appropriate assumptions about the computing architecture) finding collisions and claws [18, 20, 5], among many other quantum algorithmic speed-ups [23, 88, 68].

Currently, a small set of systems [13] are being studied intensely as possible systems to replace those broken by large-scale quantum computers. These systems can be implemented with conventional technologies and to date seem resistant to substantial quantum attacks. It is critical that these systems receive intense scrutiny for possible quantum or classical attacks. This will boost confidence in the resistance of these systems to (quantum) attacks, and allow us to fine-tune secure choices of parameters in practical implementations of these systems.

One such set of systems bases its security on the computational hardness of certain lattice problems. Since the late 1990s, there has been a lot of research into the area of lattice-based cryptography, resulting in encryption schemes [42, 79], digital signature schemes [35, 60, 27] and even fully homomorphic encryption schemes [36, 17]. Each of the lattice problems that underpin the security of these systems can be reduced to the shortest vector problem [72]. Conversely, the decisional variant of the shortest vector problem can be reduced to the average case of such lattice problems. For a more detailed summary on the security of lattice-based cryptography, see [51, 72].

In this paper, we closely study the best-known algorithms for solving the shortest vector problem, and how quantum algorithms may speed up these algorithms. By challenging and improving the best asymptotic complexities of these algorithms, we increase the confidence in the security of lattice-based schemes. Understanding these algorithms is critical when selecting key-sizes and other security parameters. Any non-trivial algorithmic advance has the potential to compromise the security of a deployed cryptosystem, for example in [14] an improvement in the index calculus method for finding discrete logarithms led to the break of a Diffie-Hellman system that had been deployed in software and was in the process of being implemented in hardware.

T.M.M. Laarhoven
Eindhoven University of Technology, Eindhoven, The Netherlands.
M. Mosca
Institute for Quantum Computing and Department of Combinatorics & Optimization, University of Waterloo, and Perimeter Institute for Theoretical Physics, Waterloo (Ontario), Canada,
Canadian Institute for Advanced Research, Toronto, Canada.
J.H. van de Pol
University of Bristol, Bristol, United Kingdom.
This article is a minor revision of the version published in Design, Codes & Cryptography: doi:10.1007/s10623-015-0067-5.
A preliminary version of this paper was published at PQCrypto 2013 [52].

1.1 Lattices

Lattices are discrete subgroups of $\mathbb{R}^n$. Given a set of n linearly independent vectors $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ in $\mathbb{R}^n$, we define the lattice generated by these vectors as $\mathscr{L} = \{\sum_{i=1}^{n} \lambda_i \mathbf{b}_i : \lambda_i \in \mathbb{Z}\}$. We call the set B a basis of the lattice $\mathscr{L}$. This basis is not unique; applying a unimodular matrix transformation to the vectors of B leads to a new basis B' of the same lattice $\mathscr{L}$.

In lattices, we generally work with the Euclidean or ℓ_2-norm, which we will denote by $\|\cdot\|$. For bases B, we write $\|B\| = \max_i \|\mathbf{b}_i\|$. We refer to a vector $\mathbf{s} \in \mathscr{L} \setminus \{\mathbf{0}\}$ such that $\|\mathbf{s}\| \leq \|\mathbf{v}\|$ for any $\mathbf{v} \in \mathscr{L} \setminus \{\mathbf{0}\}$ as a shortest (non-zero) vector of the lattice. Its length is denoted by $\lambda_1(\mathscr{L})$. Given a basis B, we write $\mathscr{P}(B) = \{\sum_{i=1}^{n} \lambda_i \mathbf{b}_i : 0 \leq \lambda_i < 1\}$ for the fundamental domain of B.

One of the most important hard problems in the theory of lattices is the *shortest vector problem* (SVP). Given a basis of a lattice, the shortest vector problem consists of finding a shortest non-zero vector in this lattice. In many applications, finding a reasonably short vector instead of a shortest vector is also sufficient. The *approximate shortest vector problem* with approximation factor δ (SVP_δ) asks to find a non-zero lattice vector $\mathbf{v} \in \mathscr{L}$ with length bounded from above by $\|\mathbf{v}\| \leq \delta \cdot \lambda_1(\mathscr{L})$.

Finding short vectors in a lattice has been studied for many reasons, including the construction of elliptic curve cryptosystems [8,31,32], the breaking of knapsack cryptosystems [55,26,75,63] and low-exponent RSA [25,89], and proving hardness results in Diffie-Hellman-type schemes [15]. For appropriately chosen lattices, the shortest vector problem appears to be hard, and may form the basis of new public-key cryptosystems.

1.2 Finding short vectors

The approximate shortest vector problem is integral in the cryptanalysis of lattice-based cryptography [33]. For small values of δ, this problem is known to be NP-hard [3,47], while for certain exponentially large δ polynomial time algorithms are known to exist that solve this problem, such as the celebrated LLL algorithm of Lenstra, Lenstra, and Lovász [56, 63]. Other algorithms trade extra running time for a better δ, such as LLL with deep insertions [85] and the BKZ algorithm of Schnorr and Euchner [85].

The current state-of-the-art for classically finding short vectors is BKZ 2.0 [22], which is essentially the original BKZ algorithm with the improved SVP subroutine of Gama et al. [34]. Implementations of this algorithm, due to Chen and Nguyen [22] and Aono and Naganuma [9], currently dominate the SVP and lattice challenge hall of fame [83, 57] together with a yet undocumented modification of the random sampling reduction (RSR) algorithm of Schnorr [86], due to Kashiwabara et al. [83].

In 2003, Ludwig [59] used quantum algorithms to speed up the original RSR algorithm. By replacing a random sampling from a big list by a quantum search, Ludwig achieves a quantum algorithm that is asymptotically faster than its classical counterpart. Ludwig also details the effect that this faster quantum algorithm would have had on the practical security of the lattice-based encryption scheme NTRU [42], had there been a quantum computer in 2005.

1.3 Finding shortest vectors

Although it is commonly sufficient to find a short vector (rather than a *shortest* vector), the BKZ algorithm and its variants all require a low-dimensional exact SVP solver as a subroutine. In theory, any of the known methods for finding a shortest vector could be used. We briefly discuss the three main classes of algorithms for finding shortest vectors below.

Enumeration. The classical method for finding shortest vectors is enumeration, dating back to work by Pohst [71], Kannan [46] and Fincke and Pohst [29] in the first half of the 1980s. In order to find a shortest vector, one enumerates all lattice vectors inside a giant ball around the origin. If the input basis is only LLL-reduced, enumeration runs in $2^{O(n^2)}$ time, where n is the lattice dimension. The algorithm by Kannan uses a stronger preprocessing of the input basis, and runs in $2^{O(n \log n)}$ time. Both approaches use only polynomial space in n.

Sieving. In 2001, Ajtai et al. [4] introduced a technique called sieving, leading to the first probabilistic algorithm to solve SVP in time $2^{O(n)}$. Several different sieving methods exist, but they all rely on somehow saturating the space of short lattice vectors, by storing all these vectors in a long list. This list will inevitably be exponential

in the dimension n, but it can be shown that these algorithms also run in single exponential time, rather than superexponential (as is the case for enumeration). Recent work has also shown that the time and space complexities of sieving improve when working with ideal lattices [43], leading to the current highest record in the ideal lattice challenge hall of fame [70].

Computing the Voronoi cell. In 2010, Micciancio and Voulgaris presented a deterministic algorithm for solving SVP based on constructing the Voronoi cell of the lattice [64]. In time $2^{2n+o(n)}$, this algorithm is able to construct an exact $2^{n+o(n)}$-space description of the Voronoi cell of the lattice, which can then be used to solve both SVP and CVP. The overall time complexity of $2^{2n+o(n)}$ was until late 2014 the best known complexity for solving SVP in high dimensions. [1]

Discrete Gaussian sampling. Very recently, an even newer technique was introduced by Aggarwal et al. [1], making extensive use of discrete Gaussians on lattices. By initially sampling $2^{n+o(n)}$ lattice vectors from a very wide discrete Gaussian distribution (with a large standard deviation), and then iteratively combining and averaging samples to generate samples from a more narrow discrete Gaussian distribution on the lattice, the standard deviation can be reduced until the point where a set of many samples of the resulting distribution is likely to contain a shortest non-zero vector of the lattice. This algorithm runs in provable $2^{n+o(n)}$ time and space.

Practice. While sieving, the Voronoi cell algorithm, and the discrete Gaussian sampling algorithm have all surpassed enumeration in terms of classical asymptotic time complexities, in practice enumeration still dominates the field. The version of enumeration that is currently used in practice is due to Schnorr and Euchner [85] with improvements by Gama et al. [34]. It does not incorporate the stronger version of preprocessing of Kannan [46] and hence has an asymptotic time complexity of $2^{O(n^2)}$. However, due to the larger hidden constants in the exponents and the exponential space complexity of the other algorithms, enumeration is actually faster than other methods for most practical values of n. That said, these other methods are still relatively new and unexplored, so a further study of these other methods may tip the balance.

1.4 Quantum search

In this paper we will study how quantum algorithms can be used to speed up the SVP algorithms outlined above. More precisely, we will consider the impact of using Grover's quantum search algorithm [38], which considers the following problem.

Given a list L of length N and a function $f : L \to \{0,1\}$, such that the number of elements $e \in L$ with $f(e) = 1$ is small. Construct an algorithm "Search" that, given L and f as input, returns an $e \in L$ with $f(e) = 1$, or determines that (with high probability) no such e exists. We assume for simplicity that f can be evaluated in unit time.

Classical algorithm. With classical computers, the natural way to find such an element is to go through the whole list, until one of these elements is found. This takes on average $O(N)$ time. This is also optimal up to a constant factor; no classical algorithm can find such an element in less than $\Omega(N)$ time.

Quantum algorithm. Using Grover's quantum search algorithm [38, 16, 19], we can find such an element in time $O(\sqrt{N})$. This is optimal up to a constant factor, as any quantum algorithm needs at least $\Omega(\sqrt{N})$ evaluations of f [11].

Throughout the paper, we will write $x \leftarrow \text{Search}\{e \in L : f(e) = 1\}$ to highlight subroutines that perform a search in some long list L, looking for an element $e \in L$ satisfying $f(e) = 1$. This assignment returns true if an element $e \in L$ with $f(e) = 1$ is found (and assigns such an element to x), and returns false if no such e exists. This allows us to give one description for both the classical and quantum versions of each algorithm, as the only difference between the two versions is which version of the subroutine is used.

[1] At the time of the initial submission of this paper, the result of Micciancio and Voulgaris was the best provable asymptotic result for classical SVP (and CVP) solvers to date. The paper [1], provably solving SVP in time $2^{n+o(n)}$, appeared only in December 2014.

1.5 RAM model

For both the classical and the quantum versions of these search algorithms, we assume a RAM model of computation where the jth entry of the list L can be looked up in constant time (or polylogarithmic time). In the case that L is a virtual list where the jth element can be computed in time polynomial in the length of j (thus polylogarithmic in the length of the list L), then look-up time is not an issue. When L is indeed an unstructured list of values, for classical computation, the assumption of a RAM-like model has usually been valid in practice. However, there are fundamental reasons for questioning it [12], and there are practical computing architectures where the assumption does not apply. In the case of quantum computation, a practical RAM-like quantum memory (e.g. [37]) looks particularly challenging, especially for first generation quantum computers. Some authors have studied the limitations of quantum algorithms in this context [39, 12, 44].

Some algorithms (e.g. [5]) must store a large database of information in regular quantum memory (that is, memory capable of storing quantum superpositions of states). In contrast, quantum searching an actual list of N (classical) strings requires the N values to be stored in quantumly addressable classical memory (e.g. as Kuperberg discusses in [50]) and $O(\log N)$ regular qubits. Quantumly addressable classical memory in principle could be much easier to realize in practice than regular qubits. Furthermore, quantum searching for a value $x \in \{0,1\}^n$ satisfying $f(x) = 1$ for a function $f : \{0,1\}^n \to \{0,1\}$ which can be implemented by a circuit on $O(n)$ qubits only requires $O(n)$ regular qubits, and there is no actual list to be stored in memory. In this paper, the quantum search algorithms used require the lists of size N to be stored in quantumly addressable classical memory and use $O(\log N)$ regular qubits and $O(\sqrt{N})$ queries into the list of numbers.

In this work, we consider (conventional) classical RAM memories for the classical algorithms, and RAM-like quantumly addressable classical memories for the quantum search algorithms. This is both a first step for future studies in assessing the impact of more practical quantum architectures, and also represents a more conservative approach in determining parameter choices for lattice-based cryptography that should be resistant against the potential power of quantum algorithmic attacks. Future work may also find ways to take advantage of advanced quantum search techniques, such as those surveyed in [80].

1.6 Contributions

In this paper, we show that quantum algorithms can significantly speed up various sieving algorithms from the literature. The constants in the time exponents generally decrease by approximately 25%, leading to an improvement in both the best provable (exact and approximate) and the best heuristic asymptotic results for solving the shortest vector problem:

- Provably, we can find a shortest vector in any lattice in time $2^{1.799n+o(n)}$.
  (Without quantum search, the best provable algorithm[1] runs in time $2^{2.000n+o(n)}$.)
- Heuristically, we can find a shortest vector in any lattice in time $2^{0.286n+o(n)}$.
  (Without quantum search, the best heuristic algorithm runs in time $2^{0.337n+o(n)}$.)
- Provably, we can solve SVP_δ in any lattice in time $2^{0.603n+o_\delta(n)}$.[2]
  (Without quantum search, the best provable algorithm runs in time $2^{0.804n+o_\delta(n)}$.)

Table 1 contains an overview of classical and quantum complexities of various SVP algorithms, and summarizes the results in this paper. While the Voronoi cell algorithm [64] is asymptotically the best algorithm in the provable classical setting[1], we show that with quantum search, both the `AKS-Birthday` algorithm described by Hanrot et al. [41] and the `ListSieve-Birthday` algorithm of Pujol and Stehlé [74] surpass the $2^{2n+o(n)}$ time complexity of the Voronoi cell algorithm. While the main focus in this paper is on sieving algorithms, we also briefly consider applying quantum search to other methods, and argue why applying the same techniques does not easily lead to significant speed-ups for those algorithms.

After the initial submission of our paper, it was shown that the provable time complexity of solving SVP can be further improved to $2^{n+o(n)}$ using a new method based on discrete Gaussian sampling [1]. Since the provable time complexity of sieving (using quantum search) is asymptotically higher than $2^{n+o(n)}$, this means that sieving on a quantum computer is no longer the best provable algorithm (asymptotically) for solving SVP exactly. In Section 9.3 we therefore also discuss the impact that quantum search may have on the discrete Gaussian sampling method.

[2] Here $o_\delta(n)$ corresponds to a function $f(\delta, n)$ satisfying $\lim_{\delta\to\infty} \lim_{n\to\infty} \frac{1}{n} f(\delta, n) \to 0$.

Table 1 A comparison of time and space complexities of SVP algorithms, both classically and quantumly. Except for the italicized algorithms, these are all results based on sieving. The top rows describe provable algorithms for SVP, the middle rows describe heuristic algorithms for SVP, and the bottom rows describe provable algorithms for solving SVP_δ, and their asymptotic complexities as $\delta, n \to \infty$.

	Algorithm	**Classical**		**Quantum**		**Roadmap**
	Name [References]	$\log_2$(Time)	$\log_2$(Space)	$\log_2$(Time)	$\log_2$(Space)	
Provable SVP	*Enumeration algorithms*	$\Omega(n \log n)$	$O(\log n)$	$\Omega(n \log n)$	$O(\log n)$	(Section 9.1)
	`AKS-Sieve` [4, 77, 69, 65, 41]	$3.398n$	$1.985n$	$2.672n$	$1.877n$	(Section 8.1)
	`ListSieve` [65]	$3.199n$	$1.327n$	$2.527n$	$1.351n$	(Section 8.2)
	`AKS-Sieve-Birthday` [41]	$2.648n$	$1.324n$	$1.986n$	$1.324n$	(Section 8.3)
	`ListSieve-Birthday` [74]	$2.465n$	$1.233n$	$1.799n$	$1.286n$	(Section 2)
	Voronoi cell algorithm	$2.000n$	$1.000n$	$2.000n$	$1.000n$	(Section 9.2)
	Discrete Gaussian sampling	$\mathbf{1.000n}$	$0.500n$	$\mathbf{1.000n}$	$0.500n$	(Section 9.3)
Heuristic SVP	`NV-Sieve` [69]	$0.415n$	$0.208n$	$0.312n$	$0.208n$	(Section 3)
	`GaussSieve` [65]	$0.415n$	$0.208n$	$0.312n$	$0.208n$	(Section 4)
	`2-Level-Sieve` [90]	$0.384n$	$0.256n$	$0.312n$	$0.208n$	(Section 8.4)
	`3-Level-Sieve` [91]	$0.378n$	$0.283n$	$0.312n$	$0.208n$	(Section 8.5)
	`Overlattice-Sieve` [10]	$0.378n$	$0.293n$	$0.312n$	$0.208n$	(Section 8.6)
	`HashSieve` [53]	$0.337n$	$0.337n$	$0.286n$	$0.286n$	(Section 5)
	`SphereSieve` [54]	$\mathbf{0.298n}$	$0.298n$	$\mathbf{0.268n}$	$0.268n$	(Section 6)
SVP_δ	*Enumeration algorithms*	$\Omega(n \log n)$	$O(\log n)$	$\Omega(n \log n)$	$O(\log n)$	(Section 9.1)
	Voronoi cell algorithm	$2.000n$	$1.000n$	$2.000n$	$1.000n$	(Section 9.2)
	Discrete Gaussian sampling	$1.000n$	$0.500n$	$1.000n$	$0.500n$	(Section 9.3)
	`ListSieve-Birthday` [58]	$\mathbf{0.802n}$	$0.401n$	$\mathbf{0.602n}$	$0.401n$	(Section 7)

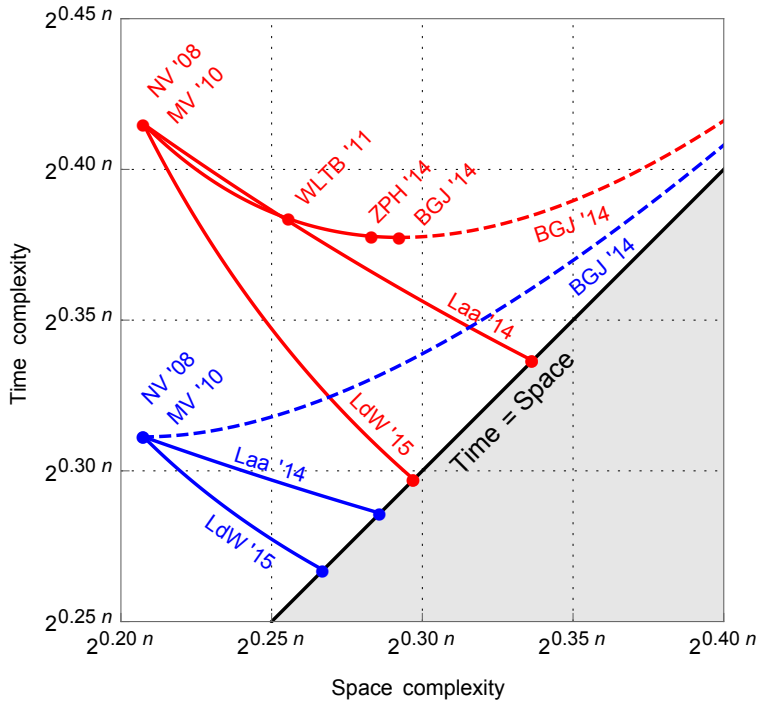

Fig. 1 The heuristic space-time trade-off of various heuristic sieve algorithms from the literature (red), and the heuristic trade-offs obtained with quantum search applied to these algorithms (blue). The optimized `2-Level-Sieve` and `3-Level-Sieve` of Wang et al. and Zhang et al. both collapse to the point $(2^{0.208n}, 2^{0.312n})$ as well. Dashed lines are increasing in both the time and the space complexity, and therefore do not offer a useful trade-off.

The heuristic improvements obtained with quantum search are also shown in Figure 1. This figure also shows the tunable trade-offs that may be obtained with various classical and quantum sieving algorithms (rather than just the single entries given in Table 1). As can be seen in the figure, we only obtain a useful trade-off between the quantum time and space complexities for the `HashSieve` and `SphereSieve` algorithms; for other algorithms the trade-offs are not really trade-offs, as both the time and the space complexity increase by changing the parameters.

Algorithm 1 The `ListSieve-Birthday` algorithm

Sample a random number $N_1' \in [0, N_1]$
Initialize an empty list L
for $i \leftarrow 1$ **to** N_1' **do**
    Sample a random perturbation vector $\mathbf{e} \leftarrow B_n(\mathbf{0}, \xi\mu)$
    Compute the translated vector $\mathbf{v}' \leftarrow \mathbf{e} \mod \mathscr{P}(B)$
    while $\mathbf{w} \leftarrow \text{Search}\{\mathbf{w} \in L : \|\mathbf{v}' \pm \mathbf{w}\| < \gamma\|\mathbf{v}'\|\}$ **do**
        Reduce $\mathbf{v}'$ with $\mathbf{w}$
    Subtract the perturbation vector $\mathbf{e}$: $\mathbf{v} \leftarrow \mathbf{v}' - \mathbf{e}$
    if $\|\mathbf{v}\| \geq R\mu$ **then**
        Add the lattice vector $\mathbf{v}$ to the list L
Initialize an empty list S
for $i \leftarrow 1$ **to** N_2 **do**
    Sample a random perturbation vector $\mathbf{e} \leftarrow B_n(\mathbf{0}, \xi\mu)$
    Compute the translated vector $\mathbf{v}' \leftarrow \mathbf{e} \mod \mathscr{P}(B)$
    while $\mathbf{w} \leftarrow \text{Search}\{\mathbf{w} \in L : \|\mathbf{v}' \pm \mathbf{w}\| < \gamma\|\mathbf{v}'\|\}$ **do**
        Reduce $\mathbf{v}'$ with $\mathbf{w}$
    Subtract the perturbation vector $\mathbf{e}$: $\mathbf{v} \leftarrow \mathbf{v}' - \mathbf{e}$
    Add the lattice vector $\mathbf{v}$ to the list S
$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \text{Search}\{(\mathbf{s}_1, \mathbf{s}_2) \in S^2 : 0 < \|\mathbf{s}_1 - \mathbf{s}_2\| < \mu\}$
return $\mathbf{s}_1 - \mathbf{s}_2$

1.7 Outline

The outline of this paper is as follows, and can also be found in Table 1. In Section 2 we first consider the current best provable sieving algorithm for solving the shortest vector problem, the `ListSieve-Birthday` algorithm of Pujol and Stehlé [74]. This is the birthday paradox variant of the `ListSieve` algorithm of Micciancio and Voulgaris [65] (which is briefly described in Section 8.2), and we get the best provable quantum time complexity by applying quantum search to this algorithm. In Sections 3 and 4 we then consider two of the most important heuristic sieving algorithms to date, the `NV-Sieve` algorithm of Nguyen and Vidick [69] and the `GaussSieve` algorithm of Micciancio and Voulgaris [65]. In Section 5 we then show how we obtain the best heuristic quantum time complexity, by applying quantum search to the very recent `HashSieve` algorithm of Laarhoven [53], which in turn builds upon the `NV-Sieve` and the `GaussSieve`. Finally, in Section 8 we discuss quantum speed-ups for various other sieving algorithms, and in Section 9 we discuss why quantum search does not seem to lead to big asymptotic improvements in the time complexity of the Voronoi cell algorithm and enumeration algorithms.

2 The provable `ListSieve-Birthday` algorithm of Pujol and Stehlé

Using the birthday paradox [63], Pujol and Stehlé [74] showed that the constant in the exponent of the time complexity of the original `ListSieve` algorithm of Micciancio and Voulgaris [65, Section 3.1] can be reduced by almost 25%. The algorithm is presented in Algorithm 1. Here $\gamma = 1 - \frac{1}{n}$, $B_n(\mathbf{0}, \xi\mu)$ denotes the ball centered at $\mathbf{0}$ of radius $\xi\mu$, and the various other parameters will be discussed below.

2.1 Description of the algorithm

The algorithm can roughly be divided in three stages, as follows.

First, the algorithm generates a long list L of lattice vectors with norms between $R\mu$ and $\|B\|$. This 'dummy' list is used for technical reasons to make the proof strategy work. The number of samples used for generating this list is taken as a random variable, which again is done to make certain proof techniques work. Note that besides the actual lattice vectors $\mathbf{v}$, to generate this list we also consider slightly perturbed vectors $\mathbf{v}'$ which are not in the lattice, but are at most $\xi\mu$ away from $\mathbf{v}$. This is yet again a technical modification purely aimed at making the proofs work, as experiments show that without such perturbed vectors, these algorithms also work fine.

After generating L, we generate a fresh list of short lattice vectors S. The procedure for generating these vectors is similar to that of generating T, with two exceptions: (i) now all sampled lattice vectors are added to S (regardless of their norms), and (ii) the vectors are reduced with the dummy list L rather than with vectors in S. The latter guarantees that the vectors in S are all independent and identically distributed.

Finally, when S has been generated, we hope that it contains two distinct lattice vectors $\mathbf{s}_1$, $\mathbf{s}_2$ that are at most $\mu \approx \lambda_1(\mathscr{L})$ apart. So we search $S \times S$ for a pair $(\mathbf{s}_1, \mathbf{s}_2)$ of close, distinct lattice vectors, and return their difference.

2.2 Classical complexities

With a classical search applied to the subroutines in Lines 6, 15, and 19, Pujol and Stehlé analyzed that the costs of the algorithm are:

- Cost of generating L: $\tilde{O}(N_1 \cdot |L|) = 2^{(c_g+2c_t)n+o(n)}$.
- Cost of generating S: $\tilde{O}(N_2 \cdot |L|) = 2^{(c_g+\frac{1}{2}c_b+c_t)n+o(n)}$.
- Cost of searching S for a pair of close vectors: $\tilde{O}(|S|^2) = 2^{(2c_g+c_b)n+o(n)}$.
- Memory requirement of storing S and L: $O(|S|+|L|) = 2^{\max(c_t, c_g+\frac{1}{2}c_b)n+o(n)}$.

The constants c_b, c_t, c_g, N_1 and N_2 above are defined as

$$c_b = 0.401 + \log_2(R), \qquad N_1 = 2^{(c_g+c_t)n+o(n)}, \tag{1}$$

$$c_t = 0.401 + \frac{1}{2}\log_2\left(1 + \frac{2\xi}{R-2\xi}\right), \qquad N_2 = 2^{(c_g+c_b/2)n+o(n)} \tag{2}$$

$$c_g = \frac{1}{2}\log_2\left(\frac{4\xi^2}{4\xi^2-1}\right). \tag{3}$$

In [74] this led to the following result on the time and space complexities.

Lemma 1 *[74] Let $\xi > \frac{1}{2}$ and $R > 2\xi$, and suppose $\mu > \lambda_1(\mathscr{L})$. Then with probability at least $\frac{1}{16}$, the `ListSieve-Birthday` algorithm returns a lattice vector $\mathbf{s} \in \mathscr{L} \setminus \{\mathbf{0}\}$ with $\|\mathbf{s}\| < \mu$, in time at most $2^{c_{\text{time}}n+o(n)}$ and space at most $2^{c_{\text{space}}n+o(n)}$, where c_{time} and c_{space} are given by*

$$c_{\text{time}} = \max\left(c_g + 2c_t, c_g + \frac{c_b}{2} + c_t, 2c_g + c_b\right), \quad c_{\text{space}} = \max\left(c_t, c_g + \frac{c_b}{2}\right). \tag{4}$$

By balancing ξ and R optimally, Pujol and Stehlé obtained the following result.

Corollary 1 *[74] Letting $\xi \approx 0.9476$ and $R \approx 3.0169$, we obtain*

$$c_{\text{time}} \approx 2.465, \quad c_{\text{space}} \approx 1.233. \tag{5}$$

Thus, using polynomially many queries to the `ListSieve-Birthday` algorithm with these parameters, we can find a shortest vector in a lattice with probability exponentially close to 1 using time at most $2^{2.465n+o(n)}$ and space at most $2^{1.233n+o(n)}$.

2.3 Quantum complexities

Applying a quantum search subroutine to Lines 6, 15, and 19, we get the following costs for the quantum algorithm based on `ListSieve-Birthday`:

- Cost of generating L: $\tilde{O}(N_1 \cdot \sqrt{|L|}) = 2^{(c_g+\frac{3}{2}c_t)n+o(n)}$.
- Cost of generating S: $\tilde{O}(N_2 \cdot \sqrt{|L|}) = 2^{(c_g+\frac{1}{2}c_b+\frac{1}{2}c_t)n+o(n)}$.
- Cost of searching S for a pair of close vectors: $\tilde{O}(\sqrt{|S|^2}) = 2^{(c_g+\frac{1}{2}c_b)n+o(n)}$.
- Memory requirement of storing S and L: $O(|S|+|L|) = 2^{\max(c_t, c_g+\frac{1}{2}c_b)n+o(n)}$.

This leads to the following general lemma about the overall quantum time and space complexities.

Lemma 2 *Let $\xi > \frac{1}{2}$ and $R > 2\xi$, and suppose $\mu > \lambda_1(\mathscr{L})$. Then with probability at least $\frac{1}{16}$, the `ListSieve-Birthday` algorithm returns a lattice vector $\mathbf{s} \in \mathscr{L} \setminus \{\mathbf{0}\}$ with $\|\mathbf{s}\| < \mu$ on a quantum computer in time at most $2^{q_{\text{time}}n+o(n)}$ and space at most $2^{q_{\text{space}}n+o(n)}$, where q_{time} and q_{space} are given by*

$$q_{\text{time}} = \max\left(c_g + \frac{3c_t}{2}, c_g + \frac{c_b}{2} + \frac{c_t}{2}, c_g + \frac{c_b}{2}\right), \quad q_{\text{space}} = \max\left(c_t, c_g + \frac{c_b}{2}\right). \tag{6}$$

Re-optimizing the parameters ξ and R subject to the given constraints, to minimize the overall time complexity, we obtain the following result.

Algorithm 2 The `NV-Sieve` algorithm

1: Sample a list L_0 of exponentially many random lattice vectors, and set $m = 0$
2: **repeat**
3: Compute the maximum norm $R_m = \max_{\mathbf{v} \in L_m} \|\mathbf{v}\|$
4: Initialize an empty list L_{m+1} and an empty list of centers C_{m+1}
5: **for each** $\mathbf{v} \in L_m$ **do**
6: **if** $\|\mathbf{v}\| \leq \gamma R_m$ **then**
7: Add $\mathbf{v}$ to the list L_{m+1}
8: Continue the loop
9: **while** $\mathbf{w} \leftarrow \text{Search}\{\mathbf{w} \in C_{m+1} : \|\mathbf{v} \pm \mathbf{w}\| \leq \|\mathbf{w}\|\}$ **do**
10: Reduce $\mathbf{v}$ with $\mathbf{w}$
11: Add $\mathbf{v}$ to the list L_{m+1}
12: Continue the outermost loop
13: Add $\mathbf{v}$ to the centers C_{m+1}
14: Increment m by 1
15: **until** L_m is empty
16: Search for a shortest vector in L_{m-1}

Theorem 1 *Letting* $\xi \approx 0.9086$ *and* $R \approx 3.1376$*, we obtain*

$$q_{\text{time}} \approx 1.799, \quad q_{\text{space}} \approx 1.286. \tag{7}$$

Thus, using polynomially many queries to the `ListSieve-Birthday` *algorithm, we can find a shortest non-zero vector in a lattice on a quantum computer with probability exponentially close to* 1*, in time at most* $2^{1.799n+o(n)}$ *and space at most* $2^{1.286n+o(n)}$.

So the constant in the exponent of the time complexity decreases by about 27% when using quantum search.

Remark. If we generate S in parallel, we can potentially achieve a time complexity of $2^{1.470n+o(n)}$, by setting $\xi \approx 1.0610$ and $R \approx 4.5166$. However, it would require exponentially many parallel quantum computers of size $O(n)$ to achieve a substantial theoretical speed-up over the $2^{1.799n+o(n)}$ of Theorem 1.

3 The heuristic `NV-Sieve` algorithm of Nguyen and Vidick

In 2008, Nguyen and Vidick [69] considered a heuristic, practical variant of the original `AKS-Sieve` algorithm of Ajtai et al. [4], which 'provably' returns a shortest vector under a certain natural, heuristic assumption. A slightly modified but essentially equivalent description of this algorithm is given in Algorithm 2.

3.1 Description of the algorithm

The algorithm starts by generating a big list L_0 of random lattice vectors with length at most $n\|B\|$. Then, by repeatedly applying a sieve to this list, shorter lists of shorter vectors are obtained, until the list is completely depleted. In that case, we go back one step and search for the shortest vector in the last non-empty list.

The sieving step consists of splitting the previous list L_m in a set of 'centers' C_{m+1} and a new list of vectors L_{m+1} that will be used for the next round. For each vector $\mathbf{v} \in L_m$, the algorithm first checks if a vector $\mathbf{w} \in C_m$ exists that is close to $\pm\mathbf{v}$. If this is the case, then we add the vector $\mathbf{v} \pm \mathbf{w}$ to L_{m+1}. Otherwise $\mathbf{v}$ is added to C_{m+1}. Since the set C_{m+1} consists of vectors with a bounded norm and any two vectors in this list have a specified minimum pairwise distance, one can bound the size of C_{m+1} from above using a result of Kabatiansky and Levenshtein [45] regarding sphere packings. In other words, C_{m+1} will be sufficiently small, so that sufficiently many vectors are left for inclusion in the list L_{m+1}. After applying the sieve, we discard all vectors in C_{m+1} and apply the sieve again to the vectors in L_{m+1}.

3.2 Classical complexities

In Line 9 of Algorithm 2, we have highlighted an application of a search subroutine that could be replaced by a quantum search. Using a standard classical search algorithm for this subroutine, under a certain heuristic assumption Nguyen and Vidick give the following estimate for the time and space complexity of their algorithm. Note

that these estimates are based on the observation that the sizes of S and C are bounded from above by $2^{c_h n+o(n)}$, so that the total space complexity is at most $O(|S|+|C|) = 2^{c_h n+o(n)}$ and the total time complexity is at most $\tilde{O}(|S|\cdot|C|) = 2^{2c_h n+o(n)}$, assuming the sieve needs to be performed a polynomial number of times.

Lemma 3 *[69] Let $\frac{2}{3} < \gamma < 1$ and let c_h be defined as*

$$c_h = -\log_2(\gamma) - \frac{1}{2}\log_2\left(1-\frac{\gamma^2}{4}\right). \tag{8}$$

Then the `NV-Sieve` algorithm heuristically returns a shortest non-zero lattice vector $\mathbf{s} \in \mathscr{L}\setminus\{\mathbf{0}\}$ in time at most $2^{c_{\text{time}} n+o(n)}$ and space at most $2^{c_{\text{space}} n+o(n)}$, where c_{time} and c_{space} are given by

$$c_{\text{time}} = 2c_h, \qquad c_{\text{space}} = c_h. \tag{9}$$

To obtain a minimum time complexity, γ should be chosen as close to 1 as possible. Letting $\gamma \to 1$ Nguyen and Vidick thus obtain the following estimates for the complexity of their heuristic algorithm.

Corollary 2 *[69] Letting $\gamma \to 1$, we obtain*

$$c_{\text{time}} \approx 0.415, \quad c_{\text{space}} \approx 0.208. \tag{10}$$

Thus, the `NV-Sieve` algorithm heuristically finds a shortest vector in time $2^{0.415n+o(n)}$ and space $2^{0.208n+o(n)}$.

3.3 Quantum complexities

If we use a quantum search subroutine in Line 9, the complexity of this subroutine decreases from $\tilde{O}(|C|)$ to $\tilde{O}(\sqrt{|C|})$. Since this search is part of the bottleneck for the time complexity, applying a quantum search here will decrease the overall running time as well. Since replacing the classical search by a quantum search does not change the internal behavior of the algorithm, the estimates and heuristics are as valid as they were in the classical setting.

Lemma 4 *Let $\frac{2}{3} < \gamma < 1$. Then the quantum version of the `NV-Sieve` algorithm heuristically returns a shortest non-zero lattice vector in time at most $2^{q_{\text{time}} n+o(n)}$ and space at most $2^{q_{\text{space}} n+o(n)}$, where q_{time} and q_{space} are given by*

$$q_{\text{time}} = \frac{3}{2}\, c_h, \qquad q_{\text{space}} = c_h. \tag{11}$$

Again, minimizing the asymptotic quantum time complexity corresponds to taking γ as close to 1 as possible, which leads to the following result.

Theorem 2 *Letting $\gamma \to 1$, we obtain*

$$q_{\text{time}} \approx 0.312, \quad q_{\text{space}} \approx 0.208. \tag{12}$$

Thus, the quantum version of the `NV-Sieve` algorithm heuristically finds a shortest vector in time $2^{0.312n+o(n)}$ and space $2^{0.208n+o(n)}$.

In other words, applying quantum search to Nguyen and Vidick's sieve algorithm leads to a 25% decrease in the asymptotic exponent of the runtime.

4 The heuristic `GaussSieve` algorithm of Micciancio and Voulgaris

In 2010, Micciancio and Voulgaris [65] described a heuristic variant of their provable `ListSieve` algorithm, for which they could not give a (heuristic) bound on the time complexity, but which has a better heuristic bound on the space complexity, and has a better practical time complexity. The algorithm is described in Algorithm 3.

Algorithm 3 The `GaussSieve` algorithm

```
Initialize an empty list L and an empty stack S
repeat
    Get a vector v from the stack (or sample a new one)
    while w ← Search{w ∈ L : ‖v ± w‖ ≤ ‖v‖} do
        Reduce v with w
    while w ← Search{w ∈ L : ‖w ± v‖ ≤ ‖w‖} do
        Remove w from the list L
        Reduce w with v
        Add w to the stack S
    if v has changed then
        Add v to the stack S
    else
        Add v to the list L
until v is a shortest vector
```

4.1 Description of the algorithm

The algorithm is similar to the `ListSieve-Birthday` algorithm described earlier, with the following main differences: (i) we do not explicitly generate two lists S, L to apply the birthday paradox in the proof; (ii) we do not use a geometric factor $\gamma < 1$ but always reduce a vector if it can be reduced; (iii) we also reduce existing list vectors $\mathbf{w} \in L$ with newly sampled vectors, so that each two vectors in the list are pairwise *Gauss-reduced*; and (iv) instead of specifying the number of iterations in advance, we run the algorithm until we get so many collisions that we are convinced we have found a shortest vector in our list.

4.2 Classical complexities

Micciancio and Voulgaris state that the algorithm above has an experimental time complexity of about $2^{0.52n}$ and a space complexity which is most likely bounded by $2^{0.208n}$ due to the kissing constant [65, Section 5]. In practice this algorithm even seems to outperform the `NV-Sieve` algorithm of Nguyen and Vidick [69]. It is therefore sometimes conjectured that this algorithm also has a time complexity of the order $2^{0.415n+o(n)}$, and the apparent extra factor $2^{0.1n}$ in the experimental time complexity may come from non-negligible polynomial factors in low dimensions. Thus one might conjecture the following.

Conjecture 1 The `GaussSieve` algorithm heuristically returns a shortest non-zero lattice vector in time at most $2^{c_{\text{time}}n+o(n)}$ and space at most $2^{c_{\text{space}}n+o(n)}$, where c_{time} and c_{space} are given by

$$c_{\text{time}} \approx 0.415, \quad c_{\text{space}} \approx 0.208. \tag{13}$$

Note that this algorithm is again (conjectured to be) quadratic in the space complexity, since each pair of list vectors needs to be compared and potentially reduced at least once (and at most a polynomial number of times) to make sure that the final list is Gauss-reduced.

4.3 Quantum complexities

To this heuristic algorithm, we can again apply the quantum speed-up using quantum search. If the number of times a vector is compared with L to look for reductions is polynomial in n, this then leads to the following result.

Conjecture 2 The quantum version of the `GaussSieve` algorithm heuristically returns a shortest non-zero lattice vector in time at most $2^{q_{\text{time}}n+o(n)}$ and space at most $2^{q_{\text{space}}n+o(n)}$, where q_{time} and q_{space} are given by

$$q_{\text{time}} \approx 0.312, \quad q_{\text{space}} \approx 0.208. \tag{14}$$

This means that the exponent in the time complexity is again conjectured to be reduced by about 25% using quantum search, and the exponents are the same as for the `NV-Sieve` algorithm. Since the `GaussSieve` seems to outperform the `NV-Sieve` in practice, applying quantum search to the `GaussSieve` will probably lead to better practical time complexities.

Algorithm 4 The `HashSieve` algorithm

1: Initialize an empty list L and an empty stack S
2: Initialize t empty hash tables
3: Sample $k \cdot t$ random hash vectors
4: **repeat**
5: Get a vector $\mathbf{v}$ from the stack (or sample a new one)
6: Let the set of candidates C be those vectors that collide with $\mathbf{v}$ in one of the hash tables
7: **while** $\mathbf{w} \leftarrow \text{Search}\{\mathbf{w} \in C : \|\mathbf{v} \pm \mathbf{w}\| \leq \|\mathbf{v}\|\}$ **do**
8: Reduce $\mathbf{v}$ with $\mathbf{w}$
9: **while** $\mathbf{w} \leftarrow \text{Search}\{\mathbf{w} \in C : \|\mathbf{w} \pm \mathbf{v}\| \leq \|\mathbf{w}\|\}$ **do**
10: Remove $\mathbf{w}$ from the list L
11: Remove $\mathbf{w}$ from the t hash tables
12: Reduce $\mathbf{w}$ with $\mathbf{v}$
13: Add $\mathbf{w}$ to the stack S
14: **if** $\mathbf{v}$ has changed **then**
15: Add $\mathbf{v}$ to the stack S
16: **else**
17: Add $\mathbf{v}$ to the list L
18: Add $\mathbf{v}$ to the t hash tables
19: **until** $\mathbf{v}$ is a shortest vector

5 The heuristic `HashSieve` algorithm of Laarhoven

5.1 Description of the algorithm

Recently, a modification of the `GaussSieve` and `NV-Sieve` algorithms was proposed in [53], improving the time complexity by using *angular locality-sensitive hashing* [21]. By storing low-dimensional sketches of the list vectors $\mathbf{w} \in L$ in these algorithms, it is possible to significantly reduce the number of list vectors $\mathbf{w}$ that need to be compared to a target vector $\mathbf{v}$ at the cost of increasing the space complexity. Using an exponential number of hash tables, where each list vector is assigned to one of the hash buckets in each hash table, and where vectors in the same bucket are more likely to be "close" in the Euclidean sense than vectors which are not in the same bin, we obtain the set of candidate close(st) vectors by computing which bucket this vector would have landed in, and taking all vectors from those bins as candidates.

5.2 Classical complexities

With a proper balancing of the parameters, it can be guaranteed (heuristically) that the number of candidate vectors for each comparison is of the order $2^{0.1290n}$. This roughly corresponds to having $O(1)$ colliding vectors in each hash table, as the number of hash tables is also of the order $t = 2^{0.1290n}$, and this choice is optimal in the sense that this leads to a minimal time complexity of $2^{0.3366n}$; the space complexity is also $2^{0.3366n}$, and thus using even more hash tables increases the space complexity beyond the time complexity, thus also increasing the time complexity further. The exact choice of parameters is given below.

Lemma 5 *[53, Corollary 1] Let* $\log_2(t) \approx 0.129$. *Then the* `HashSieve` *algorithm heuristically returns a shortest non-zero lattice vector in time at most* $2^{c_{\text{time}}n+o(n)}$ *and space at most* $2^{c_{\text{space}}n+o(n)}$, *where* c_{time} *and* c_{space} *are given by*

$$c_{\text{time}} \approx 0.337, \qquad c_{\text{space}} \approx 0.337. \tag{15}$$

In other words, using $t \approx 2^{0.129n}$ *hash tables and a hash length of* $k \approx 0.221n$, *the time and space complexities of the* `HashSieve` *algorithm are balanced at* $2^{0.337n+o(n)}$.

5.3 Quantum complexities

With a quantum search on the set of candidates in Lines 7 and 9, we can further reduce the time complexity. The optimization changes in the sense that the time to search the list of candidates with quantum search is potentially reduced from $2^{(2-\alpha)c_n n+o(n)}$ to $2^{(\frac{3}{2}-\frac{1}{2}\alpha)c_n n+o(n)}$, where $c_n = \log_2 N \approx 0.2075$ is the expected log-length of the list L and α is defined in [53]. The numerical optimization of the parameters can be performed again, and leads to the following result.

Theorem 3 *Let* $\log_2(t) \approx 0.078$. *Then the quantum* `HashSieve` *algorithm heuristically returns a shortest non-zero lattice vector in time at most* $2^{q_{\text{time}}n+o(n)}$ *and space at most* $2^{q_{\text{space}}n+o(n)}$, *where* q_{time} *and* q_{space} *are given by*

$$q_{\text{time}} \approx 0.286, \qquad q_{\text{space}} \approx 0.286. \tag{16}$$

In other words, using $t \approx 2^{0.078n}$ *hash tables and a hash length of* $k \approx 0.134n$, *the quantum time and space complexities of the algorithm are balanced at* $2^{0.286n+o(n)}$.

It is possible to obtain a continuous trade-off between the quantum time and space complexities, by choosing $\log_2(t) \in [0, 0.07843]$ differently. Similar to Figure 1 of [53], Figure 1 shows the resulting trade-off, and a comparison with previous classical heuristic time complexities.

6 The heuristic `SphereSieve` algorithm of Laarhoven and De Weger

6.1 Description of the algorithm

Even more recently, another LSH-based modification of the `NV-Sieve` algorithm was proposed in [54], improving upon the time complexity of the HashSieve using *spherical* locality-sensitive hashing. This particular hash method, introduced by Andoni et al. in 2014 [6, 7], works very well for data sets that (approximately) lie on the surface of a hypersphere, which is the case for the iterative sieving steps of the `NV-Sieve`. By dividing up the sphere into $2^{\Theta(\sqrt{n})}$ regions in a way similar as in the `2-Level-Sieve` of Wang et al. [90], it can be guaranteed that vectors have a significantly lower probability of ending up in the same region if their angle is large. Again using exponentially many hash tables, where each list vector is assigned to one of the hash buckets in each hash table, we obtain a set of candidate close(st) vectors by computing which bucket this vector would have landed in, and taking all vectors from those bins as candidates. The algorithm itself is a merge of the introduction of hash tables, as in the `HashSieve`, and the `NV-Sieve` of Nguyen and Vidick, with the important observation is that the hash functions used are different than in the `HashSieve`.

6.2 Classical complexities

With a proper balancing of the parameters, it can be guaranteed (heuristically) that the number of candidate vectors for each comparison is of the order $2^{0.0896n}$, which is again similar to the number of hash tables, which is of the order $t = 2^{0.0896n}$. This choice is optimal in that this leads to a minimal time complexity of $2^{0.2972n}$; the space complexity is also $2^{0.2972n}$, and thus using even more hash tables increases the space complexity beyond the time complexity. The exact choice of parameters is given below.

Lemma 6 *[53, Corollary 1] Let* $\log_2(t) \approx 0.0896$. *Then the* `SphereSieve` *algorithm heuristically returns a shortest non-zero lattice vector in time at most* $2^{c_{\text{time}}n+o(n)}$ *and space at most* $2^{c_{\text{space}}n+o(n)}$, *where* c_{time} *and* c_{space} *are given by*

$$c_{\text{time}} \approx 0.298, \qquad c_{\text{space}} \approx 0.298. \tag{17}$$

In other words, using $t \approx 2^{0.0896n}$ *hash tables and a hash length of* $k = \Theta(\sqrt{n})$, *the time and space complexities of the* `SphereSieve` *algorithm are balanced at* $2^{0.298n+o(n)}$.

6.3 Quantum complexities

With a quantum search on the set of candidates, we can again potentially reduce the time complexity. Again, the time to search the list of candidates with quantum search is reduced from $2^{(2-\alpha)c_n n+o(n)}$ to $2^{(\frac{3}{2}-\frac{1}{2}\alpha)c_n n+o(n)}$, where $c_n = \log_2 N \approx 0.2075$ is the expected log-length of the list L and bounds on α are defined in [54]. The numerical optimization of the parameters can be performed again, and leads to the following result.

Theorem 4 *Let* $\log_2(t) \approx 0.0413$. *Then the quantum* `SphereSieve` *algorithm heuristically returns a shortest non-zero lattice vector in time at most* $2^{q_{\text{time}}n+o(n)}$ *and space at most* $2^{q_{\text{space}}n+o(n)}$, *where* q_{time} *and* q_{space} *are given by*

$$q_{\text{time}} \approx 0.268, \qquad q_{\text{space}} \approx 0.268. \tag{18}$$

In other words, using $t \approx 2^{0.0413n}$ *hash tables and a hash length of* $k = \Theta(\sqrt{n})$, *the quantum time and space complexities of the algorithm are balanced at* $2^{0.268n+o(n)}$.

Again, one may obtain a trade-off between the quantum time and space complexities, by choosing $\log_2(t) \in [0, 0.0413]$. This trade-off is shown in Figure 1.

7 The approximate `ListSieve-Birthday` analysis of Liu et al.

7.1 Description of the algorithm

While most sieving algorithms are concerned with finding exact solutions to the shortest vector problem (i.e., finding a lattice vector whose norm is the minimum over all non-zero lattice vectors), in many cryptographic applications, finding a short (rather than shortest) vector in the lattice also suffices. Understanding the costs of finding approximations to shortest vectors (i.e., solving SVP_δ with $\delta > 1$) may therefore be as important the costs of exact SVP.

In 2011, Liu et al. [58] analyzed the impact of this relaxation of SVP on lattice sieving algorithms. In particular, they analyzed the `ListSieve-Birthday` algorithm of Section 2, taking into account the fact that an approximate solution is sufficient. The algorithm, described in [58, Algorithm 1], is effectively identical to the original `ListSieve-Birthday` algorithm of Pujol and Stehlé [74].

7.2 Classical complexities

Intuitively, the effect of large δ can be understood as that the impact of the use of perturbed lattice vectors (rather than actual lattice vectors) becomes less and less. In the limit of large δ, the impact of perturbations disappears (although it still guarantees correctness of the algorithm), and we get the same upper bound on the list size of $2^{0.401n}$ as obtained for the perturbation-free heuristic version of the *ListSieve*, the `GaussSieve` [65]. Since the runtime of the sieve remains quadratic in the list size, this leads to a time complexity of $2^{0.802n}$.

Lemma 7 *[58, Lemma 10] The* `ListSieve-Birthday` *algorithm heuristically returns a solution to* SVP_δ *in time at most* $2^{c_{\text{time}}n+o(n)}$ *and space at most* $2^{c_{\text{space}}n+o(n)}$, *where* c_{time} *and* c_{space} *are given by*

$$c_{\text{time}} \approx 0.802, \qquad c_{\text{space}} \approx 0.401. \tag{19}$$

7.3 Quantum complexities

As expected, using quantum search for the search subroutine of the `ListSieve-Birthday` algorithm leads to a gain in the exponent of 25%; a single search can be done in time $\tilde{O}(\sqrt{N})$, leading to a total time complexity of $\tilde{O}(N^{3/2})$ rather than $\tilde{O}(N^2)$.

Theorem 5 *The quantum* `ListSieve-Birthday` *algorithm heuristically returns a* δ*-approximation to the shortest non-zero lattice vector in time at most* $2^{q_{\text{time}}n+o(n)}$ *and space at most* $2^{q_{\text{space}}n+o(n)}$, *where* q_{time} *and* q_{space} *are given by*

$$q_{\text{time}} \approx 0.602, \qquad q_{\text{space}} \approx 0.401. \tag{20}$$

8 Other sieve algorithms

8.1 The provable AKS-Sieve of Ajtai et al.

Ajtai et al. [4] did not provide an analysis with concrete constants in the exponent in their original paper of the AKS-Sieve. We expect that it is possible to speed up this version of the algorithm using quantum search as well, but instead we consider several subsequent variants that are easier to analyse.

The first of these was by Regev [77], who simplified the presentation and gave concrete constants for the running time and space complexity. His variant is quadratic in the list size, which is bounded by $2^{8n+o(n)}$, leading to a worst-case time complexity of $2^{16n+o(n)}$. Using quantum search, the exponent in the runtime decreases by 25%, which results in a run-time complexity of $2^{12n+o(n)}$.

Nguyen and Vidick [69] improved this analysis by carefully choosing the parameters of the algorithm, which resulted in a space complexity of $2^{2.95n+o(n)}$. The running time of $2^{5.9n+o(n)}$ is again quadratic in the list size, and can be improved using quantum search by 25% to $2^{4.425n}$.

Micciancio and Voulgaris improve the constant as follows. Say that the initial list contains $2^{c_0 n+o(n)}$ vectors, the probability that a point is not a collision at the end is $p = 2^{-c_u n+o(n)}$ and the maximum number of points used as centers is $2^{c_s n+o(n)}$. Each step of sieving costs $2^{(c_0+c_s)n+o(n)}$ time. Now, after k sieving steps of the algorithm the number of points will be $|P_k| = \tilde{O}(2^{c_0 n} - k2^{c_s n})$, which results in $|V_k| = \tilde{O}((2^{c_0 n} - k2^{c_s n})/2^{c_u n}) \approx 2^{c_R n+o(n)}$ distinct non-perturbed lattice points. This set P_k is then searched for a pair of lattice vectors such that the difference is a non-zero shortest vector, which classically costs $|V_k| \cdot |P_k| = 2^{(2c_R+c_u)n+o(n)}$.

Classical complexities. In the above description, we have the following correspondence:

$$c_u = \log\left(\frac{\xi}{\sqrt{\xi^2 - 0.25}}\right), \qquad c_s = 0.401 + \log\left(\frac{1}{\gamma}\right), \tag{21}$$

$$c_R = 0.401 + \log\left(\xi\left(1 + \frac{1}{1-\gamma}\right)\right), \qquad c_0 = \max\{c_s, c_R + c_u\}. \tag{22}$$

where $\xi \in [0.5, \frac{1}{2}\sqrt{2})$ and $\gamma < 1$. The space complexity is $2^{c_0 n+o(n)}$ and the time complexity is $2^{c_T n+o(n)}$ with

$$c_{\text{time}} = \max\{c_0 + c_s,\ 2c_R + c_u\}, \qquad c_{\text{space}} = c_0. \tag{23}$$

Optimizing ξ and γ to minimize the classical time complexity leads to $\xi \approx 0.676$ and $\gamma \approx 0.496$ which gives space $2^{1.985n+o(n)}$ and time $2^{3.398n+o(n)}$.

Quantum complexities. Quantum searching in the sieving step speeds up this part of the algorithm to $2^{(c_0+\frac{1}{2}c_s)n+o(n)}$. In the final step quantum search can be used to speed up the search to $\sqrt{|V_k|} \cdot |P_k| = \sqrt{2^{c_R n}} \cdot 2^{(c_R+c_u)n+o(n)} = 2^{(\frac{3}{2}c_R+c_u)n+o(n)}$. Thus, the exponents of the quantum time and space become

$$q_{\text{time}} = \max\left\{c_0 + \frac{1}{2}\,c_s,\ \frac{3}{2}\,c_R + c_u\right\}, \qquad q_{\text{space}} = c_0. \tag{24}$$

Optimizing gives $\xi \to \frac{1}{2}\sqrt{2}$ and $\gamma = 0.438$, which results in a space complexity of $2^{1.876n+o(n)}$ and running time of $2^{2.672n+o(n)}$.

8.2 The provable ListSieve of Micciancio and Voulgaris

The provable ListSieve algorithm of Micciancio and Voulgaris [65] was introduced as a provable variant of their heuristic GaussSieve algorithm, achieving a better time complexity than with the optimized analysis of the AKS-Sieve. Instead of starting with a big list and repeatedly applying a sieve to reduce the length of the list (and the norms of the vectors in the list), the ListSieve builds a longer and longer list of vectors, where each new vector to be added to the list is first reduced with all other vectors in the list. (But unlike the GaussSieve, vectors already in the list are never modified.) Complete details of the algorithm and its analysis can be found in [65].

Classical complexities. First, for $\xi \in (0.5, 0.7)$ we write

$$c_1 = 0.401 + \log_2\left(\xi + \sqrt{1+\xi^2}\right), \qquad c_2 = \log_2\left(\frac{\xi}{\sqrt{\xi^2 - \frac{1}{4}}}\right). \tag{25}$$

Then the `ListSieve` algorithm has a provable complexity of at most $2^{(2c_1+c_2)n+o(n)}$ (time) and $2^{c_1 n+o(n)}$ (space) for any ξ in this interval. Minimizing the time complexity leads to $\xi \approx 0.685$, with a time complexity of $2^{3.199n+o(n)}$ and a space complexity of $2^{1.325n+o(n)}$.

Quantum complexities. Using quantum search, it can be seen that the inner search of the list of length $N = 2^{c_1 n+o(n)}$ can now be performed in time $2^{\frac{1}{2}c_1 n+o(n)}$. Thus the total time complexity becomes $2^{(\frac{3}{2}c_1+c_2)n+o(n)}$ now. Optimizing for ξ shows that the optimum is at the boundary of $\xi \to 0.7$.

Looking a bit more closely at Micciancio and Voulgaris' analysis, we see that the condition $\xi < 0.7$ comes from the condition that $\xi \times \mu \leq \frac{1}{2}\lambda_1^2$. Taking $\mu < 1.01\lambda_1$ then approximately leads to the given bound for ξ, and since in the classical case the optimum does not lie at the boundary anyway, this was sufficient for Micciancio and Voulgaris. However, now that the optimum is at the boundary, we can see that we can slightly push the boundary further and slightly relax the condition $\xi < 0.7$. For any constant $\varepsilon > 0$ we can also let $\mu < (1+\varepsilon)\lambda_1$ without losing any performance of the algorithm, and for small ε this roughly translates to the bound $\xi < \frac{1}{2}\sqrt{2} \approx 0.707$.

With this adjustment in their analysis, the optimum is at the boundary of $\xi \to \frac{1}{2}\sqrt{2}$, in which case we get a quantum time complexity of $2^{2.527n+o(n)}$ and a space complexity of $2^{1.351n+o(n)}$.

8.3 The provable `AKS-Sieve-Birthday` algorithm of Hanrot et al.

Hanrot, Pujol and Stehlé [41] described a speed-up for the `AKS-Sieve` using the birthday paradox [63], similar to the speed-up that Pujol and Stehlé describe for `Listsieve`. Recall that the `AKS-Sieve` consists of an initial sieving step that generates a list of reasonably small vectors, followed by a pairwise comparison of the remaining vectors because the difference of at least one pair is expected to be a shortest non-zero vector. If this list of reasonably small vectors are independent and identically distributed, the number of vectors required is reduced to the square root of the original amount by the birthday paradox. They describe how this can be done for the `AKS-Sieve`, which requires fixing what vectors are used as centers for every iteration. This means that when a perturbed vector does not lie close to any of the fixed centers in the generation of the list of small vectors, it is discarded.

Classical complexities. For $\xi > \frac{1}{2}$ and $\gamma < 1$ we write

$$c_t = 0.401 - \log_2(\gamma), c_b = 0.401 + \log_2\left(\xi + \frac{\xi}{1-\gamma}\right), c_g = -\frac{1}{2}\log_2\left(1 - \frac{1}{4\xi^2}\right). \tag{26}$$

The `AKS-Sieve-Birthday` algorithm now has a time complexity of $2^{c_{\text{time}}n+o(n)}$ and a space complexity of $2^{c_{\text{space}}n+o(n)}$, where

$$c_{\text{time}} = c_g + \max\left\{2c_t, c_g + c_t + \frac{c_b}{2}, c_g + c_b\right\}, \quad c_{\text{space}} = c_g + \max\left\{c_t, \frac{c_b}{2}\right\}. \tag{27}$$

Working out the details, the classically optimized constants are $\xi \to 1$ and $\gamma \approx 0.609$ leading to a time complexity of $2^{2.64791n+o(n)}$ and a space complexity of $2^{1.32396n+o(n)}$.

Quantum complexities. Replacing various steps with quantum search gives the same space exponent $q_{\text{space}} = c_{\text{space}}$ as in the classical case, and leads to the following time exponent:

$$q_{\text{time}} = c_g + \max\left\{\frac{3c_t}{2}, \frac{c_t}{2} + \frac{c_b}{2}, \frac{c_g}{2} + c_t + \frac{c_b}{4}, \frac{c_g}{2} + \frac{3c_b}{4}\right\}. \tag{28}$$

Optimizing the parameters to obtain the lowest quantum time complexity, we get the same constants $\xi \to 1$ and $\gamma \approx 0.609$ leading to a time complexity of $2^{1.98548n+o(n)}$ (which is exactly a 25% gain in the exponent) and a space complexity of $2^{1.32366n+o(n)}$.

8.4 The heuristic `2-Level-Sieve` of Wang et al.

To improve upon the time complexity of the algorithm of Nguyen and Vidick, Wang et al. [90] introduced a further trade-off between the time complexity and the space complexity. Their algorithm uses two lists of centers C_1 and C_2 and two geometric factors γ_1 and γ_2, instead of the single list C and single geometric factor γ in the algorithm of Nguyen and Vidick. For details, see [90].

Classical complexities. The classical time complexity of this algorithm is bounded from above by $\tilde{O}(|C_1| \cdot |C_2| \cdot (|C_1| + |C_2|))$, while the space required is at most $O(|C_1| \cdot |C_2|)$. Optimizing the constants γ_1 and γ_2 in their paper leads to $(\gamma_1, \gamma_2) \approx (1.0927, 1)$, with an asymptotic time complexity of less than $2^{0.384n+o(n)}$ and a space complexity of about $2^{0.256n+o(n)}$.

Quantum complexities. By using the quantum search algorithm for searching the lists C_1 and C_2, the time complexity is reduced to $\tilde{O}(|C_1| \cdot |C_2| \cdot \sqrt{|C_1| + |C_2|})$, while the space complexity remains $O(|C_1| \cdot |C_2|)$. Re-optimizing the constants for a minimum quantum time complexity leads to $(\gamma_1, \gamma_2) \approx (\sqrt{2}, 1)$, leading to the same time and space complexities as the quantum-version of the algorithm of Nguyen and Vidick. Due to the simpler algorithm and smaller constants, a quantum version of the algorithm of Nguyen and Vidick will most likely be more efficient than a quantum version of the algorithm of Wang et al.

8.5 The heuristic `3-Level-Sieve` of Zhang et al.

To further improve upon the time complexity of the `1-Level-Sieve` (`NV-Sieve`) of Nguyen and Vidick and the `2-Level-Sieve` of Wang et al. [90], Zhang et al. [91] introduced the `3-Level-Sieve`, with a further trade-off between the time complexity and the space complexity. Their algorithm generalizes the `2-Level-Sieve` with two lists of centers (with different radii) to three lists of centers. For the complete details of this algorithm, see [91].

Classical complexities. The classical time complexity of this algorithm is bounded by $\tilde{O}(|C_1| \cdot |C_2| \cdot |C_3| \cdot (|C_1| + |C_2| + |C_3|))$, while the space required is at most $O(|C_1| \cdot |C_2| \cdot |C_3|)$. Optimizing the constants $\gamma_1, \gamma_2, \gamma_3$ leads to $(\gamma_1, \gamma_2, \gamma_3) \approx (1.1399, 1.0677, 1)$, with an asymptotic time complexity of less than $2^{0.378n+o(n)}$ and a space complexity of about $2^{0.283n+o(n)}$.

Quantum complexities. By using the quantum search algorithm for searching the lists $C_{1,2,3}$, the time complexity is reduced to $\tilde{O}(|C_1| \cdot |C_2| \cdot |C_3| \cdot \sqrt{|C_1| + |C_2| + |C_3|})$, while the space complexity remains $O(|C_1| \cdot |C_2| \cdot |C_3|)$. Re-optimizing the constants for a minimum time complexity leads to $(\gamma_1, \gamma_2, \gamma_3) \approx (\sqrt{2}, \sqrt{2}, 1)$, again leading to the same time and space complexities as the quantum-version of the algorithm of Nguyen and Vidick and the quantum version of the `2-Level-Sieve` of Wang et al. Again the hidden polynomial factors of the `3-Level-Sieve` are much larger, so the quantum version of the `NV-Sieve` of Nguyen and Vidick is most likely faster.

8.6 The heuristic `Overlattice-Sieve` of Becker et al.

The `Overlattice-Sieve` works by decomposing the lattice into a sequence of overlattices such that the lattice at the bottom corresponds to the challenge lattice, whereas the lattice at the top corresponds to a lattice where enumerating short vectors is easy due to orthogonality. The algorithm begins by enumerating many short vectors in the top lattice and then iteratively moves down through the sequence of lattices by combining short vectors in the overlattice to form vectors in the lattice directly below it in the sequence. It keeps only the short non-zero vectors that are formed in this manner and uses them for the next iteration. In the last iteration, it generates short vectors in the challenge lattice, and these give a solution to the shortest vector problem. Since the algorithm actually works on cosets of the lattice, it is more naturally seen as an algorithm for the closest vector problem, which it solves as well. The algorithm relies on the assumption that the Gaussian Heuristic holds in all the lattices in the sequence.

More specifically, at any one time the algorithm deals with β^n vectors that are divided into α^n buckets with on average β^n/α^n vectors per bucket. These buckets are divided into pairs such that any vector from a bucket and any vector from its paired bucket combine into a lattice vector in the sublattice. Therefore, exactly β^{2n}/α^n combinations need to be made in each iteration.

Classical complexities. The above leads to a classical running time of $\tilde{O}(\beta^{2n}/\alpha^n)$ and a space complexity of $\tilde{O}(\beta^n)$, under the constraints that

$$1 < \alpha < \sqrt{2}, \quad \alpha^n \in \mathbb{Z}, \quad \beta\sqrt{1-\frac{\alpha^2}{4}} \geq 1+\varepsilon_n,$$

where ε_n is some function that decreases towards 0 as n grows. Optimizing α and β for the best time complexity gives $\alpha = \sqrt{\frac{4}{3}}$ and $\beta = \sqrt{\frac{3}{2}}$ for a running time of $2^{0.3774n+o(n)}$ and a space complexity of $2^{0.2925n+o(n)}$.

Quantum complexities. By using the quantum search algorithm to search for suitable combinations for every vector, the running time can be reduced to $\tilde{O}(\beta^{3n/2}/\alpha^{n/2})$. This is optimal for $\alpha \to 1$, $\beta = \sqrt{\frac{4}{3}}$, which gives a quantum time complexity of $2^{0.311n+o(n)}$ and a space complexity of $2^{0.2075n+o(n)}$. Interestingly, in the classical case there is a trade-off between α and β, which allows for a bigger α (reducing the running time) at the cost of increasing β (increasing the space complexity). In the quantum case, this trade-off is no longer possible: increasing α and β actually leads to a larger running time as well as a larger space complexity. Thus, the resulting quantum complexity is heuristically no better than the other sieving algorithms, but this algorithm solves CVP as well as SVP.

9 Other SVP algorithms

9.1 Enumeration algorithms

In enumeration, all lattice vectors are considered inside a giant ball around the origin that is known to contain at least one lattice vector. Let $\mathscr{L}$ be a lattice with basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$. Consider each lattice vector $\mathbf{u} \in \mathscr{L}$ as a linear combination of the basis vectors, i.e., $\mathbf{u} = \sum_i u_i \mathbf{b}_i$. Now, we can represent each lattice vector by its coefficient vector $(u_1, \ldots, u_n)$. We would like to have all combinations of values for $(u_1, \ldots, u_n)$ such that the corresponding vector $\mathbf{u}$ lies inside the ball. We could try any combination and see if it lies within the ball by computing the norm of the corresponding vector, but there is a smarter way that ensures we only consider vectors that lie within the ball and none that lie outside.

To this end, enumeration algorithms search from right to left, by identifying all values for u_n such that there might exist $u'_1, \ldots, u'_{n-1}$ such that the vector corresponding to $(u'_1, \ldots, u'_{n-1}, u_n)$ lies in the ball. To identify these values $u'_1, \ldots, u'_{n-1}$, enumeration algorithms use the Gram-Schmidt orthogonalization of the lattice basis as well as the projection of lattice vectors. Then, for each of these possible values for u_n, the enumeration algorithm considers all possible values for u_{n-1} and repeats the process until it reaches possible values for u_1. This leads to a search which is serial in nature, as each value of u_n will lead to different possible values for u_{n-1} and so forth. Unfortunately, we can only really apply the quantum search algorithm to problems where the list of objects to be searched is known in advance.

One might suggest to forego the smart way to find short vectors and just search all combinations of $(u_1, \ldots, u_n)$ with appropriate upper and lower bounds on the different u_i's. Then it becomes possible to apply quantum search, since we now have a predetermined list of vectors and just need to compute the norm of each vector. However, it is doubtful that this will result in a faster algorithm, because the recent heuristic changes by Gama et al. [34] have reduced the running time of enumeration dramatically (roughly by a factor $2^{n/2}$) and these changes only complicate the search area further by changing the ball to an ellipsoid. There seems to be no simple way to apply quantum search to the enumeration algorithms that are currently used in practice, but perhaps the algorithms can be modified in some way.

9.2 The Voronoi cell algorithm

Consider a set of points in the Euclidean space. For any given point in this set, its Voronoi cell is defined as the region that contains all vectors that lie closer to this point than to any of the other points in the set. Now, given a Voronoi cell, we define a relevant vector to be any vector in the set whose removal from the set will change this particular Voronoi cell. If we pick our lattice as the set and we consider the Voronoi cell around the zero vector, then any shortest vector is also a relevant vector. Furthermore, given the relevant vectors of the Voronoi cell we can solve the closest vector problem in $2^{2n+o(n)}$ time.

So how can we compute the relevant vectors of the Voronoi cell of a lattice $\mathscr{L}$? Micciancio and Voulgaris [64] show that this can be done by solving $2^n - 1$ instances of CVP in the lattice $2\mathscr{L}$. However, in order to solve CVP we would need the relevant vectors which means we are back to our original problem. Micciancio and Voulgaris show that these instances of CVP can also be solved by solving several related CVP instances in a lattice of lower rank. They give a basic and an optimized version of the algorithm. The basic version only uses LLL as preprocessing and solves all these related CVP instances in the lower rank lattice separately. As a consequence, the basic algorithm runs in time $2^{3.5n+o(n)}$ and in space $2^{n+o(n)}$. The optimized algorithm uses a stronger preprocessing for the lattice basis, which takes exponential time. But since the most expensive part is the computation of the Voronoi relevant vectors, this extra preprocessing time does not increase the asymptotic running time as it is executed only once. In fact, having the reduced basis decreases the asymptotic running time to $\tilde{O}(2^{3n})$. Furthermore, the optimized algorithm employs a trick that allows it to reduce 2^k CVP instances in a lattice of rank k to a single instance of an enumeration problem related to the same lattice. The optimized algorithm solves CVP in time $\tilde{O}(2^{2n})$ using $\tilde{O}(2^n)$ space.

Now, in the basic algorithm, it would be possible to speed up the routine that solves CVP given the Voronoi relevant vectors using a quantum computer. It would also be possible to speed up the routine that removes non-relevant vectors from the list of relevant vectors using a quantum computer. Combining these two changes gives a quantum algorithm with an asymptotic running time $\tilde{O}(2^{2.5n})$, which is still slower than the optimized classical algorithm. It is not possible to apply these same speedups to the optimized algorithm due to the aforementioned trick with the enumeration problem. The algorithm to solve this enumeration problem makes use of a priority queue, which means the search is not trivially parallellized. Once again, there does not seem to be a simple way to apply quantum search to this special enumeration algorithm. However, it may be possible that the algorithm can be modified in such a way that quantum search can be applied.

9.3 Discrete Gaussian sampling

A very recent method for finding shortest vectors in lattices is based on sampling and combining lattice vectors sampled from a discrete Gaussian on the lattice. Given a lattice, a discrete Gaussian distribution on the lattice is what you might expect it to be; the probability of sampling a non-lattice vector is 0, and the probability of sampling a lattice vector $\mathbf{x}$ is proportional to $\exp -O(\|\mathbf{x}\|^2)$, comparable to a regular multivariate Gaussian distribution. These discrete Gaussians are commonly used in lattice-based cryptographic primitives, such as lattice-based signatures [27, 60], and it is folklore that sampling from a discrete Gaussian distribution with a very large standard deviation is easy, while sampling from a distribution with a small standard deviation (often sampling short vectors) is hard.

The idea of Aggarwal et al. [1] to solve SVP with discrete Gaussian sampling is as follows. First, many vectors are sampled from a discrete Gaussian with large standard deviation. Then, to find shorter and shorter lattice vectors, list vectors are combined and averaged to obtain samples from a Gaussian with a smaller standard deviation. More precisely, two samples from a discrete Gaussian with width σ can be combined and averaged to obtain a sample from a discrete Gaussian with width $\sigma/\sqrt{2}$. To be able to combine and average list vectors, they need to be in the same coset of $2\mathscr{L}$, which means that 2^n buckets are stored, and within each bucket (coset) vectors are combined to obtain new, shorter samples. Overall, this leads to a time complexity of $2^{n+o(n)}$ for SVP, also using space $2^{n+o(n)}$.

The ideas of this algorithm are actually quite similar to the `Overlattice-Sieve` of Becker et al. [10] which we discussed in Section 8.6. Vectors are already stored in buckets, and so searching for vectors in the same coset is not costly at all. In this algorithm, the number of vectors in each bucket is even sub-exponential in n, so a quantum search speed-up does not seem to bring down the asymptotic time or space complexities at all. Due to the number of cosets of $2\mathscr{L}$ (2^n), this algorithm seems (classically and quantumly) bound by a time and space complexity of $2^{n+o(n)}$, which it already achieves.

Acknowledgments.

This report is partly a result of fruitful discussions at the Lorentz Center Workshop on Post-Quantum Cryptography and Quantum Algorithms, Nov. 5–9 2012, Leiden, The Netherlands and partly of discussions at IQC, Aug. 2014, Waterloo, Canada. In particular, we would like to thank Felix Fontein, Nadia Heninger, Stacey Jeffery, Stephen Jordan, John M. Schanck, Michael Schneider, Damien Stehlé and Benne de Weger for valuable discussions. Finally, we thank the anonymous reviewers for their helpful comments and suggestions.

The first author is supported by DIAMANT. The second author is supported by Canada's NSERC (Discovery, SPG FREQUENCY, and CREATE CryptoWorks21), MPrime, CIFAR, ORF and CFI; IQC and Perimeter Institute are supported in part by the Government of Canada and the Province of Ontario. The third author is supported in part by EPSRC via grant EP/I03126X.

References

1. Aggarwal, D., Dadush, D., Regev, O., Stephens-Davidowitz, N.: Solving the shortest vector problem in 2^n time via discrete Gaussian sampling. In: STOC (2015)
2. Aharonov, D., Regev, O.: A lattice problem in quantum NP. In: FOCS, pp. 210–219 (2003)
3. Ajtai, M.: The shortest vector problem in L_2 is NP-hard for randomized reductions. In: STOC, pp. 10–19 (1998)
4. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: STOC, pp. 601–610 (2001)
5. Ambainis, A.: Quantum walk algorithm for element distinctness. In: FOCS, pp. 22–31 (2003)
6. Andoni, A., Indyk, P., Nguyen, H. L., Razenshteyn, I.: Beyond locality-sensitive hashing. In: SODA, pp. 1018–1028 (2014)
7. Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: STOC (2015)
8. Antipa, A., Brown, D., Gallant, R., Lambert, R., Struik, R., Vanstone, S.: Accelerated verification of ECDSA signatures. In: SAC, pp. 307–318 (2006)
9. Aono, Y., Naganuma, K.: Heuristic improvements of BKZ 2.0. IEICE Tech. Rep. 112 (211), pp. 15–22 (2012)
10. Becker, A., Gama, N., Joux, A.: A sieve algorithm based on overlattices. In: ANTS, pp. 49–70 (2014)
11. Bennett, C. H., Bernstein, E., Brassard, G., Vazirani, V.: Strengths and weaknesses of quantum computing. SIAM J. Comput. 26 (5), pp. 1510–1523 (1997)
12. Bernstein, D. J.: Cost analysis of hash collisions: Will quantum computers make SHARCs obsolete?. In: SHARCS (2009)
13. Bernstein, D. J., Buchmann, J., Dahmen, E. (eds.): Post-quantum cryptography. Springer (2008)
14. Blake, I. F., Fuji-Hara, R., Mullin, R. C., Vanstone, S. A.: Computing logarithms in finite fields of characteristic two. SIAM J. Algebraic Discrete Methods 5 (2), pp. 276–285 (1984)
15. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: CRYPTO, pp. 129–142 (1996)
16. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik 46, pp. 493–505, (1998)
17. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. In: ITCS, pp. 309–325 (2012)
18. Brassard, G., Høyer P., Tapp A.: Quantum cryptanalysis of hash and claw-free functions. In: LATIN, pp. 163–169 (1998)
19. Brassard, G., Høyer P., Mosca M., and Tapp, A.: Quantum amplitude amplification and estimation. AMS Contemporary Mathematics Series Millennium Vol. entitled Quantum Computation & Information, vol. 305 (2002)
20. Buhrman, B., Dürr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness, SIAM J. Comput. 34 (6), pp. 1324–1330 (2005)
21. Charikar, M. S.: Similarity estimation techniques from rounding algorithms. In: STOC, pp. 380–388 (2002)
22. Chen, Y., Nguyen, P. Q.: BKZ 2.0: Better lattice security estimates. In: ASIACRYPT, pp. 1–20 (2011)
23. Childs, A. M., Van Dam, W.: Quantum algorithms for algebraic problems, Rev. Mod. Phys. 82, pp. 1–52 (2010)
24. Childs, A. M., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. arXiv:1012.4019 (2010)
25. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology 10 (4), pp. 233–260 (1997)
26. Coster, M. J., Joux, A., LaMacchia, B. A., Odlyzko, A. M., Schnorr, C. P., Stern, J.: Improved low-density subset sum algorithms. In: Computational Complexity 2 (2), pp. 111–128 (1992)
27. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: CRYPTO, pp. 40–56 (2013)
28. Ducas, L., Micciancio, D.: Improved short lattice signatures in the standard model. In: CRYPTO, pp. 335–352 (2014)
29. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Math. Comp. 44, pp. 463–471 (1985)
30. Fitzpatrick, R., Bischof, C., Buchmann, J., Dagdelen, Ö., Göpfert, F., Mariano, A., Yang, B.-Y.: Tuning GaussSieve for speed. In: LATINCRYPT, pp. 284–301 (2014)
31. Galbraith, S. D., Scott, M.: Exponentiation in pairing-friendly groups using homomorphisms. In: Pairing-Based Cryptography, pp. 211–224 (2008)
32. Gallant, R. P., Lambert, R. J., Vanstone, S. A.: Faster point multiplication on elliptic curves with efficient endomorphisms. In: CRYPTO, pp. 190–200 (2001)
33. Gama, N., Nguyen, P. Q.: Predicting lattice reduction. In: EUROCRYPT, pp. 31–51 (2008)
34. Gama, N., Nguyen, P. Q., Regev, O.: Lattice enumeration using extreme pruning. In: EUROCRYPT, pp. 257–278 (2010)
35. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC, pp. 197–206 (2008)
36. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178 (2009)
37. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Phys. Rev. Lett., vol. 100, 160501 (2008)
38. Grover, L. K.: A fast quantum mechanical algorithm for database search. In: STOC, pp. 212–219 (1996)
39. Grover, L. K., Rudolph, T.: How significant are the known collision and element distinctness quantum algorithms? Quantum Info. Comput. 4 (3), pp. 201–206 (2004)
40. Hallgren, S.: Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. J. ACM. 54 (1), pp. 653–658 (2007)
41. Hanrot, G., Pujol, X., Stehlé, D.: Algorithms for the shortest and closest lattice vector problems. In: IWCC, pp. 159–190 (2011)
42. Hoffstein, J., Pipher, J., Silverman, J.: NTRU: A ring-based public key cryptosystem. In: ANTS, pp. 267–288 (1998)
43. Ishiguro, T., Kiyomoto, S., Miyake, Y., Takagi, T.: Parallel Gauss Sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In: PKC, pp. 411–428 (2014)
44. Jeffery, S.: Collision finding with many classical or quantum processors. Master's thesis, University of Waterloo (2011)

45. Kabatiansky, G., Levenshtein, V. I.: On bounds for packings on a sphere and in space. Problemy Peredachi Informacii 14 (1), pp. 3–25 (1978)
46. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: STOC, pp. 193–206 (1983)
47. Khot, S.: Hardness of approximating the shortest vector problem in lattices. In: Journal of the ACM 52 (5), pp. 789–808 (2005)
48. Kuo, P.C., Schneider, M., Dagdelen, Ö., Reichelt, J., Buchmann, J., Cheng, C.M., Yang, B.Y.: Extreme enumeration on GPU and in clouds. In: CHES, pp. 176–191 (2011)
49. Kuperberg, G.: A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. SIAM J. Comput. 35 (1), pp. 170–188 (2005)
50. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. arXiv, Report 1112/3333, pp. 1–10 (2011)
51. Laarhoven, T., van de Pol, J., de Weger, B.: Solving hard lattice problems and the security of lattice-based cryptosystems. Cryptology ePrint Archive, Report 2012/533 (2012)
52. Laarhoven, T., Mosca, M., van de Pol, J.: Solving the shortest vector problem in lattices faster using quantum search. In: PQCrypto, pp. 83–101 (2013)
53. Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. Cryptology ePrint Archive, Report 2014/744 (2014)
54. Laarhoven, T., De Weger, B.: Sieving for shortest lattice vectors in time $2^{0.298n}$ using spherical locality-sensitive hashing. Submitted (2015)
55. Lagarias, J. C., Odlyzko, A. M.: Solving low-density subset sum problems. In: JACM 32 (1), pp. 229–246 (1985)
56. Lenstra, A. K., Lenstra, H., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. 261 (4), pp. 515–534 (1982)
57. Lindner, R., Rückert, M., Baumann, P., Nobach, L.: Lattice Challenge. Online at `http://www.latticechallenge.org/` (2014)
58. Liu, M., Wang, X., Xu, G., Zheng, X.: Shortest lattice vectors in the presence of gaps. Cryptology ePrint Archive, Report 2011/039 (2011)
59. Ludwig, C.: A faster lattice reduction method using quantum search. In: ISAAC, pp. 199–208 (2003)
60. Lyubashevsky, V.: Lattice signatures without trapdoors. In: EUROCRYPT, pp. 738-755 (2012)
61. Mariano, A., Timnat, S., Bischof, C.: Lock-free GaussSieve for linear speedups in parallel high performance SVP calculation. In: SBAC-PAD (2014)
62. Mariano, A., Dagdelen, Ö., Bischof, C.: A comprehensive empirical comparison of parallel ListSieve and GaussSieve. In: APCI&E (2014)
63. Menezes, A. J., van Oorschot, P. C., Vanstone, S. A.: Handbook of applied cryptography. CRC Press (1996)
64. Micciancio, D., Voulgaris, P.: A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In: STOC, pp. 351–358 (2010)
65. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: SODA, pp. 1468–1480 (2010)
66. Micciancio, D., Peikert, C.: Trapdoors for lattices: simpler, tighter, faster, smaller. In: EUROCRYPT, pp. 700–718 (2012)
67. Milde, B., Schneider, M.: A parallel implementation of GaussSieve for the shortest vector problem in lattices. In: PaCT, pp. 452–458 (2011)
68. Mosca, M.: Quantum algorithms. Encyclopedia of Complexity and Systems Science, pp. 7088–7118 (2009)
69. Nguyen, P. Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. Math. Crypt. 2 (2), pp. 181–207 (2008)
70. Plantard, T., Schneider, M.: Ideal Lattice Challenge. Online at `http://latticechallenge.org/ideallattice-challenge/` (2014)
71. Pohst, M.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. ACM SIGSAM Bulletin 15 (1), pp. 37–44 (1981)
72. van de Pol, J.: Lattice-based cryptography. Master's thesis, Eindhoven University of Technology (2011)
73. van de Pol, J., Smart, N. P.: Estimating key sizes for high dimensional lattice-based systems. In: IMACC, pp. 290–303 (2013)
74. Pujol, X., Stehlé, D.: Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, Report 2009/605 (2009)
75. Qu, M., Vanstone, S. A.: The knapsack problem in cryptography. In: Finite Fields: Theory, Applications, and Algorithms 168, pp. 291–308 (1994)
76. Regev, O.: A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. arXiv, Report 0405/151 (2004)
77. Regev, O.: Lattices in computer science. Lecture notes for a course at the Tel Aviv University (2004)
78. Regev, O.: Quantum computation and lattice problems. SIAM J. Comput. 33 (3), pp. 738–760 (2004)
79. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC, pp. 84–93 (2005)
80. Santha, M.: Quantum walk based search algorithms. In: TAMC, pp. 31–46 (2008)
81. Schneider, M.: Analysis of Gauss-Sieve for solving the shortest vector problem in lattices. In: WALCOM, pp. 89–97 (2011)
82. Schneider, M.: Sieving for short vectors in ideal lattices. In: AFRICACRYPT, pp. 375–391 (2013)
83. Schneider, M., Gama, N., Baumann, P., Nobach, L.: SVP Challenge. Online at `http://latticechallenge.org/svp-challenge` (2014)
84. Schnorr, C. P.: A hierarchy of polynomial time lattice basis reduction algorithms. Theoretical Computer Science 53 (2–3), pp. 201–224 (1987)
85. Schnorr, C. P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming 66 (2–3), pp. 181–199 (1994)
86. Schnorr, C. P.: Lattice reduction by random sampling and birthday methods. In: STACS, pp. 145–156 (2003)
87. Shor, P. W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26 (5), pp. 1484–1509 (1997)
88. Smith, J., Mosca, M.: Algorithms for quantum computers. Handbook of Natural Computing, pp. 1451–1492 (2012)
89. Vanstone, S. A., Zuccherato, R. J.: Short RSA keys and their generation. Journal of Cryptology 8 (2), pp. 101–114 (1995)
90. Wang, X., Liu, M., Tian, C., Bi, J.: Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In: ASIACCS, pp. 1–9 (2011)
91. Zhang, F., Pan, Y., Hu, G.: A three-level sieve algorithm for the shortest vector problem. In: SAC, pp. 29–47 (2013)