Solving the Shortest Vector Problem in Lattices Faster Using Quantum Search

Thijs Laarhoven[1], Michele Mosca[2], and Joop van de Pol[3]

[1] Dept. of Mathematics and Computer Science, Eindhoven University of Technology
t.m.m.laarhoven@tue.nl
[2] Institute for Quantum Computing and Dept. of C&O, University of Waterloo
and
Perimeter Institute for Theoretical Physics
michele.mosca@uwaterloo.ca
[3] Dept. of Computer Science, University of Bristol
joop.vandepol@bristol.ac.uk

Abstract. By applying Grover's quantum search algorithm to the lattice algorithms of Micciancio and Voulgaris, Nguyen and Vidick, Wang et al., and Pujol and Stehlé, we obtain improved asymptotic quantum results for solving the shortest vector problem. With quantum computers we can provably find a shortest vector in time $2^{1.799n+o(n)}$, improving upon the classical time complexity 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 $2^{0.312n+o(n)}$, improving upon the classical time complexity of $2^{0.384n+o(n)}$ of Wang et al. 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 algorithms, 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 [57] or the Principle Ideal Problem in real quadratic number fields [25], providing sub-exponential attacks for some systems based on elliptic curve isogenies [16], speeding up exhaustive searching [9,23], counting [12] and (with appropriate assumptions about the computing architecture) finding collisions and claws [4, 11, 13], among many other quantum algorithmic speed-ups [15, 42, 58].

Currently, a small set of systems [8] 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 [27, 50], digital signature schemes [20, 39] and even fully homomorphic encryption schemes [10, 21]. Each of the lattice problems that underpin the security of these systems can be reduced to the shortest vector problem. 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 [35, 45].

In this paper, we closely study the best-known algorithms for solving the shortest vector problem on a lattice, 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.

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 $L = \{\sum_{i=1}^n \lambda_i \mathbf{b}_i : \lambda_i \in \mathbb{Z}\}$. We call the set B a basis of the lattice 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 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 $\mathbf{s} \in L \setminus \{\mathbf{0}\}$ such that $\|\mathbf{s}\| \leq \|\mathbf{v}\|$ for any $\mathbf{v} \in L \setminus \{\mathbf{0}\}$ as a shortest (non-zero) vector of the lattice. Its length is denoted by $\lambda_1(L)$. Given a basis B, we write $\mathcal{P}(B) = \{\sum_{i=1}^n \lambda_i \mathbf{b}_i : 0 \leq \lambda_i < 1\}$ 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 vector in this lattice. In many applications, finding a short vector instead of a shortest vector is also sufficient. The Approximate Shortest Vector Problem with approximation factor γ (SVP_γ) asks to find a non-zero lattice vector $\mathbf{v} \in L$ with length bounded from above by $\|\mathbf{v}\| \leq \gamma\lambda_1(L)$.

1.2 Related work

The Approximate Shortest Vector problem is integral in the cryptanalysis of lattice-based cryptography [18]. For small values of γ, this problem is known to be NP-hard [2, 31], while for certain exponentially large γ, polynomial time algorithms exist, such as the LLL algorithm of Lenstra, Lenstra and Lovász [37].

Other algorithms trade extra running time for a better γ, such as LLL with deep insertions [55] and the BKZ algorithm of Schnorr and Euchner [55].

The current state-of-the-art for classically finding short vectors is BKZ 2.0 [14], which is essentially the original BKZ algorithm with the improved SVP subroutine of Gama et al. [19]. Implementations of this algorithm, due to Chen and Nguyen [14], and Aono and Naganuma [5], currently dominate the Lattice Challenge Hall of Fame [36]. The BKZ algorithm and its variants 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. For SVP solvers there is a similar online challenge [59], where the record is currently held by Kuo et al. [32].

In 2003, Ludwig [38] used quantum algorithms to speed up one such basis reduction algorithm, Random Sampling Reduction (RSR), which is due to Schnorr [56]. By replacing a random sampling from a big list by a quantum search, Ludwig achieves a quantum algorithm that is asymptotically faster than previous results. Ludwig also details the effect that this faster quantum algorithm would have had on the practical security of the lattice-based encryption scheme NTRU [27], had there been a quantum computer in 2005.

Enumeration. The classical method for finding shortest vectors is enumeration, dating back to work by Pohst [44], Kannan [30] and Fincke and Pohst [17] 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/Saturation. In 2001, Ajtai et al. [3] introduced a technique called sieving, leading to the first probabilistic algorithm to solve SVP in time $2^{O(n)}$. Starting with a huge list of short vectors, the algorithm repeatedly applies a sieve to this list to end up with a smaller list of shorter lattice vectors. Eventually, we hope to be left with a list of lattice vectors of length $O(\lambda_1(L))$. Due to the size of the list, the space requirement of sieving is $2^{O(n)}$. Later work [26, 41, 43, 48] investigated the constants in both exponents and ways to reduce these.

Recently, in 2009, Micciancio and Voulgaris [41] started a new branch of sieving algorithms, which may be more appropriately called saturation algorithms. While sieving starts out with a long list and repeatedly applies a sieve to reduce its length, saturation algorithms iteratively add vectors to an initially empty list, hoping that at some point the space of short lattice vectors is "saturated", and two of the vectors in the list are at most $\lambda_1(L)$ apart. The time and space requirements of these algorithms are also $2^{O(n)}$. In 2009, Pujol and Stehlé [46] showed that with this method, SVP can provably be solved in time $2^{2.465n+o(n)}$.

Voronoi. In 2010, Micciancio and Voulgaris presented a deterministic algorithm for solving SVP based on constructing the Voronoi cell of the lattice [40]. In time

$2^{2n+o(n)}$ and space $2^{n+o(n)}$, this algorithm is able to find a shortest vector in any lattice. Currently this is the best provable asymptotic result for classical SVP solvers.

Practice. While many methods have surpassed the enumeration algorithms in terms of classical provable asymptotic time complexities, in practice the enumeration methods still dominate the field. The version of enumeration that is currently used in practice is due to Schnorr and Euchner [55] with improvements by Gama et al. [19]. It does not incorporate the stronger version of preprocessing of Kannan [30] and hence has an asymptotic time complexity of $2^{O(n^2)}$. However, due to the small hidden constants in the exponents and the exponential space complexity of the other algorithms, enumeration is actually faster than other methods for common values of n. That said, the other methods are still quite new, so a further study of these other methods may tip the balance.

1.3 Quantum search

In this paper we will study how quantum algorithms can be used to speed up the SVP algorithms outlined above. For this, we will make use of Grover's quantum search algorithm [23], 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 quantum search [9, 12, 23], 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 [6].

Throughout the paper, we will write $x \leftarrow \text{search}_{e \in L}(f(e) = 1)$ to highlight subroutines that perform a search in a long list. This assignment returns true if an element $e \in L$ with $f(e) = 1$ exists (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.

For both of these classical and quantum 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 [7], 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. [22]) looks particularly challenging, especially for first generation quantum computers. Some authors have studied the limitations of quantum algorithms in this context [7, 24, 28].

Some algorithms (e.g. [4]) 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 [34]) 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 [51].

1.4 Contributions and outline

In this paper, we show that quantum algorithms can significantly speed up sieving and saturation algorithms. The constant in the exponent decreases by approximately 25% in all cases, leading to an improvement upon both provable and heuristic asymptotic results for solving the Shortest Vector Problem:

- Provably, we can find a shortest vector in any lattice in time $2^{1.799n+o(n)}$.
- Heuristically, we can find a shortest vector in any lattice in time $2^{0.312n+o(n)}$.
- Extrapolating from classical experiments, with quantum computers we expect to be able to find a shortest vector in any lattice in time about $2^{0.39n}$.

Table 1 contains a comparison between our contributions and previous results, in both the classical and quantum setting. While the Voronoi Cell algorithm is asymptotically the best algorithm in the provable classical setting, our quantum saturation algorithm has better asymptotics in the provable quantum setting.

Table 1. A comparison of the results as expressed in logarithmic leading order terms, with provable results above and heuristic results below.

	Classical		Quantum		
Algorithm	Time	Space	Time	Space	
(Enumeration)	$O(n \log n)$	$O(\log n)$	-	-	(Appendix C)
Pujol and Stehlé [46]	$2.47n$	$1.24n$	$1.80n$	$1.29n$	(Section 3.1)
(Voronoi)	$2.00n$	$1.00n$	-	-	(Appendix C)
Micciancio and Voulgaris [41]	$0.52n$	$0.21n$	$0.39n$	$0.21n$	(Section 3.2)
Nguyen and Vidick [43]	$0.42n$	$0.21n$	$0.32n$	$0.21n$	(Section 2.1)
Wang et al. [60]	$0.39n$	$0.26n$	$0.32n$	$0.21n$	(Section 2.2)

Why do we only consider sieving and saturation algorithms, and not the more practical enumeration or the theoretically faster Voronoi cell algorithms? It turns out that it is not as simple to significantly speed up these algorithms using similar techniques. For some intuition why this is the case, see Appendix C.

The outline of this paper is as follows. In Section 2 we look at sieving algorithms, and how quantum algorithms lead to speed-ups. In Section 3, we look at saturation algorithms, and their estimated time and space complexities on a quantum computer. Technical details regarding some of these results can be found in Appendices A and B.

2 Sieving algorithms

Sieving was first introduced by Ajtai et al. [3] and later improved theoretically [26, 41, 43, 48] and practically [43, 60] in various papers. In these algorithms, first an exponentially long list of lattice vectors is generated. Then, by iteratively applying a sieve to this list, the size of the list, as well as the lengths of the vectors in the list are reduced. After a polynomial number of applications of the sieve, we hope to be left with a short but non-empty list of very short vectors, from which we can then obtain a shortest vector of the lattice with high probability.

2.1 The Heuristic Algorithm of Nguyen and Vidick

Nguyen and Vidick [43] considered a heuristic, practical variant of the sieve algorithm of Ajtai et al. [3], which provably returns a shortest vector under a certain natural, heuristic assumption. A slightly modified but equivalent version of this algorithm is given in Algorithm 1.

Description of the algorithm. The algorithm starts by generating a big list S 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

Algorithm 1 The Heuristic Sieve Algorithm of Nguyen and Vidick

Input: An LLL-reduced basis B of L, and constants $\gamma \in (\frac{2}{3}, 1)$ and $N = 2^{O(n)}$
Output: A short non-zero lattice vector $\mathbf{s}$

1: $S \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** N **do**
3: $\quad \mathbf{v} \in_R B_n(\mathbf{0}, \|B\|) \cap L$
4: $\quad S \leftarrow S \cup \{\mathbf{v}\}$
5: **while** $S \setminus \{\mathbf{0}\} \neq \emptyset$ **do**
6: $\quad S_{\text{prev}} \leftarrow S \setminus \{\mathbf{0}\}$
7: $\quad R \leftarrow \max_{\mathbf{v} \in S_{\text{prev}}} \|\mathbf{v}\|$
8: $\quad C \leftarrow \{\mathbf{0}\}$
9: $\quad S \leftarrow \emptyset$
10: $\quad$ **for all** $\mathbf{v} \in S_{\text{prev}}$ **do**
11: $\quad\quad$ **if** $\mathbf{c} \leftarrow \text{search}_{\mathbf{c} \in C}(\|\mathbf{v} - \mathbf{c}\| \leq \gamma R)$ **then**
12: $\quad\quad\quad S \leftarrow S \cup \{\mathbf{v} - \mathbf{c}\}$
13: $\quad\quad$ **else**
14: $\quad\quad\quad C \leftarrow C \cup \{\mathbf{v}\}$
15: $\mathbf{s} \leftarrow \text{argmin}_{\mathbf{v} \in S_{\text{prev}}} \|\mathbf{v}\|$
16: **return** $\mathbf{s}$

the list is completely depleted. In that case, we go back one step, and look for the closest pair of lattice vectors in the last non-empty list.

The sieving step consists of splitting the previous list S_{prev} in a set of 'centers' C and a new list of vectors S that will be used for the next sieve. For each vector $\mathbf{v}$ in S_{prev}, the algorithm first checks if a vector $\mathbf{c}$ in C exists that is close to $\mathbf{v}$. If this is the case, then we add the difference $\mathbf{v} - \mathbf{c}$ to S_{prev}. If this is not the case, then $\mathbf{v}$ is added to C. Since the set C consists of vectors with a bounded norm and a specified minimum distance between any two points, one can bound the size of C from above using a result of Kabatiansky and Levenshtein [29] regarding sphere packings. In other words, C will be sufficiently small, so that the list S will be sufficiently large. After applying the sieve, we discard all vectors in C and apply the sieve again to the vectors in $S_{\text{prev}} = S$.

At each iteration of the sieve, the maximum norm of the vectors in the list decreases from some constant R to at most γR, where γ is some geometric factor smaller than 1. Nguyen and Vidick conjecture that throughout the algorithm, the longest vectors in S are uniformly distributed over the space of all n-dimensional vectors with norms between γR and R.

Heuristic 1 *[43] At any stage of Algorithm 1, the vectors in $S \cap C_n(\gamma R, R)$ are uniformly distributed in $C_n(\gamma R, R)$, where $C_n(r_1, r_2) = \{\mathbf{x} \in \mathbb{R}^n : r_1 \leq \|\mathbf{x}\| \leq r_2\}$.*

Classical complexities. In Line 11 of Algorithm 1, 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 this heuris-

tic assumption Nguyen and Vidick give the following estimate for the time and space complexity of their algorithm.

Lemma 1. *[43] On a classical computer, assuming that Heuristic 1 holds, Algorithm 1 will return a shortest vector of a lattice in time at most* $2^{0.415n+o(n)}$ *and space at most* $2^{0.208n+o(n)}$.

Quantum complexities. If we use a quantum search subroutine in Line 11, 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 running time significantly. Note that in Line 15, it also seems like a search of a list is performed. In reality, this final search of S_{prev} can be done in constant time by using appropriate data structures, e.g., by keeping the vectors in S and S_{prev} sorted from short to long, or by manually keeping track of the shortest vector in S.

Since replacing the classical search by a quantum search does not change the internal behaviour of the algorithm, the estimates and heuristics are as valid as they were in the classical setting. The time complexity does change, as the following theorem explains. For details, see Appendix A.

Theorem 1. *On a quantum computer, assuming that Heuristic 1 holds, Algorithm 1 will return a shortest vector of a lattice 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 exponent of the runtime.

2.2 The Heuristic Algorithm of Wang et al.

To improve upon the time complexity of the algorithm of Nguyen and Vidick, Wang et al. [60] 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 [60].

Classical complexities. The classical time complexity of this algorithm is bounded from above by $\tilde{O}(|S| \cdot (|C_1| + |C_2|))$, while the space required is at most $O(|S| + |C_1| + |C_2|)$. Optimizing the constants γ_1 and γ_2 leads to $\gamma_1 = 1.0927$ and $\gamma_2 \to 1$, 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 $\tilde{O}(|S| \cdot (\sqrt{|C_1|} + \sqrt{|C_2|}))$, while the space complexity remains $O(|S| + |C_1| + |C_2|)$. Re-optimizing the constants for a minimum time complexity leads to $\gamma_1 \to \sqrt{2}$ and $\gamma_2 \to 1$, 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.

3 Saturation algorithms

Saturation algorithms were only recently introduced by Micciancio and Voulgaris [41], and further studied by Pujol and Stehlé [46] and Schneider [52]. Instead of starting with a huge list and making the list smaller and smaller, this method starts with a small or empty list, and keeps adding more and more vectors to the list. Building upon the same result of Kabatiansky and Levenshtein about sphere packings [29], we know that if the list reaches a certain size and all vectors have a norm bounded by a sufficiently small constant, two of the vectors in the list must be close to one another. Thus, if we can guarantee that new short lattice vectors keep getting added to the list, then at some point, with high probability, we can find a shortest vector as the difference between two of the list vectors.

3.1 The Provable Algorithm of Pujol and Stehlé

Using the Birthday paradox, Pujol and Stehlé [46] showed that the constant in the exponent of the time complexity of the original algorithm of Micciancio and Voulgaris [41, Section 3.1] can be reduced by almost 25%. The algorithm is presented in Algorithm 2.

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

First, the algorithm generates a long list T of lattice vectors with norms between $R\mu$ and $\|B\|$. This 'dummy' list is only used for technical reasons, and in practice one does not seem to need such a list. 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 $r\mu$ away from $\mathbf{v}$. This is purely a technical modification to make the proofs work, as experiments show that without such perturbed vectors, saturation algorithms also work fine [40, 46, 52].

After generating T, 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 T rather than with vectors in S. The latter guarantees that the vectors in S are i.i.d.

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 μ 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.

Algorithm 2 The Provable Saturation Algorithm of Pujol and Stehlé

Input: An LLL-reduced basis B of L, $\mu \simeq \lambda_1(L)$, $\xi > \frac{1}{2}$, $R > 2\xi$, $N_1^{\max}, N_2 = 2^{O(n)}$
Output: A non-zero lattice vector $\mathbf{s}$ of norm less than μ

1: $\gamma \leftarrow 1 - \frac{1}{n}$
2: $T \leftarrow \emptyset$
3: $N_1 \in_R [0, N_1^{\max} - 1]$
4: **for** $i \leftarrow 1$ **to** N_1 **do**
5: $\quad \mathbf{x} \in_R B_n(\mathbf{0}, \xi\mu)$
6: $\quad \mathbf{v}' \leftarrow \mathbf{x} \mod \mathcal{P}(B)$
7: $\quad$ **while** $\mathbf{t} \leftarrow \text{search}_{\mathbf{t} \in T}(\|\mathbf{v}' - \mathbf{t}\| < \gamma\|\mathbf{v}'\|)$ **do**
8: $\quad\quad \mathbf{v}' \leftarrow \mathbf{v}' - \mathbf{t}$
9: $\quad \mathbf{v} \leftarrow \mathbf{v}' - \mathbf{x}$
10: $\quad$ **if** $\|\mathbf{v}\| \geq R\mu$ **then**
11: $\quad\quad T \leftarrow T \cup \{\mathbf{v}\}$
12: $S \leftarrow \emptyset$
13: **for** $i \leftarrow 1$ **to** N_2 **do**
14: $\quad \mathbf{x} \in_R B_n(\mathbf{0}, \xi\mu)$
15: $\quad \mathbf{v}' \leftarrow \mathbf{x} \mod \mathcal{P}(B)$
16: $\quad$ **while** $\mathbf{t} \leftarrow \text{search}_{\mathbf{t} \in T}(\|\mathbf{v}' - \mathbf{t}\| < \gamma\|\mathbf{v}'\|)$ **do**
17: $\quad\quad \mathbf{v}' \leftarrow \mathbf{v}' - \mathbf{t}$
18: $\quad \mathbf{v} \leftarrow \mathbf{v}' - \mathbf{x}$
19: $\quad S \leftarrow S \cup \{\mathbf{v}\}$
20: $\{\mathbf{s}_1, \mathbf{s}_2\} \leftarrow \text{search}_{\{\mathbf{s}_1, \mathbf{s}_2\} \in S \times S}(0 < \|\mathbf{s}_1 - \mathbf{s}_2\| < \mu)$
21: **return** $\mathbf{s}_1 - \mathbf{s}_2$

Classical complexities. With a classical search applied to the subroutines in Lines 7, 16, and 20, Pujol and Stehlé obtained the following results.

Lemma 2. *[46] Let $\xi \approx 0.9476$ and $R \approx 3.0169$. Then, using polynomially many queries to Algorithm 2, 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 $2^{1.233n+o(n)}$.*

Quantum complexities. Applying a quantum search algorithm to the search-subroutines in Lines 7, 16, and 20 leads to the following result. Details are given in Appendix B.

Theorem 2. *Let $\xi \approx 0.9086$ and $R \approx 3.1376$. Then, using polynomially many queries to the quantum version of Algorithm 2, we can find a shortest vector in a lattice with probability exponentially close to 1, using 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.

Algorithm 3 The Heuristic Saturation Algorithm of Micciancio and Voulgaris

Input: An LLL-reduced basis B of L, and a constant C_0
Output: A short non-zero lattice vector $\mathbf{s}$

$S \leftarrow \{\mathbf{0}\}$
$Q \leftarrow \emptyset$
$c \leftarrow 0$
while $c < C_0$ **do**
  if $Q \neq \emptyset$ **then**
    $\mathbf{v} \in_R Q$
    $Q \leftarrow Q \setminus \{\mathbf{v}\}$
  else
    $\mathbf{v} \in_R B_n(\mathbf{0}, \|B\|) \cap L$
  while $\mathbf{s} \leftarrow \text{search}_{\mathbf{s} \in S}(\max\{\|\mathbf{s}\|, \|\mathbf{v} - \mathbf{s}\|\} \leq \|\mathbf{v}\|)$ **do**
    $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{s}$
  while $\mathbf{s} \leftarrow \text{search}_{\mathbf{s} \in S}(\max\{\|\mathbf{v}\|, \|\mathbf{v} - \mathbf{s}\|\} \leq \|\mathbf{s}\|)$ **do**
    $S \leftarrow S \setminus \{\mathbf{s}\}$
    $Q \leftarrow Q \cup \{\mathbf{v} - \mathbf{s}\}$
  if $\mathbf{v} = \mathbf{0}$ **then**
    $c \leftarrow c + 1$
  else
    $S \leftarrow S \cup \{\mathbf{v}\}$
$\mathbf{s} \leftarrow \text{argmin}_{\mathbf{v} \in S \setminus \{\mathbf{0}\}} \|\mathbf{v}\|$
return $\mathbf{s}$

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 2. (Recall that quantum searching a list of c^n elements (with $c > 1$) requires the list to be stored in quantumly addressable classical memory (versus regular quantum memory) and otherwise can be searched using only $O(n)$ qubits and $O(c^{n/2})$ queries to the list.)

3.2 The Heuristic Algorithm of Micciancio and Voulgaris

In practice, just like sieving algorithms, saturation algorithms are much faster than their worst-case running times and provable time complexities suggest. Micciancio and Voulgaris [41] gave a heuristic variant of their saturation algorithm, for which they could not give a (heuristic) bound on the time complexity, but with a better bound on the space complexity, and a better practical time complexity. The algorithm is given in Algorithm 3.

Description of the algorithm. The algorithm is similar to Algorithm 2, with the following main differences: (i) we do not explicitly generate two lists S, T to apply the birthday paradox; (ii) we do not use the geometric factor $\gamma < 1$

but always reduce a vector if it can be reduced; (iii) we also reduce the existing list vectors 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, we run the algorithm until we reach a predefined number of collisions C_0.

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 from above by $2^{0.208n}$ due to the kissing constant [41, Section 5]. This is much faster than the theoretical time complexity of $2^{1.799n}$ of the quantum-enhanced saturation algorithm discussed in Section 3.1.

Remark 1. In practice, the algorithm of Micciancio and Voulgaris is faster than the one of Nguyen and Vidick of Section 2.1, even though the leading term in the exponent is larger. So asymptotically, this algorithm is dominated by the algorithm of Nguyen and Vidick, but in practice and for small dimensions, the algorithm of Micciancio and Voulgaris seems to perform better.

Remark 2. Schneider states [52] that the time complexity roughly scales like $2^{0.57n-23.5}$, instead of the $2^{0.52n}$ claimed by Micciancio and Voulgaris. Although asymptotically this time complexity is worse than the one of Micciancio and Voulgaris, the cross-over point of these rough approximations is around $n \approx 470$. So for most values of n that SVP solvers handle in practice, the term -23.5 is more significant than the small increase caused by n, and the conjectured time complexity of Schneider is better than that of Micciancio and Voulgaris.

Quantum complexities. To this heuristic algorithm, the quantum speed-ups can also be applied. Generally, these saturation algorithms generate a list S of reasonably short lattice vectors by (i) first sampling a long, random lattice vector $\mathbf{v} \in L$; (ii) reducing the vector $\mathbf{v}$ with lattice vectors already in S; (iii) possibly reducing the vectors in S with this new vector $\mathbf{v}$; and (iv) finally adding $\mathbf{v}$ to S. The total classical time complexity of these algorithms is of the order $|S|^2$ due to (ii) and (iii), but by applying quantum speed-ups to these steps, this becomes $|S|^{3/2}$. This means that the exponent in the time complexity is generally reduced by about 25%, which is comparable to the improvement in Section 3.1. In practice, we therefore expect a time complexity of about $2^{0.39n}$ for the heuristic algorithm of Micciancio and Voulgaris with quantum search speed-ups, with constants that may make this algorithm faster than the sieving algorithm of Section 2.1.

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, Leiden, The Netherlands. In particular, we would like to thank Felix Fontein, Nadia Heninger, Stacey Jeffery, Stephen Jordan, Michael Schneider, Damien Stehlé and Benne de Weger for the valuable discussions there.

Finally, we thank the anonymous reviewers for their helpful comments and suggestions.

The first author is supported by DIAMANT and ECRYPT II (ICT-2007-216676). 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. Aharonov, D., Regev, O.: A Lattice Problem in Quantum NP. In: 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 210–219. IEEE Press, New York (2003)
2. Ajtai, M.: The Shortest Vector Problem in L_2 is NP-hard for Randomized Reductions. In: 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 10–19. ACM, New York (1998)
3. Ajtai, M., Kumar, R., Sivakumar, D.: A Sieve Algorithm for the Shortest Lattice Vector Problem. In: 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 601–610. ACM, New York (2001)
4. Ambainis, A.: Quantum Walk Algorithm for Element Distinctness. In: 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 22–31. IEEE Press, New York (2003)
5. Aono, Y., Naganuma, K.: Heuristic Improvements of BKZ 2.0. IEICE Tech. Rep. 112 (211), pp. 15–22 (2012)
6. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, V.: Strengths and Weaknesses of Quantum Computing. SIAM J. Comput. 26 (5), pp. 1510–1523 (1997)
7. Bernstein, D.J.: Cost analysis of hash collisions: Will quantum computers make SHARCs obsolete?. SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems (2009)
8. Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): Post-quantum cryptography. Springer (2008)
9. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight Bounds on Quantum Searching. Fortschritte der Physik 46, pp. 493–505, (1998)
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer ScienceITCS 2012, pp. 309–325. ACM (2012)
11. Brassard, G., Høyer P., Tapp A.: Quantum cryptanalysis of hash and claw-free functions. In: LATIN'98: Theoretical Informatics, LNCS, vol. 1380, pp. 163–169 (1998)
12. Brassard, G., Høyer P., Mosca M., and Tapp A: Quantum Amplitude Amplification and Estimation, in AMS Contemporary Mathematics Series Millennium Vol. entitled Quantum Computation & Information, vol. 305, (2002).
13. 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)
14. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better Lattice Security Estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011)

15. Childs A., Van Dam, W.: Quantum algorithms for algebraic problems, Rev. Mod. Phys. 82, pp. 1–52 (2010)
16. Childs A.M., Jao D., Soukharev V.: Constructing elliptic curve isogenies in quantum subexponential time. arXiv:1012.4019 (2010)
17. 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)
18. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.): EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer (2008)
19. Gama, N., Nguyen, P.Q., Regev, O.: Lattice Enumeration Using Extreme Pruning. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 257–278. Springer, Heidelberg (2010)
20. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) STOC 2008, pp. 197–206. ACM (2008)
21. Gentry, C.: A fully homomorphic encryption scheme (Doctoral dissertation, Stanford University). (2009)
22. , Giovannetti V., Lloyd S., Maccone, L. : Quantum Random Access Memory. Phys. Rev. Lett., vol. 100, 160501 (2008)
23. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: 28th Annual ACM Symposium on Theory of Computing (STOC), pp. 212–219. ACM, New York (1996)
24. Grover L., Rudolph, T.: How significant are the known collision and element distinctness quantum algorithms?. Quantum Info. Comput. 4 (3), pp. 201–206 (2004)
25. Hallgren, S.: Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. J. ACM. 54 (1), pp. 653–658 (2007)
26. Hanrot, G., Pujol, X., Stehlé, D.: Algorithms for the Shortest and Closest Lattice Vector Problems. In: Chee, Y.M., Guo, Z., Ling, S., Shao, F., Tang, Y., Wang, H., Xing, C. (eds.) IWCC 2011. LNCS, vol. 6639, pp. 159–190. Springer, Heidelberg (2011)
27. Hoffstein, J., Pipher, J., Silverman, J.: NTRU: A ring-based public key cryptosystem. In: Buhler, J. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer (1998)
28. Jeffery, S.: Collision Finding with Many Classical or Quantum Processors. Master's thesis, University of Waterloo (2011)
29. Kabatiansky, G., Levenshtein, V.I.: On Bounds for Packings on a Sphere and in Space. Problemy Peredachi Informacii 14 (1), pp. 3–25 (1978)
30. Kannan, R.: Improved Algorithms for Integer Programming and Related Lattice Problems. In: 15th Annual ACM Symposium on Theory of Computing (STOC), pp. 193–206. ACM, New York (1983)
31. Khot, S.: Hardness of approximating the shortest vector problem in lattices. In: Journal of the ACM 52 (5), pp. 789–808 (2005)
32. 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: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 176–191. Springer (2011)
33. Kuperberg, G.: A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. SIAM J. Comput. 35 (1), pp. 170–188 (2005)
34. Kuperberg, G.: Another Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. arXiv, Report 1112/3333, pp. 1–10 (2011)
35. 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, pp. 1–43 (2012)

36. TU Darmstadt Lattice Challenge, http://www.latticechallenge.org/
37. Lenstra, A.K., Lenstra, H., Lovász, L.: Factoring Polynomials with Rational Coefficients. Math. Ann. 261 (4), pp. 515–534 (1982)
38. Ludwig, C.: A Faster Lattice Reduction Method Using Quantum Search. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 199–208. Springer, Heidelberg (2003)
39. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.): EUROCRYPT 2012. LNCS, vol. 7237, pp. 738-755. Springer (2012)
40. Micciancio, D., Voulgaris, P.: A Deterministic Single Exponential Time Algorithm for Most Lattice Problems based on Voronoi Cell Computations. In: 42nd Annual ACM Symposium on Theory of Computing (STOC), pp. 351–358. ACM, New York (2010)
41. Micciancio, D., Voulgaris, P.: Faster Exponential Time Algorithms for the Shortest Vector Problem. In: 21st Annual ACM Symposium on Discrete Algorithms (SODA), pp. 1468–1480. ACM, New York (2010)
42. Mosca, M.: Quantum Algorithms, Encyclopedia of Complexity and Systems Science (ed.: Robert Meyers) (2009)
43. Nguyen, P.Q., Vidick, T.: Sieve Algorithms for the Shortest Vector Problem are Practical. J. Math. Crypt. 2 (2), pp. 181–207 (2008)
44. 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)
45. van de Pol, J.: Lattice-based cryptography. Master's thesis. Eindhoven University of Technology (2011)
46. Pujol, X., Stehlé, D.: Solving the Shortest Lattice Vector Problem in Time $2^{2.465n}$. Cryptology ePrint Archive, Report 2009/605, pp. 1–7 (2009)
47. Regev, O.: A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space. arXiv, Report 0405/151, pp. 1–7 (2004)
48. Regev, O.: Lattices in Computer Science. Lecture Notes for a Course at the Tel Aviv University (2004)
49. Regev, O.: Quantum Computation and Lattice Problems. SIAM J. Comput. 33 (3), pp. 738–760 (2004)
50. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 84–93 (2005)
51. Santha, M.: Quantum Walk Based Search Algorithms. In: Theory and Applications of Models of Computation, 2008 (TAMC), LNCS, vol. 4978, pp. 31–46 (2008)
52. Schneider, M.: Analysis of Gauss-Sieve for Solving the Shortest Vector Problem in Lattices. In: Katoh, N., Kumar, A. (eds.) WALCOM 2011. LNCS, vol. 6552, pp. 89–97. Springer, Heidelberg (2011)
53. Schneider, M.: Sieving for Short Vectors in Ideal Lattices. Cryptology ePrint Archive, Report 2011/458, pp. 1–19 (2011)
54. Schnorr, C.P.: A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. Theoretical Computer Science 53 (2–3), pp. 201–224 (1987)
55. 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)
56. Schnorr, C.P.: Lattice reduction by random sampling and birthday methods. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 145–156. Springer (2003)

57. 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)
58. Smith J., Mosca. M.: Algorithms for Quantum Computers, Handbook of Natural Computing, pp. 1451–1492. Springer (2012)
59. SVP Challenge, `http://latticechallenge.org/svp-challenge/`
60. Wang, X., Liu, M., Tian, C., Bi, J.: Improved Nguyen-Vidick Heuristic Sieve Algorithm for Shortest Vector Problem. In: 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp. 1–9. ACM, New York (2011)

A Analysis of the Sieve Algorithm of Nguyen and Vidick

Nguyen and Vidick showed that if their heuristic assumption holds, the time and space complexities of their algorithm can be bounded from above as follows.

Lemma 3. *[43] On a classical computer, assuming Heuristic 1 holds, Algorithm 1 will return a shortest vector of a lattice in time* $2^{2c_h n+o(n)}$ *and space* $2^{c_h n+o(n)}$*, where* $\frac{2}{3} < \gamma < 1$ *and*

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

To obtain a minimum time complexity, γ should be chosen as close to 1 as possible. Letting $\gamma \to 1$ leads to an asymptotic time complexity of less than $2^{0.415n+o(n)}$ and an asymptotic space complexity of less than $2^{0.208n+o(n)}$.

To obtain these estimates, it is first noted that the sizes of S and C are bounded from above by $2^{c_h n+o(n)}$. The space complexity is therefore bounded from above by $O(|S| + |C|) = 2^{c_h n+o(n)}$, and since for every element in S the algorithm has to search the list C, the time complexity is bounded from above by $\tilde{O}(|S| \cdot |C|) = 2^{2c_h n+o(n)}$.

Using quantum search on the list C, the time complexity decreases to $\tilde{O}(|S| \cdot \sqrt{|C|}) = 2^{\frac{3}{2}c_h n+o(n)}$, while the space complexity remains the same. This leads to the following result.

Lemma 4. *On a quantum computer, assuming Heuristic 1 holds, Algorithm 1 will return a shortest vector of a lattice in time* $2^{\frac{3}{2}c_h n+o(n)}$ *and space* $2^{c_h n+o(n)}$*.*

Optimizing γ to obtain a minimum time complexity again corresponds to letting γ tend to 1 from below, leading to an asymptotic time complexity of $2^{0.312n+o(n)}$ and space complexity of $2^{0.208n+o(n)}$, as stated in Theorem 1.

B Analysis of the Saturation Algorithm of Pujol and Stehlé

In the classical setting, the time complexities of the different parts of the algorithm are as follows. The constants are explained in the lemma below.

- Cost of generating T: $\tilde{O}(N_1^{\max} \cdot |T|) = 2^{(c_g+2c_t)n+o(n)}$.
- Cost of generating S: $\tilde{O}(N_2 \cdot |T|) = 2^{(c_g+c_b/2+c_t)n+o(n)}$.
- Cost of searching S for a pair of close vectors: $\tilde{O}(|S|^2) = 2^{(2c_g+c_b)n+o(n)}$.

The space complexity is at most $O(|T| + |S|) = 2^{\max(c_t, c_g+c_b/2)n+o(n)}$. In [46], this lead to the following lemma.

Lemma 5. *[46] Let $\xi > \frac{1}{2}$ and $R > 2\xi$, and suppose $\mu > \lambda_1(L)$. Then, with c_b, c_t, c_g, N_B, N_V, N_G, $N_1^{\max}$, N_2 chosen according to:*

$$c_b = \log_2(R) + 0.401, \qquad N_B = 2^{c_b n+o(n)}, \tag{2}$$

$$c_t = \frac{1}{2}\log_2\left(1 + \frac{2\xi}{R-2\xi}\right) + 0.401, \qquad N_T = 2^{c_t n+o(n)}, \tag{3}$$

$$c_g = \frac{1}{2}\log_2\left(\frac{4\xi^2}{4\xi^2-1}\right), \qquad N_G = 2^{c_g n+o(n)}, \tag{4}$$

$$N_1^{\max} = 2^{(c_g+c_t)n+o(n)}, \qquad N_2 = 2^{(c_g+c_b/2)n+o(n)}, \tag{5}$$

with probability at least $\frac{1}{16}$, Algorithm 2 returns a lattice vector $\mathbf{s} \in L \setminus \{\mathbf{0}\}$ with $\|\mathbf{s}\| < \mu$, in time at most $2^{tn+o(n)}$ and space at most $2^{sn+o(n)}$, where t and s are given by

$$t = \max\left(c_g + 2c_t, c_g + \frac{c_b}{2} + c_t, 2c_g + c_b\right), \quad s = \max\left(c_t, c_g + \frac{c_b}{2}\right). \tag{6}$$

In the quantum setting, the costs are as follows.

- Cost of generating T: $\tilde{O}(N_1^{\max} \cdot \sqrt{|T|}) = 2^{(c_g+3c_t/2)n+o(n)}$.
- Cost of generating S: $\tilde{O}(N_2 \cdot \sqrt{|T|}) = 2^{(c_g+c_b/2+c_t/2)n+o(n)}$.
- Cost of searching S for a pair of close vectors: $\tilde{O}(\sqrt{|S|^2}) = 2^{(c_g+c_b/2)n+o(n)}$.

The total space complexity is still the same as in the classical setting, i.e., at most $O(|T| + |S|) = 2^{\max(c_t, c_g+c_b/2)n+o(n)}$. This leads to the following lemma.

Lemma 6. *Let $\xi > \frac{1}{2}$ and $R > 2\xi$, and suppose $\mu > \lambda_1(L)$. Then, with c_b, c_t, c_g, N_B, N_V, N_G, $N_1^{\max}$, N_2 chosen according to Equations (2) to (5), with probability at least $\frac{1}{16}$, Algorithm 2 returns a lattice vector $\mathbf{s} \in L \setminus \{\mathbf{0}\}$ with $\|\mathbf{s}\| < \mu$ on a quantum computer in time at most $2^{\tilde{t}n+o(n)}$ and space at most $2^{\tilde{s}n+o(n)}$, where $\tilde{t}$ and $\tilde{s}$ are given by*

$$\tilde{t} = \max\left(c_g + \frac{3c_t}{2}, c_g + \frac{c_b}{2} + \frac{c_t}{2}, c_g + \frac{c_b}{2}\right), \quad \tilde{s} = \max\left(c_t, c_g + \frac{c_b}{2}\right). \tag{7}$$

Optimizing ξ and R for the minimum time complexity, we get $\xi \approx 0.9086$ and $R \approx 3.1376$ as in Theorem 2. Note that if S is generated in parallel with exponentially many quantum computers, the cost of the second part of the algorithm becomes negligible, and the exponent in the time complexity changes to

$$\tilde{t}' = \max\left(c_g + \frac{3c_t}{2}, c_g + \frac{c_b}{2}\right). \tag{8}$$

In that case, the optimal choice of ξ and R (with respect to minimizing the time complexity) would be $\xi \approx 1.0610$ and $R \approx 4.5166$, leading to a time complexity of less than $2^{1.470n+o(n)}$.

C Other SVP algorithms

C.1 Enumeration

Recall that enumeration considers all lattice vectors inside a giant ball around the origin that is known to contain at least one lattice vector. Let L be a lattice with basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$. Consider each lattice vector $\mathbf{u} \in L$ 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 in 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. [19] 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.

C.2 Voronoi cell

Consider a set of points in the Euclidean space. For any given point in this set, its Voronoi cell is 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 L? Micciancio and Voulgaris [40] show that this can be done by solving $2^n - 1$ instances of CVP in the lattice $2L$. However, in order to solve CVP we would need the relevant vectors which means we are back to our original problem. However, 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 the 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.