Finding Closest Lattice Vectors Using Approximate Voronoi Cells

Emmanouil Doulgerakis, Thijs Laarhoven(✉), and Benne de Weger

Eindhoven University of Technology, Eindhoven, The Netherlands
{e.doulgerakis,b.m.m.d.weger}@tue.nl, mail@thijs.co

Abstract. The two traditional hard problems underlying the security of lattice-based cryptography are the shortest vector problem (SVP) and the closest vector problem (CVP). For a long time, lattice enumeration was considered the fastest method for solving these problems in high dimensions, but recent work on memory-intensive methods has resulted in lattice sieving overtaking enumeration both in theory and in practice. Some of the recent improvements [Ducas, Eurocrypt 2018; Laarhoven–Mariano, PQCrypto 2018; Albrecht–Ducas–Herold–Kirshanova–Postlethwaite–Stevens, 2018] are based on the fact that these methods find more than just one short lattice vector, and this additional data can be reused effectively later on to solve other, closely related problems faster. Similarly, results for the preprocessing version of CVP (CVPP) have demonstrated that once this initial data has been generated, instances of CVP can be solved faster than when solving them directly, albeit with worse memory complexities [Laarhoven, SAC 2016].

In this work we study CVPP in terms of approximate Voronoi cells, and obtain better time and space complexities using randomized slicing, which is similar in spirit to using randomized bases in lattice enumeration [Gama–Nguyen–Regev, Eurocrypt 2010]. With this approach, we improve upon the state-of-the-art complexities for CVPP, both theoretically and experimentally, with a practical speedup of several orders of magnitude compared to non-preprocessed SVP or CVP. Such a fast CVPP solver may give rise to faster enumeration methods, where the CVPP solver is used to replace the bottom part of the enumeration tree, consisting of a batch of CVP instances in the same lattice.

Asymptotically, we further show that we can solve an exponential number of instances of CVP in a lattice in essentially the same amount of time and space as the fastest method for solving just one CVP instance. This is in line with various recent results, showing that perhaps the biggest strength of memory-intensive methods lies in being able to reuse the generated data several times. Similar to [Ducas, Eurocrypt 2018], this further means that we can achieve a "few dimensions for free" for sieving for SVP or CVP, by doing $\Theta(d/\log d)$ levels of enumeration on top of a CVPP solver based on approximate Voronoi cells.

Keywords: Lattices · Preprocessing · Voronoi cells · Sieving algorithms · Shortest vector problem (SVP) · Closest vector problem (CVP)

J. Ding and R. Steinwandt (Eds.): PQCrypto 2019, LNCS 11505, pp. 3–22, 2019.
https://doi.org/10.1007/978-3-030-25510-7_1

1 Introduction

Lattice Problems. Lattices are discrete subgroups of $\mathbb{R}^d$: given a basis $\boldsymbol{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d\} \subset \mathbb{R}^d$, the lattice generated by $\boldsymbol{B}$ is defined as $\mathcal{L} = \mathcal{L}(\boldsymbol{B}) := \{\sum_{i=1}^{d} \lambda_i \boldsymbol{b}_i : \lambda_i \in \mathbb{Z}\}$. Given a basis of $\mathcal{L}$, the shortest vector problem (SVP) is to find a (non-zero) lattice vector $\boldsymbol{s}$ of Euclidean norm $\|\boldsymbol{s}\| = \lambda_1(\mathcal{L}) := \min_{\boldsymbol{v} \in \mathcal{L} \setminus \{\boldsymbol{0}\}} \|\boldsymbol{v}\|$. Given a basis of a lattice and a target vector $\boldsymbol{t} \in \mathbb{R}^d$, the closest vector problem (CVP) is to find a lattice vector $\boldsymbol{s} \in \mathcal{L}$ closest to $\boldsymbol{t}$. The preprocessing variant of CVP (CVPP) asks to preprocess the lattice $\mathcal{L}$ such that, when later given a target vector $\boldsymbol{t}$, one can quickly find a closest lattice vector to $\boldsymbol{t}$.

SVP and CVP are fundamental in the study of lattice-based cryptography, as the security of many schemes is directly related to their hardness. Various other hard lattice problems, such as Learning With Errors (LWE), are closely related to SVP and CVP; see, e.g., [63,74,75] for reductions among lattice problems. These reductions show that understanding the hardness of SVP and CVP is crucial for accurately estimating the security of lattice-based cryptographic schemes.

1.1 Related Work

Worst-Case SVP/CVP Analyses. Although SVP and CVP are both central in the study of lattice-based cryptography, algorithms for SVP have received somewhat more attention, including a benchmarking website to compare different methods [1]. Various SVP algorithms have been studied which can solve CVP as well, such as the polynomial-space, superexponential-time lattice enumeration studied in [14,32,38,40,47,66]. More recently, methods have been proposed which solve SVP/CVP in only single exponential time, but which also require exponential-sized memory [2,6,64]. By constructing the Voronoi cell of the lattice [4,25,64,73], Micciancio–Voulgaris showed that SVP and CVP(P) can provably be solved in time $2^{2d+o(d)}$, and Bonifas–Dadush reduced the complexity for CVPP to only $2^{d+o(d)}$. In high dimensions the best provable complexities for SVP and CVP are currently due to discrete Gaussian sampling [2,3], solving both problems in $2^{d+o(d)}$ time and space in the worst case on arbitrary lattices.

Average-Case SVP/CVP Algorithms. When considering and comparing these methods in practice on random lattices, we get a completely different picture. Currently the fastest heuristic methods for SVP and CVP in high dimensions are based on lattice sieving. After a long series of theoretical works on constructing efficient heuristic sieving algorithms [18–21,50,53,65,68,78,80] as well as applied papers studying how to further speed up these algorithms in practice [28,35,39,46,54,57–61,67,71,72], the best heuristic time complexity for solving SVP (and CVP [52]) currently stands at $2^{0.292d+o(d)}$ [18,59], using $2^{0.208d+o(d)}$ memory. The highest records in the SVP challenge [1] were recently obtained using a BKZ-sieving hybrid [7]. These recent improvements have resulted in a major shift in security estimates for lattice-based cryptography, from estimating the hardness of SVP/CVP using the best enumeration methods, to estimating this hardness based on state-of-the-art sieving results [9,24,26,27,36].

Hybrid Algorithms and Batch-CVP. In moderate dimensions, enumeration-based methods dominated for a long time, and the cross-over point with single-exponential time algorithms like sieving seemed to be far out of reach [66]. Moreover, the exponential memory of, e.g., lattice sieving will ultimately also significantly slow down these algorithms due to the large number of random memory accesses [23], and parallelizing sieving efficiently is less trivial than parallelizing enumeration [7,23,28,46,59,67,79]. Some previous work focused on obtaining a trade-off between enumeration and sieving, using less memory for sieving [17,43,44] or using more memory for enumeration [48].

Another well-known direction for a hybrid between memory-intensive methods and enumeration is to use a fast CVP(P) algorithm as a subroutine within enumeration. As described in, e.g., [40,66], at any given level in the enumeration tree, one is attempting to solve a CVP instance in a lower-rank sublattice, where the target vector is determined by the path from the root to the current node in the tree. Each node at this level in the tree corresponds to a CVP instance in the same sublattice, but with a different target. If we can preprocess this low-dimensional sublattice such that the amortized time complexity of solving a batch of CVP-instances in this sublattice is small, then this may speed up processing the bottom part of the enumeration tree.

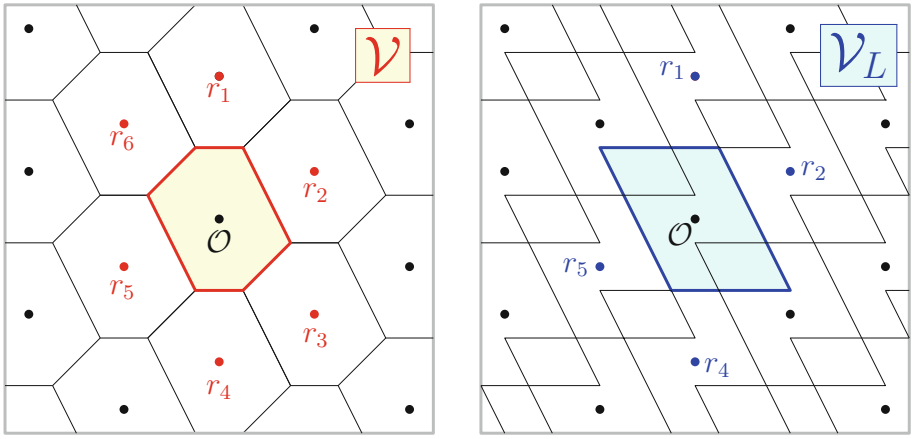
A first step in this direction was taken in [52], where it was shown that with a sufficient amount of preprocessing and space, one can achieve better amortized time complexities for batch-CVP than when solving just one instance. The large memory requirement (at least $2^{d/2+o(d)}$ memory is required to improve upon direct CVP approaches) as well as the large number of CVP instances required to get a lower amortized complexity made this approach impractical to date.

1.2 Contributions: Approximate Voronoi Cells

In this paper we revisit the preprocessing approach to CVP of [52], as well as the recent trend of speeding up these algorithms using nearest neighbor searching, and we show how to obtain significantly improved time and space complexities. These results can be viewed as a first step towards a practical, heuristic alternative to the Voronoi cell approach of Micciancio–Voulgaris [66], where instead of constructing the exact Voronoi cell, the preprocessing computes an approximation of it, requiring less time and space to compute and store.

First, our preprocessing step consists of computing a list L of most lattice vectors below a given norm.[1] This preprocessing can be done using either enumeration or sieving. The preprocessed data can best be understood as representing an *approximate* Voronoi cell $\mathcal{V}_L$ of the lattice, where the size of L determines how well $\mathcal{V}_L$ approximates the true Voronoi cell $\mathcal{V}$ of the lattice; see Fig. 1 for an example. Using this approximate Voronoi cell, we then attempt to solve CVP instances by applying the iterative slicing procedure of Sommer–Feder–Shalvi [73], with nearest neighbor optimizations to reduce the search costs [12,18].

[1] Heuristically, finding a large fraction of all lattice vectors below a given norm will suffice – one does not necessarily need to run a deterministic preprocessing algorithm to ensure all short lattice vectors are found.



(a) A tiling of $\mathbb{R}^2$ with exact Voronoi cells $\mathcal{V}$ of a lattice $\mathcal{L}$ (red/black points), generated by the set $\mathcal{R} = \{r_1, \ldots, r_6\}$ of all *relevant vectors* of $\mathcal{L}$. Here $\mathrm{vol}(\mathcal{V}) = \det(\mathcal{L})$.

(b) An overlapping tiling of $\mathbb{R}^2$ with approximate Voronoi cells $\mathcal{V}_L$ of the same lattice $\mathcal{L}$, generated by a subset of the relevant vectors, $L = \{r_1, r_2, r_4, r_5\} \subset \mathcal{R}$.

Fig. 1. Exact and approximate Voronoi cells of the same two-dimensional lattice $\mathcal{L}$. For the **exact** Voronoi cell $\mathcal{V}$ (Fig. 1a), the cells around the lattice points form a tiling of $\mathbb{R}^2$, covering each point in space exactly once. Given that a point $\boldsymbol{t}$ lies in the Voronoi cell around $\boldsymbol{s} \in \mathcal{L}$, we know that $\boldsymbol{s}$ is the closest lattice point to $\boldsymbol{t}$. For the **approximate** Voronoi cell $\mathcal{V}_L$ (Fig. 1b), the cells around the lattice points overlap, and cover a non-empty fraction of the space by multiple cells. Given that a vector $\boldsymbol{t}$ lies in an approximate Voronoi cell around a lattice point $\boldsymbol{s}$, we further do not have the definite guarantee that $\boldsymbol{s}$ is the closest lattice point to $\boldsymbol{t}$. (Color figure online)

The main difference in our work over [52] lies in generalizing how similar $\mathcal{V}_L$ (generated by the list L) needs to be to $\mathcal{V}$. We distinguish two cases below. As sketched in Fig. 1, a worse approximation leads to a larger approximate Voronoi cell, so $\mathrm{vol}(\mathcal{V}_L) \geq \mathrm{vol}(\mathcal{V})$ with equality iff $\mathcal{V} = \mathcal{V}_L$.

Good approximations: If $\mathcal{V}_L$ is a good approximation of $\mathcal{V}$ (i.e., $\mathrm{vol}(\mathcal{V}_L) \approx \mathrm{vol}(\mathcal{V})$), then with high probability over the randomness of the target vectors, the iterative slicer returns the closest lattice vector to random targets. To guarantee $\mathrm{vol}(\mathcal{V}_L) \approx \mathrm{vol}(\mathcal{V})$ we need $|L| \geq 2^{d/2+o(d)}$, where additional memory can be used to speed up the nearest neighbor part of the iterative slicer. The resulting query complexities are sketched in red in Fig. 2.

Arbitrary approximations: If the preprocessed list contains fewer than $2^{d/2}$ vectors, then $\mathrm{vol}(\mathcal{V}_L) \gg \mathrm{vol}(\mathcal{V})$ and with overwhelming probability the iterative slicer will not return the closest lattice point to a random target vector. However, similar to [40], the running time of this method is decreased by a much more significant factor than the success probability. So if we are able to *rerandomize* the problem instance and try several times, we may still be faster (and more memory-efficient) than when using a larger list L.

1.3 Contributions: Randomized Slicing

To actually find solutions to CVP instances with a "bad" approximation $\mathcal{V}_L$ to the real Voronoi cell $\mathcal{V}$, we need to be able to suitably rerandomize the iterative slicing procedure, so that if the success probability in a single run of the slicer is small, we can repeat the method several times for a high success probability. To do this, we will run the iterative slicer on randomly perturbed vectors $\boldsymbol{t}' \sim D_{\boldsymbol{t}+\mathcal{L},s}$, sampled from a discrete Gaussian over the coset $\boldsymbol{t}+\mathcal{L}$. Here the standard deviation s needs to be sufficiently large to make sampling from $D_{\boldsymbol{t}+\mathcal{L},s}$ efficient and the results of the slicer to be almost independent, and s needs to be sufficiently small to guarantee that the slicer will terminate in a limited number of steps. Algorithm 1 explicitly describes this procedure, given as input an approximate Voronoi cell $\mathcal{V}_L$ (i.e., a list $L \subset \mathcal{L}$ of short lattice vectors defining the facets of this approximate Voronoi cell).

Algorithm 1. The randomized heuristic slicer for finding closest vectors

Require: A list $L \subset \mathcal{L}$ and a target $\boldsymbol{t} \in \mathbb{R}^d$
Ensure: The algorithm outputs a closest lattice vector $\boldsymbol{s} \in \mathcal{L}$ to $\boldsymbol{t}$

$\boldsymbol{s} \leftarrow \boldsymbol{0}$ ▷ Initial guess $\boldsymbol{s}$ for closest vector to $\boldsymbol{t}$
repeat
  Sample $\boldsymbol{t}' \sim D_{\boldsymbol{t}+\mathcal{L},s}$ ▷ Randomly shift $\boldsymbol{t}$ by a vector $\boldsymbol{v} \in \mathcal{L}$
  for each $\boldsymbol{r} \in L$ **do**
    if $\|\boldsymbol{t}' - \boldsymbol{r}\| < \|\boldsymbol{t}'\|$ **then** ▷ New shorter vector $\boldsymbol{t}' \in \boldsymbol{t}+\mathcal{L}$
      Replace $\boldsymbol{t}' \leftarrow \boldsymbol{t}' - \boldsymbol{r}$ and restart the **for**-loop
  if $\|\boldsymbol{t}'\| < \|\boldsymbol{t} - \boldsymbol{s}\|$ **then**
    $\boldsymbol{s} \leftarrow \boldsymbol{t} - \boldsymbol{t}'$ ▷ New lattice vector $\boldsymbol{s}$ closer to $\boldsymbol{t}$
until $\boldsymbol{s}$ is a closest lattice vector to $\boldsymbol{t}$
return $\boldsymbol{s}$

Even though this algorithm requires sampling many vectors from the coset $\boldsymbol{t}+\mathcal{L}$ and running the iterative slicer on all of these, the overall time complexity of this procedure will still be lower, since the iterative slicer needs less time to complete when the input list L is shorter. To estimate the number of iterations necessary to guarantee that the algorithm returns the actual closest vector, we make the following assumption, stating that the probability that the iterative slicer terminates with a vector $\boldsymbol{t}' \in (\boldsymbol{t}+\mathcal{L}) \cap \mathcal{V}$, given that it must terminate to some vector $\boldsymbol{t}' \in (\boldsymbol{t}+\mathcal{L}) \cap \mathcal{V}_L$, is proportional to the ratio of the volumes of these (approximate) Voronoi cells $\mathcal{V}$ and $\mathcal{V}_L$.

Heuristic assumption 1 (Randomized slicing) *For $L \subset \mathcal{L}$ and large s,*

$$\Pr_{\boldsymbol{t}' \sim D_{\boldsymbol{t}+\mathcal{L},s}} \left[\mathrm{Slice}_L(\boldsymbol{t}') \in \mathcal{V} \right] \approx \frac{\mathrm{vol}(\mathcal{V})}{\mathrm{vol}(\mathcal{V}_L)} \,. \tag{1}$$

This is a new and critical assumption to guarantee that the claimed asymptotic complexities are correct, and we will therefore come back to this assumption later on, to show that experiments indeed suggest this assumption is justified.

1.4 Contributions: Improved CVPP Complexities

For the exact closest vector problem with preprocessing, our improved complexities over [52] mainly come from the aforementioned randomizations. To illustrate this with a simple example, suppose we run an optimized (GaussSieve-based [65]) LDSieve [18], ultimately resulting in a list of $(4/3)^{d/2+o(d)}$ of the shortest vectors in the lattice, indexed in a nearest neighbor data structure of size $(3/2)^{d/2+o(d)}$. Asymptotically, using this list as our approximate Voronoi cell, the iterative slicer succeeds with probability $p = (13/16)^{d/2+o(d)}$ (as shown in the analysis later on), while processing a query with this data structure takes time $(9/8)^{d/2+o(d)}$. By repeating a query $1/p$ times with rerandomizations of the same CVP instance, we obtain the following heuristic complexities for CVPP.

Proposition 1 (Standard sieve preprocessing). *Using the output of the LDSieve [18] as the preprocessed list and encompassing data structure, we can heuristically solve CVPP with the following query space and time complexities:*

$$\mathrm{S} = (3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}, \qquad \mathrm{T} = (18/13)^{d/2+o(d)} \approx 2^{0.235d+o(d)}.$$

This point (S, T) *is highlighted in light blue in Fig. 2.*

If we use a more general analysis of the approximate Voronoi cell approach, varying over both the nearest neighbor parameters and the size of the preprocessed list, we can obtain even better query complexities. For a memory complexity of $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$, we can achieve a query time complexity of approximately $2^{0.220d+o(d)}$ by using a shorter list of lattice vectors, and a more memory-intensive parameter setting for the nearest neighbor data structure. The following main result summarizes all the asymptotic time–space trade-offs we can obtain for heuristically solving CVPP in the average case.

Theorem 1 (Optimized CVPP complexities). *Let* $\alpha \in (1.03396, \sqrt{2})$ *and* $u \in (\sqrt{\frac{\alpha^2-1}{\alpha^2}}, \sqrt{\frac{\alpha^2}{\alpha^2-1}})$. *With approximate Voronoi cells we can heuristically solve CVPP with preprocessing space and time* S_1 *and* T_1*, and query space and time* S_2 *and* T_2*, where:*

$$\mathrm{S}_1 = \max\left\{\mathrm{S}_2, \left(\frac{4}{3}\right)^{d/2+o(d)}\right\}, \qquad \mathrm{T}_1 = \max\left\{\mathrm{S}_2, \left(\frac{3}{2}\right)^{d/2+o(d)}\right\}, \tag{2}$$

$$\mathrm{S}_2 = \left(\frac{\alpha}{\alpha - (\alpha^2-1)(\alpha u^2 - 2u\sqrt{\alpha^2-1} + \alpha)}\right)^{d/2+o(d)}, \tag{3}$$

$$\mathrm{T}_2 = \left(\frac{16\alpha^4\left(\alpha^2-1\right)}{-9\alpha^8+64\alpha^6-104\alpha^4+64\alpha^2-16} \cdot \frac{\alpha + u\sqrt{\alpha^2-1}}{-\alpha^3 + \alpha^2 u\sqrt{\alpha^2-1} + 2\alpha}\right)^{d/2+o(d)}. \tag{4}$$

The best query complexities $(\mathrm{S}_2, \mathrm{T}_2)$ *together form the blue curve in Fig. 2.*

Compared to [52], we obtain trade-offs for much lower memory complexities, and we improve upon both the best CVPP complexities of [52] and the best

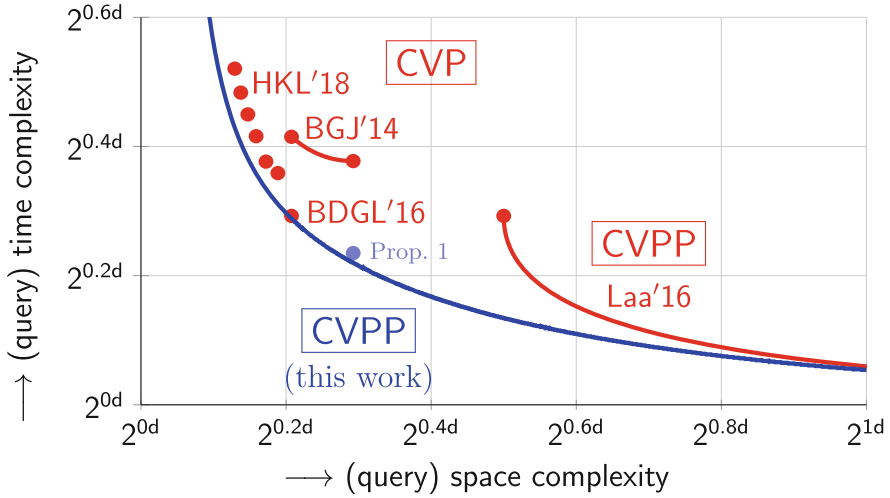


Fig. 2. Query complexities for finding closest vectors, directly (CVP) and with preprocessing (CVPP). The leftmost red points/curve show the best asymptotic SVP/CVP complexities of Becker–Gama–Joux [19], Becker–Ducas–Gama–Laarhoven [18], and Herold–Kirshanova–Laarhoven [44]. The rightmost red point and curve are the previous best CVPP complexities of [52]. The blue curve shows our new CVPP complexities. (Color figure online)

SVP/CVP complexities of [18,44].[2] Observe that our trade-off passes *below* all the best CVP results, i.e., we can always solve an exponentially large batch of $2^{\varepsilon d}$ CVP instances for small $\varepsilon > 0$ in the same amount of time as the current best complexities for solving just one instance, for any memory bound.

Due to the condition that $\alpha > 1.0339\ldots$ (which follows from the fact that the denominator in T_2 needs to remain positive), the blue curve in Fig. 2 terminates on the left side at a minimum query space complexity of $1.03396^{d+o(d)} \approx 2^{0.0482d+o(d)}$. One might wonder whether we can obtain a continuous trade-off between the query time and space complexities reaching all the way to $2^{o(d)}$ memory and $2^{\omega(d)}$ query time. The lower bound on α might be a consequence of our analysis, and perhaps a different approach would show this algorithm solves CVPP in $2^{O(d)}$ time even with less memory.

As for the other end of the blue curve, as the available space increases, one can achieve an amortized time complexity for CVP of $2^{\varepsilon d+o(d)}$ at the cost of $(1/\varepsilon)^{O(d)}$ preprocessed space for arbitrary $\varepsilon > 0$. For large query space complexities, i.e., when a lot of memory and preprocessing power is available for speeding up the queries, the blue and red curve converge, and the best parameter choice is to set $\alpha \approx \sqrt{2}$ such that $\mathcal{V}_L \approx \mathcal{V}$, as explained in Sect. 1.2.

[2] As detailed in [52], by modifying sieve algorithms for SVP, one can also solve CVP with essentially equivalent heuristic time and space complexities as for SVP.

Concrete Complexities. Although Theorem 1 and Fig. 2 illustrate how well we expect these methods to scale in high dimensions d, we would like to stress that Theorem 1 is a purely asymptotic result, with potentially large order terms hidden by the $o(d)$ in the exponents for the time and space complexities. To obtain security estimates for real-world applications, and to assess how fast this algorithm actually solves problems appearing in the cryptanalysis of lattice-based cryptosystems, it therefore remains necessary to perform extensive experiments, and to cautiously try to extrapolate from these results what the real attack costs might be for high dimensions d, necessary to attack actual instantiations of cryptosystems. Later on we will describe some preliminary experiments we performed to test the practicality of this approach, but further work is still necessary to assess the impact of these results on the concrete hardness of CVPP.

1.5 High-Level Proof Description

To prove the main results regarding the improved asymptotic CVPP complexities compared to [52], we first prove that under certain natural heuristic assumptions, we obtain the following upper bound on the volume of approximate Voronoi cells generated by the $\alpha^{d+o(d)}$ shortest vectors of a lattice. The preprocessing will consist of exactly this: generate the $\alpha^{d+o(d)}$ shortest vectors in the lattice, and store them in a nearest neighbor data structure that allows for fast look-ups of nearby points in space.

Lemma 1 (Relative volume of approximate Voronoi cells). *Let $L \subset \mathcal{L}$ consist of the $\alpha^{d+o(d)}$ shortest vectors of a lattice $\mathcal{L}$, with $\alpha \in (1.03396, \sqrt{2})$. Then heuristically,*

$$\frac{\mathrm{vol}(\mathcal{V}_L)}{\mathrm{vol}(\mathcal{V})} \leq \left(\frac{16\alpha^4 \left(\alpha^2 - 1\right)}{-9\alpha^8 + 64\alpha^6 - 104\alpha^4 + 64\alpha^2 - 16} \right)^{d/2+o(d)} . \tag{5}$$

Using this lemma and the heuristic assumption stated previously, relating the success probability of the slicer to the volume of the approximate Voronoi cell, this immediately gives us a (heuristic) lower bound on the success probability p_α of the randomized slicing procedure, given as input a preprocessed list of the $\alpha^{d+o(d)}$ shortest vectors in the lattice. Then, similar to [52], the complexity analysis is a matter of combining the costs for the preprocessing phase, the costs of the nearest neighbor data structure, and the cost of the query phase, where now we need to repeat the randomized slicing of the order $1/p_\alpha$ times – the difference in the formulas for the complexities compared to [52] comes exactly from this additional factor $1/p_\alpha \approx \mathrm{vol}(\mathcal{V}_L)/\mathrm{vol}(\mathcal{V})$.

To prove the above lemma regarding the volume of approximate Voronoi cells, we will prove the following statements. First, we show that if the list L contains the $\alpha^{d+o(d)}$ shortest vectors of a random lattice $\mathcal{L}$, then on input a target vector $\boldsymbol{t}$, we heuristically expect the slicer to terminate on a reduced vector $\boldsymbol{t}' \in \boldsymbol{t} + \mathcal{L}$ of norm at most $\|\boldsymbol{t}'\| \leq \beta \cdot \lambda_1(\mathcal{L})$, where β is determined by the parameter α. The relation between α and β can be succinctly described by the following relation

$$\beta = \alpha^2/\sqrt{4\alpha^2 - 4}. \tag{6}$$

More precisely, we show that as long as $\|\boldsymbol{t}'\| \gg \beta \cdot \lambda_1(\mathcal{L})$, then with high probability we expect to be able to combine $\boldsymbol{t}'$ with vectors in L to form a shorter vector $\boldsymbol{t}'' \in \boldsymbol{t} + \mathcal{L}$ with $\|\boldsymbol{t}''\| < \|\boldsymbol{t}'\|$. On the other hand, if we have a vector $\boldsymbol{t}' \in \boldsymbol{t} + \mathcal{L}$ of norm less than $\beta \cdot \lambda_1(\mathcal{L})$, then we only expect to be able to combine $\boldsymbol{t}'$ with a vector in L to form a shorter vector with exponentially small probability $2^{-\Theta(d)}$. In other words, reducing to a vector of norm $\beta \cdot \lambda_1(\mathcal{L})$ can be done almost "effortlessly", while after that even making small progress in reducing the length of $\boldsymbol{t}'$ comes at an exponential loss in the success probability.

Good Approximations. Next, from the above relation between the size of the input list, $|L|$ (or α), and the reduced norm of the shifted target vector, $\|\boldsymbol{t}'\|$ (or β), the previous result of [52] immediately follows – to achieve $\boldsymbol{t}' \in \mathcal{V}$ we heuristically need $\beta = 1 + o(1)$. This implies that $\alpha = \sqrt{2}$ is the minimal parameter that guarantees we will be able to effortlessly reduce to the exact Voronoi cell, and so L must contain the $\alpha^{d+o(d)} = 2^{d/2+o(d)}$ shortest vectors in the lattice. In that case the success probability is constant, and the costs of the query phase are determined by a single reduction of $\boldsymbol{t}$ with the iterative slicer.

Arbitrary Approximations. However, even if $\alpha < \sqrt{2}$ is smaller, and the corresponding β is therefore larger than 1, the slicer might still succeed with (exponentially) small probability. To analyze the success probability, note that from the Gaussian heuristic we may assume that the closest vector to our target $\boldsymbol{t}$ lies uniformly at random in a ball (or sphere) of radius $\lambda_1(\mathcal{L})$ around $\boldsymbol{t}$. Then, also for the reduced vector $\boldsymbol{t}'$ of norm at most $\beta \cdot \lambda_1(\mathcal{L})$, the closest lattice vector lies in a ball of radius $\lambda_1(\mathcal{L})$ around it. Since our list L contains all vectors of norm less than $\alpha \cdot \lambda_1(\mathcal{L})$, we will clearly find the closest lattice vector in the list L if the closest lattice vector lies in the intersection of two balls of radii $\lambda_1(\mathcal{L})$ (resp. $\alpha \cdot \lambda_1(\mathcal{L})$) around $\boldsymbol{t}'$ (resp. $\mathbf{0}$). Estimating the volume of this intersection of balls, relative to the volume of the ball of radius $\lambda_1(\mathcal{L})$ around $\boldsymbol{t}'$, then gives us a lower bound on the success probability of the slicer, and a heuristic upper bound on the volume of the corresponding approximate Voronoi cell. This analysis ultimately leads to the aforementioned lemma.

Tightness of the Proof. Note that the above proof technique only gives us a *lower bound* on the success probability, and an *upper bound* on the volume of the approximate Voronoi cell: when the target vector has been reduced to a vector of norm at most $\beta \cdot \lambda_1(\mathcal{L})$, we bound the success probability of the slicer by the probability that the slicer now terminates successfully in a *single* iteration. Since the algorithm might also succeed in more than one additional iteration, the actual success probability may be higher. A tighter analysis, perhaps showing that the given heuristic bound can be improved upon, is left for future work.

1.6 Intermezzo: Another Few Dimensions for Free

Recently, Ducas [35] showed that in practice, one can effectively use the additional vectors found by lattice sieving to solve a few extra dimensions of SVP "for free". More precisely, by running a lattice sieve in a base dimension d, one can solve SVP in dimension $d' = d + \Theta(d/\log d)$ at little additional cost. This is done by taking all vectors returned by a d-dimensional lattice sieve, and running Babai's nearest plane algorithm [16] on all these vectors in the d'-dimensional lattice to find short vectors in the full lattice. If d' is close enough to d, one of these vectors will then be "rounded" to a shortest vector of the full lattice.

On a high level, Ducas' approach can be viewed as a sieving/enumeration hybrid, where the *top* part of enumeration is replaced with sieving, and the bottom part is done regularly as in enumeration, which is essentially equivalent to doing Babai rounding [16]. The approach of using a CVPP-solver inside enumeration is in a sense dual to Ducas' idea, as here the *bottom* part of the enumeration tree is replaced with a (sieving-like) CVPP routine. Since our CVPP complexities are strictly better than the best SVP/CVP complexities, we can also gain up to $\Theta(d/\log d)$ dimensions for free as follows:

1. First, we initialize an enumeration tree in the full lattice $\mathcal{L}$ of dimension $d' = d+k$, and we process the top $k = \varepsilon \cdot d/\log d$ levels as usual in enumeration. This will result in $2^{\Theta(k \log k)} = 2^{\Theta(d)}$ target vectors at level k, and this requires a similar time complexity of $2^{\Theta(d)}$ to generate all these target vectors.
2. Then, we run the CVPP preprocessing on the d-dimensional sublattice of $\mathcal{L}$ corresponding to the bottom part of the enumeration tree. This may for instance take time $2^{0.292d+o(d)}$ and space $2^{0.208d+o(d)}$ using the sieve of [18].
3. Finally, we take the batch of $2^{\Theta(d)}$ target vectors at level k in the enumeration tree, and we solve CVP for each of them with our approximate Voronoi cell and randomized slicing algorithm, with query time $2^{0.220d+o(d)}$ each.

By setting $k = \varepsilon \cdot d/\log d$ as above with small, constant $\varepsilon > 0$, the costs for solving SVP or CVP in dimension d' are asymptotically dominated by the costs of the preprocessing step, which is as costly as solving SVP or CVP in dimension d. So similar to [35], asymptotically we also get $\Theta(d/\log d)$ dimensions "for free". However, unlike for Ducas' idea, in practice the dimensions are likely not quite as free here, as there is more overhead for doing the CVPP-version of sieving than for Ducas' additional batch of Babai nearest plane calls.

Even More Dimensions for Free. A natural question one might ask now is: can both ideas be combined to get even more dimensions "for free"? At first sight, this seems hard to accomplish, as Ducas' idea speeds up SVP rather than CVPP. Furthermore, note that when solving SVP without Ducas' trick, one gets $2^{0.208d+o(d)}$ short lattice vectors when only one shortest vector is needed, and so in a sense one might "naturally" hope to gain something by going for only one short output vector. Here the analysis of the iterative slicer is already based on the fact that ultimately, we hope to reduce a single target vector to its closest neighbor in the lattice. There might be a way of combining both ideas to get even more dimensions for free, but for now this is left as an open problem.

1.7 Contributions: Experimental Results

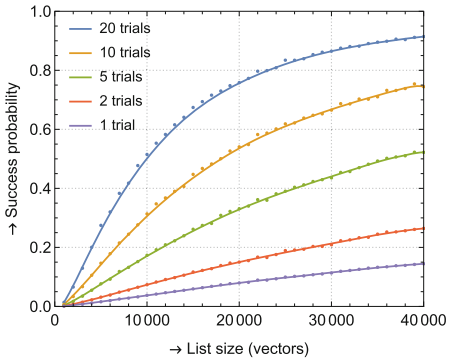
Besides the theoretical contributions mentioned above, with improved heuristic time and space complexities compared to [52], for the first time we also implemented a (sieving-based) CVPP solver using approximate Voronoi cells. For the preprocessing we used a slight modification of a lattice sieve, returning more vectors than a standard sieve, allowing us to vary the list size in our experiments. Our implementations serve two purposes: validating the additional heuristic assumption we make, and to see how well the algorithm performs.

Validation of the Randomization Assumption. To obtain the aforementioned improved asymptotic complexities for solving CVPP, we required a new heuristic assumption, stating that if the iterative slicer succeeds with some probability p on a CVP instance $\boldsymbol{t}$, then we can repeat it $1/p$ times with perturbations $\boldsymbol{t}' \sim D_{\boldsymbol{t}+\mathcal{L},s}$ to achieve a high success probability for the same target $\boldsymbol{t}$. To verify this assumption, we implemented our method and tested it on lattices of dimension 50 with a range of randomly chosen targets to see whether, if the probability of success is small, repeating the method m times will increase the success rate by a factor m. Figure 3 shows performance metrics for various numbers of repetitions and for varying list sizes. In particular, Fig. 3a illustrates the increased success probability as the number of repetitions increases, and Fig. 3c shows that the normalized success probability per trial[3] seems independent of the number of repetitions. Therefore, the "expected time" metric as illustrated in Fig. 3b appears to be independent of the number of trials.

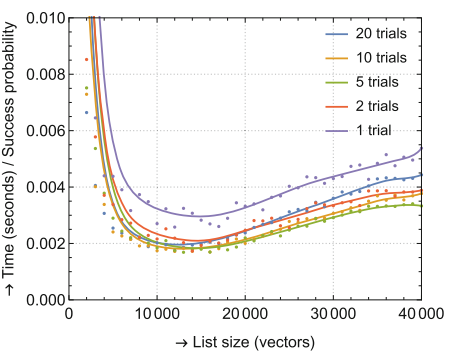
Experimental Performance. Unlike the success probabilities, the time complexity might vary a lot depending on the underlying nearest neighbor data structure. For our experiments we used hyperplane LSH [29] as also used in the HashSieve [50,58], as it is easy to implement, has few parameters to set, and performs better in low dimensions ($d = 50$) than the LDSieve [18,59].

To put the complexities of Fig. 3b into perspective, let us compare the normalized time complexities for CVPP with the complexities of sieving for SVP, which by [52] are comparable to the costs for CVP. First, we note that the HashSieve algorithm solves SVP in approximately 4 s on the same machine. This means that in dimension 50, the expected time complexity for CVPP with the HashSieve (roughly 2 ms) is approximately 2000 times smaller than the time for solving SVP. To explain this gap, observe that the list size for solving SVP is approximately 4000, and so the HashSieve algorithm needs to perform in the order of 4000 reductions of newly sampled vectors with a list of size 4000. For solving CVPP, we only need to reduce 1 target vector, with a slightly larger list of 10 000 to 15 000 vectors. So we save a factor 4000 on the number of reductions, but the searches are more expensive, leading to a speed-up of less than a factor 4000.

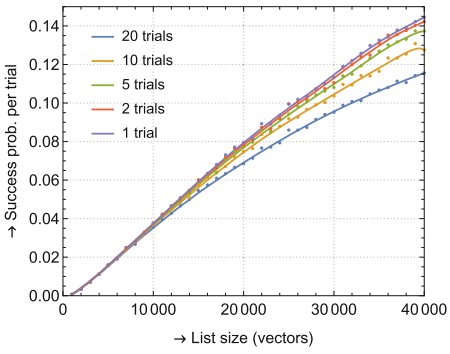
[3] As the success prob. q for m trials scales as $q = 1-(1-p)^m$ if each trial independently has success prob. p, we computed the success prob. per trial as $p = 1 - (1-q)^{1/m}$.



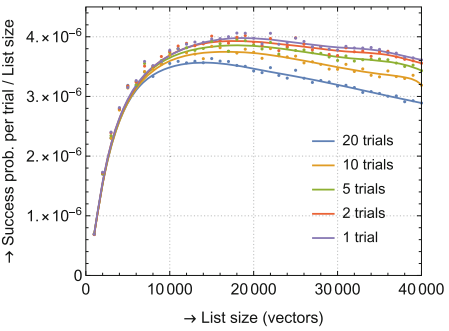
(a) Success prob. with rerandomizations



(b) Expected time per CVP instance



(c) Average success prob. per trial



(d) Success prob. per trial, per vector

Fig. 3. Experimental results for solving CVPP with randomized slicing in dimension 50. Each data point corresponds to 10 000 random target vectors for those parameters.

Predictions and Extrapolations. For solving SVP or CVP, the Hash-Sieve [50] reports time complexities in dimension d of $2^{0.45d-19}$ s, corresponding to 11 s in dimension 50, i.e., a factor 3 slower than here. This is based on doing $n \approx 2^{0.21d}$ reductions of vectors with the list. If doing only one of these searches takes a factor $2^{0.21d}$ less time, and we take into account that for SVP the time complexity is now a factor 3 less than in [50], then we obtain an estimated complexity for CVPP in dimension d of $2^{0.24d-19}/3$, which for $d = 50$ corresponds to approximately 2.6 ms. A rough extrapolation would then lead to a time complexity in dimension 100 of only 11 s. This however seems to be rather optimistic – preliminary experiments in dimensions 60 and 70 suggest that the overhead of using a lot of memory may be rather high here, as the list size is usually even larger than for standard sieving.

1.8 Contributions: Asymptotics for Variants of CVPP

For easier variants of CVP, such as when the target lies closer to the lattice than expected or an approximate solution to CVP suffices as a solution, we obtain considerable gains in both the time and space complexities when using preprocessing. We explicitly consider two variants of CVPP below.

BDDP$_\delta$. For bounded distance decoding with preprocessing (BDDP), we are given a target vector $\boldsymbol{t}$ and a guarantee that $\boldsymbol{t}$ lies within distance $\delta \cdot \lambda_1(\mathcal{L})$ to the nearest lattice vector, for some parameter $\delta > 0$. By the Gaussian heuristic, setting $\delta = 1$ makes this problem as hard as general CVPP without a distance guarantee, while for small $\delta \to 0$ polynomial-time algorithms exist [16].

By adjusting the analysis leading up to Theorem 1 for BDDP, we obtain the same result as Theorem 1 with two modifications: T_2 is replaced by $\mathrm{T}_2^{(\delta)}$ below, and the range of admissable values α changes to (α_0, α_1), with α_0 the smallest root larger than 1 of the denominator of the left-most term in $\mathrm{T}_2^{(\delta)}$, and α_1 the smallest value larger than 1 such that the left-most term in $\mathrm{T}_2^{(\delta)}$ equals 1. The resulting optimized trade-offs for various $\delta \in (0, 1)$ are plotted in Fig. 4a.

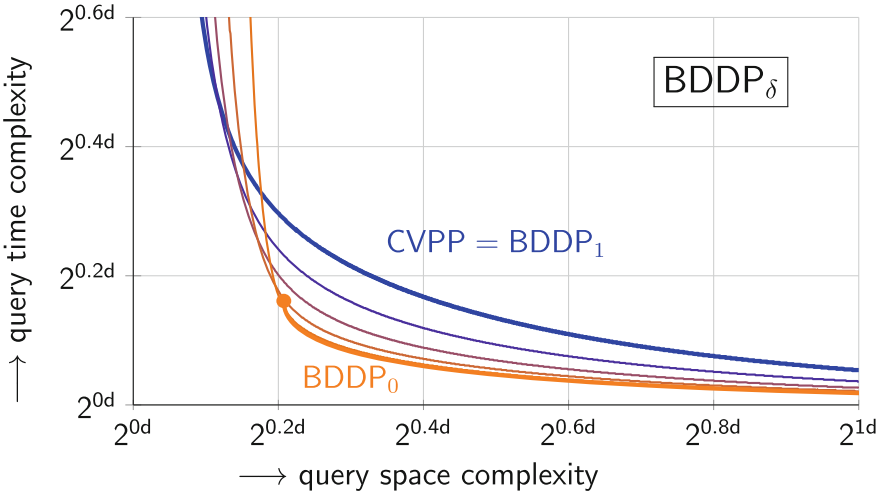
$$\mathrm{T}_2^{(\delta)} = \left(\frac{16\alpha^4 \left(\alpha^2 - 1\right) \delta^2}{-9\alpha^8 + 8\alpha^6(3+5\delta^2) - 8\alpha^4(2+9\delta^2+2\delta^4) + 32\alpha^2(\delta^2+\delta^4) - 16\delta^4} \cdot [\dots] \right)^{d/2+o(d)} . \quad (7)$$

Note that in the limit of $\delta \to 0$, our algorithm tries to reduce a target close to the lattice to the origin. This is similar to reducing a vector to the $\mathbf{0}$-vector in the GaussSieve [65], and even with a long list of all short lattice vectors this does not occur with probability 1. Here also the limiting curve in Fig. 4a shows that for $\delta \to 0$ with suitable parameterization we can do better than just with sieving, but we do not get polynomial time and space complexities.

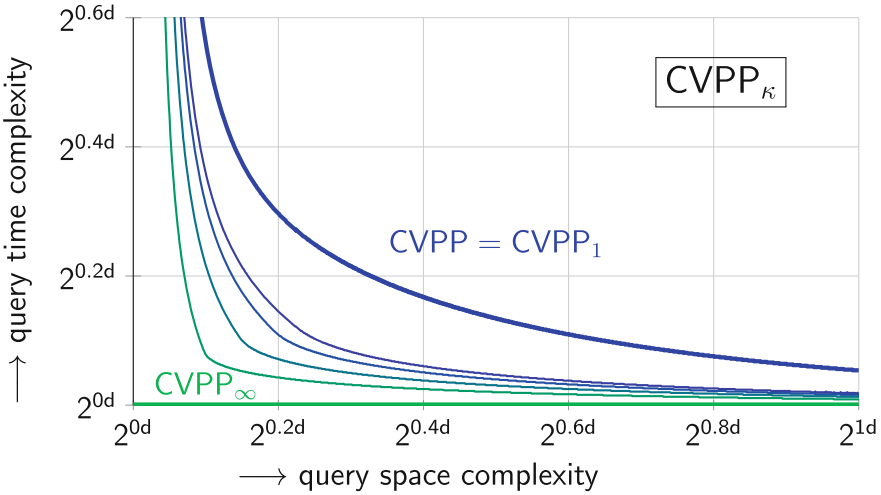
CVPP$_\kappa$. For the approximate version of CVPP, a lattice vector $\boldsymbol{v}$ qualifies as a solution for $\boldsymbol{t}$ if it lies at most a factor κ further from the real distance of $\boldsymbol{t}$ from the lattice, for some $\kappa \geq 1$. Heuristically, this is essentially equivalent to looking for any lattice vector within radius $\kappa \cdot \lambda_1(\mathcal{L})$ of the target, and similar to BDDP the resulting trade-offs can be summarized by Theorem 1 where T_2 is replaced by $\mathrm{T}_2^{(\kappa)}$ below, and the range of admissable values α again changes to (α_0, α_1) as before.

$$\mathrm{T}_2^{(\kappa)} = \left(\frac{16\alpha^4 \left(\alpha^2 - 1\right)}{-9\alpha^8 + 8\alpha^6(3+5\kappa^2) - 8\alpha^4(2+9\kappa^2+2\kappa^4) + 32\alpha^2(\kappa^2+\kappa^4) - 16\kappa^4} \cdot [\dots] \right)^{d/2+o(d)} . \quad (8)$$

For increasing approximation factors $\kappa \to \infty$, our algorithm tries to reduce a target vector to vector of norm less than $\kappa \cdot \lambda_1(\mathcal{L})$. For large κ this is increasingly easy to achieve, and as $\kappa \to \infty$, both the query time and space complexities in our analysis converge to zero as expected. Figure 4b highlights this asymptote, and illustrates the other trade-offs through some examples for small $\kappa > 1$.



(a) Heuristic complexities for BDDP_δ for different values $\delta \in \{0, 0.2, \ldots, 0.8, 1\}$. Smaller δ correspond to easier problems but also to a larger lower bound α_0 on α. The trade-off for $\delta \to 0$ is indicated by the thick orange line.



(b) Heuristic query complexities for CVPP_κ for different approximation factors $\kappa \in \{\sqrt{4/3}, 1.2, 1.3, 1.5, \infty\}$. The thick green line shows the limit as $\kappa \to \infty$.

Fig. 4. Asymptotics for solving variants of CVP(P) with approximate Voronoi cells: (a) BDDP_δ and (b) CVPP_κ. Note that the (tail of the) curve for $\mathrm{CVPP}_{\sqrt{4/3}}$ overlaps with the curve for BDDP_0.

1.9 Open Problems

Combination with Other Techniques. The focus of this work was on the asymptotic complexities we can achieve for high dimensions d, and therefore we focused only on including techniques from the literature that lead to the best asymptotics. In practice however, there may be various other techniques that can help speed up these methods in moderate dimensions. This for instance includes Ducas' dimensions for free [35], progressive sieving [35,54], the recent sieving-BKZ hybrid [7], and faster NNS techniques [7,11]. Incorporating such techniques will likely affect the experimental performance as well, and future work may show how well the proposed techniques truly perform in practice when all the state-of-the-art techniques are combined into one.

Faster Enumeration with Approximate Voronoi Cells. As explained above, one potential application of our CVPP algorithm is as a subroutine within enumeration, to speed up the searches in the bottom part of the tree. Such an algorithm can be viewed as a trade-off between enumeration and sieving, where the level at which we insert the CVPP oracle determines whether we are closer to enumeration or to sieving. An open question remains whether this would lead to faster algorithms in practice, or if the preprocessing/query costs are too high. Note that depending on at which level of the tree the CVPP oracle is inserted, and on the amount of pruning in enumeration, the hardness of the CVP instances at these levels also changes. Optimizing all parameters involved in such a combination appears to be a complex task, and is left for future work.

Sieving in the Dual Lattice. For the application of CVPP within enumeration, observe that a decisional CVPP oracle, deciding whether a vector lies close to the lattice or not, may actually be sufficient; most branches of the enumeration tree will not lead to a solution, and therefore in most cases running an accurate decision-CVPP oracle is enough to determine that this subtree is not the right subtree. For those few subtrees that potentially do contain a solution, one could then run a full CVP(P) algorithm at a slightly higher cost. Improving the complexities for the decision-version of CVPP may therefore be an interesting future direction, and perhaps one approach could be to combine this with ideas from [5], by running a lattice sieve on the dual lattice to find many short vectors in the dual lattice, which can then be used to check if a target vector lies close to the primal lattice or not.

Quantum Complexities. As one of the strengths of lattice-based cryptography is its conjectured resistance to quantum attacks [22], it is important to study the potential impact of quantum improvements to SVP and CVP algorithms, so that the parameters can be chosen to be secure in a post-quantum world [15,55]. For lattice sieving for solving SVP, the time complexity exponent potentially decreases by approximately 25% [55], and for CVPP we expect the exponents may decrease by approximately 25% as well. Studying the exact

quantum asymptotics of solving CVPP with approximate Voronoi cells is left for future work.

1.10 Outline

Due to space restrictions, the remainder of the paper, including full details on all claims, is given in the appendix.[4] Below we briefly outline the contents of these appendices for the interested reader.

Appendix A – Preliminaries
This section describes preliminary results and notation for the technical contents, formally states the main hard problems discussed in the paper, formalizes the heuristic assumptions made throughout the paper, and describes existing results on nearest neighbor searching, lattice sieving algorithms, Voronoi cells, and Voronoi cell algorithms.

Appendix B – Approximate Voronoi cells
In Appendix B we formalize the CVPP approach considered in this paper in terms of our approximate Voronoi cell framework with randomized slicing, and we derive our main results regarding improved asymptotic complexities for exact CVPP. Approximate Voronoi cells are formally introduced, the main results are stated and proved in terms of this framework, and all corresponding algorithms are given in pseudocode.

Appendix C – Experimental results
Appendix C describes the experiments we performed with these methods in more detail, both to verify the (additional) heuristic assumptions we made for this paper, and to assess the practicality of our CVPP algorithm. Here we also briefly compare our results to various published complexities for SVP or CVP(P), to put these numbers into context.

Appendix D – Asymptotics for variants of CVPP
The last appendix finally discusses asymptotic results for variants of CVPP, namely approximate CVPP and BDDP. This section contains a more formal statement of the results given in Sect. 1.8, and explains how the analysis changes compared to the analysis for exact CVPP, and how this leads to improved complexities for these slightly easier variants of (exact) CVPP.

Acknowledgments. The authors are indebted to Léo Ducas, whose ideas and suggestions on this topic motivated work on this paper. The authors are further grateful to the reviewers, whose thorough study of the contents (with one review even exceeding the page limit for the conference) significantly helped improve the contents of the paper, as well as improve the presentation of the results. Emmanouil Doulgerakis is supported by the NWO under grant 628.001.028 (FASOR). At the time of writing a preliminary version of this paper, Thijs Laarhoven was supported by the SNSF ERC Transfer Grant CRETP2-166734 FELICITY. At the time of publishing, Thijs Laarhoven is supported by a Veni Innovational Research Grant from NWO under project number 016.Veni.192.005.

[4] The full version of this paper including all appendices will be made available online at https://eprint.iacr.org/2016/888.

References

1. SVP challenge (2018). http://latticechallenge.org/svp-challenge/
2. Aggarwal, D., Dadush, D., Regev, O., Stephens-Davidowitz, N.: Solving the shortest vector problem in 2^n time via discrete Gaussian sampling. In: STOC, pp. 733–742 (2015)
3. Aggarwal, D., Dadush, D., Stephens-Davidowitz, N.: Solving the closest vector problem in 2^n time - the discrete Gaussian strikes again! In: FOCS, pp. 563–582 (2015)
4. Agrell, E., Eriksson, T., Vardy, A., Zeger, K.: Closest point search in lattices. IEEE Transact. Inf. Theor. **48**(8), 2201–2214 (2002)
5. Aharonov, D., Regev, O.: Lattice problems in $\mathsf{NP} \cap \mathsf{coNP}$. In: FOCS, pp. 362–371 (2004)
6. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: STOC, pp. 601–610 (2001)
7. Albrecht, M., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E., Stevens, M.: The general sieve kernel and new records in lattice reduction. Preprint, 2018
8. Alekhnovich, M., Khot, S., Kindler, G., Vishnoi, N.: Hardness of approximating the closest vector problem with pre-processing. In: FOCS, pp. 216–225 (2005)
9. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: USENIX Security Symposium, pp. 327–343 (2016)
10. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: FOCS, pp. 459–468 (2006)
11. Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., Schmidt, L.: Practical and optimal LSH for angular distance. In: NIPS, pp. 1225–1233 (2015)
12. Andoni, A., Laarhoven, T., Razenshteyn, I., Waingarten, E.: Optimal hashing-based time-space trade-offs for approximate near neighbors. In: SODA, pp. 47–66 (2017)
13. Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: STOC, pp. 793–801 (2015)
14. Aono, Y., Nguyen, P.Q.: Random sampling revisited: lattice enumeration with discrete pruning. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 65–102. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_3
15. Aono, Y., Nguyen, P.Q., Shen, Y.: Quantum lattice enumeration and tweaking discrete pruning. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11272, pp. 405–434. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03326-2_14
16. Babai, L.: On Lovasz lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986)
17. Bai, S., Laarhoven, T., Stehlé, D.: Tuple lattice sieving. In: ANTS, pp. 146–162 (2016)
18. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: SODA, pp. 10–24 (2016)
19. Becker, A., Gama, N., Joux, A.: A sieve algorithm based on overlattices. In: ANTS, pp. 49–70 (2014)
20. Becker, A., Gama, N., Joux, A.: Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, pp. 1–14 (2015)

21. Becker, A., Laarhoven, T.: Efficient (ideal) lattice sieving using cross-polytope LSH. In: AFRICACRYPT, pp. 3–23 (2016)
22. Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): Post-quantum Cryptography. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-88702-7
23. Bernstein, D.J., Chuengsatiansup, C., Lange, T., van Vredendaal, C.: NTRU prime: reducing attack surface at low cost. In: Adams, C., Camenisch, J. (eds.) SAC 2017. LNCS, vol. 10719, pp. 235–260. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72565-9_12
24. Bhattacharya, S., et al.: Round5: Compact and fast post-quantum public-key encryption. Cryptology ePrint Archive, Report 2018/725 (2018)
25. Bonifas, N., Dadush, D.: Short paths on the Voronoi graph and the closest vector problem with preprocessing. In: SODA, pp. 295–314 (2015)
26. Bos, J., et al.: Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In: CCS, pp. 1006–1018 (2016)
27. Bos, J., et al.: CRYSTALS - Kyber: a CCA-secure module-lattice-based KEM. In: Euro S&P, pp. 353–367 (2018)
28. Bos, J.W., Naehrig, M., van de Pol, J.: Sieving for shortest vectors in ideal lattices: a practical perspective. Int. J. Appl. Crypt. **3**(4), 313–329 (2016)
29. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: STOC, pp. 380–388 (2002)
30. Christiani, T.: A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In: SODA, pp. 31–46 (2017)
31. Conway, J.H., Sloane, N.J.A.: Sphere Packings, Lattices and Groups. Springer, Heidelberg (1999). https://doi.org/10.1007/978-1-4757-6568-7
32. Correia, F., Mariano, A., Proenca, A., Bischof, C., Agrell, E.: Parallel improved Schnorr-Euchner enumeration SE++ for the CVP and SVP. In: PDP, pp. 596–603 (2016)
33. Dadush, D., Regev, O., Stephens-Davidowitz, N.: On the closest vector problem with a distance guarantee. In: CCC, pp. 98–109 (2014)
34. The FPLLL development team. FPLLL, a lattice reduction library (2016). https://github.com/fplll/fplll
35. Ducas, L.: Shortest vector from lattice sieving: a few dimensions for free. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 125–145. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_5
36. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS - Dilithium: Digital signatures from module lattices. CHES **2018**, 238–268 (2018)
37. Feige, U., Micciancio, D.: The inapproximability of lattice and coding problems with preprocessing. In: CCC, pp. 32–40 (2002)
38. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice. Math. Comput. **44**(170), 463–471 (1985)
39. Fitzpatrick, R., et al.: Tuning gausssieve for speed. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 288–305. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16295-9_16
40. 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). https://doi.org/10.1007/978-3-642-13190-5_13
41. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC, pp. 197–206 (2008)

42. Hermans, J., Schneider, M., Buchmann, J., Vercauteren, F., Preneel, B.: Parallel shortest lattice vector enumeration on graphics cards. In: AFRICACRYPT, pp. 52–68 (2010)
43. Herold, G., Kirshanova, E.: Improved algorithms for the approximate k-list problem in euclidean norm. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10174, pp. 16–40. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54365-8_2
44. Herold, G., Kirshanova, E., Laarhoven, T.: Speed-ups and time–memory trade-offs for tuple lattice sieving. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10769, pp. 407–436. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76578-5_14
45. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC, pp. 604–613 (1998)
46. Ishiguro, T., Kiyomoto, S., Miyake, Y., Takagi, T.: Parallel gauss sieve algorithm: solving the SVP challenge over a 128-dimensional ideal lattice. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 411–428. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_24
47. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: STOC, pp. 193–206 (1983)
48. Kirchner, P., Fouque, P.-A.: Time-memory trade-off for lattice enumeration in a ball. Cryptology ePrint Archive, Report 2016/222 (2016)
49. Klein, P.: Finding the closest lattice vector when it's unusually close. In: SODA, pp. 937–941 (2000)
50. Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 3–22. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_1
51. Laarhoven, T.: Tradeoffs for nearest neighbors on the sphere. arXiv:1511.07527 [cs.DS], pp. 1–16 (2015)
52. Laarhoven, T.: Sieving for closest lattice vectors (with preprocessing). In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 523–542. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_28
53. Laarhoven, T., de Weger, B.: Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 101–118. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_6
54. Laarhoven, T., Mariano, A.: Progressive lattice sieving. In: PQCrypto, pp. 292–311 (2018)
55. Laarhoven, T., Mosca, M., van de Pol, J.: Finding shortest lattice vectors faster using quantum search. Des. Codes Crypt. **77**(2), 375–400 (2015)
56. Lagarias, J.C., Lenstra, H.W., Schnorr, C.-P.: Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice. Combinatorica **10**(4), 333–348 (1990)
57. Mariano, A., Bischof, C.: Enhancing the scalability and memory usage of HashSieve on multi-core CPUs. In: PDP, pp. 545–552 (2016)
58. Mariano, A., Laarhoven, T., Bischof, C.: Parallel (probable) lock-free HashSieve: a practical sieving algorithm for the SVP. In: ICPP, pp. 590–599 (2015)
59. Mariano, A., Laarhoven, T., Bischof, C.: A parallel variant of LDSieve for the SVP on lattices. In: PDP, pp. 23–30 (2017)

60. Mariano, A., Dagdelen, Ö., Bischof, C.: A comprehensive empirical comparison of parallel listsieve and gausssieve. In: Lopes, L., et al. (eds.) Euro-Par 2014. LNCS, vol. 8805, pp. 48–59. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14325-5_5
61. Mariano, A., Timnat, S., Bischof, C.: Lock-free GaussSieve for linear speedups in parallel high performance SVP calculation. In: SBAC-PAD, pp. 278–285 (2014)
62. Micciancio, D.: The hardness of the closest vector problem with preprocessing. IEEE Transact. Inf. Theory **47**(3), 1212–1215 (2001)
63. Micciancio, D.: Efficient reductions among lattice problems. In: SODA, pp. 84–93 (2008)
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., Walter, M.: Fast lattice point enumeration with minimal overhead. In: SODA, pp. 276–294 (2015)
67. Milde, B., Schneider, M.: A parallel implementation of gausssieve for the shortest vector problem in lattices. In: Malyshkin, V. (ed.) PaCT 2011. LNCS, vol. 6873, pp. 452–458. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23178-0_40
68. Nguyên, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. Math. Cryptol. **2**(2), 181–207 (2008)
69. Dagdelen, Ö., Schneider, M.: Parallel enumeration of shortest lattice vectors. In: D'Ambra, P., Guarracino, M., Talia, D. (eds.) Euro-Par 2010. LNCS, vol. 6272, pp. 211–222. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15291-7_21
70. Regev, O.: Improved inapproximability of lattice and coding problems with preprocessing. IEEE Transact. Inf. Theory **50**(9), 2031–2037 (2004)
71. 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). https://doi.org/10.1007/978-3-642-19094-0_11
72. Schneider, M.: Sieving for shortest vectors in ideal lattices. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 375–391. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38553-7_22
73. Sommer, N., Feder, M., Shalvi, O.: Finding the closest lattice point by iterative slicing. SIAM J. Discret. Math. **23**(2), 715–731 (2009)
74. Stephens-Davidowitz, N.: Dimension-preserving reductions between lattice problems, pp. 1–6 (2016). http://noahsd.com/latticeproblems.pdf
75. van de Pol, J.: Lattice-based cryptography. Master's thesis, Eindhoven University of Technology (2011)
76. Viterbo, E., Biglieri, E.: Computing the voronoi cell of a lattice: the diamond-cutting algorithm. IEEE Transact. Inf. Theory **42**(1), 161–171 (1996)
77. Wang, J., Shen, H.T., Song, J., Ji, J.: Hashing for similarity search: a survey. arXiv:1408.2927 [cs.DS], pp. 1–29 (2014)
78. Wang, X., Liu, M., Tian, C., Bi, J.: Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In: ASIACCS, pp. 1–9 (2011)
79. Yang, S.-Y., Kuo, P.-C., Yang, B.-Y., Cheng, C.-M.: Gauss sieve algorithm on GPUs. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 39–57. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_3
80. Zhang, F., Pan, Y., Hu, G.: A three-level sieve algorithm for the shortest vector problem. In: SAC, pp. 29–47 (2013)