Lattice Reduction with Approximate Enumeration Oracles

Practical Algorithms and Concrete Performance

Martin R. Albrecht[1(✉)], Shi Bai[2], Jianwei Li[1], and Joe Rowell[1]

[1] Information Security Group, Royal Holloway, University of London, Egham, UK
martin.albrecht@royalholloway.ac.uk , Jianwei.Li@rhul.ac.uk
[2] Department of Mathematical Sciences, Florida Atlantic University, Boca Raton, USA

Abstract. This work provides a systematic investigation of the use of approximate enumeration oracles in BKZ, building on recent technical progress on speeding-up lattice enumeration: *relaxing* (the search radius of) enumeration and *extended preprocessing* which preprocesses in a larger rank than the enumeration rank. First, we heuristically justify that relaxing enumeration with certain extreme pruning asymptotically achieves an exponential speed-up for reaching the same root Hermite factor (RHF). Second, we perform simulations/experiments to validate this and the performance for relaxed enumeration with numerically optimised pruning for both regular and extended preprocessing.

Upgrading BKZ with such approximate enumeration oracles gives rise to our main result, namely a practical and faster (wrt. previous work) polynomial-space lattice reduction algorithm for reaching the same RHF in practical and cryptographic parameter ranges. We assess its concrete time/quality performance with extensive simulations and experiments.

1 Introduction

Lattices are discrete subgroups of $\mathbb{R}^m$. A lattice $\mathcal{L}$ in $\mathbb{R}^m$ is represented as a set of all integer linear combinations of n linearly independent vectors $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1}$ in $\mathbb{R}^m$: $\mathcal{L} = \left\{\sum_{i=0}^{n-1} x_i \cdot \boldsymbol{b}_i,\ x_i \in \mathbb{Z}\right\}$. The matrix $\boldsymbol{B} := (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ forms a *basis* of $\mathcal{L}$, and the integer n is the *rank* of $\mathcal{L}$. Any lattice of rank ≥ 2 has infinitely many bases.

A central lattice problem is the *shortest vector problem* (SVP): given a basis of a lattice $\mathcal{L}$ (endowed with the Euclidean norm), SVP is to find a shortest nonzero vector in $\mathcal{L}$. SVP is known to be NP-hard under randomised reductions [3]. The

J. Rowell—This work was supported in part by EPSRC grants EP/S020330/1, EP/S02087X/1, EP/P009301/1, by European Union Horizon 2020 Research and Innovation Program Grant 780701, by Innovate UK grant AQuaSec, by NIST award 60NANB18D216 and by National Science Foundation under Grant No. 2044855. Part of this work was done while MA visited the Simons Institute for the Theory of Computing. The full version of this work is available as [5].

T. Malkin and C. Peikert (Eds.): CRYPTO 2021, LNCS 12826, pp. 732–759, 2021.
https://doi.org/10.1007/978-3-030-84245-1_25

hardness of solving SVP and in particular its applications in cryptography have led to the study of approximate variants.

For $\delta \geq 1$, the δ-approximate variant of SVP (δ-SVP) is to find a non-zero vector $\boldsymbol{v}$ in $\mathcal{L}$ such that $\|\boldsymbol{v}\| \leq \delta \cdot \lambda_1(\mathcal{L})$, where $\lambda_1(\mathcal{L}) := \min_{\boldsymbol{x} \in \mathcal{L}_{\neq \mathbf{0}}} \|\boldsymbol{x}\|$ denotes the length of the shortest nonzero vector in $\mathcal{L}$. Solving δ-SVP is also NP-hard for any $\delta \leq n^{c/\log\log n}$ with some constant $c > 0$ under reasonable complexity assumptions [30,32,37,38]. A closely related problem is *δ-Hermite SVP* (δ-HSVP), which asks to find a non-zero vector $\boldsymbol{v}$ in $\mathcal{L}$ such that $\|\boldsymbol{v}\| \leq \delta \cdot \mathrm{vol}(\mathcal{L})^{1/n}$, where $\mathrm{vol}(\mathcal{L})$ denotes the *volume* of $\mathcal{L}$. Many cryptographic primitives base their security on the worst-case hardness of δ-SVP or related lattice problems [2,27,43,46]. Security estimates of these constructions depend on solving δ-HSVP, typically for $\delta = \mathrm{poly}(n)$ [9,10]. The output quality of a δ-HSVP solver in rank n is typically assessed with the so-called *root Hermite factor* (RHF) $\delta^{1/(n-1)}$.[1]

To solve the approximate versions of SVP, the standard approach is *lattice reduction*, which finds reduced bases consisting of reasonably short and relatively orthogonal vectors. Its "modern" history began with the celebrated LLL algorithm [34] and continued with stronger blockwise algorithms [1,4,23,40,49,51]. Lattice reduction has numerous applications in mathematics, computer science and especially cryptanalysis.

Lovász [36] showed that any δ-HSVP solver in rank n can be used to efficiently solve δ^2-SVP in rank n. For random lattices $\mathcal{L}$ of rank n, the classical *Gaussian heuristic* claims $\lambda_1(\mathcal{L}) \approx \mathrm{GH}(\mathcal{L}) := \mathrm{GH}(n) \cdot \mathrm{vol}(\mathcal{L})^{1/n}$. Here, $\mathrm{GH}(n)$ denotes the radius of the unit-volume n-dimensional ball. Thus, any δ-HSVP solver in rank n for $\delta \geq \sqrt{n}$ can possibly be used to solve $(\delta/\sqrt{n})$-SVP in the same rank in practice (see [24, §3.2]).

In this work we consider the practical aspects of solving δ-HSVP using blockwise lattice reduction algorithms. The Schnorr–Euchner BKZ algorithm [51] and its modern incarnations [4,7,12,13,17] provide the best time/quality trade-off in practice. The BKZ algorithm takes a parameter k controlling its time/quality trade-off: the larger k is, the more reduced the output basis, but the running time grows at least exponentially with k. BKZ is commonly available in software libraries (such as FP(y)LLL [21,22], NTL [53] and PBKZ [12]) and has been used in many lattice record computations [7,19,48]. G6K [7,19] currently provides the fastest public BKZ implementation by replacing the enumeration-based SVP oracle in BKZ with a sieving-based oracle. As such, it achieves a running time of $2^{\Theta(k)}$ at the cost of also requiring $2^{\Theta(k)}$ memory. However, this memory requirement may prove prohibitively expensive in some settings. Moreover, in a massively parallelised computation the communication overhead required for sieving may limit its performance advantage.

In this work we reduce the performance gap between enumeration-based and sieving-based BKZ. That is, we focus on enumeration-based lattice reduction for solving δ-HSVP, i.e. the polynomial-memory regime, building on recent technical

[1] The normalisation by the $(n-1)$-th root is justified by that the algorithms considered here achieve RHFs that are bounded independently of the lattice rank n.

progress on speeding-up lattice enumeration: *relaxed pruned enumeration* [35] and *extended preprocessing* [4].

Recently, [35] heuristically justified that if relaxing the search radius by a small constant $\alpha > 1$, then enumeration with certain extreme cylinder pruning [25,52] asymptotically achieves an exponential speed-up. Intuitively, this relaxation strategy allows to upgrade the enumeration subroutine for BKZ (2.0) [17,51] with one more optional parameter α. Here and in what follows, we omit pruning parameters due to the use of FP(y)LLL's numerical `pruning` module [21,22].

Concurrently, a variant of BKZ presented in [4] can achieve RHF $\mathrm{GH}(k)^{1/(k-1)}$ in time $k^{k/8+o(k)}$, which is super-exponentially faster than the cost record $k^{k/(2e)+o(k)}$ of [29,31] for reaching the same RHF. The idea behind the BKZ variant [4] is to preprocess in a larger rank than the enumeration rank. That is, [4] upgraded the HSVP-oracle of BKZ to exact (pruned) enumeration in rank k with *extended preprocessing* in rank $\lceil (1+c)\cdot k \rceil$ for some small constant $c \geq 0$. Intuitively, this preprocessing strategy upgrades the enumeration subroutine for BKZ (2.0) [17,51] with an additional optional parameter c.

Contributions. This work investigates the impact of improved enumeration subroutines in BKZ by integrating the relaxation strategy [1,12,35] with the extended preprocessing strategy [4], i.e. we propose the use of *relaxed pruned enumeration with extended preprocessing* in BKZ.

First, in Sect. 3, we justify and empirically validate that relaxed enumeration with certain extreme cylinder pruning [25,52] asymptotically achieves better time/quality trade-offs for certain approximation regimes based on standard heuristics. More precisely, for large enough k, the resulting $\alpha \cdot \mathrm{GH}(k_\alpha)$-HSVP-oracle in rank k_α is exponentially faster than a $\mathrm{GH}(k)$-HSVP-oracle in rank k for any constant $\alpha \in (1, 2]$. Here, k_α is the smallest integer greater than k such that the corresponding RHF would not become larger after relaxation:

$$\mathrm{GH}\left(k\right)^{\frac{1}{k-1}} \geq \left(\alpha \cdot \mathrm{GH}(k_\alpha)\right)^{\frac{1}{k_\alpha - 1}}.$$

Prior work [35] only treated the speed-up of $\alpha \cdot \mathrm{GH}(k)$ compared with $\mathrm{GH}(k)$.

Second, in Sect. 4, we explore the concrete cost estimates of relaxed enumeration with FP(y)LLL's `pruning` module [21,22] with or without extended preprocessing, using simulations and experiments. We validate that with the same preprocessing in rank $\lceil (1+c)\cdot k \rceil$ for $c \in [0, 0.4]$, the resulting $\alpha \cdot \mathrm{GH}(k)$-HSVP-oracle in rank k is exponentially faster than a $\mathrm{GH}(k)$-HSVP-oracle in rank k for constants $\alpha \in (1, 1.3]$.[2]

Third, our main result is a practical BKZ variant presented in Sect. 5, which uses an $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α with preprocessing in rank $\lceil (1+c)\cdot k_\alpha \rceil$. Intuitively, it upgrades the enumeration subroutine for BKZ (2.0) [17,51] with two more optional parameters (α, c), and generalises the BKZ

[2] We also observed a small speed-up of $c = 0.15$ over $c = 0.25$ (claimed to be the "optimal" in [4]) and verified it using the original simulation code from [4] in the full version of this work.

variant in [4] with one more optional parameter α. This additional freedom results in the best current time/quality trade-off for enumeration-based BKZ implementations: our algorithm achieves RHF $\text{GH}(k)^{\frac{1}{k-1}}$ in time $\approx 2^{\frac{k \log k}{8} - 0.654\,k + 25.84}$. This improves on the cost record $2^{\frac{k \log k}{8} - 0.547\,k + 10.4}$ given in [4]. As a side result, by setting $c = 0$ (i.e. without extended preprocessing), our algorithm achieves RHF $\text{GH}(k)^{\frac{1}{k-1}}$ in time $\approx 2^{\frac{k \log k}{2\,e} - 1.077\,k + 29.12}$, which also improves on the cost for BKZ 2.0 [17] reported in [4]: $2^{\frac{k \log k}{2\,e} - 0.995\,k + 16.25}$. A comparison between our results and those reported in [4] is given in Fig. 1: it illustrates that our BKZ variant is exponentially faster than previous BKZ variants in the polynomial-memory setting. Comparing our best fit with the results reported in [4], we obtain a crossover rank of 145, or approximately 2^{61} operations.[3]

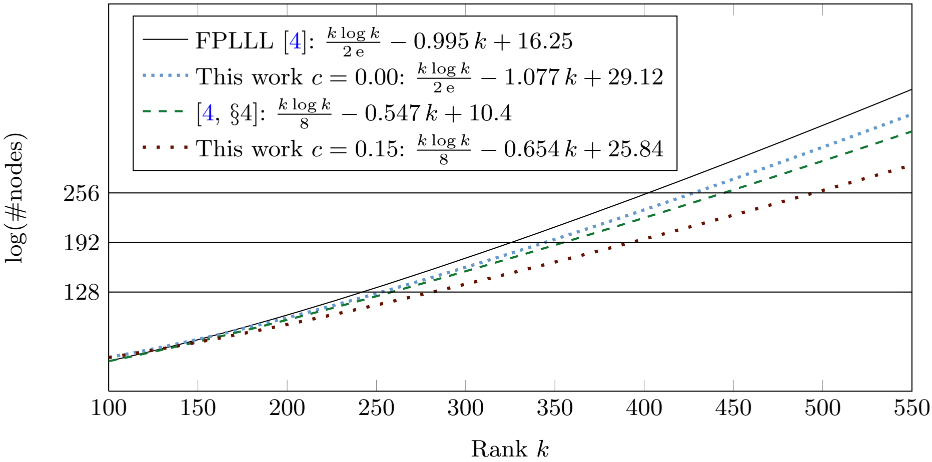


Fig. 1. Cost comparison.

Since our results critically depend on our simulation and implementation results, we provide the complete source code (used to produce our simulation data and experimental verification) with the full version of this work.

Impact on security estimates. Security estimates for lattice-based cryptographic primitives typically rely upon sieving algorithms [6]. In the classical (i.e. non-quantum) setting this is backed by both the asymptotic [14] and concrete [7,19] performance of sieving algorithms. Our results do not affect this

[3] To put this into perspective, [55] reports solving 1.05-HSVP in rank 150 using a distributed implementation of an enumeration algorithm. As a result, we expect the speedups demonstrated in this work to be practical.

state of the art.[4] As can be gleaned from Fig. 1, all known enumeration-based algorithms, including those based on the strategies in this work, perform similarly up to rank $k \approx 100$. On the other hand, G6K [7] outperforms FPLLL's implementation of enumeration for ranks $\gtrsim 70$.

In the quantum setting the situation is considerably more complicated. Quantum enumeration algorithms asymptotically produce a quadratic speed-up over classical enumeration algorithms [11] in the "query model", but each such queries may have significant (polynomial) cost, implying that such an estimate is likely a significant underestimate of the true cost. On the other hand, quantum sieving improves the cost from $2^{0.292\,k+o(k)}$ to $2^{0.265\,k+o(k)}$ [33], assuming no depth restriction on quantum computation. In [8] some quantum resource estimates are given for the dominant part of various lattice sieving algorithms. These costs, however, are derived assuming unit cost for accessing quantum accessible RAM, an optimistic assumption. Overall, given the lack of clarity on the cost of the two families of algorithms under consideration in a quantum setting, it is currently not possible to assess the crossover rank when quantum lattice sieving outperforms quantum lattice-point enumeration. This suggests an analogous investigation to [8] for quantum enumeration as a pressing research question.

Faced with the difficulty of assessing the cost of quantum algorithms, the literature routinely relies on rough low bounds to estimate the cost of lattice reduction, see e.g. [15,26,45].[5] In particular, the quantum version of the Core-SVP methodology [10] assigns a cost of $2^{0.265\,k}$ to performing lattice reduction with RHF $\mathrm{GH}(k)^{1/(k-1)}$. Now, comparing this figure with a naive square-root of our enumeration costs would give a crossover rank of $k = 547$. Yet, even then, i.e. even presuming the square-root advantage applies as is to our algorithm including preprocessing, accepting the assumptions of suppressing (potentially significant) polynomial factors, no depth restriction on quantum computation and unit-cost qRAM, this would not imply a downward correction of Category 1 NIST PQC Round 3 submission parameters and similar parameters for lattice-based schemes. That is, we stress that this work does *not* invalidate the claimed NIST Security Level of such submissions. This is because a given security level is defined by both a classical and a quantum cost: roughly 2^{λ} classically and $2^{\lambda/2}$ quantumly. For example, for Level 1 this is the cost of classically and quantumly breaking AES-128. Submissions targeting a classical security level 2^{λ} relying on the cost of classical sieving $2^{0.292\,k+o(k)}$ have a quantum security level much higher than $2^{\lambda/2}$ under the $2^{0.265\,k}$ cost model. In other words, this work does not lower the cost of quantum enumeration sufficiently to invalidate NIST Security Level claims since known quantum algorithms provide only a minor speed-up in the chosen cost model over classical algorithms when compared to Grover's algorithm for, say, AES.

[4] We discuss the (apparent lack of) applicability of our approach to the sieving setting in the full version of this work.

[5] This does not imply, though, that those works endorse this mode of comparison, e.g. [15] explicates its objections to it.

2 Background

Notation. To be compatible with software implementations such as FP(y)LLL, we let matrix indices start with 0 and use row-representation for both vectors and matrices in this work. Bold lower-case and upper-case letters denote row vectors and matrices respectively. The set of $n \times m$ matrices with coefficients in the ring $\mathbb{A}$ is denoted by $\mathbb{A}^{n\times m}$, and we identify $\mathbb{A}^m$ with $\mathbb{A}^{1\times m}$. The notations $\log(\cdot)$ and $\ln(\cdot)$ stand for the base 2 and natural logarithms respectively.

2.1 Lattices

Orthogonalisation. Let $\boldsymbol{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1}) \in \mathbb{R}^{n\times m}$ be a basis of a lattice $\mathcal{L}$. Lattice algorithms often involve the orthogonal projections $\pi_i : \mathbb{R}^m \mapsto \operatorname{span}(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{i-1})^{\perp}$ for $i = 0, \ldots, n-1$. The *Gram–Schmidt orthogonalisation* (GSO) of $\boldsymbol{B}$ is $\boldsymbol{B}^* = (\boldsymbol{b}_0^*, \ldots, \boldsymbol{b}_{n-1}^*)$, where the Gram–Schmidt vector $\boldsymbol{b}_i^*$ is $\pi_i(\boldsymbol{b}_i)$. Then $\boldsymbol{b}_0^* = \boldsymbol{b}_0$ and $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=0}^{i-1} \mu_{i,j} \cdot \boldsymbol{b}_j^*$ for $i = 1, \ldots, n-1$, where $\mu_{i,j} = \frac{\langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle}{\langle \boldsymbol{b}_j^*, \boldsymbol{b}_j^* \rangle}$. The projected block $(\pi_i(\boldsymbol{b}_i), \pi_i(\boldsymbol{b}_{i+1}), \ldots, \pi_i(\boldsymbol{b}_{j-1}))$ is denoted by $\boldsymbol{B}_{[i,j)}$. Then the volume of the parallelepiped generated by $\boldsymbol{B}_{[i,j)}$ is $\operatorname{vol}(\boldsymbol{B}_{[i,j)}) = \prod_{k=i}^{j-1} \|\boldsymbol{b}_k^*\|$. In particular, $\boldsymbol{B}_{[0,j)} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{j-1})$ and $\operatorname{vol}(\mathcal{L}) = \operatorname{vol}(\boldsymbol{B}) = \prod_{k=0}^{n-1} \|\boldsymbol{b}_k^*\|$.

Hermite's constant. *Hermite's constant* of dimension n is the maximum $\gamma_n = \max\left(\lambda_1(\mathcal{L})/\operatorname{vol}(\mathcal{L})^{1/n}\right)^2$ over all n-rank lattices $\mathcal{L}$, where $\lambda_1(\mathcal{L}) = \min_{\boldsymbol{v}\in\mathcal{L}\setminus\{\boldsymbol{0}\}} \|\boldsymbol{v}\|$ is the *first minimum* of $\mathcal{L}$. The best asymptotical bounds known are [18,41]: $\frac{n}{2\pi e} + \frac{\log(\pi n)}{2\pi e} \leq \gamma_n \leq \frac{1.744n}{2\pi e} + o(n)$.

Lattice reduction. Let $\boldsymbol{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ be a basis of a lattice $\mathcal{L}$.

$\boldsymbol{B}$ is *size-reduced* if $|\mu_{i,j}| \leq \frac{1}{2}$ for all $0 \leq j < i < n$. $\boldsymbol{B}$ is *LLL-reduced* [34] if it is size-reduced and every 2-rank projected block $\boldsymbol{B}_{[i,i+2)}$ satisfies Lovász's condition: $\frac{3}{4} \cdot \|\boldsymbol{b}_i^*\|^2 \leq \|\mu_{i+1,i} \cdot \boldsymbol{b}_i^* + \boldsymbol{b}_{i+1}^*\|^2$ for $0 \leq i \leq n-2$. In practice, the parameter $\frac{3}{4}$ can be replaced with any constant in the interval $(\frac{1}{4}, 1)$.

$\boldsymbol{B}$ is *SVP-reduced* if $\|\boldsymbol{b}_0\| = \lambda_1(\mathcal{L})$. There are two relaxations with $\delta \geq 1$: $\boldsymbol{B}$ is *δ-SVP-reduced* if $\|\boldsymbol{b}_0\| \leq \delta \cdot \lambda_1(\mathcal{L})$; $\boldsymbol{B}$ is *δ-HSVP-reduced* if $\|\boldsymbol{b}_0\| \leq \delta \cdot \operatorname{vol}(\mathcal{L})^{1/n}$.

$\boldsymbol{B}$ is *HKZ-reduced* if it is size-reduced and $\boldsymbol{B}_{[i,n)}$ is SVP-reduced for $i = 0, \ldots, n-1$; $\boldsymbol{B}$ is *k-BKZ-reduced* [49] if it is size-reduced and $\boldsymbol{B}_{[i,\min\{i+k,n\})}$ is SVP-reduced for $i = 0, \ldots, n-1$.

Primitive vector. Let $\mathcal{L}$ be a lattice with basis $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$. A vector $\boldsymbol{b} = \sum_{i=0}^{n-1} x_i \boldsymbol{b}_i \in \mathcal{L}$ with $x_i \in \mathbb{Z}$ is *primitive* for $\mathcal{L}$ iff it can be extended to a basis of $\mathcal{L}$, or equivalently, $\gcd(x_0, \ldots, x_{n-1}) = 1$ [54, Theorem 32].

HSVP-oracle and RHF. A δ-HSVP-oracle with factor $\delta > 0$ is any algorithm which, given as input an n-rank lattice $\mathcal{L}$ specified by a basis, outputs a primitive vector $\boldsymbol{v}$ in $\mathcal{L}$ such that $\|\mathbf{v}\| \leq \delta \cdot \mathrm{vol}(\mathcal{L})^{1/n}$. The resulting *root-Hermite-factor* (RHF) is $\left(\frac{\|\mathbf{v}\|}{\mathrm{vol}(\mathcal{L})^{1/n}}\right)^{1/(n-1)}$, which is less than $\delta^{1/(n-1)}$. In other words, the worst-case RHF of this δ-HSVP-oracle on an n-rank lattice is $\delta^{1/(n-1)}$. For instance, any exact SVP-solver working on an n-rank lattice is a $\sqrt{\gamma_n}$-HSVP-oracle, whose corresponding worse-case RHF is $\gamma_n^{\frac{1}{2(n-1)}}$.

Geometric Series Assumption. Let $\boldsymbol{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ be a basis. Schnorr's *Geometric Series Assumption* (GSA) [50] says that $\boldsymbol{B}$ follows the GSA wrt. some constant $r \in [3/4, 1)$ (depending on the reduction algorithm) if its Gram–Schmidt lengths decay geometrically wrt. r, namely $\|\boldsymbol{b}_{i+1}^*\|/\|\boldsymbol{b}_i^*\| = r$ for all $i = 0, \ldots, n-2$. In practice, it has been observed that a reduced basis produced by the LLL algorithm [34] satisfies the GSA in an approximate sense when the input basis is sufficiently randomised.

Gaussian heuristic. Given a full-rank lattice $\mathcal{L}$ in $\mathbb{R}^n$ and a measurable set $S \subseteq \mathbb{R}^n$, the cardinality of $S \cap \mathcal{L}$ is approximately $\mathrm{vol}(S)/\mathrm{vol}(\mathcal{L})$. Under the heuristic, there are about α^n points in $\mathcal{L}$ of norm $\leq \alpha \cdot \mathrm{GH}(\mathcal{L})$, and one would expect $\lambda_1(\mathcal{L})$ to be close to $\mathrm{GH}(\mathcal{L})$. Here, $\mathrm{GH}(\mathcal{L}) := \mathrm{GH}(n) \cdot \mathrm{vol}(\mathcal{L})^{1/n}$ with

$$\mathrm{GH}(n) := \frac{\varGamma(n/2+1)^{1/n}}{\sqrt{\pi}} \approx \sqrt{\frac{n}{2\pi e}} \cdot (\pi n)^{\frac{1}{2n}}$$

by Stirling's formula. In fact, for a random lattice $\mathcal{L}$, $\lambda_1(\mathcal{L})$ is close to $\mathrm{GH}(\mathcal{L})$ with high probability [47]; for any lattice $\mathcal{L}$ of rank $n > 24$, it follows from Blichfeldt's inequality $\gamma_n \leq 2 \cdot \mathrm{GH}(n)^2$ [16] that $\lambda_1(\mathcal{L}) \leq \sqrt{2} \cdot \mathrm{GH}(\mathcal{L})$.

2.2 Enumeration: Pruning Plus Relaxation

Enumeration [4,20,31,39,44,51] is the simplest algorithm for solving SVP and requires only polynomial memory: given a full-rank lattice $\mathcal{L}$ in $\mathbb{R}^n$ and a radius $R > 0$, enumeration outputs $\mathcal{L} \bigcap \mathrm{Ball}_n(R)$ by a depth-first tree search. If $R \geq \lambda_1(\mathcal{L})$, then it is trivial to extract a nonzero lattice vector of length $\leq R$: moreover, by comparing all the norms of vectors in $\mathcal{L} \bigcap \mathrm{Ball}_n(R)$, one can find a shortest nonzero lattice vector.

Cylinder pruning [25,52] speeds up enumeration by replacing the search region $\mathrm{Ball}_n(R)$ with a (much smaller) subset $\mathrm{P}_f(\boldsymbol{B}, R)$ defined by a bounding function $f : \{1, \ldots, n\} \to [0, 1]$, a basis $\boldsymbol{B}$ of $\mathcal{L}$ and R:

$$\mathrm{P}_f(\boldsymbol{B}, R) = \{\boldsymbol{x} \in \mathbb{R}^n : \|\pi_{n-k}(\boldsymbol{x})\| \leq f(k) \cdot R \text{ for all } 1 \leq k \leq n\} \subseteq \mathrm{Ball}_n(R).$$

Algorithm 1 recalls enumeration with extreme cylinder pruning, which repeats enumeration with cylinder pruning many times over different subsets $\mathrm{P}_f(\boldsymbol{B}, R)$ by randomising $\boldsymbol{B}$. Here, each Step 3 is a single cylinder pruning.

Algorithm 1. Extreme cylinder pruning [25, Algorithm 1]

Require: $(\mathcal{L}, R, f)$, where $\mathcal{L}$ is a full-rank lattice in $\mathbb{R}^n$ specified by a basis, $R > 0$ is a radius and f is a bounding function.
Ensure: A nonzero vector in $\mathcal{L} \bigcap \mathrm{Ball}_n(R)$.
1: **WHILE** no nonzero vector in $\mathcal{L} \bigcap \mathrm{Ball}_n(R)$ has been found:
2: Compute a (randomised) reduced basis $\boldsymbol{B}$ by applying basis reduction to a "random" basis of $\mathcal{L}$.
3: Compute $\mathcal{L} \bigcap \mathrm{P}_f(\boldsymbol{B}, R)$ by enumeration with cylinder pruning

The use of enumeration with extreme cylinder pruning in blockwise lattice reduction requires finding just one nonzero point in $\mathcal{L} \bigcap \mathrm{P}_f(\boldsymbol{B}, R)$ for some basis $\boldsymbol{B}$ produced at Step 2: it allows to suitably relax radius R for speedup, which was already exploited in solving SVP challenges [48].

Recently, Li and Nguyen [35] clarified the heuristic asymptotic speedup achieved by enumeration with relaxed radius and with certain extreme cylinder pruning. It uses the following two heuristic assumptions as in [25]:

Heuristic 1 *The cost of Algorithm 1 is dominated by enumeration with cylinder pruning at Step 3, rather than the repeated reductions of Step 2.*

Heuristic 2 *All the reduced bases $\boldsymbol{B}$ of Algorithm 1 follow the GSA wrt. the same positive constant.*

Theorem 1 ([35, Theorem 6]). *Let $\mathcal{L}$ be a full-rank lattice in $\mathbb{R}^n$. Let $\alpha \geq 1$ and $\rho \in (0, \frac{1}{2})$ such that $4\,\alpha^4 \cdot \rho \cdot (1-\rho) < 1$. Let $R = \alpha \cdot \mathrm{GH}(\mathcal{L})$ and*

$$f(i) = \begin{cases} \sqrt{\rho} & \text{if } 1 \leq i \leq n/2, \\ 1 & \text{otherwise.} \end{cases}$$

Under Heuristics 1 and 2, the time complexity $T_{\alpha,\rho}(n)$ of Algorithm 1 on $(\mathcal{L}, R, f)$ equals, up to polynomial factors, $T(n)$ of a full enumeration on $\mathcal{L} \bigcap \mathit{Ball}_n(\mathrm{GH}(\mathcal{L}))$ reduced by a multiplicative factor $(4\alpha^2(1-\rho))^{n/4}$:

$$T_{\alpha,\rho}(n) \approx \frac{T(n)}{(4\alpha^2(1-\rho))^{n/4}}.$$

Here (and for the remainder of this work) the cost of enumeration is expressed as the number of nodes visited during the enumeration process.

2.3 Schnorr–Euchner's BKZ and its Accelerated Variant in [4]

BKZ. The (original) BKZ algorithm introduced by Schnorr and Euchner [51] is the most widely used lattice reduction algorithm besides LLL [34] and a central tool in lattice-based cryptanalysis. Its performance drives the setting of concrete parameters (such as keysizes) for concrete lattice-based cryptographic primitives (see e.g. [6]).

Originally, the SVP subroutine implemented in [51] was the simplest form of lattice enumeration, but it is now replaced by better subroutines, such as pruned enumeration [25] in BKZ 2.0 [17] and FP(y)LLL [21,22] and (asymptotically) faster sieving in the General Sieve Kernel [7,19]. In practice, BKZ is typically implemented with an approximate (rather than exact) SVP-subroutine. Thus, Algorithm 2 slightly generalises BKZ by allowing the use of a relaxed HSVP-oracle at Step 3, as well as full LLL (instead of partial LLL) at Step 5: both are justified by Li–Nguyen's analysis [35].

At a high level, Algorithm 2 reduces a basis in high rank, using HSVP-oracles in low rank ($\leq k$) as subroutines and running the LLL algorithm [34] to remove the linear dependency right after inserting a lattice vector (found by the oracle) in the current basis.

Algorithm 2. BKZ: Schnorr–Euchner's BKZ algorithm [51]

Require: A block size $k \in (2, n)$, the number of tours $N \in \mathbb{Z}^+$, a relaxation factor $\alpha \geq 1$, and an LLL-reduced basis $\boldsymbol{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ of a lattice $\mathcal{L} \subseteq \mathbb{Z}^m$.
Ensure: A new basis of $\mathcal{L}$.
1: **for** $\ell = 0$ to $N - 1$ **do**
2:     **for** $j = 0$ to $n - 2$ **do**
3:         Find a primitive vector $\boldsymbol{b}$ for the sublattice generated by the basis vectors $\boldsymbol{b}_j, \ldots, \boldsymbol{b}_{h-1}$ where $h = \min\{j + k, n\}$ s.t. $\|\pi_j(\boldsymbol{b})\| \leq \alpha \sqrt{\gamma_{h-j}} \cdot \mathrm{vol}(\boldsymbol{B}_{[j,h)})^{1/(h-j)}$
4:         **if** $\|\boldsymbol{b}_j^*\| > \|\pi_j(\boldsymbol{b})\|$ **then**
5:             LLL-reduce $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{j-1}, \boldsymbol{b}, \boldsymbol{b}_j, \ldots, \boldsymbol{b}_{n-1})$ to remove linear dependencies
6:         **end if**
7:     **end for** //A BKZ tour refers to a single execution of Steps 2-7.
8: **end for**
9: **return** $\boldsymbol{B}$.

Building on Hanrot–Pujol–Stehlé's analysis of a certain BKZ variant (removing internal LLL calls) [28], Li and Nguyen [35] justified the popular "early termination" strategy in practice of BKZ:

Theorem 2 **([35, Theorem 2]).** *Let $n > k \geq 2$ be integers and let $0 < \varepsilon \leq 1 \leq \alpha \leq \frac{2^{(k-1)/4}}{\sqrt{\gamma_k}}$. Given as input a block size k, a relaxation factor α, and an LLL-reduced basis of an n-rank lattice $\mathcal{L} \subset \mathbb{R}^m$, if $N \geq 4(\ln 2)\frac{n^2}{k^2} \log \frac{n^{1.5}}{(4\sqrt{3})\varepsilon}$, then Algorithm 2 outputs a basis $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ of $\mathcal{L}$ such that*

$$\|\boldsymbol{b}_0\| \leq (1 + \varepsilon) \cdot (\alpha^2 \gamma_k)^{\frac{n-1}{2(k-1)} + \frac{k \cdot (k-2)}{2n \cdot (k-1)}} \cdot \mathrm{vol}(\mathcal{L})^{1/n}.$$

It was also mentioned in [35] that for $n > k > 8e\pi$, there is a k-BKZ reduced basis $\boldsymbol{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ satisfying $\|\boldsymbol{b}_0\| = \left(\frac{k-1}{8e\pi}\right)^{\frac{n-1}{2k}} \cdot \mathrm{vol}(\boldsymbol{B})^{1/n}$. Since $\gamma_k = \Theta(k)$, this means that BKZ with early termination indeed provides bases almost as reduced as the full BKZ algorithm. Theorem 2 has a heuristic version (i.e. [35, Th. 5]), which heuristically models the practical behaviour of BKZ.

The accelerated BKZ variant in [4]**.** Recently, in [4] a practical and faster BKZ variant within the class of polynomial-space algorithms was introduced, based on the idea that its HSVP-oracle performs an exact enumeration with *extended preprocessing*.

Extended preprocessing is to preprocess in a larger rank than the enumeration rank. Exact enumeration with extended preprocessing refers to the procedure that the $\delta(k)$-HSVP-oracle in "block size" $\lceil(1+c)\cdot k\rceil$ (for some small constant $c \geq 0$ and an integer $k \geq 2$) first preprocesses a given projected block of rank $\lceil(1+c)\cdot k\rceil$ (using this BKZ variant recursively in lower levels) into a reduced block (say,) $\boldsymbol{C}$ and then performs a (pruned) enumeration for solving SVP exactly on the k-rank head block of $\boldsymbol{C}$ to find a short nonzero vector $\boldsymbol{v} \in \mathcal{L}(\boldsymbol{C})$.

The performance parameter k dominates the time/quality trade-off:

- Quality aspect: $\boldsymbol{v}$ is a shortest nonzero vector in the lattice generated by the k-rank head block $\boldsymbol{C}_{[0,k)}$ of $\boldsymbol{C}$, so that $\|\boldsymbol{v}\| \leq \sqrt{\gamma_k}\cdot \mathrm{vol}(\boldsymbol{C}_{[0,k)})^{1/k}$. The BKZ-preprocessing on $\boldsymbol{C}$ ensures that $\mathrm{vol}(\boldsymbol{C}_{[0,k)})/\mathrm{vol}(\boldsymbol{C})^{k/\lceil(1+c)k\rceil}$ can be upper bounded well, so that $\|\boldsymbol{v}\| \leq \delta(k)\cdot \mathrm{vol}(\boldsymbol{C})^{1/\lceil(1+c)k\rceil}$.
- Cost aspect: Due to the extended preprocessing on $\boldsymbol{C}$, the k-rank head block $\boldsymbol{C}_{[0,k)}$ has good quality for enumeration, i.e. $\boldsymbol{C}_{[0,k)}$ almost satisfies the GSA. As a result, enumeration on $\boldsymbol{C}_{[0,k)}$ costs at most $k^{k/8}\cdot 2^{O(k)}$ (matching the Gaussian heuristic estimate under the GSA). Both the GSA shape and the cost estimate were validated by [4]'s simulations and experiments.

We revisit [4, § 4]'s BKZ variant in Algorithms 3 and 4. We refer the reader to [4] for definitions of the functions tail() and pre() called in Algorithm 4.

When $c = 0$, Algorithm 3 is essentially Schnorr-Euchner's BKZ algorithm [51] (i.e. using enumeration but with recursive BKZ preprocessing as an SVP-oracle).

Algorithm 3. BKZ variant in [4, Algorithm 4]

Require: $(\boldsymbol{B}, k, c)$, where $\boldsymbol{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ is an LLL-reduced basis of an n-rank lattice $\mathcal{L}$ in $\mathbb{Z}^m$, $k \in [2, n)$ is a performance parameter, $c \geq 0$ is an overshooting parameter and $N \in \mathbb{Z}^+$ is the number of tours.
Ensure: A reduced basis of $\mathcal{L}$.
for $\ell = 0$ to $N-1$ **do**
  for $j = 0$ to $n-2$ **do**
    Find a short nonzero vector $\boldsymbol{v}$ in the lattice $\mathcal{L}_{[j,h)}$ (generated by the projected block $\boldsymbol{B}_{[j,h)}$ where $h = \min\{j + \lceil(1+c)k\rceil, n\}$), by calling Alg. 4 on $(\boldsymbol{B}_{[j,h)}, k, c)$
    if $\|\boldsymbol{b}_j^*\| > \|\boldsymbol{v}\|$ **then**
      Lift $\boldsymbol{v}$ into a primitive vector $\boldsymbol{b}$ for the sublattice generated by the basis vectors $\boldsymbol{b}_j, \ldots, \boldsymbol{b}_{h-1}$ such that $\|\pi_j(\boldsymbol{b})\| \leq \|\boldsymbol{v}\|$
      LLL-reduce $(\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{j-1}, \boldsymbol{b}, \boldsymbol{b}_j, \ldots, \boldsymbol{b}_{n-1})$ to remove linear dependencies
    end if
  end for
end for
return $\boldsymbol{B}$.

Without formal analysis but with concrete simulations and experiments, [4] reported that the following instantiation of Algorithm 3 seems to provide the best practical performance: $(c, N) = (0.25, 4)$ and Algorithm 4 performing pruned enumeration at both Step 4 and Step 8. The resulting procedure achieves RHF $\approx \mathrm{GH}(k)^{1/(k-1)}$ in time $\approx 2^{\frac{k \log k}{8} - 0.547\,k + 10.4}$, at least up to $k \approx 500$.

2.4 Simulating BKZ

To understand the behaviour of lattice reduction algorithms, a useful approach is to conduct simulations. The underlying idea is to model the practical behaviour of the evolution of the Gram–Schmidt norms during the algorithm execution, without running a costly lattice reduction algorithm. Note that this requires only the Gram–Schmidt norms rather than the basis itself. Chen and Nguyen first provided a BKZ simulator [17] based on the Gaussian heuristic and with an experiment-driven modification for the tail blocks of the basis. It relies on the assumption that each SVP solver on the projected blocks (except the tail ones of the basis) finds a vector whose norm corresponds to the Gaussian heuristic applied to that local block.

We extend/adapt this simulator to also estimate the cost and not only the evolution of the Gram–Schmidt norms. To find the enumeration cost with pruning, we make use of FPyLLL's `pruning` module which numerically optimises pruning parameters for a time/success probability trade-off using a gradient descent. In small block sizes, the enumeration cost is dominated by calls to LLL. In our code, we simply assume that one LLL call in rank k costs the equivalent of visiting k^3 enumeration nodes. While this is clearly not the cost of LLL [42], this choice produces costs that match the observed running times (see e.g. Fig. 4) closest among the choices we experimented with. We hypothesise that this behaviour can be explained by that the basis vectors $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{j-1}, \boldsymbol{b}_j, \ldots, \boldsymbol{b}_{n-1}$ appearing at, say, Step 6 of Algorithm 3 are already (better than) LLL-reduced. This assumption enables us to bootstrap our cost estimates. BKZ in block size up to (say,) 40 only requires LLL preprocessing, allowing us to estimate the cost of preprocessing with block size up to 40, which in turn enables us to estimate the cost (including preprocessing) for larger block sizes etc. Our simulation source

---

Algorithm 4. An approx-HSVP oracle on $(\boldsymbol{B}_{[j,h)}, k, c)$ using exact enumeration in rank k^* with extended preprocessing in rank $(h-j)$ [4, Algorithm 3]

1: Find the enumeration rank $k^* \leftarrow \mathrm{tail}(k, c, h-j)$
2: Numerically find the preprocessing parameter $k' \leftarrow \mathrm{pre}(k^*, \|\boldsymbol{b}_j^*\|, \ldots, \|\boldsymbol{b}_{h-1}^*\|)$
3: **if** $k' \geq 3$ **then**
4:     Run Alg. 3 on $(\boldsymbol{B}_{[j,h)}, k', c)$ to obtain a reduced basis $\boldsymbol{C} \in \mathbb{Q}^{(h-j)\times m}$ of $\mathcal{L}_{[j,h)}$
5: **else**
6:     LLL-reduce $\boldsymbol{B}_{[j,h)}$ into a basis $\boldsymbol{C} \in \mathbb{Q}^{(h-j)\times m}$ of $\mathcal{L}_{[j,h)}$
7: **end if** //Steps 3-7 preprocess $\boldsymbol{B}_{[j,h)}$ for the next local enumeration
8: Enumerate on the head block $\boldsymbol{C}_{[0,k^*)}$ of $\boldsymbol{C}$ to find a shortest nonzero vector $\boldsymbol{v}$ in the lattice generated by $\boldsymbol{C}_{[0,k^*)}$

---

code is available as `simu.py`, as an attachment to the electronic version of the full version of this document.

3 Asymptotic Time/Quality Trade-Offs

In this section, we show asymptotically that relaxed (rather than exact) enumeration with certain extreme cylinder pruning does achieve better time/quality trade-offs for certain approximation regimes, especially for small enough RHFs.

3.1 An Elementary Lemma

We will use the following notation for the remainder of this work:

- δ-HSVP enumeration oracle: it denotes a δ-HSVP-solver using (relaxed) enumeration with (extreme) pruning, i.e. setting the radius $R = \delta \cdot \mathrm{vol}(\mathcal{L})^{1/n}$ for enumeration on a given n-rank lattice $\mathcal{L}$.
- k_α: for real $\alpha \geq 1$ and integer $k \geq 36$, let k_α be the smallest integer greater than k such that
$$\mathrm{GH}\,(k)^{\frac{1}{k-1}} \geq (\alpha \cdot \mathrm{GH}(k_\alpha))^{\frac{1}{k_\alpha - 1}}. \tag{1}$$

The integer k_α is well-defined, due to the following fact:

Fact 3. *With the definition* $\mathrm{GH}\,(i) = \frac{\Gamma(i/2+1)^{1/i}}{\sqrt{\pi}}$, $\mathrm{GH}\,(i)^{\frac{1}{i-1}}$ *strictly decreases for integers* $i \geq 36$.

Our following analysis relies on a key observation that the ratio $\frac{k_\alpha}{k}$ "almost" decreases for $k \geq \lceil 2\pi e^2 \rceil = 47$ and tends to 1 as k tends to infinity. More precisely, we will use the following key elementary lemma:

Lemma 1. *Let* $\alpha \geq 1$ *be a real and* $k \geq 36$ *be an integer.*

1. *Monotonicity: For any fixed* k, k_α *increases with* $\alpha \geq 1$.
2. *Lower bound:* $k_\alpha \geq k + \frac{k \log \alpha}{\log k}$.
3. *Upper bound: If* $k \geq (2\pi e^2)^{\frac{\eta}{\eta-2}}$ *for some variable* $\eta > 2$*, then*
$$k_\alpha \leq k + \left\lceil \frac{\eta\, k \log \alpha}{\log k} \right\rceil.$$

The proofs of Fact 3 and Lemma 1 can be found in the full version of this work.

Lemma 1 indicates that asymptotically for a fixed constant α, the larger the integer k, the smaller we can assign the variable η in Item 3, then the smaller both the upper bound $1 + \frac{\eta \log \alpha}{\log k} + \frac{1}{k}$ and the lower bound $1 + \frac{\log \alpha}{\log k}$ of the ratio $\frac{k_\alpha}{k}$. Figure 2 verifies this numerically for several values of α and k.

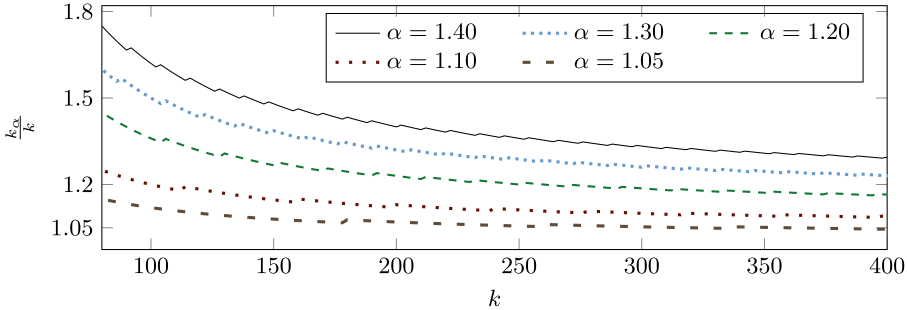


Fig. 2. Evolution of the ratio $\frac{k_\alpha}{k}$ wrt. constant $\alpha \in \{1.05, 1.1, 1.2, 1.3, 1.4\}$ and integer $k = 80, \ldots, 400$.

3.2 Asymptotic Time/Quality Trade-Offs

Theorem 1 implies that with certain extreme cylinder pruning, relaxing enumeration would result in an exponential speedup, with a minor loss in the approximation factor:

Corollary 1. *Let $\mathcal{L}$ be a full-rank lattice in $\mathbb{R}^n$. Let $\alpha \geq 1$ and $\rho \in (0, \frac{1}{2})$ such that $4\alpha^4\rho(1-\rho) < 1$. Let $R = \mathrm{GH}(\mathcal{L})$, $R_\alpha = \alpha \cdot \mathrm{GH}(\mathcal{L})$ and*

$$f(i) = \begin{cases} \sqrt{\rho} & \text{if } 1 \leq i \leq n/2, \\ 1 & \text{otherwise.} \end{cases}$$

Under Heuristics 1 and 2, the heuristic time complexity of Algorithm 1 with radius R_α is less than that of Algorithm 1 with radius R by a multiplicative factor $\alpha^{n/2}$ (up to some polynomial factor).

Proof. Let $T(n)$ denote the standard heuristic estimate for the cost of full enumeration on $\mathcal{L} \bigcap \mathrm{Ball}_n(\mathrm{GH}(\mathcal{L}))$. It follows from Theorem 1 that the heuristic cost estimates of Algorithm 1 with radius R_α and with radius R are respectively

$$\frac{T(n)}{(4\alpha^2(1-\rho))^{n/4}} \quad \text{and} \quad \frac{T(n)}{(4(1-\rho))^{n/4}}$$

up to some polynomial factors. This implies the conclusion. □

The corollary indicates that, in the same extreme pruning regime (i.e. with the same bounding function f), if one is interested in finding just one short nonzero vector (rather than one shortest nonzero vector) for a given lattice, then it is faster to run a relaxed (rather than exact) enumeration.

A more interesting question is whether such benefits can be carried over without sacrificing the quality. Thus what remains to be established is how the cost gain compares to the corresponding quality loss. For instance, we take $k = 50$ and $\alpha = 2$. For reaching the same RHF $\mathrm{GH}(50)^{\frac{1}{49}} \approx 1.012$, it is unlikely that the

$(2 \cdot \mathrm{GH}(152))$-HSVP enumeration oracle in rank 152 is faster than the GH(50)-HSVP enumeration oracle in rank 50. Thus, we now clarify that asymptotically relaxed (rather than exact) enumeration with certain extreme cylinder pruning does achieve better time/quality trade-offs for certain approximation regimes, especially for small enough RHFs. To do so, we compare costs of δ-HSVP enumeration oracles with different factors δ aiming for the same output quality.

More precisely, Lemma 1 allows us to prove that for reaching the same RHF $\mathrm{GH}(k)^{\frac{1}{k-1}}$, the $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α is exponentially faster than the $\mathrm{GH}(k)$-HSVP enumeration oracle in rank k, provided that k is sufficiently large and $\alpha > 1$ is reasonably small.

Theorem 4. *Let $\alpha > 1$ and $\rho \in (0, \frac{1}{2})$ be constants such that $4\,\alpha^4\,\rho \cdot (1-\rho) < 1$. Let*

$$f(i) = \begin{cases} \sqrt{\rho} \text{ if } 1 \le i \le n/2, \\ 1 \quad \text{otherwise.} \end{cases}$$

In addition to Heuristics 1 and 2, assume that up to some polynomial factor, the heuristic runtime of full enumeration on any n-rank integer lattice with radius equal to the Gaussian heuristic is $T(n) := n^{c_0 n} \cdot 2^{c_1 n}$ with constant coefficients c_0, c_1 such that $0 < c_0 < \frac{1}{4}$. Let k be an arbitrary positive integer satisfying $k > \max\left\{(2\pi e^2)^{\frac{1}{1-4c_0}}, 2^{-\frac{c_1}{c_0}}\right\}$. For any real $\eta \in [\frac{2\ln k}{\ln k - \ln(2\pi e^2)}, \frac{1}{2c_0})$, if $1 < \alpha \le (k^{c_0} \cdot 2^{c_1})^2$, then the $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α (using Algorithm 1) is exponentially faster than the $\mathrm{GH}(k)$-HSVP enumeration oracle in rank k (using Algorithm 1) by a multiplicative factor of at least

$$\alpha^{\left(\frac{1}{2}-c_0\eta\right)k} \cdot \left(4(1-\rho)\left(\frac{\sqrt{\alpha}}{(2e)^{c_0}\,2^{c_1}}\right)^{4\eta}\right)^{\frac{k\log\alpha}{4\log k}} \quad \text{(up to some polynomial factor).}$$

Proof. We omit some polynomial factors in the following complexity analysis. By the assumption, it follows from Theorem 1 that the heuristic runtime of the $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α and the $\mathrm{GH}(k)$-HSVP enumeration oracle in rank k are respectively

$$\begin{aligned} T_\alpha \approx \frac{T(k_\alpha)}{(4\alpha^2(1-\rho))^{k_\alpha/4}} &= k_\alpha^{c_0 k_\alpha} \cdot 2^{c_1 k_\alpha} \cdot \alpha^{-k_\alpha/2} \cdot (4(1-\rho))^{-k_\alpha/4} \\ &= 2^{\left(c_0 \log k_\alpha + c_1 - \frac{\log\alpha}{2}\right)k_\alpha} \cdot (4(1-\rho))^{-k_\alpha/4}, \\ T_1 \approx \frac{T(k)}{(4(1-\rho))^{k/4}} &= k^{c_0 k} \cdot 2^{c_1 k} \cdot (4(1-\rho))^{-k/4}. \end{aligned}$$

For simplicity, let $u_\alpha := k + \phi_\alpha \in \mathbb{Z}^+$ with $\phi_\alpha := \left\lceil \frac{\eta k \log\alpha}{\log k} \right\rceil$. Since $\eta \in [\frac{2\ln k}{\ln k - \ln(2\pi e^2)}, \frac{1}{2c_0})$ and $k > (2\pi e^2)^{\frac{1}{1-4c_0}}$, we have $\eta > 2$ and $k \ge (2\pi e^2)^{\frac{\eta}{\eta-2}} > (2\pi e^2)^{\frac{1}{1-4c_0}}$. Then Item 3 of Lemma 1 implies $k_\alpha \le u_\alpha$. Since $1 < \alpha \le (k^{c_0} \cdot 2^{c_1})^2$, Item 2 of Lemma 1 implies $k_\alpha > k \ge \alpha^{\frac{1}{c_0}} 2^{\frac{|c_1|}{c_0}}$. Then $c_0 \log k_\alpha + c_1 - \frac{\log\alpha}{2} > 0$.

Thus,

$$T_\alpha \lessapprox 2^{\left(c_0 \log u_\alpha + c_1 - \frac{\log \alpha}{2}\right) u_\alpha} \cdot (4(1-\rho))^{-k_\alpha/4} = u_\alpha^{c_0 u_\alpha} \cdot 2^{c_1 u_\alpha} \cdot \alpha^{-u_\alpha/2} \cdot (4(1-\rho))^{-k_\alpha/4}.$$

As a result, we have

$$\begin{aligned}
\frac{T_1}{T_\alpha} &\gtrapprox \frac{k^{c_0 k} \cdot 2^{c_1 k} \cdot \alpha^{u_\alpha/2} \cdot (4(1-\rho))^{k_\alpha/4}}{u_\alpha^{c_0 u_\alpha} \cdot 2^{c_1 u_\alpha} \cdot (4(1-\rho))^{k/4}} \\
&= \frac{\alpha^{(k+\phi_\alpha)/2}}{k^{c_0 \phi_\alpha} \cdot (1+\frac{\phi_\alpha}{k})^{c_0 \cdot (k+\phi_\alpha)} \cdot 2^{c_1 \phi_\alpha}} \cdot (4(1-\rho))^{\frac{(k_\alpha - k)}{4}} \\
&\geq \frac{\alpha^{(k+\phi_\alpha)/2}}{k^{c_0 \phi_\alpha} \cdot e^{c_0 \cdot \phi_\alpha} \cdot (1+\frac{\phi_\alpha}{k})^{c_0 \phi_\alpha} \cdot 2^{c_1 \phi_\alpha}} \cdot (4(1-\rho))^{\frac{(k_\alpha - k)}{4}} \qquad \left(\text{using} \left(1+\frac{\phi_\alpha}{k}\right)^k \leq e^{\phi_\alpha}\right) \\
&\geq \frac{\alpha^{(k+\phi_\alpha)/2}}{k^{c_0 \phi_\alpha} \cdot (2e)^{c_0 \phi_\alpha} \cdot 2^{c_1 \phi_\alpha}} \cdot (4(1-\rho))^{\frac{(k_\alpha - k)}{4}} \qquad \left(\text{using } 1+\frac{\phi_\alpha}{k} \leq 2\right) \\
&\geq \frac{\alpha^{(k+\phi_\alpha)/2}}{\alpha^{c_0 \eta k} \cdot k^{c_0} \cdot (2e)^{c_0 \phi_\alpha} \cdot 2^{c_1 \phi_\alpha}} \cdot (4(1-\rho))^{\frac{(k_\alpha - k)}{4}} \qquad (\text{using } k^{c_0 \phi_\alpha} \leq \alpha^{c_0 \eta k} \cdot k^{c_0}) \\
&\geq \alpha^{\left(\frac{1}{2} - c_0 \eta\right) k} \cdot \left(\frac{\sqrt{\alpha}}{(2e)^{c_0} 2^{c_1}}\right)^{\phi_\alpha} \cdot k^{-c_0} \cdot (4(1-\rho))^{\frac{k \log \alpha}{4 \log k}}. \qquad (\text{by Item 2 of Lemma 1})
\end{aligned}$$

Substituting $\phi_\alpha = \left\lceil \frac{\eta k \log \alpha}{\log k} \right\rceil$, we conclude that

$$\frac{T_1}{T_\alpha} \gtrapprox \alpha^{\left(\frac{1}{2} - c_0 \eta\right) k} \cdot \left(\frac{\sqrt{\alpha}}{(2e)^{c_0} 2^{c_1}}\right)^{\frac{\eta k \log \alpha}{\log k}} \cdot (4(1-\rho))^{\frac{k \log \alpha}{4 \log k}}$$

up to some polynomial factor. This completes the proof. □

By Theorem 4, the smaller the time coefficient c_0 and the larger the relaxation constant α (satisfying both $4\,\alpha^4\,\rho \cdot (1-\rho) < 1$ and $1 < \alpha \leq (k^{c_0} \cdot 2^{c_1})^2$), the larger the exponential speedup factor $\alpha^{\left(\frac{1}{2} - c_0 \eta\right) k}$. This suggests that if some full enumeration algorithm of time $n^{c_0 n} \cdot 2^{O(n)}$ with smaller coefficient c_0 is found, then relaxing such an algorithm in the certain extreme cylinder pruning regime would result in better time/quality trade-offs for certain (including larger) RHFs. In brief, an enumeration oracle with smaller coefficient c_0 would benefit more from (larger) relaxation.

3.3 Numerical Validation

To validate Corollary 1 for concrete parameters, we simulated enumeration up to rank $k = 500$ when fixing $\rho = 0.01$ for varying α. For this, we first simulated both the output and the corresponding cost of pre-processing with k'-BKZ for some index $k' < k$. We note that for our pre-processing, we always assume a k'-rank SVP oracle inside BKZ. By combining the (recursive) preprocessing cost with the expected (repeated) enumeration cost, we arrive at an expected overall enumeration cost (denoted by $t_\alpha(k)$ in Table 1). For the top-most enumeration, we pick pruning parameters as suggested by Corollary 1 for $\rho = 0.01$ and for all values of α. Our simulation runs a simple linear search for k' such that the total expected cost is minimised. We then used `SciPy`'s `scipy.optimize.curve_fit` function [56] to fit simulation data into cost functions of form $k^{\frac{k}{2e}} \cdot 2^{c_1 k + c_2}$ with constant coefficients c_1 and c_2. For fitting we use always the indices $k = \lceil \alpha \cdot 100 \rceil, \lceil \alpha \cdot 100 \rceil + 1, \ldots, \lceil \alpha \cdot 250 \rceil$, which depend on α due to numerical stability issues. The results are given in Table 1.

Furthermore, several heuristics (such as the Geometric Series Assumption) are required to hold to instantiate Corollary 1 and Theorem 4. We check these experimentally in the full version of this work. In those experiments, the preprocessing cost is not taken into account and thus these algorithms are hypothetical. As a consequence, they give lower-bound estimates rather than predict costs.

Table 1. Speedups of relaxed enumeration with certain extreme cylinder pruning derived from our simulation for $\rho = 0.01$ and claimed by Corollary 1.

α	$\log t_\alpha(k)$ Simulation	$\log \frac{t_1(k)}{t_\alpha(k)}$ Simulation	$\log \frac{t_1(k)}{t_\alpha(k)} \approx \frac{\log \alpha}{2} k$ Corollary 1
1.00	$\frac{k \log k}{2\,\mathrm{e}} - 0.581\,k + 9.07$	0.00	0.00
1.05	$\frac{k \log k}{2\,\mathrm{e}} - 0.638\,k + 10.91$	$0.057\,k - 1.84$	$0.035k$
1.10	$\frac{k \log k}{2\,\mathrm{e}} - 0.691\,k + 12.34$	$0.110\,k - 3.27$	$0.069k$
1.15	$\frac{k \log k}{2\,\mathrm{e}} - 0.731\,k + 11.97$	$0.150\,k - 2.90$	$0.101k$
1.20	$\frac{k \log k}{2\,\mathrm{e}} - 0.767\,k + 11.21$	$0.186\,k - 2.14$	$0.132k$
1.25	$\frac{k \log k}{2\,\mathrm{e}} - 0.800\,k + 10.37$	$0.219\,k - 1.30$	$0.161k$
1.30	$\frac{k \log k}{2\,\mathrm{e}} - 0.836\,k + 10.75$	$0.255\,k - 1.69$	$0.189k$

Here, $t_\alpha(k)$ denotes the "expected cost" of the $(\alpha \cdot \mathrm{GH}(k))$-HSVP enumeration oracle in rank $k \in [\lceil \alpha \cdot 100 \rceil, \lceil \alpha \cdot 250 \rceil]$, including preprocessing.

4 Practical Approximate Enumeration Oracles

Table 1 highlights the relative speedups obtainable by relaxed enumeration with certain extreme cylinder pruning. It does not, however, present speedups over the state-of-the-art for enumeration, which can be observed by comparing the second column of Table 1 with the known cost $2^{\frac{k \log k}{2\,\mathrm{e}} - 0.995\,k + 16.25}$ of enumeration with optimised BKZ 2.0 [17] preprocessing (see [4, Fig. 2]).

In this section, we provide simulation data – fitted curves and experimental validation – to show that with FP(y)LLL's `pruning` module [21,22] and with or without extended preprocessing, relaxed enumeration does achieve exponential speedups, but with a loss in the approximation factor: it can be viewed as a practical version of Corollary 1. We will consider the performance gain when targeting the same RHF as an exact oracle in Sect. 5. In the full version of this work, we also provide additional experiments to check the accuracy of the underlying cost estimation module in FP(y)LLL, with respect to relaxed pruned enumeration. Furthermore, a curious artefact of our parameters is that they do not suggest extreme pruning. Rather, they imply a small number of repetitions only. We elaborate on this in the full version of this work.

4.1 Simulations and Cost Estimates

As in Sect. 3.3, we run the top-most enumeration as an $(\alpha \cdot \mathrm{GH}(k))$-HSVP-oracle in rank k and perform a linear search over parameter k' $(< k)$ for preprocessing such that the overall enumeration cost is minimised. We first simulate calling Algorithm 2 with block size k' (i.e. k'-BKZ) to preprocess a given basis of rank $\lceil (1 + c) \cdot k \rceil$ and then simulate running relaxed enumeration on it. That is, we simulate the "expected cost" of the $(\alpha \cdot \mathrm{GH}(k))$-HSVP enumeration oracle in rank k with preprocessing in rank $\lceil (1 + c) \cdot k \rceil$, i.e. enumeration on a k-rank head block $\boldsymbol{B}$ with FPyLLL's optimised cylinder pruning and with relaxed radius $R = \alpha \cdot \mathrm{GH}(\mathcal{L}(\boldsymbol{B}))$. Here, the "expected cost" of each oracle call includes both the expected (repeated) enumeration cost and all recursive preprocessing costs.

We illustrate the fitted cost estimates in Table 2 (columns "$\alpha' = 1$"), which confirm that relaxed enumeration does achieve exponential speedups. We also give some example data and curve fits in Fig. 3.

Remark 1. In Table 2 we are seeing a slight advantage when picking $c = 0.15$ over picking $c = 0.25$. It slightly deviates from a claim in [4] that for $\alpha = 1$, $c = 0.25$ seems to provide the best performance among $c \geq 0$. We hence reproduce this advantage using the original simulation code from [4] in the full version of this work. This simulation confirms that the choice of $c = 0.15$ also provides a minor performance improvement for $\alpha = 1$.

4.2 Consistency with Experiments

In Fig. 4, we give experimental data comparing our implementation with our simulations of the $(\alpha \cdot \mathrm{GH}(k))$-HSVP enumeration oracle in rank k with preprocessing in rank $\lceil (1 + c) \cdot k \rceil$ for $c \in \{0.00, 0.15, 0.25\}$.[6] It shows that our simulation for cost estimates is reasonably accurate for larger instances with a minor bias towards underestimating the cost. The data should be understood as follows:

[6] The reader may consult [4, Fig. 4] for the case $c = 0.00, \alpha = 1.00$.

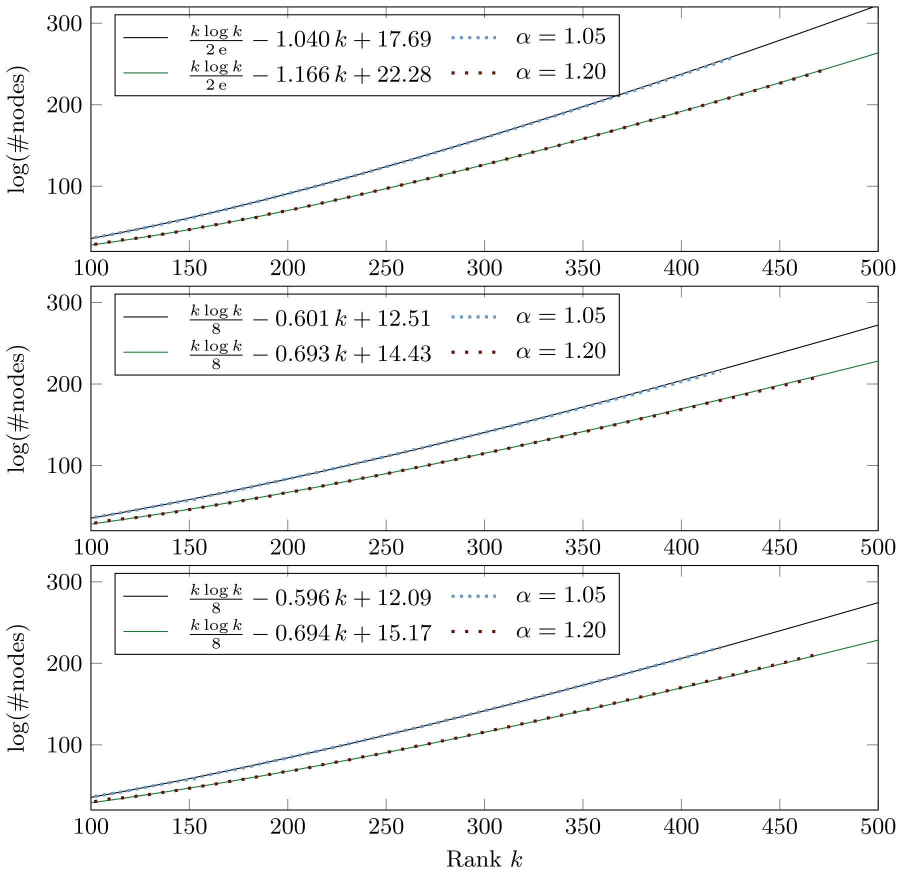


Fig. 3. Selected "expected costs" from simulations for $(\alpha \cdot \mathrm{GH}(k))$-HSVP enumeration oracles in rank k for $c \in \{0.00, 0.15, 0.25\}$ (in turn).

- "Simulation" is the output of our simulation code `simu.py`.
- "Runtime" is the walltime for running FPLLL, converted to "nodes visited" units, assuming 64 CPU cycles per node. It is scaled by $3.3 \cdot 10^9/64$ because it runs on a "Intel(R) Xeon(R) CPU E5-2667 v2 @ 3.30 GHz" (strombenzin).
- "Nodes" is the number of enumeration nodes visited reported by FPLLL. "Runtime" also includes the cost of recursive LLL calls, but "Nodes" does not.

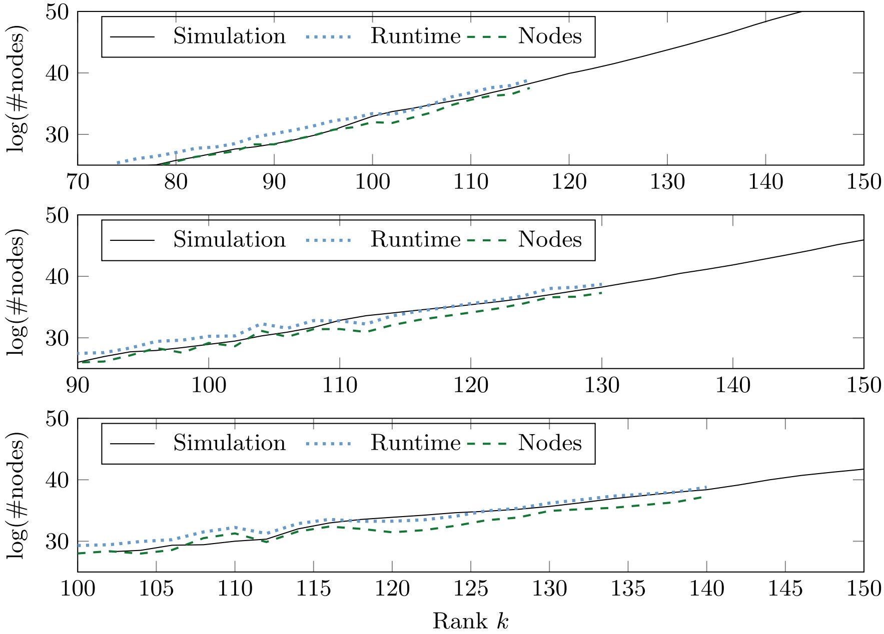


Fig. 4. Experimental verification of simulation results for the $(\alpha \cdot \mathrm{GH}(k))$-HSVP enumeration oracle in rank k with example $\alpha \in \{1.10, 1.20, 1.30\}$ (in turn) and $c = 0.15$. We ran 16 experiments.

5 A Practical BKZ Variant

While Sect. 4 establishes a practical exponential speed-up of relaxed enumeration in the same rank k, it does not yet account for the loss in quality. In this section, we consider relaxed enumeration in rank k_α to obtain a RHF of $\approx \mathrm{GH}(k)^{1/(k-1)}$. This enables us to define a practical variant of the BKZ algorithm utilising relaxed enumeration. This, in turn, enables us to use relaxed enumeration recursively to preprocess bases for relaxed enumeration.

To this end, we present a generalisation of the BKZ variant in [4] with one more optional parameter. This generalisation integrates the idea of extended preprocessing (introduced by [4]) with the relaxation strategy (formalised in [1,35]) on enumeration-based HSVP-oracles. That is, given a performance parameter k (akin to the 'block size' of Algorithm 2), we equip Schnorr–Euchner's BKZ with approximate enumeration oracles as illustrated in Sect. 4, namely an $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α with preprocessing in rank $\lceil (1 + c) \cdot k_\alpha \rceil$ for some small constant $c \geq 0$ and an optional relaxation constant $\alpha \geq 1$. This BKZ variant uses three parameters (k, c, α), while [4]'s variant relies on two parameters (k, c) and BKZ (2.0) [17,51] uses one parameter k. In particular, our BKZ variant can be viewed as a practical version of Theorem 4.

With extensive experiments and simulations, we investigate the performances of this BKZ variant for both practical and cryptographic parameter ranges: it does achieve better time/quality trade-offs for certain approximation regimes than both [4]'s variant and BKZ 2.0 [17].

Main result. Given as input a performance parameter k—our simulations cover $k \in [100, 400]$—an overshooting parameter $c \in [0, 0.4]$, and a basis of an integer lattice of rank $n \geq (1+c) \cdot k_{1.3}$, our BKZ variant first picks the "optimal" relaxation constant $\alpha \in \{1, 1.05, 1.1, 1.15, 1.2, 1.25, 1.3\}$ to minimise the expected cost of one oracle call and achieves RHF $\mathrm{GH}(k)^{\frac{1}{k-1}}$ with simulated cost estimates:

- Case $c = 0$: the expected cost of one oracle call is about $2^{\frac{k \log k}{2\,\mathrm{e}} - 1.077\,k + 29.12}$, which is lower than BKZ 2.0's record $2^{\frac{k \log k}{2\,\mathrm{e}} - 0.995\,k + 16.25}$ reported in [4, Fig. 2];
- Case $c = 0.25$: the expected cost of one oracle call is about $2^{\frac{k \log k}{8} - 0.632\,k + 21.94}$, which is lower than the record in [4]: $2^{\frac{k \log k}{8} - 0.547\,k + 10.4}$;
- Case $c = 0.15$: the expected cost of one oracle call is about $2^{\frac{k \log k}{8} - 0.654\,k + 25.84}$.

Our results are illustrated in Fig. 1. Our simulations were performed on q-ary lattices of dimensions $n = \lceil (1+c) \cdot k_\alpha \rceil$ with volume $q^{n/2}$ for $q = 2^{30}$.

5.1 Algorithm

Algorithm 5 is our BKZ variant which, given as input a performance parameter $k \geq 2$, an overshooting parameter $c \geq 0$, a relaxation parameter $\alpha \geq 1$, and a basis of an integer lattice $\mathcal{L}$ of rank $n \geq (1+c) \cdot k_\alpha$, outputs a reduced basis of $\mathcal{L}$.

It calls the $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α with preprocessing in rank $\lceil (1+c) \cdot k_\alpha \rceil$ as an HSVP subroutine. This oracle includes recursive preprocessing: when $\alpha = 1$ then Algorithm 6 is essentially Algorithm 4, and hence calls a function $\mathrm{pre}(\cdot, \cdot)$ for returning the preprocessing parameter. When $(c, \alpha) = (0, 1)$, Algorithm 5 is essentially BKZ 2.0 [17] and Schnorr-Euchner's BKZ algorithm [51].

Restricted to the state-of-the-art power in practice, we choose $c \in [0, 0.4]$ and $\alpha \in \{1.00, 1.05, 1.10, 1.15, 1.20, 1.25, 1.30\}$ for simplicity in our simulations.

Remark 2. In our experiments, the choice of α in Algorithm 5 is determined from an optimised strategy profile built upon our simulated data for each $k \in [2, 400]$. We remark that it is also possible to determine such α on-the-fly based on simulations on the current basis.

Algorithm 5. A new BKZ variant with three parameters (k, c, α)

Require: $(\boldsymbol{B}, k, c, \alpha)$, where $\boldsymbol{B} = (\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1})$ is an LLL-reduced basis of an n-rank lattice $\mathcal{L}$ in $\mathbb{Z}^m$, $k \in [2, n)$ is a performance parameter, $c \geq 0$ is an overshooting parameter, $\alpha \geq 1$ is a relaxation parameter satisfying $n \geq (1+c) \cdot k_\alpha$, and $N \in \mathbb{Z}^+$ denotes the number of tours.

Ensure: A reduced basis of $\mathcal{L}$.

```
for ℓ = 0 to N − 1 do
    for j = 0 to n − 2 do
        Find a short nonzero vector v in the lattice L_[j,h) (generated by the projected
        block B_[j,h) where h = min{j+⌈(1+c)·k_α⌉, n}), by calling Alg. 6 on (B_[j,h), k, c, α)
        if ‖b*_j‖ > ‖v‖ then
            Lift v into a primitive vector b for the sublattice generated by the basis
            vectors b_j, ..., b_{h−1} such that ‖π_j(b)‖ ≤ ‖v‖
            LLL-reduce (b_0, ..., b_{j−1}, b, b_j, ..., b_{n−1}) to remove linear dependencies
        end if
    end for
end for
return B.
```

Handling the tail. Just like all known BKZ variants (such as the variant in [4] and BKZ 2.0 [17]), it is tricky to handle tail projected blocks of the current basis during execution, because of the decreasing ranks over $d = \lceil (1+c) \cdot k_\alpha \rceil, \lceil (1+c) \cdot k_\alpha \rceil - 1, \ldots, 2$. We hence generalise [4]'s tail function $\mathrm{tail}(\cdot, \cdot, \cdot)$ with one more parameter α for computing the enumeration rank.

For given integer $k \geq 2$, constant $c \geq 0$ and relaxation constant $\alpha \geq 1$, our approximate enumeration oracle first finds the enumeration rank k^* using the function $\mathrm{tail}(k, c, \alpha, d)$ for $d = 2, \ldots, \lceil (1+c) \cdot k_\alpha \rceil$:

$$k^* \leftarrow \mathrm{tail}(k, c, \alpha, d) = \max\left\{ \min\left\{ d, \left\lceil k_\alpha - \frac{\lceil (1+c) \cdot k_\alpha \rceil - d}{2} \right\rceil \right\}, 2 \right\}.$$

Then $k^* = k_\alpha$ when $d = \lceil (1+c) \cdot k_\alpha \rceil$. It can be checked that k^* is strictly less than d if d is large enough and is exactly equal to d otherwise:

$$\mathrm{tail}(k, c, \alpha, d) = \begin{cases} k_\alpha + \left\lceil \frac{d - \lceil (1+c) \cdot k_\alpha \rceil}{2} \right\rceil & \text{if } (1-c) \cdot k_\alpha < d \leq \lceil (1+c) \cdot k_\alpha \rceil \\ d & \text{if } 2 \leq d \leq (1-c) \cdot k_\alpha \end{cases} \in [2, k_\alpha]. \quad (2)$$

Algorithm 5 calls the $(\alpha \cdot \mathrm{GH}(k^*))$-HSVP enumeration oracle in rank k^* with preprocessing in rank d to reduce each tail projected block, namely Algorithm 6.

Preprocessing parameter. Given a projected block (say,) $(\boldsymbol{b}_0, \dots, \boldsymbol{b}_{d-1})$ of rank $d \in [2, \lceil (1+c) \cdot k_\alpha \rceil]$, the preprocessing function $\mathrm{pre}(k^*, \|\boldsymbol{b}_0^*\|, \dots, \|\boldsymbol{b}_{d-1}^*\|)$ returns the "optimal" preprocessing parameter $k' \in [2, k^*]$, possibly based on simulations, such that the cost of enumeration on the k^*-rank head block is minimised (e.g., at most $k^{k/8} \cdot 2^{O(k)}$ when $c = 0.15$), after preprocessing on $(\boldsymbol{b}_0, \dots, \boldsymbol{b}_{d-1})$ using Algorithm 5 recursively in lower levels, i.e. equipped with a similar HSVP-oracle with parameters (k', c, α') (instead of the current level (k, c, α)).

Since $k_\alpha \geq k^* \geq k' \geq 2$, each enumeration throughout all recursive levels of Algorithm 5 would not be more expensive than the top-most enumeration-based HSVP-oracle (i.e., the $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α with preprocessing in rank $\lceil (1+c) \cdot k_\alpha \rceil$).

5.2 Performance of Our BKZ Variant

Using simulations and data from our implementation, we now validate the performance of our algorithm. We first show that preprocessing with relaxed enumeration has a performance benefit (for $c > 0$) and then validate the output quality of our algorithm. Combining the two, we obtain our main result in Fig. 1, as claimed above.

Algorithm 6. An approx-HSVP oracle on $(\boldsymbol{B}_{[j,h)}, k, c, \alpha)$ using relaxed enumeration in rank k^* with extended preprocessing in rank $(h-j)$

1: Find the enumeration rank $k^* \leftarrow \mathrm{tail}(k, c, \alpha, h-j)$ by Eq. (2)
2: Numerically find the preprocessing parameter $k' \leftarrow \mathrm{pre}(k^*, \|\boldsymbol{b}_j^*\|, \dots, \|\boldsymbol{b}_{h-1}^*\|)$
3: **if** $k' \geq 3$ **then**
4: Run Alg. 5 on $(\boldsymbol{B}_{[j,h)}, k', c, \alpha')$ with some $\alpha' \geq 1$ to obtain a reduced basis $\boldsymbol{C} \in \mathbb{Q}^{(h-j)\times m}$ of $\mathcal{L}_{[j,h)}$
5: **else**
6: LLL-reduce $\boldsymbol{B}_{[j,h)}$ into a basis $\boldsymbol{C} \in \mathbb{Q}^{(h-j)\times m}$ of $\mathcal{L}_{[j,h)}$
7: **end if** //Steps 3-7 preprocess $\boldsymbol{B}_{[j,h)}$ for the relaxed enumeration.
8: Call the $(\alpha \cdot \mathrm{GH}(k^*))$-HSVP enumeration oracle in rank k^* on the head block $\boldsymbol{C}_{[0,k^*)}$ of $\boldsymbol{C}$ to find a short nonzero vector $\boldsymbol{v}$ in the lattice $\mathcal{L}_{[j,h)}$

$\boldsymbol{\alpha} \cdot$ GH($\boldsymbol{k}$)-HSVP oracle performance. In the columns labelled "$\alpha' \geq 1$" in Table 2, we present the speed-ups over $\alpha = 1$ attained by our BKZ variant. That is, the performance of solving $\alpha \cdot \mathrm{GH}(k)$-HSVP when using recursive preprocessing with $\alpha' \geq 1$. We can observe the following from Table 2:

Table 2. Speedups of relaxed enumeration with extreme cylinder pruning derived from our simulation with FPyLLL's optimised cylinder pruning and recursive relaxed enumeration compared with that claimed by Corollary 1.

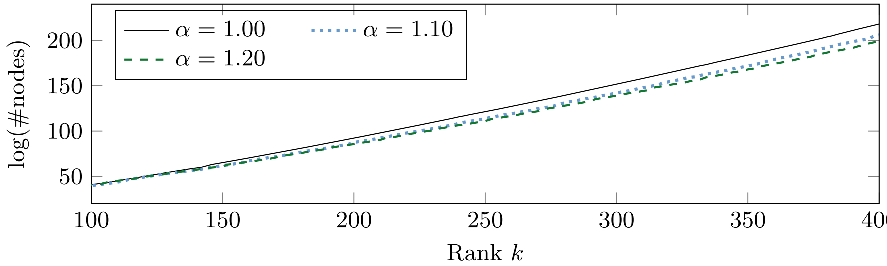
α	$\log \frac{t_1(k)}{t_\alpha(k)}$ Cor. 1	$\log t_\alpha(k)$ Sim. ($\alpha' = 1$)	$\log \frac{t_1(k)}{t_\alpha(k)}$ Sim. ($\alpha' = 1$)	$\log t_\alpha(k)$ Sim. ($\alpha' \geq 1$)	$\log \frac{t_1(k)}{t_\alpha(k)}$ Sim. ($\alpha' \geq 1$)
			$c = 0.00$		
1.00	0.00	$\frac{k \log k}{2\,e} - 0.994\,k + 17.94$	0.00	$\frac{k \log k}{2\,e} - 0.946\,k + 11.31$	0.00
1.05	$0.035k$	$\frac{k \log k}{2\,e} - 1.040\,k + 17.69$	$0.046\,k + 0.24$	$\frac{k \log k}{2\,e} - 0.984\,k + 9.82$	$0.038\,k + 1.49$,
1.10	$0.069k$	$\frac{k \log k}{2\,e} - 1.088\,k + 18.56$	$0.093\,k - 0.63$	$\frac{k \log k}{2\,e} - 1.027\,k + 9.99$	$0.081\,k + 1.32$
1.15	$0.101k$	$\frac{k \log k}{2\,e} - 1.132\,k + 20.55$	$0.137\,k - 2.61$	$\frac{k \log k}{2\,e} - 1.078\,k + 12.75$	$0.132\,k - 1.45$
1.20	$0.132k$	$\frac{k \log k}{2\,e} - 1.166\,k + 22.28$	$0.171\,k - 4.34$	$\frac{k \log k}{2\,e} - 1.123\,k + 15.73$	$0.176\,k - 4.43$
1.25	$0.161k$	$\frac{k \log k}{2\,e} - 1.193\,k + 23.84$	$0.199\,k - 5.90$	$\frac{k \log k}{2\,e} - 1.157\,k + 17.93$	$0.211\,k - 6.62$
1.30	$0.189k$	$\frac{k \log k}{2\,e} - 1.217\,k + 25.42$	$0.223\,k - 7.48$	$\frac{k \log k}{2\,e} - 1.187\,k + 20.31$	$0.241\,k - 9.00$
			$c = 0.15$		
1.00	0.00	$\frac{k \log k}{8} - 0.552\,k + 12.53$	0.00	$\frac{k \log k}{8} - 0.566\,k + 14.28$	0.00
1.05	$0.035k$	$\frac{k \log k}{8} - 0.601\,k + 12.51$	$0.049\,k + 0.02$	$\frac{k \log k}{8} - 0.617\,k + 14.69$	$0.052\,k - 0.41$
1.10	$0.069k$	$\frac{k \log k}{8} - 0.641\,k + 13.13$	$0.089\,k - 0.60$	$\frac{k \log k}{8} - 0.660\,k + 15.68$	$0.094\,k - 1.40$
1.15	$0.101k$	$\frac{k \log k}{8} - 0.670\,k + 13.79$	$0.118\,k - 1.26$	$\frac{k \log k}{8} - 0.691\,k + 16.71$	$0.126\,k - 2.43$
1.20	$0.132k$	$\frac{k \log k}{8} - 0.693\,k + 14.43$	$0.142\,k - 1.90$	$\frac{k \log k}{8} - 0.716\,k + 17.73$	$0.151\,k - 3.45$
1.25	$0.161k$	$\frac{k \log k}{8} - 0.713\,k + 15.19$	$0.161\,k - 2.66$	$\frac{k \log k}{8} - 0.738\,k + 18.91$	$0.172\,k - 4.63$
1.30	$0.189k$	$\frac{k \log k}{8} - 0.730\,k + 15.95$	$0.178\,k - 3.42$	$\frac{k \log k}{8} - 0.757\,k + 20.01$	$0.191\,k - 5.73$
			$c = 0.25$		
1.00	0.00	$\frac{k \log k}{8} - 0.549\,k + 12.33$	0.00	$\frac{k \log k}{8} - 0.571\,k + 15.39$	0.00
1.05	$0.035k$	$\frac{k \log k}{8} - 0.596\,k + 12.09$	$0.047\,k + 0.24$	$\frac{k \log k}{8} - 0.616\,k + 14.80$	$0.044\,k + 0.60$
1.10	$0.069k$	$\frac{k \log k}{8} - 0.639\,k + 13.15$	$0.090\,k - 0.82$	$\frac{k \log k}{8} - 0.651\,k + 14.84$	$0.080\,k + 0.55$
1.15	$0.101k$	$\frac{k \log k}{8} - 0.669\,k + 14.08$	$0.121\,k - 1.75$	$\frac{k \log k}{8} - 0.683\,k + 15.93$	$0.112\,k - 0.53$
1.20	$0.132k$	$\frac{k \log k}{8} - 0.694\,k + 15.17$	$0.145\,k - 2.84$	$\frac{k \log k}{8} - 0.712\,k + 17.59$	$0.140\,k - 2.20$
1.25	$0.161k$	$\frac{k \log k}{8} - 0.713\,k + 15.92$	$0.164\,k - 3.59$	$\frac{k \log k}{8} - 0.735\,k + 19.09$	$0.164\,k - 3.70$
1.30	$0.189k$	$\frac{k \log k}{8} - 0.728\,k + 16.62$	$0.180\,k - 4.29$	$\frac{k \log k}{8} - 0.755\,k + 20.50$	$0.183\,k - 5.11$

Here, $t_\alpha(k)$ denotes the "expected cost" of the $(\alpha \cdot \mathrm{GH}(k))$-HSVP enumeration oracle in rank $k \in [\lceil \alpha \cdot 100 \rceil, \lceil \alpha \cdot 250 \rceil]$, with preprocessing in rank $\lceil (1 + c)\,k \rceil$, using relaxed enumeration recursively.

– Without extended preprocessing (i.e. setting the overshooting parameter $c = 0$), Table 2 indicates that it is better for preprocessing in rank k to call the $(\alpha' \cdot \mathrm{GH}(k'))$-HSVP enumeration oracle in rank k' with $\alpha' = 1$ than $\alpha' > 1$.
– In contrast, Table 2 indicates that in the case $c > 0$, it is better for preprocessing in rank $\lceil (1 + c) \cdot k \rceil$ to call the $(\alpha' \cdot \mathrm{GH}(k'))$-HSVP enumeration oracle in rank k' with some $\alpha' \geq 1$ than $\alpha' = 1$, i.e. to proceed as outlined above.

Table 2 does not normalise time/quality trade-offs. Thus, in Fig. 5 we illustrate the performance gain of relaxed enumeration for reaching the same RHF.

(a) Expected cost $t_\alpha(k_\alpha)$ of the $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α for reaching RHF $\mathrm{GH}(k)^{\frac{1}{k-1}}$.



(b) Cost advantage $\log \frac{t_1(k)}{t_\alpha(k_\alpha)}$ of the $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α for reaching RHF $\mathrm{GH}(k)^{\frac{1}{k-1}}$.

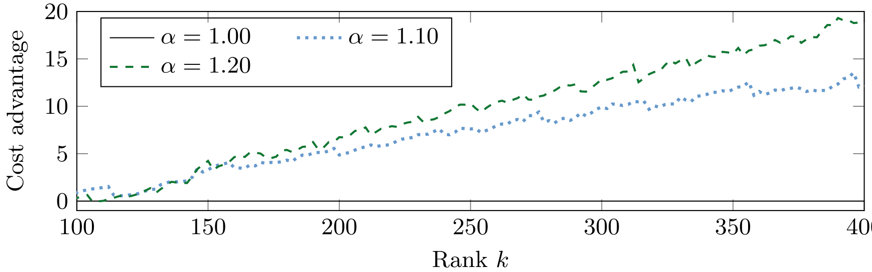
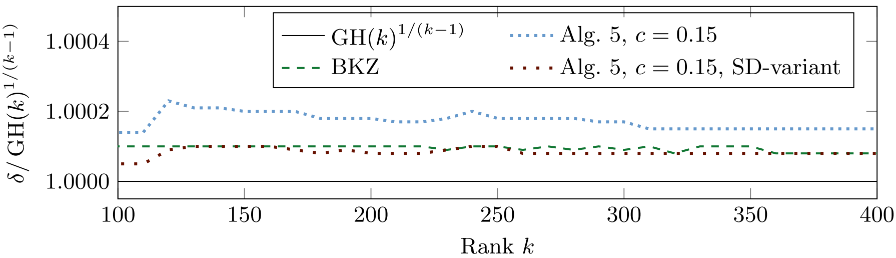


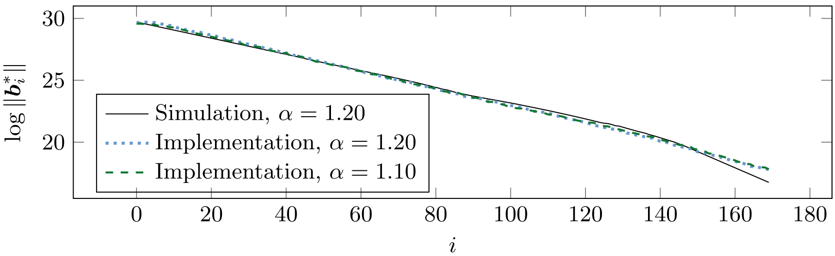
Fig. 5. Expected performance of $(\alpha \cdot \mathrm{GH}(k_\alpha))$-HSVP enumeration oracle in rank k_α; case $c = 0.15$; preprocessing with $\alpha' \geq 1.00$.

Quality. To validate the output quality of our BKZ variant, we compared the RHF predicted by the simulations for BKZ, Algorithm 5 and a self-dual variant of Algorithm 5 in Fig. 6a, following the strategy of [4]. As Fig. 6a illustrates, our variant achieves the same RHF as BKZ, when run in "self-dual" mode.

We also verified the behaviour of the practical implementation of Algorithm 5 against our simulation and give an example in Fig. 6b. As this figure illustrates, our implementation agrees with our simulation except in the tail.



(a) We compare $\delta := (\|\boldsymbol{b}_0\|/\text{vol}(\Lambda)^{1/n})^{1/(n-1)}$ as predicted by simulation algorithms to $\text{GH}(k)^{1/(k-1)}$ for $n = 2\,k$ and random q-ary lattices. For "BKZ" we use eight tours of the simulator from [18]. For "Alg. 5, $c = 0.15$" we use eight tours of our simulator. For "Alg. 5, $c = 0.15$, SD-variant" we use our simulator on the dual basis (four tours) followed by the same on the primal basis (four tours).



(b) We compare the basis shape predicted by our simulations with that produced by our implementation of Algorithm 5 for q-ary lattices Λ with $q = 2^{24} + 43$, $n = 170$, $\text{vol}(\Lambda) = q^{n/2}$, $k = 60$ and $c = 0.25$. Implementation data is averaged over eight runs.

Fig. 6. Basis quality.

References

1. Aggarwal, D., Li, J., Nguyen, P.Q., Stephens-Davidowitz, N.: Slide reduction, revisited—filling the gaps in SVP approximation. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 274–295. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_10
2. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: 28th ACM STOC, pp. 99–108. ACM Press (May 1996)
3. Ajtai, M.: The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In: 30th ACM STOC, pp. 10–19. ACM Press (May 1998)
4. Albrecht, M.R., Bai, S., Fouque, P.A., Kirchner, P., Stehlé, D., Wen, W.: Faster enumeration-based lattice reduction: root Hermite factor $k^{1/(2k)}$ Time $k^{k/8+o(k)}$. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 186–212. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_7
5. Albrecht, M.R., Bai, S., Li, J., Rowell, J.: Lattice reduction with approximate enumeration oracles: practical algorithms and concrete performance. Cryptology ePrint Archive, Report 2020/1260 (2020). https://eprint.iacr.org/2020/1260

6. Albrecht, M.R., et al.: Estimate all the LWE, NTRU schemes!. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 351–367. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_19
7. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 717–746. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_25
8. Albrecht, M.R., Gheorghiu, V., Postlethwaite, E.W., Schanck, J.M.: Estimating quantum speedups for lattice sieves. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 583–613. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_20
9. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving uSVP and applications to LWE. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 297–322. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_11
10. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - a new hope. In: Holz, T., Savage, S. (eds.) USENIX Security 2016, pp. 327–343. USENIX Association (August 2016)
11. Aono, Y., Nguyen, P.Q., Shen, Y.: Quantum lattice enumeration and tweaking discrete pruning. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part I. LNCS, vol. 11272, pp. 405–434. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03326-2_14
12. Aono, Y., Wang, Y., Hayashi, T., Takagi, T.: Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 789–819. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_30
13. Bai, S., Stehlé, D., Wen, W.: Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part I. LNCS, vol. 11272, pp. 369–404. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03326-2_13
14. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th SODA, pp. 10–24. ACM-SIAM (January 2016)
15. Bernstein, D.J., et al.: NTRU prime. Tech. rep., National Institute of Standards and Technology (2020). https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions
16. Blichfeldt, H.F.: A new principle in the geometry of numbers, with some applications. Trans. Am. Math. Soc. **16**, 227–235 (1914)
17. 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). https://doi.org/10.1007/978-3-642-25385-0_1
18. Conway, J.H., Sloane, N.J.A.: Sphere-Packings, Lattices, and Groups. Springer, Heidelberg (1987). https://doi.org/10.1007/978-1-4757-6568-7
19. Ducas, L., Stevens, M., van Woerden, W.: Advanced lattice sieving on GPUs, with tensor cores (2021). to appear in Eurocrypt 2021. https://eprint.iacr.org/2021/141
20. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Math. Comput. **44**(170), 463–471 (1985)
21. FPLLL development team: FPLLL, a lattice reduction library (2019). https://github.com/fplll/fplll
22. FPyLLL development team: FPyLLL, a Python interface to FPLLL (2020). https://github.com/fplll/fpylll

23. Gama, N., Nguyen, P.Q.: Finding short lattice vectors within Mordell's inequality. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 207–216. ACM Press (May 2008)
24. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_3
25. 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
26. Garcia-Morchon, O., et al.: Round5. Tech. rep., National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
27. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 197–206. ACM Press (May 2008)
28. Hanrot, G., Pujol, X., Stehlé, D.: Analyzing blockwise lattice algorithms using dynamical systems. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 447–464. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_25
29. Hanrot, G., Stehlé, D.: Improved analysis of Kannan's shortest lattice vector algorithm. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 170–186. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_10
30. Haviv, I., Regev, O.: Tensor-based hardness of the shortest vector problem to within almost polynomial factors. Theory Comput. **8**(1), 513–531 (2012). preliminary version in Proceedings of STOC '07
31. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: 15th ACM STOC, pp. 193–206. ACM Press (April 1983)
32. Khot, S.: Hardness of approximating the shortest vector problem in lattices. J. ACM **52**(5), 789–808 (2005). preliminary version in Proceedings of FOCS '04
33. Laarhoven, T.: Search problems in crpytography. Ph.D. thesis, Eindhoven University of Technology (2015)
34. Lenstra, A.K., Lenstra Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**, 366–389 (1982)
35. Li, J., Nguyen, P.Q.: A complete analysis of the BKZ lattice reduction algorithm (2020). https://eprint.iacr.org/2020/1237.pdf
36. Lovász, L.: An algorithmic theory of numbers, graphs and convexity. Society for Industrial and Applied Mathematics (1986)
37. Micciancio, D.: The shortest vector in a lattice is hard to approximate to within some constant. SIAM J. Comput. **30**(6), 2008–2035 (2001). preliminary version in Proceedings of FOCS '98
38. Micciancio, D.: Inapproximability of the shortest vector problem: toward a deterministic reduction. Theory Comput. **8**(22), 487–512 (2012). http://www.theoryofcomputing.org/articles/v008a022
39. Micciancio, D., Walter, M.: Fast lattice point enumeration with minimal overhead. In: Indyk, P. (ed.) 26th SODA, pp. 276–294. ACM-SIAM (January 2015)
40. Micciancio, D., Walter, M.: Practical, predictable lattice basis reduction. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 820–849. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_31
41. Milnor, J., Husemoller, D.: Symmetric Bilinear Forms. Springer, Heidelberg (1973). https://doi.org/10.1007/978-3-642-88330-9

42. Nguên, P.Q., Stehlé, D.: Floating-point LLL revisited. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 215–233. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_13
43. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 333–342. ACM Press (May/June 2009)
44. Pohst, M.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. SIGSAM Bull. **15**, 37–44 (1981)
45. Poppelmann, T., et al.: NewHope. Tech. rep., National Institute of Standards and Technology (2019). https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
46. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press (May 2005)
47. Rogers, C.A.: The number of lattice points in a set. Proc. Lond. Math. Soc. **3**, 305–320 (1956)
48. Schneider, M., Gama, N.: Darmstadt SVP challenges (2010). https://www.latticechallenge.org/svp-challenge/index.php. Accessed 17 Aug 2018
49. Schnorr, C.P.: A hierarchy of polynomial time lattice basis reduction algorithms. Theor. Comput. Sci. **53**, 201–224 (1987)
50. 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, Heidelberg (2003). https://doi.org/10.1007/3-540-36494-3_14
51. Schnorr, C.P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Program. **66**, 181–199 (1994)
52. Schnorr, C.P., Hörner, H.H.: Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In: Guillou, L.C., Quisquater, J.J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 1–12. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-49264-X_1
53. Shoup, V.: NTL 11.4.3: number theory c++ library (2020). http://www.shoup.net/ntl/
54. Siegel, C.L.: Lectures on the Geometry of Numbers. Springer, New York (1989). https://doi.org/10.1007/978-3-662-08287-4
55. Teruya, T., Kashiwabara, K., Hanaoka, G.: Fast lattice basis reduction suitable for massive parallelization and its application to the shortest vector problem. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 437–460. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76578-5_15
56. Virtanen, P., et al.: SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Methods **17**, 261–272 (2020)