

# Solving shortest and closest vector problems: The decomposition approach

Anja Becker

Nicolas Gama

Antoine Joux

June 13, 2014

## Abstract

In this paper, we present a heuristic algorithm for solving exact, as well as approximate, shortest vector and closest vector problems on lattices. The algorithm can be seen as a modified sieving algorithm for which the vectors of the intermediate sets lie in overlattices or translated cosets of overlattices. The key idea is hence to no longer work with a single lattice but to move the problems around in a tower of related lattices. We initiate the algorithm by sampling very short vectors in an overlattice of the original lattice that admits a quasi-orthonormal basis and hence an efficient enumeration of vectors of bounded norm. Taking sums of vectors in the sample, we construct short vectors in the next lattice. Finally, we obtain solution vector(s) in the initial lattice as a sum of vectors of an overlattice. The complexity analysis relies on the Gaussian heuristic. This heuristic is backed by experiments in low and high dimensions that closely reflect these estimates when solving hard lattice problems in the average case.

This new approach allows us to solve not only shortest vector problems, but also closest vector problems, in lattices of dimension  $n$  in time  $2^{0.3774n}$  using memory  $2^{0.2925n}$ . Moreover, the algorithm is straightforward to parallelize on most computer architectures.

**Note on this version.** The great part of the paper is published at ANTS 2014 under the title “A Sieve Algorithm Based on Overlattices”. We added here the section “Example for co-cyclic lattices or  $q$ -ary lattices” which gives a concrete example of a tower of lattices one might consider at first trial.

## 1 Introduction

Hard lattice problems, such as the shortest vector problem (SVP) and the closest vector problem (CVP), have a long standing relationship to number theory and cryptology. In number theory, they can for example be used to find Diophantine approximations. In cryptology, they were used as cryptanalytic tools for a long time, first through a direct approach as in [20] and then more indirectly using Coppersmith’s small roots algorithms [8, 9]. More recently, these hard problems have also been used to construct cryptosystems. Lattice-based cryptography is also a promising area due to the simple additive, parallelizable structure of a lattice. The two basic hard problems SVP and CVP are known to be NP-hard<sup>1</sup> to solve exactly [1, 22] and also NP-hard to approximate [10, 27] within at least constant factors. The time complexity of known algorithms that find the *exact* solution are at least exponential in the dimension of the lattice. These algorithms also serve as subroutines for strong polynomial time *approximation* algorithms. Algorithms for the exact problem hence enable us to choose appropriate parameters.

A *shortest* vector can be found by enumeration [37, 21], sieving [3, 32, 29, 39] or the Voronoi-cell algorithm [28]. Enumeration uses a negligible amount of memory and its running time is between  $n^{\mathcal{O}(n)}$  and  $2^{\mathcal{O}(n^2)}$  depending on the amount and quality of the preprocessing. Probabilistic sieving algorithms, as well as the deterministic Voronoi-cell algorithm are simply exponential in time and

---

<sup>1</sup>Under randomized reductions in the case of SVP.

Table 1: Complexity of currently known SVP/CVP algorithms.

Algorithm	Time	Memory	CVP	SVP	
Kannan-Enumeration [18]	$n^{n/2+o(n)}$	$\text{poly}(n)$	✓		proven
	$n^{n/(2e)+o(n)}$	$\text{poly}(n)$		✓	proven
Voronoi-cell [28]	$2^{2n}$	$2^n$	✓	✓	proven
ListSieve-Birthday [34]	$2^{2.465n+o(n)}$	$2^{1.233n+o(n)}$	×	✓	proven
GaussSieve [29]	$2^{0.415n+o(n)?}$	$2^{0.2075n+o(n)?}$	?	✓	heuristic
Nguyen-Vidick sieve [32]	$2^{0.415n+o(n)}$	$2^{0.2075n+o(n)}$	×	✓	heuristic
WLTB sieve [39]	$2^{0.3836n+o(n)}$	$2^{0.2557n+o(n)}$	×	✓	heuristic
Three-level sieve [40]	$2^{0.3778n+o(n)}$	$2^{0.2833n+o(n)}$	×	✓	heuristic
Our algorithm	from $2^{0.4150n}$	$2^{0.2075n}$	✓	✓	heuristic
	to $2^{0.3774n}$	$2^{0.2925n}$			

memory. A *closest* vector can be found by enumeration and by the Voronoi-cell algorithm, however, to the best of our knowledge, no sieve algorithm is known to provably solve CVP instances, and it would be interesting to study other sieve algorithms which also work in the CVP case. Table 1 presents the complexities of currently known SVP and CVP algorithms including our new algorithm. In particular, it shows that the asymptotic time complexity of our new approach (slightly) outperforms the complexity of the best pre-existing sieving algorithm and that, as a bonus, it can for the same price serve as a CVP algorithm. The high memory requirement limits the size of accessible dimensions, for example we need 3 TB of storage in dimension 90 which we divide into 25 groups of 120 GB in RAM, and we would need the double in dimension 96. For this reason, the algorithm, as well as other classical sieving techniques, is in practice not competitive with the fastest memoryless methods such as pruned enumeration or aborted BKZ. However, our experiments suggest that despite the higher memory requirements, the sequential running time of our algorithm is of the same order of magnitude as the Gauss sieve, but with an easier to parallelize algorithm.

A long standing open question was to find ways to decrease the complexity of enumeration-based algorithms to a single exponential time complexity. On an LLL- or BKZ-reduced basis [24, 37] the running time of Schnorr-Euchner’s enumeration is doubly exponential in the dimension. If we further reduce the basis to an HKZ-reduced basis [23], the complexity becomes  $2^{\mathcal{O}(n \log n)}$  [21, 18]. Enumeration would become simply exponential if a quasi-orthonormal basis, as defined in Sect. 2, could be found. Unfortunately, most lattices do not possess such a favorable quasi-orthonormal basis. Also for random lattices the lower bound on the Rankin invariant is of size  $2^{\Theta(n \log n)}$  and determines the minimal complexity for enumeration that operates exclusively on the original lattice. We provide a more detailed discussion in Sect. 2.

Our approach circumvents this problem by making use of overlattices that admit a quasi-orthonormal basis. These overlattices are found in polynomial time by an algorithm relying on the structural reduction introduced by [14], as described in Sect. 3.3. Once we have an overlattice and its quasi-orthonormal basis, we may efficiently enumerate short vectors at a constant factor of the first minimum in the overlattice. Our main task is to turn these small samples into a solution vector in the initial lattice. The construction is very similar to an observation by Mordell [31] in 1935 which presented the first algorithmic proof of Minkowski’s inequality using only finite elements. Namely, he observed that given a lattice  $\mathcal{L}_i$  and an overlattice  $\mathcal{L}_{i+1} \supset \mathcal{L}_i$  such that  $[\mathcal{L}_{i+1} : \mathcal{L}_i] = r$ , in any pool of at least  $r + 1$  short vectors of  $\mathcal{L}_{i+1}$ , there exist at least two vectors whose difference is a short non-zero vector in  $\mathcal{L}_i$ . This construction has also been implicitly used in worst-case to average case reductions, where a short overlattice basis is used to sample a pool of short Gaussian overlattice vectors, which are then combined by a SIS (short integer solution) oracle into polynomially longer vectors of the original lattice. In our setting, the overlattice basis is quasi-orthonormal, which allows an efficient enumeration of the shortest overlattice vectors. These vectors are then combined to the shortest vectors of the original

lattice by a concrete, albeit exponential-time, algorithm.

The new algorithm solves SVP and CVP for random lattices in the spirit of a sieving algorithm, except that intermediate vectors lie in overlattices or cosets of overlattices whose geometry vary from dense lattices to quasi-orthogonal lattices. Alternatively, the algorithm can be viewed as an adaptation of the representation technique that solves knapsack problems [4] and decoding problems [25, 5] to the domain of lattices. Due to the richer structure of lattices, the adaptation is far from straightforward. To give a brief analogy, instead of searching for a knapsack solution, assume that we want to find a short vector in an integer lattice. An upper-bound on the Euclidean norm of the solution vector provides a geometric constraint, which induces a very large search space. The short vector we seek can be decomposed in many ways as the sum of two shorter vectors with integer coefficients. Assuming that these sums provide  $N$  different representations of the same solution vector, we can then choose any arbitrary constraint which eliminates all but a fraction  $\approx 1/N$  of all representations. With this additional constraint, the solution vector can still be efficiently found, in a search space reduced by a factor  $N$ . From a broader perspective, this technique can be used to transform a problem with a hard geometric constraint, like short lattice vectors, into an easier subproblem, like short integer vectors (because  $\mathbb{Z}^n$  has an orthonormal basis), together with a custom additional constraint, which is in general linear or modular, which allow an efficient recombination of the solutions to the subproblems. The biggest challenge is to bootstrap the algorithm by finding suitable and easier subproblems using overlattices. We propose a method that achieves this thanks to a well-chosen overlattice allowing an efficient deterministic enumeration of vectors of bounded norm. In this way, we can compute a starting set of vectors that is used to initiate a sequence of recombinations that ends up solving the initially considered problem.

**Our contribution:** We present a new heuristic algorithm for the exact SVP and CVP for  $n$ -dimensional lattices using a tower of  $k$  overlattices  $\mathcal{L}_i$ , where  $\mathcal{L} = \mathcal{L}_0 \subseteq \dots \subseteq \mathcal{L}_k$ . In this tower, we choose the lattice  $\mathcal{L}_k$  at the bottom of the tower in a way that ensures that we can efficiently compute a sufficiently large pool of very short vectors in  $\mathcal{L}_k$ . Starting from this pool of short vectors, we move from each lattice of our tower to the one above using *summation* of vectors while controlling the growth of norms.

For random lattices and under heuristic assumptions, two  $\mathcal{L}_{i+1}$ -vectors sum up to an  $\mathcal{L}_i$ -vector with probability  $\frac{1}{\alpha^n}$ , where  $\text{vol}(\mathcal{L}_i)/\text{vol}(\mathcal{L}_{i+1}) = \alpha^n > 1$ . We allow the norm to increase by a moderate factor  $\alpha$  in each step, in order to preserve the size of our pool of available vectors per lattice in our tower.

Our method can be used to find vectors of bounded norm in a lattice  $\mathcal{L}$  or, alternatively, in a coset  $\mathbf{x} + \mathcal{L}$ ,  $\mathbf{x} \notin \mathcal{L}$ . Thus, in contrast to classical sieving techniques, it allows us to solve both SVP or CVP, and more generally, to enumerate all lattice points within a ball of fixed radius. The average running time in the asymptotic case is  $2^{0.3774n}$ , requiring a memory of  $2^{0.2925n}$ . It is also possible to choose different time-memory tradeoffs and devise slower algorithms that need less memory. We report our experiments on random lattices and SVP challenges of dimension 40 to 90, whose results confirm our theoretical analysis and show that the algorithm works well in practice. We also study various options to parallelize the algorithm and show that parallelization works well on a wide range of computer architectures.

## 2 Background and notation

**Lattices and cosets.** A lattice  $\mathcal{L}$  of dimension  $n$  is a discrete subgroup of  $\mathbb{R}^m$  than spans an  $n$ -dimensional linear subspace. A lattice can be described as the set of all integer combinations  $\{\sum_{i=1}^n \alpha_i \mathbf{b}_i \mid \alpha_i \in \mathbb{Z}\}$  of  $n$  linearly independent vectors  $\mathbf{b}_i$  of  $\mathbb{R}^m$ . Such vectors  $\mathbf{b}_1, \dots, \mathbf{b}_n$  are called a basis of  $\mathcal{L}$ . The volume of the lattice  $\mathcal{L}$  is the volume of  $\text{span}(\mathcal{L})/\mathcal{L}$ , and can be computed as  $\sqrt{|\det(BB^t)|}$ , for any basis  $B$ . Any lattice has a shortest non-zero vector of Euclidean length  $\lambda_1(\mathcal{L})$

which can be upper bounded by Minkowski's theorem as  $\lambda_1(\mathcal{L}) \leq \sqrt{n} \text{vol}(\mathcal{L})^{1/n}$ . We call a *coset* of  $\mathcal{L}$  a translation  $\mathbf{x} + \mathcal{L} = \{\mathbf{x} + \mathbf{v} \mid \mathbf{v} \in \mathcal{L}\}$  of  $\mathcal{L}$  by a vector  $\mathbf{x} \in \text{span}(L)$ .

**Overlattice and index.** A lattice  $\mathcal{L}'$  of dimension  $n$  such that  $\mathcal{L} \subseteq \mathcal{L}'$  is called an overlattice of  $\mathcal{L}$ . The quotient group  $\mathcal{L}'/\mathcal{L}$  is a finite abelian group of order  $\text{vol}(\mathcal{L})/\text{vol}(\mathcal{L}') = [\mathcal{L}' : \mathcal{L}]$ .

**Hyperballs.** Let  $\text{Ball}_n(R)$  denote the ball of radius  $R$  in dimension  $n$  where we omit  $n$  if it is implied by the context. The volume  $V_n$  of the  $n$ -dimensional ball of radius 1 and the radius  $r_n$  of the  $n$ -dimensional ball of volume 1 are respectively:

$$V_n = \frac{\sqrt{\pi}^n}{\Gamma\left(\frac{n}{2} + 1\right)} \text{ and } r_n = V_n^{-1/n} = \sqrt{\frac{n}{2\pi e}}(1 + o(1)).$$

**Gaussian heuristic.** In many cases, when we wish to estimate the number of lattice points in a "nice enough" set  $S$ , we can use the Gaussian heuristic. In this paper, we will quantify the Gaussian Heuristic as follows:

**Heuristic 2.1 ((Gaussian Heuristic))** *There exists a constant<sup>2</sup>  $G_H \geq 1$  such that for all the lattices  $\mathcal{L}$  and all the sets  $S$  that we consider in this paper, the number of points in  $S \cap \mathcal{L}$  satisfies:*

$$\frac{1}{G_H} \cdot \frac{\text{vol}(S)}{\text{vol}(\mathcal{L})} \leq \#(S \cap \mathcal{L}) \leq G_H \cdot \frac{\text{vol}(S)}{\text{vol}(\mathcal{L})}.$$

In fact, it can be proved that for any bounded measurable set  $S$ , the expectation, over random unit volume lattices drawn from the Haar distribution [2], of the number of non-zero points is always the volume of  $S$ . Also, for any fixed lattice of unit volume and any fixed bounded measurable set  $S$ , the expectation of the number of lattice points in  $\mathbf{t} + S$  for uniform  $\mathbf{t}$  modulo  $\mathcal{L}$  is exactly  $\text{vol}(S)$ . However, fewer results are known about the standard deviation, or whether these distributions are concentrated enough around the expectation so that almost all instances satisfy the upper and lower bounds.

In this paper, the geometry of lattices vary between random integer lattices of large enough volume and quasi-orthonormal lattices. We will assume that in these lattices, the length  $\lambda_1$  of the shortest vector is the one given by the Gaussian heuristic, i.e. the radius of a ball of volume  $\text{vol}(\mathcal{L})$ :  $\lambda_1(\mathcal{L}) \approx r_n \cdot \sqrt[n]{\text{vol}(\mathcal{L})}$ . Furthermore, the sets  $S$  we consider in this paper are either balls of radius larger than  $\sqrt{3/2} r_n \sqrt[n]{\text{vol}(\mathcal{L})}$ , and whose center is uniform modulo the lattice (*i.e.* far from  $\mathbf{0}$ ), or the intersection of two of such balls whose centers are close enough. In these cases, our experiments indicate that the number of lattice points in these sets are almost always between 50% and 110% of  $\text{vol}(S)/\text{vol}(\mathcal{L})$ , thus Heuristic 2.1 holds in practice for  $G_H = 2$ .

**Gram-Schmidt orthogonalization (GSO).** The GSO of a non-singular square matrix  $B$  is the unique decomposition as  $B = \mu \cdot B^*$ , where  $\mu$  is a lower triangular matrix with unit diagonal and  $B^*$  consist of mutually orthogonal rows. For each  $i \in [1, n]$ , we call  $\pi_i$  the orthogonal projection over  $\text{span}(\mathbf{b}_1, \dots, \mathbf{b}_{i-1})^\perp$ . In particular, one has  $\pi_i(\mathbf{b}_i) = \mathbf{b}_i^*$ , which is the  $i$ -th row of  $B^*$ . We use the notation  $B_{[i,j]}$  for the projected block  $[\pi_i(\mathbf{b}_i), \dots, \pi_i(\mathbf{b}_j)]$ .

**Rankin factor and quasi-orthonormal basis.** Let  $B$  be an  $n$  dimensional basis of a lattice  $\mathcal{L}$ , and  $j \leq n$ . We call the ratio

$$\gamma_{n,j}(B) = \frac{\text{vol}(B_{[1,j]})}{\text{vol}(\mathcal{L})^{j/n}} = \frac{\text{vol}(\mathcal{L})^{(n-j)/n}}{\text{vol}(\pi_{j+1}(\mathcal{L}))}$$

<sup>2</sup>The algorithms and proofs would also work with  $G_H = \text{poly}(n)$  or  $G_H = (1 + \epsilon)^n$ , giving slightly worse complexities.

the Rankin factor of  $B$  with index  $j$ . The well known Rankin invariants of the lattice,  $\gamma_{n,j}(\mathcal{L})$ , introduced by Rankin [35] are simply the squares of the minimal Rankin factors of index  $j$  over all bases of  $\mathcal{L}$ . This allows to define a *quasi-orthonormal* basis.

**Definition 2.2 ((quasi-orthonormal basis))** *A basis  $B$  is quasi-orthonormal if and only if its Rankin factors satisfy  $1 \leq \gamma_{n,j}(B) \leq n$  for all  $j \in [1, n]$ .*

In the above definition, we chose the upper-bound  $n$  over a general  $\text{poly}(n)$  only because we are able to achieve this factor. More generally, any polynomial function would be sufficient for the asymptotical analysis and for the running time. For example, any real triangular matrix with identical diagonal coefficients forms a quasi-orthogonal basis. More generally, any basis whose  $\|\mathbf{b}_i^*\|$  are almost equal is quasi-orthogonal. This is a very strong notion of reduction, since average LLL-reduced or BKZ-reduced bases only achieve a  $2^{\mathcal{O}(n^2)}$  Rankin factor and HKZ-reduced bases of random lattices have a  $2^{\mathcal{O}(n \log n)}$  Rankin factor. Finally, Rankin's invariants are lower-bounded [6, 38, 13] by  $2^{\Theta(n \log n)}$  for almost all lattices<sup>3</sup>, which means that only lattices in a tiny subclass possess a quasi-orthonormal basis.

**Schnorr-Euchner enumeration** Given a basis  $B$  of an integer lattice  $\mathcal{L} \subseteq \mathbb{R}^n$ , Schnorr-Euchner's enumeration algorithm [37] allows to enumerate all vectors of Euclidean norm  $\leq R$  in the bounded coset  $C = (\mathbf{z} + \mathcal{L}) \cap \text{Ball}_n(R)$  where  $\mathbf{z} \in \mathbb{R}^n$ . The running time of this algorithm is

$$\mathcal{T}_{SE} = \sum_{i=1}^n \#(\pi_{n+1-i}(\mathbf{z} + \mathcal{L}) \cap \text{Ball}_i(R)) \quad , \quad (1)$$

which is equivalent to

$$\mathcal{T}_{SE} \approx \sum_{i=1}^n \frac{\text{vol}(\text{Ball}_i(R))}{\text{vol}(\pi_{n+1-i}(\mathcal{L}))} \quad (2)$$

under Heuristic 2.1. The last term in the sums (1) and (2) denotes the number of solutions  $\#C$ . Thus, the complexity of enumeration is approximately  $\mathcal{T}_{SE} \approx \tilde{\mathcal{O}}(\#C) \cdot \max_{j \in [1, n]} \gamma_{n,j}(B)$ . This is why a reduced basis of smallest Rankin factor is favorable. The lower bound on Rankin's invariant of  $\gamma_{n,n/2}(\mathcal{L}) = 2^{\Theta(n \log n)}$  for most lattices therefore determines the minimal complexity of enumeration that is achievable while working with the original lattice, provided that one can actually compute a basis of  $\mathcal{L}$  minimizing the Rankin factors, which is also NP-hard. If the input basis is quasi-orthonormal, the upper-bound  $\gamma_{n,j}(B) \leq n$  from Definition 2.2 implies that the enumeration algorithm runs in time  $\tilde{\mathcal{O}}(\#C)$ , which is optimal. Without knowledge of a good basis one can aim to decompose the problem into more favorable cases that finally allow to apply Schnorr-Euchner's algorithm as we describe in the following.

### 3 Enumeration of short vectors by intersection of hyperballs

The section presents the new algorithm that enumerates  $\beta^n$  shortest vectors in any coset  $\mathbf{t} + \mathcal{L}$  of a lattice  $\mathcal{L}$  for a constant  $\beta \approx \sqrt{3/2}$ . It can be used to solve the NP-hard problems SVP, CVP, ApproxSVP $_{\beta}$  and ApproxCVP $_{\beta}$ : Given a lattice  $\mathcal{L}$ , the SVP can be reduced to enumerating vectors of Euclidean norm  $\mathcal{O}(\lambda_1(\mathcal{L}))$  in the coset  $\mathbf{0} + \mathcal{L}$  while a CVP instance can be solved by enumerating vectors of norm at most  $\text{dist}(\mathbf{t}, \mathcal{L})$  in the coset  $-\mathbf{t} + \mathcal{L}$ . These bounded cosets,  $(\mathbf{t} + \mathcal{L}) \cap \text{Ball}_n(R)$  for suitable radius  $R$ , can be constructed in an iterative way by use of overlattices. The searched vectors are expressed as a sum of short vectors of suitable translated overlattices of smaller volume. The search for a unique element in a lattice as required in the SVP or CVP is delegated to the problem of enumerating bounded cosets. Any non-trivial element found by our algorithm is naturally a solution to the corresponding ApproxSVP $_{\beta}$  or ApproxCVP $_{\beta}$ .

<sup>3</sup> $\gamma_{2n,n}(\mathcal{L}) \geq (n/12)^n$  with probability  $\approx 1$  on random real lattices of volume 1 drawn from the Haar distribution.

We present the new algorithm solving lattice problems based on intersections of hyperballs in Section 3.1 and the generic initialization of our algorithm as described in Section 3.3. Finally, Section 3.4 describes the cost of the first step in the algorithm.

### 3.1 General description of the new algorithm

Assume that we are given a tower of  $k = \mathcal{O}(n)$  lattices  $\mathcal{L}_i \subset \mathbb{R}^n$  of dimension  $n$  where  $\mathcal{L}_i \subseteq \mathcal{L}_{i+1}$  and the volume of any two consecutive lattices differs by a factor  $N = \lceil \alpha^n \rceil \in \mathbb{N}_{>1}$ . We also assume that the bottom lattice  $\mathcal{L}_k$  permits an efficient enumeration of the  $\beta^n$  shortest vectors in any coset  $\mathbf{t} + \mathcal{L}_k$  for  $1 < \beta < \sqrt{3/2}$ . The ultimate goal is to find the  $\beta^n$  shortest vectors in some coset  $\mathbf{t}_0 + \mathcal{L}_0$  of  $\mathcal{L}_0$ . We postpone how to find suitable lattices  $\mathcal{L}_i$ ,  $i \geq 1$ , to the following two sections.

We also assume in this section, that the Gaussian heuristic (Heuristic 1) holds. Under this assumption, the problem of finding the  $\beta^n$  shortest elements in some coset  $\mathbf{t} + \mathcal{L}$  is roughly equivalent to enumerating all lattice vectors of  $\mathcal{L}$  in the ball of radius  $\beta \cdot r_n \cdot \sqrt[n]{\text{vol}(\mathcal{L})}$  centered at  $-\mathbf{t} \in \mathbb{R}^n$ .

Each step for  $i = k - 1$  down to 0 of the algorithm is based on the following intuition: We are given the  $\approx \beta^n$  shortest vectors  $\mathbf{v}_j$  in  $\mathbf{t}_i/2 + \mathcal{L}_{i+1}$ . By summation, we can then find vectors  $(\mathbf{v}_j + \mathbf{v}_l)_{j \leq l}$  that lie in  $\mathbf{t}_i + \mathcal{L}_{i+1}$ . We select those who actually lie in  $\mathbf{t}_i + \mathcal{L}_i$  and whose norm is small enough, and consider them as the input pool for the next step. For suitable parameters, namely  $\alpha$  small enough and  $\beta$  large enough, we thus recover the  $\approx \beta^n$  shortest vectors of  $\mathbf{t}_i + \mathcal{L}_i$ .

More precisely, for each  $i \in [0, k]$ , we call  $C_i$  the *bounded coset*  $C_i$  that contains the  $\beta^n$  shortest vectors of the coset  $\mathbf{t}_i + \mathcal{L}_i$  where  $\mathbf{t}_i = \mathbf{t}_0/2^i \in \mathbb{R}^n$ . More formally, let us define

$$R_i = \beta \cdot r_n \sqrt[n]{\text{vol}(\mathcal{L}_i)} \text{ and } C_i = (\mathbf{t}_i + \mathcal{L}_i) \cap \text{Ball}(\mathbf{0}, R_i)$$

such that

$$\#C_i \approx \text{vol}(\text{Ball}(R_i))/\text{vol}(\mathcal{L}_i) = \beta^n \text{ ,}$$

which follows from the Heuristic 2.1. In addition, we recall that

$$\mathcal{L}_i \subset \mathcal{L}_{i+1} \text{ where } \text{vol}(\mathcal{L}_i)/\text{vol}(\mathcal{L}_{i+1}) = \lceil \alpha^n \rceil \text{ .}$$

In order to enumerate  $C_0$ , our algorithm successively enumerates  $C_i$ , starting from  $i = k$  down to zero, Figure 1 illustrates the sequence of enumerated lists.

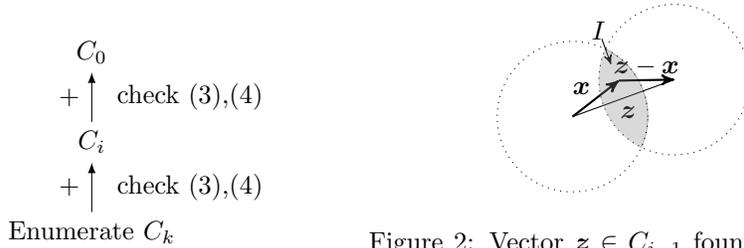


Figure 1: Iterative creation of lists.

Figure 2: Vector  $\mathbf{z} \in C_{i-1}$  found as sum between  $\mathbf{x} \in C_i$  and  $\mathbf{z} - \mathbf{x} \in C_i \Leftrightarrow I \cap (\mathbf{t}_i + \mathcal{L}_i) \neq \emptyset$ .

During the construction of the tower of lattices, which is studied in the next sections, we already ensure that  $C_k$  is easy to obtain. We now explain how we can compute  $C_{i-1}$  from  $C_i$ . To do this, we compute all sums  $\mathbf{x} + \mathbf{y}$  of vector pairs of  $C_i \times C_i$  which satisfy the conditions

$$\mathbf{x} + \mathbf{y} \in \mathbf{t}_{i-1} + \mathcal{L}_{i-1} \text{ and} \tag{3}$$

$$\|\mathbf{x} + \mathbf{y}\| \leq \beta \cdot r_n \cdot \sqrt[n]{\text{vol}(\mathcal{L}_i)} \text{ .} \tag{4}$$

This means that we collect the  $\beta^n$  shortest vectors of the coset  $C_{i-1} = \mathbf{t}_{i-1} + \mathcal{L}_{i-1}$  by going through  $C_i = \mathbf{t}_i + \mathcal{L}_i$ . In practice, an equivalent way to check if condition (3) holds, is to use an efficient computation for the map  $\varphi_{i-1} : C_i \rightarrow \mathcal{L}_i/\mathcal{L}_{i-1}$ ,  $\mathbf{z} \rightarrow \mathbf{z} - \mathbf{t}_i \pmod{\mathcal{L}_{i-1}}$  and to verify that  $\varphi_{i-1}(\mathbf{x}) + \varphi_{i-1}(\mathbf{y}) = 0$ . Algorithm 1 summarizes our approach.

---

**Algorithm 1** Coset enumeration
 

---

Constants:  $\alpha \approx \sqrt{4/3}, \beta \approx \sqrt{3/2}$  Parameters:  $k$

**Input:** A LLL-reduced basis  $B$  of  $\mathcal{L}_0$  and a center  $\mathbf{t} \in \mathbb{R}^n$

**Output:** Elements of  $\mathbf{t} + \mathcal{L}_0$  of norm  $\leq R_0 = \beta r_n \text{vol}(\mathcal{L}_0)^{1/n}$

- 1: Randomize the input target by sampling  $\mathbf{t}_0 \in \mathbf{t} + \mathcal{L}$ . Use for example a Discrete Gaussian Distribution of parameter  $\sqrt{n}\|B^*\|$ . This defines all the sub-targets  $\mathbf{t}_i = \mathbf{t}_0/2^i$
  - 2: Compute a tower of lattices  $\mathcal{L}_0, \dots, \mathcal{L}_k$  by use of Alg. 3 such that
    - $\mathcal{L}_0 \subset \mathcal{L}_1 \subset \dots \subset \mathcal{L}_k$  and  $\text{vol}(\mathcal{L}_i)/\text{vol}(\mathcal{L}_{i-1}) = N = \lceil \alpha^n \rceil$
    - lattice enumeration is easy on  $\mathcal{L}_k$
    - testing-morphisms  $\varphi_{i-1}$  from  $\mathbf{t}_i + \mathcal{L}_i$  to  $\mathcal{L}_i/\mathcal{L}_{i-1}$  are efficient to evaluate.
  - 3: Enumerate bottom coset  $C_k$  (with Schnorr-Euchner)
  - 4: **for**  $i = k - 1$  **downto** 0 **do**
  - 5:      $C_i \leftarrow \text{Merge}(C_{i+1}, \varphi_i, R_i = \beta r_n \text{vol}(\mathcal{L}_i)^{1/n})$  (Alg. 2)
  - 6: **end for**
  - 7: **return**  $C_0$
- 

A naive implementation of the merge routine that creates  $C_{i-1}$  from  $C_i$  would just run through the  $\beta^{2n}$  pairs of vectors from  $C_i \times C_i$ , and eliminate those that do not satisfy the constraints (3) and (4). By regrouping the elements of  $C_i$  into  $\alpha^n$  buckets, according to their value modulo  $\mathcal{L}_{i-1}$ , condition (3) implies that each element of  $C_i$  only needs to be paired with the elements of a single bucket, see Algorithm 2. Heuristic 2.1 implies that each bucket contains at most  $G_H(\beta/\alpha)^n$  elements, therefore the merge operation can then be performed in time  $G_H^2(\beta^2/\alpha)^n$ .

---

**Algorithm 2** Merge by collision
 

---

*/\* Efficiently find pairs of vectors of  $C_{i+1}$  s.t. their sum is in  $C_i$  \*/*

*/\*  $C_i$  denotes  $(\mathbf{t}_i + \mathcal{L}_i) \cap \text{Ball}(R_i)$  \*/*

**Input:** The bounded coset  $C_{i+1}$ , a testing morphism  $\varphi_i$  and a radius  $R_i$

**Output:** The bounded coset  $C_i$

- 1:  $C_i \leftarrow \emptyset$
  - 2: Reorganize  $C_{i+1}$  into buckets indexed by the values of  $\varphi_i$
  - 3: **for each**  $\mathbf{v} \in C_{i+1}$  **do**
  - 4:     **for each**  $\mathbf{u}$  in the bucket of index  $-\varphi_i(\mathbf{v})$  **do**
  - 5:         **if**  $\|\mathbf{u} + \mathbf{v}\| \leq R_i$  **then**
  - 6:              $C_i \leftarrow C_i \cup \{\mathbf{u} + \mathbf{v}\}$
  - 7:         **end if**
  - 8:     **end for**
  - 9: **end for**
  - 10: **return**  $C_i$
- 

**Complexity and constraints for parameters  $\alpha$  and  $\beta$ .** We now prove the complexity and correctness of Algorithm 2.

**Theorem 3.1** *Assuming Heuristic 2.1, and provided that  $\beta^n \geq G_H \sqrt{n}/0.692 \sqrt{1 - \alpha^2/4}^n$ , then: Given as input the bounded coset  $C_{i+1}$ , Alg. 2 outputs the coset  $C_i$  within  $G_H^2(\beta^2/\alpha)^n$  Euclidean norm computations. The memory is bounded by the size of the input and output:  $G_H \beta^n$   $n$ -dimensional vectors.*

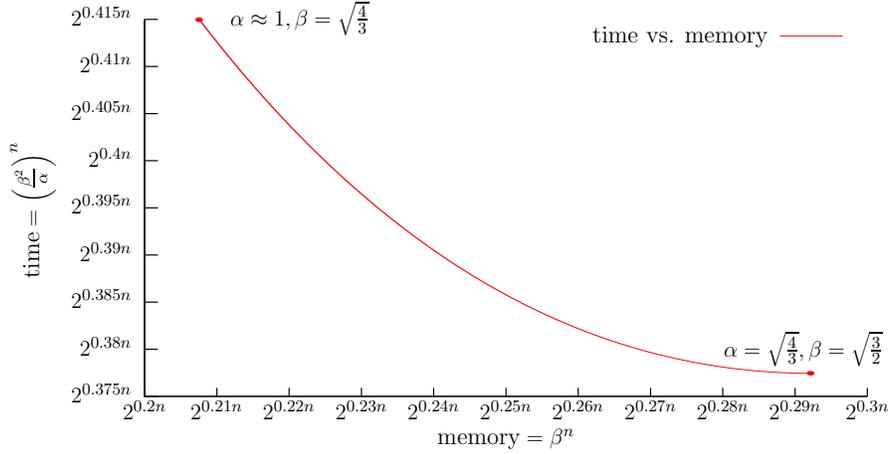


Figure 3: Trade-off between memory and time for varying choices of  $\alpha$  and  $\beta$ .

**Proof:** It is clear that at each level, conditions (3) and (4) imply that Alg. 2 outputs a subset of  $C_i$ . We now need to prove that there exist constants  $\alpha$  and  $\beta$  such that all points of  $C_i$  are present in the output. Equivalently, all points of  $C_i$  must be expressed as the sum of two points in  $C_{i+1}$ , see Fig. 2 for an illustration. This geometric constraint can be simply rephrased as follows: a vector  $\mathbf{z} \in C_i$  is found if and only if there exists at least one vector  $\mathbf{x}$  of the coset  $\mathbf{t}_{i+1} + \mathcal{L}_{i+1}$  in the intersection of two balls of radius  $R_{i+1}$ , the first one centered in  $\mathbf{0}$ , and the second one in  $\mathbf{z}$ . It is clear that  $\mathbf{z} - \mathbf{x} \in C_{i+1} = \mathbf{t}_{i+1} + \mathcal{L}_{i+1}$  since  $2\mathbf{t}_{i+1} = \mathbf{t}_i$  and  $\mathcal{L}_i \subseteq \mathcal{L}_{i+1}$ . So if there is a point  $\mathbf{x} \in C_{i+1}$  in the intersection  $I = \text{Ball}(\mathbf{0}, R_{i+1}) \cap \text{Ball}(\mathbf{z}, R_{i+1})$ , we obtain  $\mathbf{z} \in C_i$  as a sum between  $\mathbf{x} \in C_{i+1}$  and  $\mathbf{z} - \mathbf{x} \in C_{i+1}$ . Under Heuristic 2.1, this occurs as soon as the intersection  $I$  of the two balls has a volume larger than  $G_H \text{vol}(\mathcal{L}_{i+1})$ . We thus require that  $\text{vol}(I) / \text{vol}(\mathcal{L}_{i+1}) \geq G_H$ .  $\square$

From Lemma A.1 and its corollary in the appendix, we derive that the intersection of two balls of radius  $R_i$  at distance at most  $R_{i-1} = \alpha R_i$  is larger than  $0.692 \cdot \text{vol}(\text{Ball}(R_i \cdot \sqrt{1 - (\alpha/2)^2})) / \sqrt{n}$ . A sufficient condition on  $\alpha$  and  $\beta$  is then

$$\left(\beta \cdot \sqrt{1 - (\alpha/2)^2}\right)^n \geq G_H \sqrt{n} / 0.692 \text{ or alternatively} \quad (5)$$

$$\beta \sqrt{1 - (\alpha/2)^2} \geq (1 + \varepsilon_n) \quad (6)$$

where  $\varepsilon_n = (G_H \sqrt{n} / 0.692)^{1/n} - 1$  decreases towards 0 when  $n$  grows.

Of course, for optimization reasons, we want to minimize the size of the lists  $\beta^n$ , and the number of steps  $(\beta^2/\alpha)^n$  in the merge. Therefore we want to minimize  $\beta$  and maximize  $\alpha$  under the above constraint. The total running time of Alg. 1 is given by  $\mathcal{B} + \text{poly}(n) (\beta^2/\alpha)^n$  where  $\mathcal{B}$  represents the running time of the initial enumeration at level  $k$  (details in Sect. 3.4). For optimal parameters, inequality (6) is in fact an equality. Asymptotically, the shortest running time occurs for  $\alpha = \sqrt{4/3}$  and  $\beta = \sqrt{3/2}$  for which a merge costs around  $(\beta^2/\alpha)^n \approx 2^{0.3774n}$  and the size of the lists is  $\beta^n \approx 2^{0.2925n}$ .

**Time-memory trade-off.** Other choices of  $\alpha$  and  $\beta$  that satisfy (6) provide a trade-off between running time and required memory. Figure 3 shows the logarithmic size of the lists the algorithm needs to store depending on the time one is willing to spend. If one has access to only  $\beta^n \approx 2^{0.21n}$  in memory, the time complexity increases to  $(\beta^2/\alpha)^n \approx 2^{0.41n}$ . In practice, we choose  $\alpha > 1$  and  $\beta > 0$  satisfying (5) with the constraint that  $\alpha^n$  is integer.

### 3.2 Example for co-cyclic lattices or $q$ -ary lattices.

We now give a simple intuition on how we could define the overlattice tower in the case of random co-cyclic lattices and  $q$ -ary lattices. These examples help to understand the idea that even for hard lattices, it is fairly easy to find quasi-orthonormal bases in overlattices. In the next section, we will present a more general method to create randomized overlattices, which performs well in practice for all types of lattices, including cocyclic or  $q$ -ary lattices, and ensures the estimated complexity as denoted in Sect. 3.1 which is based on Heuristic 2.1.

In the following description, the tower of lattices remains implicit in the sense that we do not need to find a basis for each of the  $k+1$  lattices  $\mathcal{L}_i$ . We only need a description of the initial and the bottom lattice as we test membership to a coset by evaluating  $\varphi_i$ .

Let  $\mathcal{L} \subseteq \mathbb{Z}^n$  be a co-cyclic lattice given as  $\mathcal{L} = \{\mathbf{x} \in \mathbb{Z}^n, \sum_{i=1}^n a_i x_i = 0 \pmod{M}\}$  for large  $M \in \mathbb{N}$  and random integers  $a_1, \dots, a_n \in [0, M-1]$ . The task is to enumerate  $C = (\mathbf{t} + \mathcal{L}) \cap \text{Ball}_n(R)$  where  $R = \beta \cdot r_n \cdot \text{vol}(\mathcal{L})^{1/n}$  for a given  $\beta > 1$ . For  $k = \mathcal{O}(n)$ , the connection with random subset sum instances, as well as newer adaptations of worst-case to average case proofs (see [14]) support the claim that random instances are hard. Choose  $\alpha$  such that  $M = \alpha^{nk} \in \mathbb{N}$  and define  $N = \alpha^n \in \mathbb{N}$ . We can naturally define the tower consisting of lattices

$$\mathcal{L}_i = \{\mathbf{y} \in \mathbb{Z}^n, \sum_{i=1}^n a_i y_i = 0 \pmod{N^{k-i}}\} .$$

At the level  $k$ , we have  $\mathcal{L}_k = \mathbb{Z}^n$  so that we can efficiently enumerate any coset  $C$  by use of the Schnorr-Euchner algorithm [37] in time  $\text{poly}(n) \cdot |C|$  as we argue in Sect. 2. The coset testing function  $\varphi_i$ , which represents  $\mathbf{x} - \mathbf{t}_i \pmod{\mathcal{L}_{i-1}}$ , can be implemented as  $\langle \mathbf{a}, \mathbf{x} - \mathbf{t}_i \rangle / N^{k-i} \pmod{N}$ .

A second example is the class of  $q$ -ary lattices. Let  $\mathcal{L}$  be the lattice of dimension  $n$  and volume  $q^k$  such that for  $\mathbf{x} \in \mathbb{Z}^n$ ,

$$\mathbf{x} \in \mathcal{L} \iff [(a_{1,1}x_1 + \dots + a_{1,n}x_n \equiv_q 0) \wedge \dots \wedge (a_{k,1}x_1 + \dots + a_{k,n}x_n \equiv_q 0)] \quad (7)$$

where  $a_{i,j}$  are uniform in  $\mathbb{Z}/q\mathbb{Z}$ . For  $q = \alpha^n$  classical worst-case to average-case reductions prove that these lattices provide difficult lattice problems on average [1]. Here, a lattice  $\mathcal{L}_i$  could be defined as the lattice satisfying the last  $i$  equations of (7). Again,  $\mathcal{L}_k$  is  $\mathbb{Z}^n$ ,  $\mathcal{L}_{i-1} \subseteq \mathcal{L}_i$  and  $\text{vol}(\mathcal{L}_{i-1})/\text{vol}(\mathcal{L}_i) = q$ . The coset testing function  $\varphi_i$  can be computed as  $\langle \mathbf{a}_i, \mathbf{x} - \mathbf{t}_i \rangle \pmod{q}$ .

As elegant as it may seem, these simple towers of lattices are not as efficient as one could expect, because the top overlattice is  $\mathbb{Z}^n$ , and the Gaussian heuristic does not apply to its bounded coset  $C_k = \mathbb{Z}^n \cap \text{Ball}_n(R_k)$ , whose radius  $R_k$  is too close to  $\sqrt{n}$ . Indeed, the number of points of  $\mathbb{Z}^n$  in a ball of radius  $R_k \approx \sqrt{n}$  varies by exponential factors depending on the center of the ball [26]. If the target is very close to 0, like in an SVP-setting, the coset  $C_k$  contains around  $2^{0.513n}$  vectors<sup>4</sup>, which differs considerably from  $\beta^n \approx 2^{0.292n}$  that we could expect of a random lattice. The initial coset would be very costly to store already in moderate dimensions.

Even if we store only a fraction of the bottom coset, Heuristic 2.1 would prevent the first merge by collision from working. Indeed, it relies on the number of points in intersections of balls of radius  $R_k$  centered in an exponential number of different points. Unfortunately, balls of radius  $R_k$  centered in random points contain an exponentially smaller number of integer vectors than  $\beta^n$ , and their intersections contain in general no integer point at all. Thus the collision by merge would fail to recover  $C_{k-1}$ .

This means that because of the Gaussian heuristic, the  $\mathbb{Z}^n$  lattice should never be used as the starting point of an overlattice tower. Fortunately, random quasi-orthonormal lattices are a valid replacement of  $\mathbb{Z}^n$ , as our experiments show. Furthermore, we can still build in polynomial time a tower of lattices ending with a quasi-orthonormal basis. We give the details of a generic generation of a tower of lattices in the following section.

<sup>4</sup>Computation based on saddle point method as in [26] for a radius  $\sqrt{\beta^2/(2\pi e) \cdot n} \approx \sqrt{0.0878 \cdot n}$ .

### 3.3 Generic creation of the tower

For any integer lattice  $\mathcal{L}$ , the simplest choice of an overlattice is  $\mathcal{L}' = \mathbb{Z}^n$ . However, this creates two problems.

1.  $\mathbb{Z}^n$  does not satisfy our quantified version of the Gaussian heuristic 2.1, as it would require an  $n$ -exponential  $G_H$ .
2. The index  $[\mathbb{Z}^n : \mathcal{L}]$ , which is the volume of  $\mathcal{L}$  might be too large. Indeed, our sieve requires that the index is simply exponential but an integer basis with polynomial entries may have a superexponential volume  $2^{\text{poly}(n)}$ .

For the first problem, it suffices to replace the square lattice  $\mathbb{Z}^n$  by a quasi-orthogonal lattice  $\mathcal{L}'$ . Indeed, although the expectation of the number of lattice points in a randomly centered ball of radius  $r_n \beta \text{vol}(\mathcal{L}')^{1/n}$  is always  $\beta^n$ , their repartition is much more concentrated around the expectation when the lattice is quasi-orthonormal than when it is  $\mathbb{Z}^n$ . In practice, almost all of these balls contain at least  $\beta^n/2$  lattice points, and Heuristic 2.1 is valid with  $G_H = 2$ . The most noticeable exception is when the center is too close to a lattice point. In this very rare case, the number of points would exceed the upper-bound by an exponential factor. Luckily, the lower-bound in Heuristic 2.1 is more important than the upper-bound: the lower-bound is used an exponential number of times in Eq. (5) to prove the correctness of the merge, whereas the upper-bound is used only with a polynomial number of different centers, to obtain the time and memory complexities. Thus bad centers are easy to avoid.

For the second problem, one can see that the difficulty of creating the sieve is not just to find a quasi-orthonormal overlattice (which is always possible), the difficulty is to find one of small index. That is why we need a several overlattices instead of just one.

We now present a generic method of computing the tower of  $\mathcal{L}_i$ 's that works well in practice for high dimensions as we have verified in our experiments. Algorithm 3 summarizes the following steps.

We take as input a randomized LLL-reduced or BKZ-30-reduced basis  $B$  of an  $n$ -dimensional lattice  $\mathcal{L}$ . We choose constants  $\alpha > 1$  and  $\beta > 0$  satisfying equation (6) with the additional constraint that  $N = \alpha^n$  is an integer.

The Gram-Schmidt coefficients of  $B$  usually decrease geometrically, and we can safely assume that  $\min_i \|\mathbf{b}_i^*\| \geq \max_i \|\mathbf{b}_i^*\| / \sqrt{4/3}^n$ . Otherwise, the LLL-reduced basis would immediately reveal a sublattice of dimension  $< n$  containing the shortest vectors of  $\mathcal{L}$ . This means that there exists a smallest integer  $k = \mathcal{O}(n)$  such that  $\min_{i \in [1, n]} \|\mathbf{b}_i^*\| \geq \left(\frac{\text{vol}(\mathcal{L})}{N^k}\right)^{\frac{1}{n}} = \sigma$ . The integer  $k$  determines the number of levels in our tower and  $\sigma$  is the  $n$ -th root of the volume of the last overlattice  $\mathcal{L}_k$ .

---

**Algorithm 3** Compute the tower of overlattices

---

**Input:**  $B$  a (randomized) LLL-reduced basis of  $\mathcal{L}$  of dimension  $n$

**Output:** Bases  $B^{(i)}$  of a tower of overlattices  $\mathcal{L} = \mathcal{L}_0 \subset \dots \subset \mathcal{L}_k$ . Note that given a target  $\mathbf{t}_{i+1}$ , the testing morphism  $\varphi_i$  from  $\mathbf{t}_{i+1} + \mathcal{L}_{i+1}$  to  $\mathbb{Z}_N$  is implicitly defined by  $\varphi_i \left( \mathbf{t}_{i+1} + \sum_{j=1}^n \mu_j \mathbf{b}_j^{(i+1)} \right) = \mu_1 \pmod N$

- 1: Let  $N = \lceil \alpha^n \rceil$ .
  - 2: Let  $k$  be the smallest integer s.t.  $N^k \geq \text{vol}(\mathcal{L}) / \min_i \|\mathbf{b}_i^*\|^n$ .
  - 3: Let  $\sigma = (\text{vol}(\mathcal{L}) / N^k)^{\frac{1}{n}}$ , thus  $\sigma \leq \min_i \|\mathbf{b}_i^*\|$ .
  - 4: Apply Alg. 4 on input  $(B, \sigma)$  to find a basis  $\hat{B} = [\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \dots, \hat{\mathbf{b}}_n]$  of  $\mathcal{L}$ .
  - 5:  $B^{(i)} \leftarrow \left[ \frac{\hat{\mathbf{b}}_1}{N^i}, \hat{\mathbf{b}}_2, \dots, \hat{\mathbf{b}}_n \right]$  foreach  $i \in [0, k]$
  - 6: **return**  $B^{(i)}$  for all  $i$
- 

It remains to find the tower  $(\mathcal{L}_i)_{i \in [1, k]}$  of overlattices of  $\mathcal{L}$ , together with a quasi-orthonormal basis  $B^{(k)}$  of  $\mathcal{L}_k$ , given a structural condition  $\mathcal{L}^{(i)} / \mathcal{L}^{(i-1)} \simeq \mathbb{Z} / N\mathbb{Z}$  (Alg. 3). This problem is closely related to the structural reduction, introduced in [14], which aims at finding a short basis  $\bar{B}$  of an overlattice

$\bar{\mathcal{L}}$  such that  $\bar{\mathcal{L}}/\mathcal{L}$  is isomorphic to some fixed abelian group  $G$ . However, the primary goal of [14] was to decrease the Gram-Schmidt norm of  $\bar{B}$  in order to sample a pool of Gaussian overlattice vectors of norm  $\Theta(\sqrt{n \log n} \|\bar{B}^*\|)$ . These vectors would be too large for our purpose, since the bottom level of our decomposition algorithm needs a pool of vectors of length  $\Theta(\sqrt[n]{\text{vol}(\bar{\mathcal{L}})})$ .

In the present paper, we prove that when the group  $G$  is large enough, the *unbalanced reduction* of [14] can in fact efficiently construct a basis  $C$  of  $\mathcal{L}$  such that  $[\mathbf{c}_1/N^k, \mathbf{c}_2, \dots, \mathbf{c}_n]$  is quasi-orthonormal. This naturally defines the tower of  $k+1$  overlattices  $\mathcal{L}_i$ , where  $\mathcal{L}_i$  is generated by the corresponding basis  $B^{(i)} = [\frac{\mathbf{c}_1}{N^i}, \mathbf{c}_2, \dots, \mathbf{c}_n]$  for  $i = 0, \dots, k$ . Then, the Gaussian sampling algorithm on  $\mathcal{L}_k$  can be replaced by Schnorr-Euchner's enumeration – with or without pruning – using  $B^{(k)}$ , and thus, the norm of the overlattice vectors can be decreased to  $r_n \beta \sqrt[n]{\text{vol}(\mathcal{L}_k)}$ . For completeness, we recall in Appendix the pseudo-code of the unbalanced reduction from [14]. Here, we prove that it allows to produce a quasi-orthonormal basis. Compared to [14], we added the condition that  $\sigma \leq \min \|\mathbf{b}_i^*\|$  on the input parameters, and consequently, one of the test cases in the main loop of the algorithm in [14] never occurs.

Theorem 3.2 and its corollary below prove that running the unbalanced reduction with a smaller parameter  $\sigma$  than what is considered in [14] allows to construct a quasi-orthonormal overlattice basis in polynomial time. Note that in [14], the goal was to find an overlattice basis whose Gram-Schmidt length was smaller than  $\sigma$  without any constraints on their Rankin factors. Here, the first vector of the overlattice basis may be larger than  $\sigma$ , we just ensure that the  $n$ -th root of the volume is exactly  $\sigma$  and that all Rankin factors are polynomially bounded. Thus, the main Equation (10) of Theorem 3.2 cannot be directly deduced from [14], and we provide a full proof in the appendix.<sup>5</sup>

**Theorem 3.2 ((Unbalanced reduction))** *Let  $\mathcal{L}(B)$  be an  $n$ -dimensional integer lattice with an LLL-reduced basis  $B = [b_1, \dots, b_n]$ . Let  $\sigma$  be a target length  $\leq \min \|\mathbf{b}_i^*\|$ . Algorithm 4 outputs in polynomial time a basis  $C$  of  $\mathcal{L}$  satisfying*

$$\|\mathbf{c}_i^*\| \leq \sigma \text{ for all } i \in [2, n] \quad (8)$$

$$\|\mathbf{c}_1\| \leq \sigma n \cdot \text{vol}(\mathcal{L}) / \sigma^n \quad (9)$$

$$\frac{\sigma^{n+1-i}}{\text{vol}(C_{[i,n]})} \leq n+1-i \text{ for all } i \in [2, n] \quad (10)$$

Since  $\sigma$  is by construction the  $n$ -th root of the bottom lattice  $\mathcal{L}_k$ , we immediately deduce the following elementary corollary, which proves that Algorithm 3 computes a tower of overlattices suitable for the decomposition algorithm.

**Corollary 1** *Given as input a LLL-reduced basis  $B$  of  $\mathcal{L}$  such that  $\max \|\mathbf{b}_i^*\| / \min \|\mathbf{b}_i^*\| \leq 2^{\mathcal{O}(n)}$ , Algorithm 3 outputs a sequence of bases  $B^{(0)}, \dots, B^{(k)}$  such that  $B^{(0)}$  is a basis of  $\mathcal{L}$ ,  $B^{(k)}$  is quasi-orthogonal, and  $\mathcal{L}(B^{(i)}) / \mathcal{L}(B^{(i-1)}) \simeq \mathbb{Z}/N\mathbb{Z}$  for all  $i \in [1, k]$ .*

**Proof:** The condition  $\max \|\mathbf{b}_i^*\| / \min \|\mathbf{b}_i^*\| \leq 2^{\mathcal{O}(n)}$  ensures that the number  $k$  of levels computed in Step 2 is linear in  $n$ . Thus,  $\sigma$  computed in Step 3 is  $\leq \min \|\mathbf{b}_i^*\|$ . From the definition of  $B^{(k)}$ , we deduce that  $\text{vol}(\mathcal{L}(B^{(k)})) = \text{vol}(\mathcal{L}) / N^k = \sigma^n$ , and for all  $i \geq 2$ ,  $B_{[i,n]}^{(k)} = \hat{B}_{[i,n]}$ . Thus, Eq. (10) proves that the Rankin factor  $\gamma_{n,i}(B^{(k)}) = \sigma^{n-i} / \text{vol}(B_{[i+1,n]}^{(k)})$  is  $\leq n-i$  for all  $i \in [1, n-1]$ .  $\square$   
The proofs of Theorem 3.2 and Alg. 4 are given in Appendix B.

<sup>5</sup>Equations (8) and (9) are in common with [14] and correspond more or less to the definition of the unbalanced reduction algorithm

### 3.4 Cost for initial enumeration at level $k$ and pruning

The cost of a full enumeration of any bounded coset  $(z + \mathcal{L}_k) \cap \text{Ball}_n(r_n\beta\sigma)$  at level  $k$  is:

$$\mathcal{T}_{\text{SE}} = \sum_{i=1}^n \frac{\text{vol}(\text{Ball}_i(r_n\beta\sigma))}{\text{vol}(B_{[n+1-i,n]}^{(k)})} \leq n \sum_{i=1}^n V_i \cdot (r_n\beta)^i = \tilde{\mathcal{O}}(2^{0.398n}) \quad (11)$$

where for  $n \rightarrow \infty$  the maximal term in the sum,  $\sim \left(\frac{\sqrt{n}\beta}{\sqrt{i}}\right)^i$ , appears for  $i = n\beta^2/e$ . It is of size  $\tilde{\mathcal{O}}(2^{0.398n})$  because  $\sqrt{e}^{\beta^2/e} \approx 2^{0.398n}$ . Experiments show that the above estimate is close to what we observe in practice as we present in Sect. 4. The number of steps in the full enumeration is an exponential factor  $< 2^{0.03n}$  larger than the complexity of the merge. In practical dimensions  $\leq 100$ , the actual running time of the full enumeration is already smaller than the time for the merge by collision in the consecutive steps, as elementary operations in the enumeration are faster than memory access and vector additions in the merge. However, more work must be done in large dimensions. For instance, a light pruning [15, 12] can be used to divide the running time of the initial enumeration by a small exponential factor of  $2^{0.03n}$ , but it will only recover a subset  $S_k \subseteq C_k$ . For instance, by use of the linear bounding function of [15], it will recover a fraction  $1/n$  of  $C_k$ , and since depth  $n+1-i$  in the pruned enumeration only explores a subset of the  $i$ -dimensional ball of radius  $\sqrt{i/n}r_n\beta\sigma$  instead of  $r_n\beta\sigma$ , the running time  $\mathcal{T}_{\text{SE}}$  provably decreases below

$$\mathcal{T}_{\text{SE}}^{\text{lin.prun.}} \leq \sum_{i=1}^n \frac{\text{vol}(\text{Ball}_i(\sqrt{\frac{i}{n}}r_n\beta\sigma))}{\text{vol}(B_{[n+1-i,n]}^{(k)})} \leq n \sum_{i=1}^n V_i \left(r_n\sqrt{\frac{i}{n}}\beta\right)^i \approx n^2\beta^n = \tilde{\mathcal{O}}(2^{0.292n}). \quad (12)$$

Of course, there are lots of pruning trade-offs between Eqs. (11) and (12). This leads to a natural question on the stability of the algorithm, namely if the input of the merge at level  $i$  is an incomplete subset  $S_{i+1}$  containing only a constant or polynomially small fraction  $\nu$  of all elements of  $C_{i+1}$ , is the merge algorithm still be able to retrieve the whole set  $C_i$ . Intuitively, under some reasonable independence heuristics,  $\beta$  should then be increased so that the volume of each ball intersection grows by a factor  $1/\nu^2$ . Thus condition (5) becomes  $\beta^n \sqrt{1 - (\alpha/2)^2}^n \geq \sqrt{n}G_H/0.692\nu^2$ . But on the other hand,  $G_H$  can now be decreased from some large enough constant down to almost 1, since the Gaussian heuristic 2.1 only needs to be valid for a fraction  $\geq \nu$  of all intersections of balls, in order to get a fraction  $\geq \nu$  of  $C_i$  in the output. Working with incomplete cosets also raises additional questions, namely how likely are short elements to be present in the incomplete output coset, and can this probability be increased with randomization and standard repetition arguments.

In the next section, we address these questions in our experimental results which implicitly uses  $G_H = 1$  for efficiency reasons.

## 4 Experimental validation

In this section we present our experimental results of a C++-implementation of our Algorithm 1, presented in Section 3. We make use of the newNTL [16] and fplll [7] libraries as well as the Open MP [33] and GMP [11] library. We tested the algorithm on random lattices of dimensions up to  $n = 90$  as input.

### 4.1 Overview

Tests in smaller and larger dimensions confirm the choice of parameters  $\alpha$  and  $\beta$  that we computed for the asymptotic case. We are hence able to enumerate vectors of a target coset  $C_0 = (t_0 + \mathcal{L}_0) \cap \text{Ball}(R_0)$  and in this way we solve SVP as well as CVP. Indeed, unlike classical sieving algorithm, short elements, *i.e.*, either a short vector or a close vector, have a higher probability to be found than larger elements.

Thus, even though we might miss some elements of the target coset, we almost always solve the respective SVP or CVP. For instance, the algorithm finds the same shortest vectors as solutions for the SVP challenges published in [36]. The memory requirement and running time in the course of execution closely match our estimates and the intermediate helper lattices  $\mathcal{L}_i$  behave as predicted.

Besides the search for one smallest/closest vector, each run of the algorithm, with appropriate parameters, finds a non-negligible fraction of the whole bounded coset  $C_0$ . Repeating the search for vectors in  $C_0$  several times on a randomized LLL-reduced basis will discover the complete bounded coset. Our experiments reflect this behavior where we can use the Gaussian heuristic or Schnorr-Euchner enumeration to verify the proportion of recovered elements of  $C_0$ .

All these tasks can be performed by a single machine or independently by a cluster as a distributed computation.

## 4.2 Recovering $C_0$ in practice for smaller dimensions

For design reasons we have described an algorithm that produces the same number of elements per list in each iteration in order to find all of  $C_0$ . All lists contain  $\#C_0 = \#((\mathbf{t}_0 + \mathcal{L}_0) \cap \text{Ball}_n(R_0)) \approx (1 + \varepsilon_n)^n \beta^n$  elements on average where  $\varepsilon_n$  can be neglected for very large dimensions, (see also (5)). For accessible dimensions, we need to increase the radii of the balls slightly, by a small factor  $1 + \varepsilon_n$ , that compensates for small variations from the heuristic estimate. We here present results for different values  $\varepsilon_n \leq 0.08$  and dimension  $n \in \{40, 45, 50, 55, 60\}$ . The larger the dimension, the better Heuristic 2.1 holds, which means that  $\varepsilon_n$  can be chosen smaller, see (6). Figure 4 shows the relation between varying  $\varepsilon_n$  and the fraction of found vectors of  $C_0$  for dimension  $n \in \{40, 45, 50, 55, 60\}$ . The optimal choice for  $\varepsilon_n$  depends on  $n$  and the fraction of  $C_0$  we wish to enumerate.

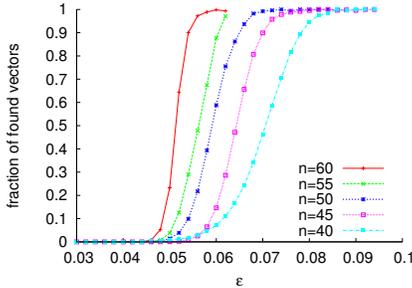


Figure 4: Fraction of vectors in  $C_0$  found for varying  $\varepsilon_n$ .

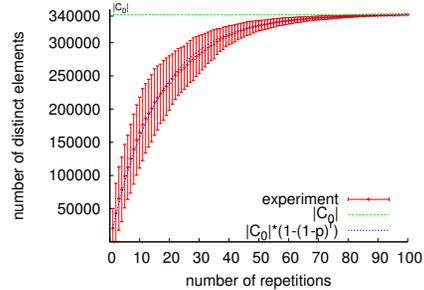


Figure 5: Success probability after  $r$  repetitions,  $n = 50$ ,  $p = 0.06$ .

## 4.3 Probability of success for randomized repetitions - example: small dimension

The success ratio of recovering all of  $C_0$  rises with increasing  $n$ . We here present the case of smaller dimensions  $n = \{50, 55\}$  to show how it evolves.

Suppose that we want to enumerate 100% of a coset  $C_0$  in dimension 50. According to Fig. 4, we need to choose  $\varepsilon_n$  at least 0.07, which results in lists of size  $(1 + \varepsilon_n)^{50} \beta^{50} \approx 29.4 \beta^{50}$  and a running time  $(1 + \varepsilon_n)^{100} (\beta^2 / \alpha)^{50} \approx 867.7 (\beta^2 / \alpha)^{50}$  on average. An alternative, which is less memory consuming, is to choose a smaller  $\varepsilon_n$ , and to run the algorithm several times on randomized input bases. For instance, if one chooses  $\varepsilon = 0.0535$ , one should expect to recover  $p = 6\%$  of  $C_0$  per iteration on average. Then, assuming that the recovered vectors are uniformly and independently distributed in  $C_0$ , we expect to find a fraction of  $1 - (1 - p)^r$  after  $r$  repetitions.

To confirm this independence assumption, we tested repeated execution for SVP instances with parameters  $n = 50$ ,  $(1 + \varepsilon)\beta = 1.0535\sqrt{3/2}$ ,  $\alpha = \sqrt{4/3}$ . Figure 5 shows the average number of distinct vectors of  $C_0$  recovered as a function of the number of repetition  $r$  (and the observed standard

deviation) in comparison to the expected number of elements  $C_0 \cdot (1 - (1 - 0.06)^r)$ . The experiments match closely the estimate.

For a random lattice of dimension  $n = 50$  and  $\varepsilon = 0.0535$ , the size of the coset  $C_0$  is roughly 342 000. In our experiments, we found 164 662 vectors (48%) after 10 repetitions in which we randomized the basis. After 20 trials, we found 239 231 elements which corresponds to 70%, and after 70 trials, we found 337 016 elements (99% of  $C_0$ ). We obtained the following results in dimension  $n = 55$ . After 10 trials with  $\varepsilon = 0.0535$ , we obtain 96.5% of the vectors of  $C_0$  which is significantly higher in comparison to the 48% recovered after 10 trials in dimension 50.

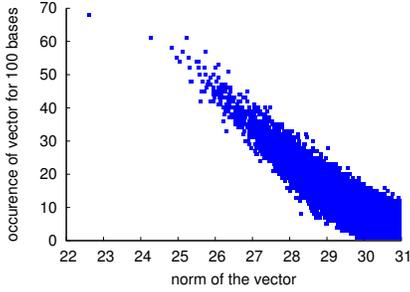


Figure 6: Correlation of occurrence of vectors and their length.

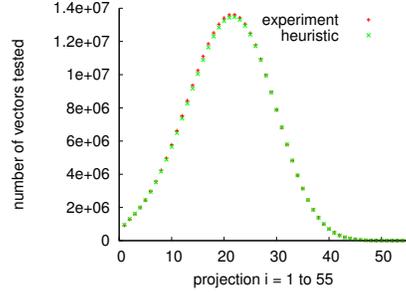


Figure 7: Comparison between the actual number of nodes during enumeration and the Gaussian heuristic predictions for dimension 55.

#### 4.4 Shorter or closer vectors are easier to find

During the merge operations, we can find a vector  $v \in C_i$  if there exist vectors in the intersection between two balls of the same radius, centered at the end points of  $v$ . As the intersection is larger when  $v$  is shorter, see Fig. 8, we can deduce that with the practical variant, short vectors of a coset are easier to find than longer ones.

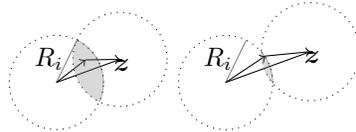


Figure 8: Volume of intersection varies for vectors  $z$  of different length.

As we work with cosets, this means that vectors which are closer to the target (*i.e.*, short lattice vectors when the target is 0) should appear more often for different runs on randomized input basis. We verified this observation experimentally by comparing the norm of a vector with the number of appearances during 100 repetitions in dimension 50, with  $\varepsilon = 0.0535$ , see Fig. 6.

#### 4.5 Parallelization

The algorithm itself is highly parallelizable for various types of hardware architectures. Of course, the dominant operations are  $n$ -dimensional vector additions and Euclidean norm computations, which can be optimized on any hardware containing vector instructions. Additionally, unlike sieving techniques, each iteration of the outer for-loop of the merge algorithm (Alg. 2, line 3) can be run simultaneously, as every vector is treated independently of the output. Furthermore, one may divide the pool of vectors into  $p \leq \alpha^n/2$  groups of buckets at each level, as soon as any two opposite buckets belong to the same group. Thus, the merge operation can operate on a group independently of all other groups.

Table 2: Experimental results for  $n \in \{70, 80\}$ ,  $\alpha = \sqrt{4/3}$  and  $\beta = \sqrt{3/2}$ .

	level = $i$	8	7	6	5	4	3	2	1	0
$n = 80$	$\#S_i$ in millions	253	149	132	142	163	194	230	265	336
$\varepsilon = 0.044$	% of Gauss. heuristic	73	43	38	41	47	56	66	76	97
$n = 70$	$\#S_i$ in millions	-	38.8	20.3	19.0	20.0	20.3	23.1	26.5	29.8
$\varepsilon = 0.049$	% of Gauss. heuristic	-	95	50	46	50	56	65	73	87
$n = 70$	$\#S_i$ in millions	-	33.1	16.0	13.4	12.3	11.4	10.7	9.7	7
$\varepsilon = 0.046$	% of Gauss. heuristic	-	95	46	38	35	32	30.6	27.8	20

This allows to efficiently run the algorithm when the available RAM is too small to store lists of size  $(1 + \varepsilon)^n \beta^n$ . It also allows to distribute the merge step on a cluster. For instance, in dimension  $n = 90$  using  $\varepsilon = 0.0416$ , storing the full lists would require 3 TB of RAM. We divided the lists into 25 groups of 120 GB each, which we treated one at a time in RAM while the others were kept on hard drive. This did not produce any noticeable slowdown. Finally, the number of elements in each bucket can be estimated precisely in advance using Heuristic 2.1, and each group performs exactly the same vector operations (floating point addition, Euclidean norm computation) at the same time. This makes the algorithm suitable for SIMD implementation, not only multi-threading.

#### 4.6 Experiments in low- and middle-sized dimensions

Our experiments in dimension 40 to 90 on challenges in [36] show that we find the same short vectors as previously reported and found as shortest vector by use of BKZ or sieving. To solve SVP or CVP by use of the decomposition technique, it is in fact not necessary to enumerate the complete bounded coset  $C_0$  and to ensure that the lists are always of size  $(1 + \varepsilon_n)^n \beta^n$  as we describe in the following paragraphs.

We give more details for medium dimensions  $n = 70$  and  $n = 80$  with  $\alpha = \sqrt{4/3}$  and  $\beta = \sqrt{3/2}$  in the following. The algorithm ran on a machine with an Opteron 6176 processor, containing 48 cores at 2.3 GHz, and having 256 GB of RAM. Table 2 presents the observed size of the lists  $S_i \subseteq C_i$  for each level in dimension 70 and 80.

In dimension 80, we chose aborted-BKZ-30 [17] as a preprocessing. The algorithm has 8 levels and we chose  $\varepsilon = 0.044$  to obtain 97% of  $C_0$  after a single run. The initial enumeration on one core took a very short time of 6.5 CPU hours (so less than 10 minutes with our multi-thread implementation of the enumeration) while each of the 8 levels of the merge took between 20 and 36 CPU hours (so less than 45 minutes per level in our parallel implementation).

The number of elements in lower levels lies below the heuristic estimate and we keep losing elements during the merge for the deepest levels. For example, in dimension 80 we start with 73% of  $C_8$  and recover only 43% of  $C_7$  after one step. Towards higher levels, we slowly begin to recover more and more elements. In dimension 80, the size of the lists starts to increase from level 5 on as  $S_5, S_4$  and  $S_3$  cover 41%, 47% and 56% of the vectors, respectively. This continues until the final step where we find 97% of the elements of  $C_0$ .

#### 4.7 Pruning of the merge step in practice - larger dimension $n = 75$ and $n = 90$

In Section 3.1, we obtain conditions on the parameters as we request the intersection  $I$  of two balls to be non-empty, which means that  $\text{vol}(I)/\text{vol}(\mathcal{L}) \geq K$  for some number  $K > 1$  under Heuristic 2.1. This condition suggests that at each level, each coset element in an output list  $S_{i-1} \subseteq C_{i-1}$  of a merge is obtained on average about  $K$  times. If the input list  $S_i$  is shorter than expected, one will indeed recover fewer than  $K$  copies of each element, but we may still have one representative of each element of  $C_{i-1}$ . Our experiments confirm this fact, see Tab. 2 and Tab. 3. To solve SVP or CVP, one may shorten the time and memory necessary to find a solution vector by interrupting each level whenever

Table 3: Experimental results **with pruning**,  $n \in \{75, 80, 90\}$ ,  $\alpha = \sqrt{4/3}$  and  $\beta = \sqrt{3/2}$ .

	level = $i$	9	8	7	6	5	4	3	2	1	0	SVP
$n = 75$ , $\varepsilon = 0.044$	% of Gauss. heuris.	-	-	50	50	47	46	46	48	50	69	solved
todo $n = 75$ , $\varepsilon = 0.044$	% of Gauss. heuris.	-	-	50	35	30	25	20	15	8	6.4	solved
$n = 90$ , $\varepsilon = 0.0416$	% of Gauss. heuris.	70	40	40	40	40	40	40	40	40	70	solved
$n = 90$ , $\varepsilon = 0.0416$	% of Gauss. heuris.	70	33	33	33	33	33	33	33	33	61	solved

the output list contains a sufficiently large fraction of the elements of the bounded cosets. For example, we ran our algorithm on the 75-dimensional basis of the SVP challenge [36] with seed 38. We chose  $\varepsilon = 0.044$  and interrupted the merge if the size of the intermediate set  $S_i$  reached 50% or 35% of  $\#C_i$  for  $i \in [1, k - 1]$ . Tab. 3 presents the intermediate list sizes. In the end, we recovered 69% and 6.4% of  $\#C_0$ , respectively, and the shortest vector was found in both cases. The running time for the merge in the intermediate levels decreases compared to no pruning by a factor 0.49 and 0.29, respectively, as one would expect for lists that are smaller by at least a factor 0.5 and 0.35, respectively.

In dimension 90, we ran our algorithm on the 90 dimensional SVP-challenge with seed 11, using  $\varepsilon = 0.0416$ . We chose to keep at most 33% of  $C_i$  for  $i \in [1, k - 1]$ . Despite this harsh cut, the size of the intermediate lists remained stable after the first merge. And interestingly, after only 65 hours on 32 threads, we recovered 61% of  $\#C_0$  in the end, including the published shortest vector. Note that as we interrupt the merge, we in fact do not read all elements of the starting list  $S_k$ . One might hence simply not apply a full enumeration in practice but stop the Schnorr-Euchner enumeration once enough elements are enumerated.

#### 4.8 Notes on the Gaussian heuristic for intermediate levels

Our quasi-orthogonal lattices at the bottom level behave randomly and follow the Gaussian heuristic. The most basic method to fill the bottom list  $S_k$  is to run Schnorr-Euchner enumeration (see Sect. 2) where the expected number of nodes in the enumeration tree is given by (11) based on Heuristic 2.1. Previous research has established that this estimate is accurate for random BKZ-reduced bases of random lattices in high dimension. Here, since we work with quasi-orthogonal bases, which are very specific, we redo the experiments, and confirm the findings also for quasi-orthogonal bases. Already for small dimensions ( $n = 40, 50, 55$ ), experiments show that the actual number of nodes in a Schnorr-Euchner enumeration is very close to the expected value. Figure 7 shows that experiment and heuristic estimate for dimension 55, for example, are almost indistinguishable.

We also make use of Heuristic 2.1 when we estimate the number of coset vectors in the intersection of two balls. As the lower lattices in the tower are not "random" enough, they have close to quasi-orthonormal bases, we observe smaller lists in the lower levels and thus a deviation from the heuristic. Beside the geometry of lattices, the deviation depends on the center of the balls or the center of the intersection. Randomly centered cosets of quasi-orthonormal lattices contain experimentally an average number of points a constant factor below  $(1 + \varepsilon_n)^n \beta^n$ . Zero-centered cosets contain more points, and should be avoided. The randomization of the initial target used in Alg. 1 ensures that the centers are random modulo  $\mathcal{L}_k$ , even in an SVP setting. The number of vectors stays hence below, but close to the estimate  $(1 + \varepsilon_n)^n \beta^n$  after the first collision steps. The following steps can only improve the situation. The lattices in higher levels are more and more random and we observe that the algorithm recovers the expected number of vectors. This is a sign that our algorithm is stable even when the input pools  $S_i$  are incomplete.

Finally, experiments support the claim that the number of elements per bucket during the merge by collision corresponds to the average value  $(\beta/\alpha)^n$ . For example, in dimension  $n = 80$ , for parameters

$\alpha = \sqrt{4/3}$ ,  $\beta = \sqrt{3/2}$ ,  $\varepsilon = 0.044$ , we observe that the largest bucket contains only 10% more elements than the average value, and that 60% of the buckets are within  $\pm 2\%$  of the average value.

#### 4.9 Comparison to experimental results of a parallel Gauss sieve algorithm

From a very general point of view, our algorithm can be viewed as a Sieving algorithm. The algorithm is decomposed into a polynomial number of levels, each one corresponds to a certain upper-bound  $R_i$  on the norm. At each level, we use an exponential pool of lattice vectors, perform linear combinations, and select the shortest of them for the next level.

We now list the specificities of our algorithm compared to previous sieving algorithms:

- We start from short vectors (in overlattices) and at each level, the norm  $R_i$  geometrically increases by a factor  $\alpha$ .
- At each level, we maintain an (almost) complete set of all coset vectors of norm  $\leq R_i$ . For this reason, our algorithm has the stability property that short coset vectors are more likely to be found. Classical sieving techniques satisfy the opposite: short vectors have a negligible property to appear spontaneously. Our algorithm is compatible with pruning, and it can solve the Exact CVP. Reducing the list sizes for classical sieving leads in general to catastrophic results.
- Our algorithm is highly parallelizable as it allows to use up to  $\alpha^n$  independent threads per merge operation as explained in Sect. 4.5. However the accessible dimension is naturally limited by the exponential memory requirement of order  $\beta^n$ . There exists parallel versions of the Gauss sieve [30, 19], which leads to faster practical running time in dimensions 70 to 96, however the efficiency of the parallelization decreases fast when the number of threads increases because of the list size and the communication cost [19].
- Our algorithm is essentially a CVP solver and is not specialized for SVP: if classical Sieving algorithms were to be turned into CVP solvers, then it would obviously be impossible to regroup each vector with its opposite, and the lists of vectors would be twice as large. Furthermore, classical Sieving techniques rely on the fact that vectors which cannot be reduced by others, necessarily become poles to reduce others. By replacing subtractions with additions in order to preserve the target, these two options – can a vector  $v$  be reduced by others vs. can  $-v$  be considered as a pole – cease to be mutually exclusive, and both would have to be tested. Thus, turning classical Sieving algorithms into CVP solvers would likely increase their running time by a factor 4 and their memory requirement by a factor 2, with no guarantee that they would actually find the solution.

We give some concrete timings: To solve instances in dimension 80 and 90, our algorithm takes more time than the currently fastest implementation of the Gauss sieve algorithm [19]. Ishiguro et al. report in [19] to solve the SVP challenge in dimension 80 in 29 sequential hours and an instance of dimension 96 in 6400 sequential hours. Our algorithm however needs 65 sequential hours in dimension 80 and 2080 hours in dimension 90. It is slower than the Gauss Sieve used as an SVP solver, yet, the slowdown factor remains smaller than 4, which would be expected (as a minimum) to turn it into a CVP solver.

## 5 Conclusion

We have presented an alternative approach to solve the hard lattice problems SVP and CVP for random lattices. It makes use of a new technique that is different from the ones used so far in enumeration or sieving algorithms and works by moving short vectors along a tower of nested lattices. Our experiments show that the method works well in practice. An open question in the case of ideal lattices is to find a structural reduction that preserves the cyclic structure or provides an other structure for which speed-ups can be applied.

## References

- [1] M. Ajtai. The shortest vector problem in  $L_2$  is NP-hard for randomized reductions (extended abstract). In *STOC'98*, pages 10–19, 1998.
- [2] M. Ajtai. Random lattices and a conjectured 0 - 1 law about their polynomial time computable properties. In *FOCS*, pages 733–742, 2002.
- [3] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proc. 33rd STOC*, pages 601–610, 2001.
- [4] A. Becker, J.-S. Coron, and A. Joux. Improved generic algorithms for hard knapsacks. In *Proc. of Eurocrypt 2011*, LNCS 6632, pages 364–385. Springer-Verlag, 2011.
- [5] A. Becker, A. Joux, A. May, and A. Meurer. Decoding random binary linear codes in  $2^{n/20}$ : How  $1 + 1 = 0$  improves information set decoding. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012.
- [6] M. I. Boguslavsky. Radon transforms and packings. *Discrete Applied Mathematics*, 111(1-2):3–22, 2001.
- [7] X. Cadé, D. Pujol and D. Stehlé. fplll 4.0.4, May 2013.
- [8] D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *EUROCRYPT*, pages 178–189, 1996.
- [9] D. Coppersmith. Finding a small root of a univariate modular equation. In *EUROCRYPT*, pages 155–165, 1996.
- [10] I. Dinur, G. Kindler, and S. Safra. Approximating-cvp to within almost-polynomial factors is np-hard. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 99–, Washington, DC, USA, 1998. IEEE Computer Society.
- [11] T. G. et al. GNU multiple precision arithmetic library 5.1.3, September 2013. <https://gmplib.org/>.
- [12] M. Fukase and K. Yamaguchi. Finding a very short lattice vector in the extended search space. *JIP*, 20(3):785–795, 2012.
- [13] N. Gama, N. Howgrave-Graham, H. Koy, and P. Q. Nguyen. Rankin’s constant and blockwise lattice reduction. In *CRYPTO*, pages 112–130, 2006.
- [14] N. Gama, M. Izabachene, P. Q. Nguyen, and X. Xie. Structural lattice reduction: Generalized worst-case to average-case reductions. Eprint report 2014/283, 2014.
- [15] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT*, pages 257–278, 2010.
- [16] N. Gama, J. van de Pol, and J. M. Schanck. Fork of V. Shoup’s number theory library NTL, with improved lattice functionalities. <http://www.prism.uvsq.fr/~gama/newntl.html>, February 2013.
- [17] G. Hanrot, X. Pujol, and D. Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *CRYPTO*, pages 447–464, 2011.
- [18] G. Hanrot and D. Stehlé. Improved analysis of Kannan’s shortest lattice vector algorithm (extended abstract). In *Proceedings of Crypto 2007*, volume 4622 of *LNCS*, pages 170–186. Springer-Verlag, 2007.

- [19] T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi. Parallel gauss sieve algorithm : Solving the svp in the ideal lattice of 128-dimensions. Cryptology ePrint Archive, Report 2013/388, 2013.
- [20] A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *J. Cryptology*, 11(3):161–185, 1998.
- [21] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 193–206, New York, NY, USA, 1983. ACM.
- [22] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.
- [23] A. Korkine and G. Zolotarev. Sur les formes quadratiques. *Mathematische Annalen* 6, pages 336–389, 1973.
- [24] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.
- [25] A. May, A. Meurer, and E. Thomae. Decoding random linear codes in  $2^{0.054n}$ . In *ASIACRYPT*, pages 107–124, 2011.
- [26] J. E. Mazo and A. M. Odlyzko. Lattice points in high-dimensional spheres. *Monatshefte für Mathematik*, 110:47–62, 1990.
- [27] D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 92–, Washington, DC, USA, 1998. IEEE Computer Society.
- [28] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 351–358, New York, NY, USA, 2010. ACM.
- [29] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480. ACM/SIAM, 2010.
- [30] B. Milde and M. Schneider. A parallel implementation of gausssieve for the shortest vector problem in lattices. In *PaCT*, pages 452–458, 2011.
- [31] L. J. Mordell. On some arithmetical results in the geometry of numbers. *Compositio Mathematica*, 1:248–253, 1935.
- [32] P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2008.
- [33] OpenMP Architecture Review Board. OpenMP API version 4.0, 2013.
- [34] X. Pujol and D. Stehlé. Solving the shortest lattice vector problem in time  $2^{2.465n}$ . *IACR Cryptology ePrint Archive*, 2009:605, 2009.
- [35] R. Rankin. On positive definite quadratic forms. *J. Lond. Math. Soc.*, 28:309–314, 1953.
- [36] M. Schneider, N. Gama, P. Baumann, and P. Nobach. <http://www.latticechallenge.org/svp-challenge/halloffame.php>.
- [37] K.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

- [38] J. L. Thunder. Higher-dimensional analogs of Hermite’s constant. *Michigan Math. J.*, 45(2):301–314, 1998.
- [39] X. Wang, M. Liu, C. Tian, and J. Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS ’11, pages 1–9, New York, NY, USA, 2011. ACM.
- [40] F. Zhang, Y. Pan, and G. Hu. A Three-Level Sieve Algorithm for the Shortest Vector Problem. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013 - 20th International Conference on Selected Areas in Cryptography*, volume Lecture Notes in Computer Science, Burnaby, Canada, Aug. 2013. Springer.

## A Intersection of hyperballs

The volume of the intersection,  $\text{vol}_I(d)$ , of two  $n$ -dimensional hyperballs of radius 1 at distance  $d \in [0.817; 2]$  can be approximated for large  $n$  by the volume of the  $n$ -dimensional ball of radius  $D = \sqrt{1 - (\frac{d}{2})^2}$ , see Lemma A.1 below. If we consider the intersection of two balls of radius  $R$ , the volume gets multiplied by a factor  $R^n$  as stated in Corollary 2.

**Lemma A.1** *The volume of the intersection of two  $n$ -dimensional hyperballs of radius 1 at distance  $d \in [0.817; 2]$  is*

$$\frac{2V_{n-1}}{(n+1)V_n} \arccos\left(\frac{d}{2}\right) \leq \frac{\text{vol}_I(d)}{\text{vol}(\text{Ball}_n(D))} \leq \frac{2V_{n-1}}{(\frac{n}{2}+1)V_n} \arccos\left(\frac{d}{2}\right)$$

where  $D = \sqrt{1 - (\frac{d}{2})^2}$ .

**Proof:** The intersection of two balls of radius 1 whose centers are at distance  $d \in [0, 2]$  of each other can be expressed as

$$\text{vol}_I(d) = 2 \cdot \int_{\frac{d}{2}}^1 V_{n-1} \left(\sqrt{1-x^2}\right)^{n-1} dx = 2V_{n-1} \int_0^{\arccos(d/2)} \sin^n(\theta) d\theta$$

where  $V_{n-1}$  equals the volume of the  $n-1$ -dimensional ball of radius 1. For  $d \in [0.817; 2]$  one can bound the sinus term in the integral:

$$\frac{D}{\arccos(d/2)} \theta \leq \sin(\theta) \leq \frac{D}{\sqrt{\arccos(d/2)}} \sqrt{\theta} .$$

Therefore, we obtain bounds for the volume of the intersection:

$$\text{vol}_I(d) \leq \frac{2V_{n-1}}{\frac{n}{2}+1} \arccos\left(\frac{d}{2}\right) D^n$$

and

$$\text{vol}_I(d) \geq \frac{2V_{n-1}}{n+1} \arccos\left(\frac{d}{2}\right) D^n$$

which proves the lemma.  $\square$

We can use the lower-bound of Lemma A.1 and obtain a numerical lowerbound on the volume of the intersection of balls of radius  $R$  at distance at most  $\sqrt{4/3}R$  used in our algorithm:

**Corollary 2** For all dimensions  $n \geq 10$ , the volume of the intersection of two  $n$ -dimensional hyperballs of radius  $R$  at distance  $dR$  where  $d \leq \sqrt{4/3}$  is lower-bounded by:

$$R^n \text{vol}_T(d) \geq \frac{0.692}{\sqrt{n}} \cdot R^n \text{vol} \left( \text{Ball}_n \left( \sqrt{1 - \left(\frac{d}{2}\right)^2} \right) \right).$$

## B Proof of Theorem 3.2 and Algorithm 4:

---

**Algorithm 4** Unbalanced Reduction from [14], specialized for  $\sigma \leq \min \|\mathbf{b}_i^*\|$

---

**Input:** A LLL-reduced basis  $B$  of an integer lattice  $\mathcal{L}$ , and a target length  $\sigma \leq \min \|\mathbf{b}_i^*\|$

**Output:** A basis  $C$  of  $\mathcal{L}$  satisfying  $\|\mathbf{c}_1\| \leq \sigma n \text{vol}(\mathcal{L}) / \sigma^n$ , and for all  $i \in [2, n]$ ,  $\|\mathbf{c}_i^*\| \leq \sigma$  and  $\frac{\sigma^{n+1-i}}{\text{vol}(C_{[i,n]})} \leq n+1-i$ .

- 1:  $C \leftarrow B$
  - 2: Compute the Gram-Schmidt matrices  $\mu$  and  $C^*$
  - 3: Let  $k$  be the largest index such that  $\|\mathbf{c}_k^*\| > \sigma$
  - 4: **for**  $i = k-1, \dots, 1$  **do**
  - 5:    $\gamma \leftarrow \left[ -\mu_{i+1,i} + \frac{\|\mathbf{c}_{i+1}^*\|}{\|\mathbf{c}_i^*\|} \sqrt{\left(\frac{\|\mathbf{c}_i^*\|}{\sigma}\right)^2 - 1} \right]$
  - 6:    $(\mathbf{c}_i, \mathbf{c}_{i+1}) \leftarrow (\mathbf{c}_{i+1} + \gamma \cdot \mathbf{c}_i, \mathbf{c}_i)$
  - 7:   Update the Gram-Schmidt matrices  $\mu$  and  $C^*$ .
  - 8: **end for**
  - 9: **return**  $C$
- 

We use the suffix “old” and “new” to denote the values of the variables at the beginning and at the end of the “for” loop of Alg. 4, respectively. Furthermore, we call  $x_i$  the value  $\|\mathbf{b}_i^{\text{new}}\|$  during iteration  $i$ . Note that  $x_i$  is also  $\|\mathbf{b}_i^{\text{old}}\|$  during the next iteration (of index  $i-1$  since  $i$  goes backwards).

For  $i \in [1, n]$ , let  $a_i = \|\mathbf{b}_i^*\| / \sigma$ . Note that  $a_i$  is always  $\geq 1$ . We show by induction over  $i$  that the following invariant holds at the end of each iteration of Alg. 4:

$$a_i x_{i+1} \leq x_i \leq a_i x_{i+1} + \sigma a_i. \quad (13)$$

At the first iteration ( $i = k-1$ ), it is clear that  $x_k = \|\mathbf{b}_k^{\text{old}}\| = \sigma a_k$ . At the beginning of iteration  $i$ , we always have  $\|\mathbf{b}_i^{\text{old}}\| > \sigma$ , and by induction,  $\|\mathbf{b}_{i+1}^{\text{old}}\| > \sigma$ . We transform the block so that the norm of the first vector satisfies

$$R \leq \|\mathbf{b}_i^{\text{new}}\| \leq R + \|\mathbf{b}_i^{\text{old}}\| \quad (14)$$

where  $R = \|\mathbf{b}_{i+1}^{\text{old}}\| \|\mathbf{b}_i^{\text{old}}\| / \sigma$ .

This condition can always be fulfilled with a primitive vector of the form  $\mathbf{b}_i^{\text{new}} = \mathbf{b}_{i+1}^{\text{old}} + \gamma \mathbf{b}_i^{\text{old}}$  for some  $\gamma \in \mathbb{Z}$ . Since the volume is invariant, the new  $\|\mathbf{b}_{i+1}^{\text{new}}\|$  is upper-bounded by  $\sigma$ . And by construction, Equation (14) is equivalent to the invariant (13) since  $\|\mathbf{b}_i^{\text{old}}\| = a_i \sigma$ ,  $\|\mathbf{b}_i^{\text{new}}\| = x_i$  and  $\|\mathbf{b}_{i+1}^{\text{old}}\| = x_{i+1}$ .

By developping (13), we derive a bound on  $x_1$ :

$$x_1 \leq \sigma \sum_{i=1}^k a_1 \dots a_i \leq \sigma n \prod_{i=1}^k a_i \leq n \sigma \text{vol}(\mathcal{L}) / \sigma^n$$

which proves (9). Similarly, one obtains that  $x_i \leq (n+1-i) \sigma \text{vol}(B_{[i,n]}) / \sigma^{n+1-i}$ , which is equivalent

to (10). Note that the transformation matrix of the unbalanced reduction algorithm is

$$\left[ \begin{array}{cccc|ccc} \gamma_1 & \cdots & \gamma_{k-1} & 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 & \vdots & & \vdots \\ 0 & \ddots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ \hline 0 & \cdots & \cdots & 0 & 1 & 0 & 0 \\ \vdots & & & \vdots & 0 & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 0 & 0 & 1 \end{array} \right]$$

where  $\gamma_i$  is  $\left[ -\mu_{i+1,i} + \frac{x_{i+1}}{\sigma} \sqrt{1 - \frac{1}{a_i^2}} \right]$ . Since each  $x_{i+1}$  is bounded by

$$\prod_{j=i+1}^n a_j = \prod_{j=i+1}^n \max(1, \|b_j^*\|_2 / \sigma) ,$$

all coefficients have a size polynomial in the input basis. This proves that Alg. 4 has polynomial running time.  $\square$

**Anja Becker** *EPFL, École Polytechnique Fédérale de Lausanne,  
Laboratory for cryptologic algorithms (LACAL),  
Switzerland.  
anja.becker@epfl.ch*

**Nicolas Gama** *UVSQ/PRISM, Université de Versailles, France.  
nicolas.gama@prism.uvsq.fr*

**Antoine Joux** *CryptoExperts,  
INRIA/Ouragan,  
Chaire de Cryptologie de la Fondation de l'UPMC,  
Sorbonne Universités, UPMC Univ Paris 06, CNRS UMR 7606, LIP 6, France.  
Antoine.Joux@m4x.org*