Submitted exclusively to the London Mathematical Society
doi:10.1112/0000/000000

Solving shortest and closest vector problems: The decomposition approach

Abstract

In this paper, we present a heuristic algorithm for solving exact, as well as approximate, SVP and CVP for lattices. This algorithm is based on a new approach which is very different from and complementary to the sieving technique. This new approach allows us to solve not only shortest vector problems, but also closest vector problems, in lattices of dimension n in time $2^{0.3774\,n}$ using memory $2^{0.2925\,n}$. Moreover, it is straightforward to parallelize on most computer architectures. The key idea is 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 of our tower. Repeating this, we climb all the way to the top of the tower and finally obtain solution vector(s) in the initial lattice as a sum of vectors of the overlattice just below it. 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.

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, for a long time, they were used as cryptanalytic tools, first through a direct approach as in [**19**] 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 [†] to solve exactly [**1**, **21**] and also NP-hard to approximate [**10**, **26**] 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 [**34**, **20**], sieving [**3**, **29**, **28**, **36**] or the Voronoi-cell algorithm [**27**]. 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 preprocessing. Probabilistic sieving algorithms, as well as the deterministic Voronoi-cell algorithm are simply exponential in time and memory. A *closest* vector can be found by enumeration and by the Voronoi-cell algorithm, however, state-of-the-art sieving techniques cannot be directly applied to solve CVP instances. 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

2000 *Mathematics Subject Classification* 00000.

[†] Under randomized reductions in the case of SVP.

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.

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 [**23**, **34**] the running time of Schnorr-Euchner's enumeration is double exponential in the dimension. If we further reduce the basis to a HKZ-reduced basis [**22**], the complexity becomes $2^{\mathcal{O}(n \log n)}$ [**20**, **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 and which are found in polynomial time by a special case of structural reduction as described in Sect. 3.3. Once we have an overlattice and its quasi-orthonormal basis, our main task is to find a solution vector in the initial lattice given a sample of short vectors in the overlattice. This is similar to hardness results frequently found in worst-case to average-case proofs. Usually, 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 and follows a novel approach to tackle them. It represents an adaptation of the representation technique that solves knapsack problems [**4**] and decoding problems [**24**, **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.

TABLE 1. *Complexity of currently known SVP/CVP algorithms.*

Algorithm	Time	Memory	CVP	SVP	
Kannan-Enumeration [**18**]	$n^{n/2+o(n)}$	poly(n)	✓		proven
	$n^{n/(2e)+o(n)}$	poly(n)		✓	proven
Voronoi-cell [**27**]	$2^{2\,n}$	2^n	✓	✓	proven
ListSieve-Birthday [**31**]	$2^{2.465\,n+o(n)}$	$2^{1.233\,n+o(n)}$	×	✓	proven
GaussSieve [**28**]	$2^{0.415\,n+o(n)}$?	$2^{0.2075\,n+o(n)}$?	?	✓	heuristic
Nguyen-Vidick sieve [**29**]	$2^{0.415\,n+o(n)}$	$2^{0.2075\,n+o(n)}$	×	✓	heuristic
WLTB sieve [**36**]	$2^{0.3836\,n+o(n)}$	$2^{0.2557\,n+o(n)}$	×	✓	heuristic
Three-level sieve [**37**]	$2^{0.3778\,n+o(n)}$	$2^{0.2833\,n+o(n)}$	×	✓	heuristic
Our algorithm	from $2^{0.4150\,n}$	$2^{0.2075\,n}$	✓	✓	heuristic
	to $2^{0.3774\,n}$	$2^{0.2925\,n}$			

The biggest challenge is to bootstrap the algorithm by finding suitable and easier subproblems related to overlattices. We propose a generic method that achieves this thanks to a well-chosen overlattice for which a deterministic enumeration of vectors of bounded norm is efficient. In this way, we can compute a starting set of vectors that can be used as the starting point of a sequence of recombinations that ends up solving the initially considered problem.

1.1. *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 $\mathcal{L} = \mathcal{L}_0 \subseteq .. \subseteq \mathcal{L}_k$. 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 $\mathcal{L}_i$-vectors sum up to an $\mathcal{L}_{i-1}$-vector with probability $\frac{1}{\alpha^n}$, where $\mathrm{vol}\left(\mathcal{L}_{i-1}\right)/\mathrm{vol}\left(\mathcal{L}_i\right) = \alpha^n > 1$. We allow the norm to increase by a moderate factor α 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 $\boldsymbol{x} + \mathcal{L}$, $\boldsymbol{x} \notin \mathcal{L}$. 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. Furthermore, the time and memory complexity are no longer linked to the kissing number. The average running time in the asymptotic case is $2^{0.3774\,n}$, requiring a memory of $2^{0.2925\,n}$. 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 the 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 $\mathbb{R}^m$. A lattice can be described as the set of all integer combinations $\{\sum_{i=1}^{n} \alpha_i \boldsymbol{b}_i \mid \alpha_i \in \mathbb{Z}\}$ of n linearly independent vectors $\boldsymbol{b}_i$ of $\mathbb{R}^m$. In this case the vectors $\boldsymbol{b}_1, .., \boldsymbol{b}_n$ are called a basis of $\mathcal{L}$. The volume of the lattice $\mathcal{L}$ is the volume of $\mathrm{span}(\mathcal{L})/\mathcal{L}$, and can be easily 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 $\lambda_1(\mathcal{L}) \leq \sqrt{n}\,\mathrm{vol}\left(\mathcal{L}\right)^{1/n}$. We call a *coset* of a lattice a translation $\boldsymbol{x} + \mathcal{L} = \{\boldsymbol{x} + \boldsymbol{v} \mid \boldsymbol{v} \in \mathcal{L}\}$ of $\mathcal{L}$ by a vector $\boldsymbol{x} \in \mathrm{span}(L)$.

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 $\mathrm{vol}(\mathcal{L})/\mathrm{vol}(\mathcal{L}') = [\mathcal{L}' : \mathcal{L}]$.

Hyperballs. Let $\mathrm{Ball}_n(R)$ denote the ball of radius R in dimension n where we omit n if it is implied from 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:

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

Gaussian heuristic. In many cases, when we wish to estimate the number of lattice points in a "nice enough" set S, we use the following approximation called the Gaussian heuristic:

Heuristic 2.1 (Gaussian Heuristic). *Given a lattice $\mathcal{L}$ and a suitable set S, the number of points in $S \cap \mathcal{L}$ can be approximated by $\mathrm{vol}\left(S\right)/\mathrm{vol}\left(\mathcal{L}\right)$.*

When S is a ball of radius at least $\sqrt{n}^{\epsilon}\,\mathrm{vol}(\mathcal{L})^{1/n}$ for some fixed $\epsilon > 0$, we can prove that this estimate holds for almost all real lattices, and almost all integer lattices of large volume [**2**]. It has been widely experimentally verified that for random integer cocyclic real lattices of large volume, this estimate also holds when S is a smaller ball of radius close to $\sqrt{n}\,\mathrm{vol}(\mathcal{L})^{1/n}$. This allows to estimate the length of the shortest vector of a random lattice as the radius of a ball of volume $\mathrm{vol}\,(\mathcal{L})$: $\lambda_1(\mathcal{L}) \approx r_n \cdot \mathrm{vol}\,(\mathcal{L})^{1/n}$. It also indicates that a ball of radius $\beta\, r_n \mathrm{vol}\,(\mathcal{L})^{1/n}$, for all real $\beta > 0$, should asymptotically contain about β^n lattice points. However, this heuristic may not hold for specific lattices. For example, the number of lattice points of $\mathbb{Z}^n$ contained in a ball varies significantly depending on the center of the ball; it differs from the heuristic by an exponential factor in n [**25**]. In general, any use of Heuristic 2.1 requires an experimental validation. We describe experiments validating the use of the Gaussian heuristic in our algorithm in Sect. 4.

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

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

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

the *Rankin factor* of B with index j. The well known Rankin invariants of the lattice, $\gamma_{n,j}(\mathcal{L})$, introduced by Rankin [**32**] 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 $j \in [1, n]$.*

For example, any real triangular matrix with identical diagonal coefficients forms a quasi-orthogonal basis. More generally, any basis whose $\|\boldsymbol{b}_i^*\|$ 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**, **35**, **13**] by $2^{\Theta(n \log n)}$ for almost all lattices[†], 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 [**34**] allows to enumerate all vectors of Euclidean norm $\leq R$ in the bounded coset $C = (\boldsymbol{z} + \mathcal{L}) \cap \mathrm{Ball}_n(R)$ where $\boldsymbol{z} \in \mathbb{R}^n$. The running time of this algorithm is

$$\mathcal{T}_{SE} = \sum_{i=1}^{n} \# \left(\pi_{n+1-i}(\boldsymbol{z} + \mathcal{L}) \cap \mathrm{Ball}_i(R)\right) \quad , \tag{2.1}$$

[†] $\gamma_{2n,n}(\mathcal{L}) \geq (n/12)^n$ with probability ≈ 1 on random real lattices of volume 1 drawn from the Haar distribution.

which is equivalent to

$$\mathcal{T}_{SE} \approx \sum_{i=1}^{n} \frac{\mathrm{vol}(\mathrm{Ball}_i(R))}{\mathrm{vol}(\pi_{n+1-i}(\mathcal{L}))} \tag{2.2}$$

under Heuristic 2.1. The last term in the sums (2.1) and (2.2) denotes the number of solutions $\#C$. Thus, the complexity of enumeration is approximately $\mathcal{T}_{SE} \approx \tilde{\mathcal{O}}\left(\#C\right) \cdot \max_{j\in[1,n]} \gamma_{n,j}(B)$. 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 $\tilde{\mathcal{O}}\left(\#C\right)$, 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 β^n shortest vectors in any coset $\boldsymbol{t} + \mathcal{L}$ 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 $\mathrm{dist}(\boldsymbol{t}, \mathcal{L})$ in the coset $-\boldsymbol{t} + \mathcal{L}$. These bounded cosets, $(\boldsymbol{t} + \mathcal{L}) \cap \mathrm{Ball}_n(R)$ 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$_\beta$.

We present the new algorithm solving lattice problems based on intersections of hyperballs in Sect. 3.1 and application to co-cyclic lattices and q-ary lattices as an example in Sect. 3.2. These examples motivate the generic initialization of our algorithm as described in Sect. 3.3.

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 $\alpha^n \in \mathbb{N}_{>1}$. We also assume that the bottom lattice $\mathcal{L}_k$ permits an efficient enumeration of the β^n shortest vectors in any coset $\boldsymbol{t} + \mathcal{L}_k$ for $1 < \beta < \sqrt{3/2}$. The ultimate goal is to find the β^n shortest vectors in some coset $\boldsymbol{t}_0 + \mathcal{L}_0$ 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 β^n shortest elements in some coset $\boldsymbol{t} + \mathcal{L}$ is roughly equivalent to enumerating all lattice vectors of $\mathcal{L}$ in the ball of radius $\beta \cdot r_n \cdot \sqrt[n]{\mathrm{vol}\left(\mathcal{L}\right)}$ centered at $-\boldsymbol{t} \in \mathbb{R}^n$.

For each $i \in [0, k]$, we define a real vector $\boldsymbol{t}_i = \boldsymbol{t}_0/2^i \in \mathbb{R}^n$, and a *bounded coset* C_i that contains the β^n shortest vectors of the coset $\boldsymbol{t}_i + \mathcal{L}_i$. More formally, let us define

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

such that

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

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

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

The goal of our algorithm is to enumerate C_0, and to do that, it successively enumerates subsets $S_i \subseteq C_i$, starting from $i = k$ down to zero, containing a majority of all elements which means that $\#S_i \approx \#C_i$. Figure 1 illustrates the sequence of enumerated lists.

$$\begin{tikzcd} S_0 \subseteq C_0 \\ S_i \subseteq C_i \arrow[u, "+"', "\text{check (3.1),(3.2)}"] \\ \text{Enumerate } S_k = C_k \arrow[u, "+"', "\text{check (3.1),(3.2)}"] \end{tikzcd}$$

FIGURE 1. *Iterative creation of lists.*

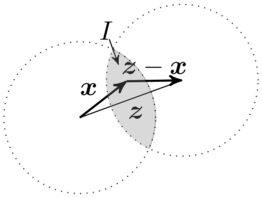

FIGURE 2. *Vector $\boldsymbol{z} \in C_{i-1}$ found as sum between $\boldsymbol{x} \in C_i$ and $\boldsymbol{z} - \boldsymbol{x} \in C_i \Leftrightarrow I \cap (\boldsymbol{t}_i + \mathcal{L}_i) \neq \emptyset$.*

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

$$\boldsymbol{x} + \boldsymbol{y} \in \boldsymbol{t}_{i-1} + \mathcal{L}_{i-1} \text{ and} \tag{3.1}$$

$$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \beta \cdot r_n \cdot \sqrt[n]{\mathrm{vol}\,(\mathcal{L}_i)} \ . \tag{3.2}$$

This means that we collect the β^n shortest vectors of the coset $C_{i-1} = \boldsymbol{t}_{i-1} + \mathcal{L}_{i-1}$ by going through the list S_i of short elements belonging to $C_i = \boldsymbol{t}_i + \mathcal{L}_i$. In practice, an equivalent way to check if condition (3.1) holds, is to use an efficient computation for the map $\varphi_{i-1} : C_i \to \mathcal{L}_i/\mathcal{L}_{i-1}$, $\boldsymbol{z} \to \boldsymbol{z} - \boldsymbol{t}_i \mod \mathcal{L}_{i-1}$ and to verify that $\varphi_{i-1}(\boldsymbol{x}) + \varphi_{i-1}(\boldsymbol{y}) = 0$. Section 3.2 shows concrete examples for φ_i which are easy to implement. Alg. 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 $\boldsymbol{t} \in \mathbb{R}^n$
Output: Almost all the $\beta^n$ shortest elements of $\boldsymbol{t} + \mathcal{L}_0$
Randomize the input target by sampling $\boldsymbol{t}_0 \in \boldsymbol{t} + \mathcal{L}$. Use for example a Discrete Gaussian
  Distribution of parameter $\sqrt{n}\|B^*\|$. This defines all the sub-targets $\boldsymbol{t}_i = \boldsymbol{t}_0/2^i$
Compute a tower of lattices $\mathcal{L}_0, .., \mathcal{L}_k$ by use of Alg. 3 such that
    - $\mathcal{L}_0 \subset \mathcal{L}_1 \subset ... \subset \mathcal{L}_k$ and $\mathrm{vol}(\mathcal{L}_i)/\mathrm{vol}(\mathcal{L}_{i-1}) = \alpha^n$
    - lattice enumeration is easy on $\mathcal{L}_k$
    - testing-morphisms $\varphi_{i-1}$ from $\boldsymbol{t}_i + \mathcal{L}_i$ to $\mathcal{L}_i/\mathcal{L}_{i-1}$ are efficient to evaluate.
$S_k \leftarrow$ Enumerate bottom coset $C_k$ (Schnorr-Euchner)
for $i = k - 1$ downto 0 do
    $S_i \leftarrow \mathrm{Merge}(S_{i+1}, \varphi_i, R_i = \beta r_n \mathrm{vol}(\mathcal{L}_i))$ (Alg. 2)
end for
return $S_0$
```

A naive implementation of the merge routine that creates S_{i-1} from S_i would just run through the β^{2n} pairs of vectors from $S_i \times S_i$, and eliminate those that do not satisfy the constraints (3.1) and (3.2). By regrouping the elements of S_i into α^n buckets, according to their value modulo $\mathcal{L}_{i-1}$, condition (3.1) implies that each element of S_i only needs to be paired with the elements of a single bucket, see Alg. 2. Heuristic 2.1 implies that each bucket contains $\approx (\beta/\alpha)^n$ elements, therefore the merge operation can then be performed in time $\left(\beta^2/\alpha\right)^n$.

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 $(\boldsymbol{t}_i + \mathcal{L}_i) \cap Ball(R_i)$ */
Input: A set of vectors $S_{i+1} \subseteq C_{i+1}$, a testing morphism $\varphi_i$ and a radius $R_i$
Output: A set $S_i$ of elements of $C_i$
$S_i \leftarrow \emptyset$
Reorganize $S_{i+1}$ into buckets indexed by the values of $\varphi_i$
for each $\boldsymbol{v} \in S_{i+1}$ do
    for each $\boldsymbol{u}$ in the bucket of index $-\varphi_i(\boldsymbol{v})$ do
        if $\|\boldsymbol{u}+\boldsymbol{v}\| \le R_i$ then
            $S_i \leftarrow S_i \cup \{\boldsymbol{u}+\boldsymbol{v}\}$
        end if
    end for
end for
return $S_i$
```

Complexity and constraints for parameters α and β. It is clear that at each level, conditions (3.1) and (3.2) imply that S_i is a subset of C_i. We now need to prove that there exist constants α and β such that a constructed list S_i contains all or at least a vast majority of C_i. If, by decreasing induction on i, S_i is close to C_i, the main requirement is that almost all points of C_{i-1} can be expressed as the sum of two points in C_i, see Fig. 2 for an illustration. This geometric constraint can be simply rephrased as follows: a vector $\boldsymbol{z} \in C_{i-1}$ is found if and only if there exists at least one vector $\boldsymbol{x}$ of the coset $\boldsymbol{t}_i + \mathcal{L}_i$ in the intersection of two balls of radius R_i, the first one centered in 0, and the second one in $\boldsymbol{z}$. It is clear that $\boldsymbol{z} - \boldsymbol{x} \in C_i = \boldsymbol{t}_i + \mathcal{L}_i$ since $2\,\boldsymbol{t}_i = \boldsymbol{t}_{i-1}$ and $\mathcal{L}_{i-1} \subseteq \mathcal{L}_i$. So if there is a point $\boldsymbol{x} \in C_i$ in the intersection $I = \mathrm{Ball}(\boldsymbol{0}, R_i) \cap \mathrm{Ball}(\boldsymbol{z}, R_i)$, we obtain $\boldsymbol{z} \in C_{i-1}$ as a sum between $\boldsymbol{x} \in C_i$ and $\boldsymbol{z} - \boldsymbol{x} \in C_i$. Under Heuristic 2.1, this occurs with high probability as soon as the intersection I of the two balls has a larger volume than $\mathcal{L}_i$. We thus require that $\mathrm{vol}\,(I)\,/\,\mathrm{vol}\,(\mathcal{L}_i) \ge K$ for some constant $K > 1$.

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 $0.692 \cdot \mathrm{vol}(\mathrm{Ball}(R_i \cdot \sqrt{1-(\alpha/2)^2}))/\sqrt{n}$. A sufficient condition on α and β is then

$$\left(\beta \cdot \sqrt{1-(\alpha/2)^2}\right)^n \ge K\sqrt{n} \text{ or alternatively} \tag{3.3}$$

$$\beta\sqrt{1-(\alpha/2)^2} \ge (1+\varepsilon_n) \tag{3.4}$$

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

Of course, for optimization reasons, we want to minimize the size of the lists β^n, and the number of steps $(\beta^2/\alpha)^n$ in the merge. Therefore we want to minimize β and maximize α under the above constraint. The total running time of Alg. 1 is given by $\mathcal{B} + \mathrm{poly}(n)\left(\beta^2/\alpha\right)^n$ where $\mathcal{B}$ represents the running time of the initial enumeration at level k (details in Sect. 3.4). For optimal parameters, inequality (3.4) 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 $(\beta^2/\alpha)^n \approx 2^{0.3774\,n}$ and the size of the lists is $\beta^n \approx 2^{0.2925\,n}$.

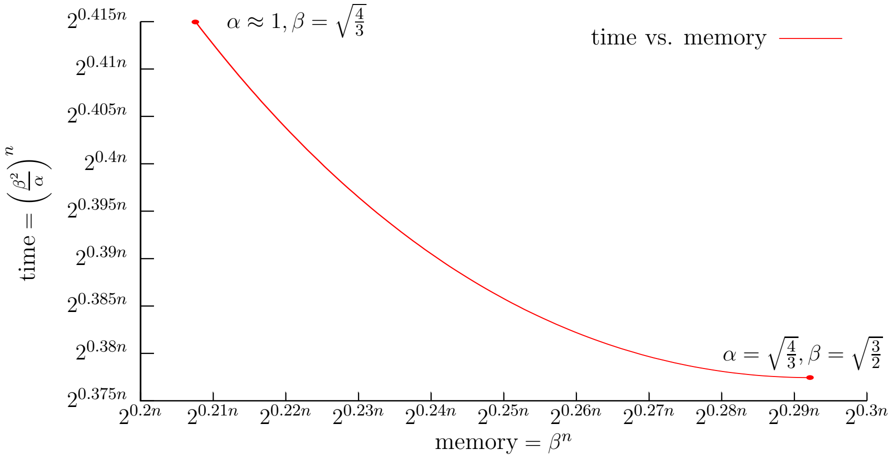

FIGURE 3. *Trade-off between memory and time for varying choices of α and β.*

Time-memory trade-off. Other choices of α and β that satisfy (3.4) 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 $\beta^n \approx 2^{0.21\,n}$ in memory, the time complexity increases to $(\beta^2/\alpha)^n \approx 2^{0.41\,n}$. In practice, we choose $\alpha > 1$ and $\beta > 0$ satisfying (3.3) with the constraint that α^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 φ_i.
Let $\mathcal{L} \subseteq \mathbb{Z}^n$ be a co-cyclic lattice given as $\mathcal{L} = \{\boldsymbol{x} \in \mathbb{Z}^n, \sum_{i=1}^n a_i x_i = 0 \mod M\}$ for large $M \in \mathbb{N}$ and random integers $a_1, .., a_n \in [0, M-1]$. The task is to enumerate $C = (\boldsymbol{t} + \mathcal{L}) \cap \mathrm{Ball}_n(R)$ where $R = \beta \cdot r_n \cdot \mathrm{vol}(\mathcal{L})^{1/n}$ 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 α 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

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

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 [**34**] in time $poly(n) \cdot |C|$ as we argue in Sect. 2. The coset testing function φ_i, which represents $\boldsymbol{x} - \boldsymbol{t}_i \mod \mathcal{L}_{i-1}$, can be implemented as $\langle \boldsymbol{a}, \boldsymbol{x} - \boldsymbol{t}_i \rangle / N^{k-i} \mod N$.

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 $\boldsymbol{x} \in \mathbb{Z}^n$,

$$\boldsymbol{x} \in \mathcal{L} \iff [(a_{1,1}x_1 + .. + a_{1,n}x_n \equiv_q 0) \wedge .. \wedge (a_{k,1}x_1 + .. + a_{k,n}x_n \equiv_q 0)] \qquad (3.5)$$

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 (3.5). Again, $\mathcal{L}_k$ is $\mathbb{Z}^n$, $\mathcal{L}_{i-1} \subseteq \mathcal{L}_i$ and $\mathrm{vol}\left(\mathcal{L}_{i-1}\right)/\mathrm{vol}\left(\mathcal{L}_i\right) = q$. The coset testing function φ_i can be computed as $\langle \boldsymbol{a}_i, \boldsymbol{x} - \boldsymbol{t}_i \rangle \mod q$.

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 $C_k = \mathbb{Z}^n \cap \mathrm{Ball}_n(R_k)$, 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 [**25**]. If the target is very close to 0, like in an SVP-setting, the coset C_k contains around $2^{0.513\,n}$ vectors[†], which differs considerably from $\beta^n \approx 2^{0.292\,n}$ 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 β^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.

3.3. *Generic creation of the tower*

Here, we present a generic method of computing the tower of $\mathcal{L}_i$'s that overcomes the problems we have shown in the previous section and 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 (3.4) 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 $\min_i \|\boldsymbol{b}_i^*\| \geq \max_i \|\boldsymbol{b}_i^*\| / \sqrt{4/3}^n$. 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 $\min_{i \in [1,n]} \|\boldsymbol{b}_i^*\| \geq \left(\frac{N^k}{n\,\mathrm{vol}(\mathcal{L})}\right)^{\frac{1}{n-1}} = \sigma$. The integer k determines the number of levels in our tower and σ is the $(n-1)$-th root of the volume of the last overlattice $\mathcal{L}_k$.

Finally, we use a slightly modified version, Alg. 4, of the unbalanced reduction algorithm from [**14**] to compute a basis $\hat{B} = [\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n]$ of $\mathcal{L}$ such that $[\hat{b}_1/N^k, \hat{b}_2, \ldots, \hat{b}_n]$ is quasi-orthogonal and provides a basis for a lattice $\mathcal{L}_k$. This naturally defines the tower of $k+1$ overlattices $\mathcal{L}_i$, where $\mathcal{L}_i$ is generated by the corresponding basis $B^{(i)} = [\frac{\hat{b}_1}{N^i}, \hat{b}_2, \ldots, \hat{b}_n]$ for $i = 0, .., k$.

Alg. 4 can be viewed as a reversed LLL-reduction algorithm: in each 2×2 dimensional projected block $B_{[i,i+1]}$, the LLL algorithm would shorten the first vector as much as possible. The unbalanced reduction focuses on decreasing the second projection $\|\boldsymbol{b}_{i+1}^*\|$ just below σ. By conservation of the volume, it suffices to replace $\boldsymbol{b}_i$ by a sufficiently large combination $\boldsymbol{b}_{i+1} + \gamma\, \boldsymbol{b}_i$. What is not trivial, is to prove that each block can be visited only once, and that

[†] Computation based on saddle point method as in [**25**] for a radius $\sqrt{\beta^2/(2\pi e) \cdot n} \approx \sqrt{0.0878 \cdot n}$.

Algorithm 3 Compute the tower of overlattices

```
Input: $B$ a (randomized) LLL-reduced basis of $\mathcal{L}$ of dimension $n$, and a target $\boldsymbol{t} \in \operatorname{span}(\mathcal{L})$
Output: Bases $B^{(i)}$ of a tower of overlattices $\mathcal{L} = \mathcal{L}_0 \subset \cdots \subset \mathcal{L}_k$. Note that given a target $\boldsymbol{t}_{i+1}$, the testing morphism $\varphi_i$ from $\boldsymbol{t}_{i+1} + \mathcal{L}_{i+1}$ to $\mathbb{Z}_N$ is implicitely defined by $\varphi_i\left(\boldsymbol{t}_{i+1} + \sum_{j=1}^{n} \alpha_j \boldsymbol{b}_j^{(i+1)}\right) = \alpha_j \mod N$
Let $N = \alpha^n$.
Let $k$ be the smallest integer s.t. $N^k \geq n \cdot \operatorname{vol}(\mathcal{L}) / \min_i \ \|\boldsymbol{b}_i^*\|^n$.
Let $\sigma = (N^k / n\operatorname{vol}(\mathcal{L}))^{\frac{1}{n}}$, thus $\sigma \leq \min_i \ \|\boldsymbol{b}_i^*\|$.
Apply Alg. 4 on input $(B, \sigma)$ to find a basis $\hat{B} = [\hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2 \ldots, \hat{\boldsymbol{b}}_n]$ of $\mathcal{L}$.
$B^{(i)} \leftarrow \left[\frac{\hat{\boldsymbol{b}}_1}{N^i}, \hat{\boldsymbol{b}}_2, \ldots, \hat{\boldsymbol{b}}_n\right]$ foreach $i \in [0, k]$
return $B^{(i)}$ for all $i$
```

tight choices of the combination coefficients γ effectively lead to a quasi-orthonormal basis, and therefore to an efficient enumeration for $\mathcal{L}_k$.

Theorem 3.1 below states the requirements for which the unbalanced reduction, Alg. 4, is of polynomial time. All steps that we need to take in order to compute the tower of overlattices are hence of polynomial complexity.

Algorithm 4 Unbalanced Reduction

```
Input: A LLL-reduced basis $B$ of an integer lattice $\mathcal{L}$ such that $\max \boldsymbol{b}_i^* / \min \|\boldsymbol{b}_i^*\| \leq \sqrt{4/3}^n$, and a target length $\sigma \leq \min \|\boldsymbol{b}_i^*\|$
Output: A basis $C$ of $\mathcal{L}$ satisfying $\|\boldsymbol{c}_1\| \leq \sigma n \operatorname{vol}(\mathcal{L}) / \sigma^n$, and for all $i \in [2, n]$, $\|\boldsymbol{c}_i^*\| \leq \sigma$ and $\frac{\sigma^{n+1-i}}{\operatorname{vol}(C_{[i,n]})} \leq n + 1 - i$.
$C \leftarrow B$
Compute the Gram-Schmidt matrices $\mu$ and $C^*$
Let $k$ be the largest index such that $\|\boldsymbol{c}_k^*\| > \sigma$
for $i = k - 1, \ldots, 1$ do
    $\gamma \leftarrow \left\lceil -\mu_{i+1,i} + \frac{\|\boldsymbol{c}_{i+1}^*\|}{\|\boldsymbol{c}_i^*\|} \sqrt{\left(\frac{\|\boldsymbol{c}_i^*\|}{\sigma}\right)^2 - 1} \right\rceil$
    $(\boldsymbol{c}_i, \boldsymbol{c}_{i+1}) \leftarrow (\boldsymbol{c}_{i+1} + \gamma \cdot \boldsymbol{c}_i, \boldsymbol{c}_i)$
    Update the Gram-Schmidt matrices $\mu$ and $C^*$.
end for
return $C$
```

Theorem 3.1 Unbalanced reduction. *Let $\mathcal{L}(B)$ be an n-dimensional integer lattice with an LLL-reduced basis $B = [b_1, .., b_n]$. Let σ be a target length $\leq \min \|\boldsymbol{b}_i^*\|$. Algorithm 4 outputs in polynomial time a basis C of $\mathcal{L}$ satisfying*

$$\|\boldsymbol{c}_i^*\| \leq \sigma \text{ for all } i \in [2, n] \tag{3.6}$$

$$\|\boldsymbol{c}_1\| \leq \sigma n \cdot vol(\mathcal{L}) / \sigma^n \tag{3.7}$$

$$\frac{\sigma^{n+1-i}}{vol(C_{[i,n]})} \leq n + 1 - i \text{ for all } i \in [2, n] \tag{3.8}$$

The proofs of Theorem 3.1 and Alg. 4 are given in Appendix B.

3.4. *Cost for initial enumeration at level k*

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

$$\mathcal{T}_{\mathrm{SE}} = \sum_{i=1}^{n} \frac{\mathrm{vol}(\mathrm{Ball}_i(r_n\beta\sigma))}{\mathrm{vol}(B^{(k)}_{[n+1-i,n]})} \leq n \sum_{i=1}^{n} V_i \cdot (r_n\beta)^i = \tilde{\mathcal{O}}\left(2^{0.398\,n}\right) \tag{3.9}$$

where the maximal term in the sum is of size $\tilde{\mathcal{O}}\left(2^{0.398\,n}\right)$. Experiments show that the above estimate is close to what we observe in practice as we present in Sect. 4.

This number of steps in the full enumeration is an exponential factor $< 2^{0.03\,n}$ larger than the complexity of the merge. In large dimensions, classical pruning techniques [**15**, **12**] can be used to cancel this exponential factor as small as $2^{0.03\,n}$. The algorithm behaves very well for a pruned enumeration of C_k and can recover solutions to SVP and CVP anyhow. Also, in practice, *i.e.*, for dimensions ≤ 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.

4. *Experimental validation*

In this section we present our experimental results of a C++-implementation of our algorithm, Alg. 1, presented in Sect. 3. We make use of the newNTL [**16**] and fplll [**7**] libraries as well as the Open MP [**30**] 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 α and β that we computed for the asymptotic case. We are hence able to enumerate vectors of a target coset $C_0 = (\boldsymbol{t}_0 + \mathcal{L}_0) \cap \mathrm{Ball}(R_0)$ 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 [**33**]. The memory requirement and running time in the course of execution match closely 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 $\#C_0 = \#\left((\boldsymbol{t}_0 + \mathcal{L}_0) \cap \mathrm{Ball}_n(R_0)\right) \approx (1 + \varepsilon_n)^n \beta^n$ elements on average where ε_n can be neglected for very large dimensions, (see also (3.3)). For accessible dimensions, we need to increase the radii of the balls slightly, by a small factor $\varepsilon_n \ll 1$, 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 ε_n can be chosen smaller, see (3.4). Figure 4 shows the relation between varying ε_n and the fraction of found vectors of C_0 for dimension $n \in \{40, 45, 50, 55, 60\}$. The optimal choice for ε_n depends on n and the fraction of C_0 we wish to enumerate.

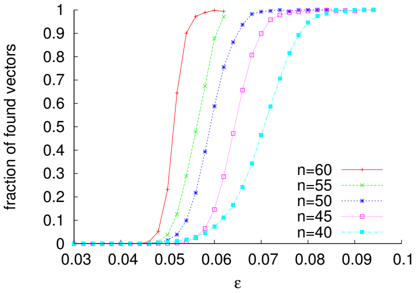

FIGURE 4. *Fraction of vectors in C_0 found for varying ε_n.*

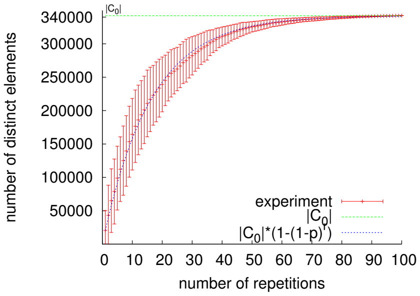

FIGURE 5. *Success probability after r repetitions, $n = 50$, $p = 0.06$.*

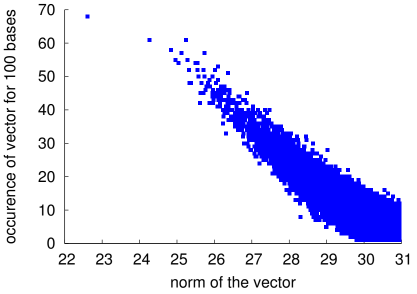

FIGURE 6. *Correlation of occurrence of vectors and their length.*

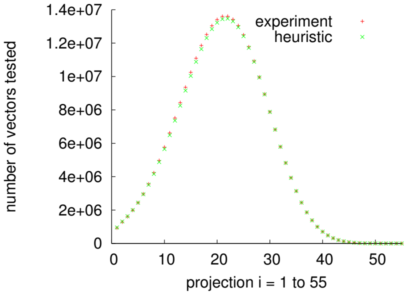

FIGURE 7. *Comparison between the actual number of nodes during enumeration and the Gaussian heuristic predictions for dimension 55.*

4.3. *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 ε_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 ε_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.

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 short vectors of a coset are easier to find than longer ones.

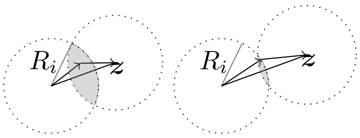

FIGURE 8. *Volume of intersection varies for vectors* $\boldsymbol{z}$ *of different length.*

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. 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 [**33**] 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,

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

containing 48 cores at 2.3 GHz, and having 256 GB of RAM. Tab. 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 loosing 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 $\mathrm{vol}(I)/\mathrm{vol}(\mathcal{L}) \geq K$ 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 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 [**33**] 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 $\varepsilon = 0.041$. 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.

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

Table 3. *Experimental results with pruning, $n \in \{75, 90\}$, $\alpha = \sqrt{4/3}$ 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 cut	47	46	46	48	50 cut	69	solved
$n = 75$, $\varepsilon = 0.044$	% of Gauss. heuris.	-	-	35 cut	35 cut	30	25	20	15	8	6.4	solved
$n = 90$, $\varepsilon = 0.0416$	% of Gauss. heuris.	70 cut	40 cut	40 cut	40 cut	40 cut	40 cut	40 cut	40 cut	40 cut	70	solved
$n = 90$, $\varepsilon = 0.0416$	% of Gauss. heuris.	70 cut	33 cut	33 cut	33 cut	33 cut	33 cut	33 cut	33 cut	33 cut	61	solved

by (3.9) 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. Fig. 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 ±2% of the average value.

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.

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, and P. Q. Nguyen. Worst-case to average-case reduction for almost all lattices. To appear, 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 funtionalities. `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. A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *J. Cryptology*, 11(3):161–185, 1998.

20. 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.

21. 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.

22. A. Korkine and G. Zolotarev. Sur les formes quadratiques. *Mathematische Annalen 6*, pages 336–389, 1973.

23. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

24. A. May, A. Meurer, and E. Thomae. Decoding random linear codes in $2^{0.054n}$. In *ASIACRYPT*, pages 107–124, 2011.

25. J. E. Mazo and A. M. Odlyzko. Lattice points in high-dimensional spheres. *Monatshefte für Mathematik*, 110:47–62, 1990.

26. 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.

27. 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.

28. D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480. ACM/SIAM, 2010.

29. P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2008.

30. OpenMP Architecture Review Board. OpenMP API version 4.0, 2013.

31. X. Pujol and D. Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009:605, 2009.

32. R. Rankin. On positive definite quadratic forms. *J. Lond. Math. Soc.*, 28:309–314, 1953.

33. M. Schneider, N. Gama, P. Baumann, and P. Nobach. http://www.latticechallenge.org/svp-challenge/halloffame.php.

34. K.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

35. J. L. Thunder. Higher-dimensional analogs of Hermite's constant. *Michigan Math. J.*, 45(2):301–314, 1998.

36. 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.

37. 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.

Appendix A. *Intersection of hyperballs*

The volume of the intersection, $vol_I(d)$, 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 $D = \sqrt{1 - \left(\frac{d}{2}\right)^2}$, 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 1.

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

$$\frac{2V_{n-1}}{(n+1)V_n} \arccos\left(\frac{d}{2}\right) \leq \frac{vol_I(d)}{vol(Ball_n(D))} \leq \frac{2V_{n-1}}{(\frac{n}{2}+1)V_n} \arccos\left(\frac{d}{2}\right)$$

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

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

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

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:

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

and

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

which proves the lemma. □

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 1. *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:*

$$R^n vol_I\,(d) \geq \frac{0.692}{\sqrt{n}} \cdot R^n vol\left(Ball_n\left(\sqrt{1 - \left(\frac{d}{2}\right)^2}\right)\right).$$

Appendix B. *Proof of Theorem 3.1 and Algorithm 4:*

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 $\|\boldsymbol{b}_i^{*\text{new}}\|$ during iteration i. Note that x_i is also $\|\boldsymbol{b}_i^{*\text{old}}\|$ during the next iteration (of index $i-1$ since i goes backwards).

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

$$a_i x_{i+1} \leq x_i \leq a_i x_{i+1} + \sigma a_i \quad . \tag{B.1}$$

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

$$\begin{gathered} R \leq \|\boldsymbol{b}_i^{*\text{new}}\| \leq R + \|\boldsymbol{b}_i^{*\text{old}}\| \\ \text{where } R = \|\boldsymbol{b}_{i+1}^{*\text{old}}\| \|\boldsymbol{b}_i^{*\text{old}}\|/\sigma \quad . \end{gathered} \tag{B.2}$$

This condition can always be fulfilled with a primitive vector of the form $\boldsymbol{b}_i^{\text{new}} = \boldsymbol{b}_{i+1}^{\text{old}} + \gamma \boldsymbol{b}_i^{\text{old}}$ for some $\gamma \in \mathbb{Z}$. Since the volume is invariant, the new $\|\boldsymbol{b}_{i+1}^{*\text{new}}\|$ is upper-bounded by σ. And by construction, Equation (B.2) is equivalent to the invariant (B.1) since $\|\boldsymbol{b}_i^{*\text{old}}\| = a_i\sigma$, $\|\boldsymbol{b}_i^{*\text{new}}\| = x_i$ and $\|\boldsymbol{b}_{i+1}^{*\text{old}}\| = x_{i+1}$.

By developping (B.1), we derive a bound on x_1:

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

which proves (3.7). Similarly, one obtains that $x_i \leq (n+1-i)\, \sigma \, \mathrm{vol}(B_{[i,n]})/\sigma^{n+1-i}$, which is equivalent to (3.8). 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 γ_i is $\left\lceil -\mu_{i+1,i} + \frac{x_{i+1}}{\sigma}\sqrt{1 - \frac{1}{a_i^2}} \right\rceil$. Since each x_{i+1} is bounded by

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

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