Another L makes it better? Lagrange meets LLL and may improve BKZ pre-processing

Sébastien Balny[*] Claire Delaplace[†] Gilles Dequen[‡]

Abstract
We present a new variant of the LLL lattice reduction algorithm, inspired by Lagrange notion of pair-wise reduction, called L4. Similar to LLL, our algorithm is polynomial in the dimension of the input lattice, as well as in $\log M$, where M is an upper-bound on the norm of the longest vector of the input basis. We experimentally compared the norm of the first basis vector obtained with LLL and L4 up to dimension 200. On average we obtain vectors that are up to 16% shorter. We also used our algorithm as a pre-processing step for the BKZ lattice reduction algorithm with blocksize 24. In practice, up to dimension 140, this allows us to reduce the norm of the shortest basis vector on average by 3%, while the runtime does not significantly increases. In 10% of our tests, the whole process was even faster.

Keywords. Lattice Reduction. LLL. Short lattice vectors.

Acknowledgement: We would like to thank the MatriCS HPC Platform from UPJV[1] that provided computational resources needed to exhibit the experimental results of this works. Resources are publicly available at the following link:
`https://zenodo.org/records/13847623`

1 Introduction

A lattice is a discrete subgroup of $\mathbb{R}^n$. Usually it is defined as the set

$$\mathcal{L}(\mathbf{B}) = \left\{ \mathbf{B}\mathbf{u} : \mathbf{u} \in \mathbb{Z}^d \right\},$$

where $\mathbf{B}$ is an n-by-d full-rank matrix with $d \leq n$, called the basis of the lattice. There are infinitely many bases of a given lattice Λ. Those bases are said to be equivalent. However, it does not mean that they all share the same qualities. In most applications, we are interested in finding either short lattice vectors or a lattice vector which is close to some target in $\mathbb{R}^n$. These problems are known to be hard to solve given a random basis $\mathbf{B}$ of Λ, but become easier if $\mathbf{B}$ satisfy some "good" properties, typically $\mathbf{B}$ consisting of short and "somehow orthogonal" vectors. Such a basis is called reduced basis.

The idea of lattice reduction is not new and can be traced back as early as the 1850's with a first definition of size reduction proposed by Hermite [14]. Korkine and Zolotarev [17, 18] later gave a stronger notion which is called the HKZ (for Hermite-Korkine-Zolotarev) reduction. Amongst other properties, it has the following: if a basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_d]$ is HKZ reduced, then $\mathbf{b}_1$ is a non zero-vector with minimal norm in $\mathcal{L}(\mathbf{B})$. However computing an HKZ-reduced basis from an arbitrary one is not an easy task. Using basic enumeration, it requires $2^{\mathcal{O}(d^2)}$ operations. Kannan later showed how to decrease this complexity down to $2^{\mathcal{O}(d \log d)}$ using a recursive procedure [15]. The famous LLL algorithm [19], called after its designers Lenstra, Lenstra and Lovász, offers a weaker, yet more practical reduction. This algorithm has known many variants and improvements along the years (e.g., [23, 25, 26]), and has many fields of application, such as cryptography, polynomial factorisation, as well as Integer Linear Programming. Another well-known lattice reduction algorithm, called Block-Korkine-Zolotarev (BKZ) [27, 28] and its later improvement BKZ2.0 [7] offers a tradeoff between efficiency and quality. Roughly speaking, BKZ with blocksize β (BKZ-β) computes blockwise HKZ-reductions on smaller β-dimensional sub-lattices of the original lattice Λ. The closer β is to the dimension d, the better reduction, however the time complexity grows super-exponentially in β. Finally, even prior to Hermite's work, the particular case of 2-dimensional lattices had been studied by Lagrange [8]. Lagrange's Algorithm, often wrongly attributed to Gauss, allows to compute an HKZ-reduced lattice basis quite efficiently, but this method does not seem easy to generalize to higher dimensions d, although some attempts have been made when d is small [24, 29].

Arguably the most famous lattice problem is the Shortest Vector Problem (SVP). Given as input a basis of a lattice Λ, the goal is to find a non-zero vector of Λ whose norm is minimal. This problem has been proved to be NP-Hard under randomized reductions for the Euclidean Norm by Ajtai [3]. As such various relaxations of the problem have been defined, where the goal is to find any vector whose norm is at most γ times (an estimation of) the norm of the shortest non-zero vector. If γ is a constant, these γ-approximate variants are known to remain NP-Hard [20]. However it can

[*]MIS, Université de Picardie Jules Verne, Amiens, France
[†]MIS, Université de Picardie Jules Verne, Amiens, France
[‡]MIS, Université de Picardie Jules Verne, Amiens, France
[1]`https://www.matrics.u-picardie.fr/`

be solved in $\mathsf{poly}(d)$ operations if $\gamma = \Theta(2^d)$ [19]. We do not know the exact difficulty of these approximate variants for other parameters, although there are several theoretical results that suggest that they cease to be NP-Hard as soon as $\gamma = \Omega(\sqrt{d/\log d})$ [2, 12]. Another famous problem is the Closest Vector Problem (CVP), where the goal is to find the closest lattice vector to target in $\mathbb{R}^n$. This problem is also NP-Hard [31, 16], and has several relaxations and variants, such as the Approximate Closest Vector Problem, or the Bounded Distance Decoding Problem. We can see why lattice problems are easier to solve if we are given as input a basis which already consists of small vectors. If by chance, the basis already contains a vector of minimal norm, then SVP is trivially solved. Thus, an important part of lattice's algorithms focuses on reducing the input basis.

1.1 Our contributions. We present a new variant of the LLL algorithm, inspired by the two-dimensional notion of Lagrange-reduction, which we call the L4 algorithm, for Lagrange-LLL. For a given lattice, if we denote by $\mathbf{B}^{L^3}$ an LLL-reduced basis, and $\mathbf{B}^{L^4}$ the output of our L4 algorithm, empirically, we see that the first vector of $\mathbf{B}^{L^4}$ is up 16% shorter on average than the first vector of $\mathbf{B}^{L^3}$ for dimension 200.

More interestingly, we also notice that, when used as a pre-processing step for BKZ, our algorithm slightly improves the quality of the output, compared to BKZ with LLL as a preprocessing. More precisely for dimension 60 to 140 we obtain on average a shortest basis vector which is 3% shorter while the runtime of the overall process is only slightly slower, and there are even cases where it is faster than BKZ-24. Furthermore, we argue that the gap in performance could possibly be bridged with a better implementation of L4, since our code was made mostly for testing the quality of the output and we did not try to optimise it. All in all, we think that, even though the gain seems incremental, L4 offers an elegant alternative to LLL when used as a pre-processing to BKZ, and leaves room to further improvements.

1.2 Experiments. In this paper, we are mostly interested in the cryptographic setting. As such, all experiments were performed on random lattices generated using the Darmstadt SVP Challenge generator [5]. The lattices are random in the sense of Goldstein and Mayer [13], which are known to provide hard instances for the Shortest Vector Problem. Our algorithm has been implemented in Python, using the FPyLLL library [9] and the code is available here:

`https://zenodo.org/records/13847623`[2]

Most of our tests were run on an Intel(R) Xeon(R) cluster with a 2.40 GHz processor. We used 32 GB of RAM for tests up to dimension 170 and 128 GB for dimension 180 and more. This first machine is referred to as Computer 1 in this paper. Some additional tests were performed on an Intel i5-6300U laptop, with 3.0 GHz, using 8 GB of RAM. We call this machine Computer 2.

2 Preliminary

2.1 Notation. Let $\mathbf{v}$ be a vector of $\mathbb{R}^n$, we denote by $\|\mathbf{v}\|$ the euclidean norm of $\mathbf{v}$. For a set of vectors $\mathcal{S}$, we denote by $\#\mathcal{S}$ the cardinal of $\mathcal{S}$ and by $\mathcal{N}_{\max}(\mathcal{S})$ (resp. $\mathcal{N}_{\min}(\mathcal{S})$) the norm of the longest (resp. the shortest) vector in $\mathcal{S}$. Formally

$$\mathcal{N}_{\max}(\mathcal{S}) = \max_{\mathbf{v}\in\mathcal{S}}\{\|\mathbf{v}\|\} \quad \text{and} \quad \mathcal{N}_{\min}(\mathcal{S}) = \min_{\mathbf{v}\in\mathcal{S}}\{\|\mathbf{v}\|\}$$

For any column vector $\mathbf{v}$ (resp. matrix $\mathbf{M}$), we denote the transpose vector (resp. transpose matrix) by $\mathbf{v}^t$ (resp. $\mathbf{M}^t$). For any two vectors $\mathbf{v}$ and $\mathbf{w}$, $\mathbf{v}^t\mathbf{w}$ represents the dot product between $\mathbf{v}$ and $\mathbf{w}$. Given two vectors $\mathbf{u}$ and $\mathbf{v}$, $\mathbf{u} \pm \mathbf{v}$ is the vector such that

$$\mathbf{u} \pm \mathbf{v} = \begin{cases} \mathbf{u} - \mathbf{v} & \text{if } \mathbf{u}^t\mathbf{v} \geq 0, \\ \mathbf{u} + \mathbf{v} & \text{otherwise.} \end{cases} \tag{2.1}$$

If we denote by $|\cdot|$ the absolute value, notice that

$$\|\mathbf{u} \pm \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 - 2|\mathbf{u}^t\mathbf{v}| \leq \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2.$$

We call $\mathbf{u} \pm \mathbf{v}$ the difference between $\mathbf{u}$ and $\mathbf{v}$. For any square matrix $\mathbf{M}$ of $\mathbb{R}^{n\times n}$, we denote by $\det(\mathbf{M})$ the determinant of $\mathbf{M}$.

2.2 Lattices background. Given two positive integers, $0 < d \leq n$, and a set $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d]$ of d linearly independent vectors of $\mathbb{R}^n$, we define Λ, the lattice spanned by $\mathbf{B}$ as

$$\Lambda = \mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{d} x_i\mathbf{b}_i \;:\; x_i \in \mathbb{Z} \right\}.$$

We say that $\mathbf{B}$ is a basis of Λ, $d \leq n$ its dimension or rank. If $d = n$, we say that Λ is full-rank. We call integral lattice a lattice spanned by a basis $\mathbf{B}$ whose vectors consist only of integer coefficients. These are the lattices that are interesting in the cryptographic setting.

Given two bases $\mathbf{B}_1$ and $\mathbf{B}_2$, $\mathcal{L}(\mathbf{B}_1) = \mathcal{L}(\mathbf{B}_2)$ if and only if there is an unimodular matrix $\mathbf{U} \in \mathbb{Z}^{d\times d}$ (i.e.,

[2]Also available via github `https://github.com/sbalny/L4`

$\det(\mathbf{U}) = \pm 1$) such that $\mathbf{B}_2 = \mathbf{B}_1\mathbf{U}$. We say that $\mathbf{B}_1$ and $\mathbf{B}_2$ are equivalent, since they are both bases of the same lattice.

For $m \geq d$, we call generating set of a d-dimensional lattice Λ any set $\mathcal{S} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ such that any vector $\mathbf{w} \in \Lambda$ is an integer linear combination of the vectors of $\mathcal{S}$, and vice-versa. We use the following notation

$$\Lambda = \mathcal{L}(\mathcal{S}).$$

Given a lattice $\Lambda = \mathcal{L}(\mathbf{B})$, the volume of Λ is defined as

$$\mathrm{vol}(\Lambda) = \det(\mathbf{B}^t\mathbf{B})^{1/2}.$$

For a specific lattice, this value is a constant and thus does not depend on the basis $\mathbf{B}$.

For a given lattice Λ, we denote by $\lambda_1(\Lambda)$ the norm of the shortest non-zero vector. We usually do not know the exact value of $\lambda_1(\Lambda)$, but we have some rather precise estimations. Minkowski's First Theorem [21] gives us an upper bound stating that, if $\Lambda = \mathcal{L}(\mathbf{B})$ is a full-rank lattice of dimension n, $\lambda_1(\Lambda) \leq \sqrt{n} \cdot \mathrm{vol}(\Lambda)^{1/n}$. Additionally, if Λ is a full-rank lattice of dimension n, the Gaussian Heuristic gives us an estimation of the number of lattice points that can be found inside any measurable subset of $\mathbb{R}^n$ leading to the following estimation of $\lambda_1(\Lambda)$:

$$\mathrm{GH}(\Lambda) = \frac{\Gamma(n/2+1)^{1/n}}{\sqrt{n}} \mathrm{vol}(\Lambda)^{1/n}. \tag{2.2}$$

Similar results can also be obtained for lattices of rank $d < n$ [4, 22, 6].

2.3 Shortest Vector Problem.

One of the most famous lattice problem is the Shortest Vector Problem (SVP) defined as follows.

Definition 2.1. (Shortest Vector Problem) *Given a d-dimensional lattice Λ represented by a basis $\mathbf{B}$, find a vector $\mathbf{v}_0 \in \Lambda$ such that $\|\mathbf{v}_0\| = \lambda_1(\Lambda)$.*

As previously mentioned, we usually do not know $\lambda_1(\Lambda)$. Furthermore, since this problem is hard to solve, several relaxations have been introduced. For instance, the Approximate Shortest Vector Problem, with approximation factor $\gamma \geq 1$ (ASVP$_\gamma$), consists in finding a non zero vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}\| \leq \gamma\lambda_1(\Lambda)$. For a full-rank lattice Λ of dimension n, the Hermite Shortest Vector Problem (HSVP$_\gamma$) aims to find a non-zero vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v}\| \leq \gamma \mathrm{vol}(\Lambda)^{1/n}$.

In section 4, we consider the variant proposed in the Darmstadt SVP Challenge [1], where the goal is to find a non zero vector in a full-rank lattice whose norm is less than $\gamma\mathrm{GH}(\Lambda)$.

2.4 Lattice Reduction.

In the case of a vector space, there is a well known algorithm that takes as input a random basis $\mathbf{B}$ and reduces it in order to produce a basis $\mathbf{B}^*$ whose vectors are all pair-wise orthogonal. This is the Gram-Schmidt orthogonalization process we recall below.

Definition 2.2. *For a sequence of d linearly independent vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d$, we define their Gram-Schmidt Orthogonalization (GSO) as the sequence of vectors $\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_d^*$ such that*

$$\mathbf{b}_1^* = \mathbf{b}_1 \quad \textit{and} \quad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$$

where $\mu_{i,j} = \dfrac{\mathbf{b}_i^t\mathbf{b}_j^*}{\|\mathbf{b}_j^*\|^2}$.

One may be tempted to apply a similar method to reduce the basis of a lattice. However, the issue here lies in the fact that the $\mu_{i,j}$ coefficients are rational, not integers.

Special case of dimension 2. Consider first the case of dimension 2. Given a basis $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2]$, Assuming that $\|\mathbf{b}_1\|^2 \leq \|\mathbf{b}_2\|^2$, Lagrange's [8] following algorithm can be seen as a discrete variant of the GSO process.

1. Reduce $\mathbf{b}_2$ by setting $\mathbf{b}_2 \leftarrow \mathbf{b}_2 - \lfloor\mu_{2,1}\rceil\mathbf{b}_1$.
2. If $\|\mathbf{b}_2\|^2 < \|\mathbf{b}_1\|^2$ swap the two vectors and go to step 1.
3. Stop when $|\mathbf{b}_1^t\mathbf{b}_2| \leq \min(\|\mathbf{b}_1\|^2, \|\mathbf{b}_2\|^2)/2$.

At the end, the vectors cannot be reduced anymore.

Definition 2.3. (L-reduction) *A pair of vectors $(\mathbf{b}_1, \mathbf{b}_2)$ is Lagrange (L)-reduced if*

$$|\mathbf{b}_1^t\mathbf{b}_2| \leq \frac{\min\left(\|\mathbf{b}_1\|^2, \|\mathbf{b}_2\|^2\right)}{2}$$

We first extend this definition to the following

Definition 2.4. (Pair-wise L-reduction) *A set of linearly independent vectors $\mathcal{S}$ is said to be L-reduced if for all $(\mathbf{u}, \mathbf{v}) \in \mathcal{S}^2$, $(\mathbf{u}, \mathbf{v})$ is L-reduced.*

In [29], Semaev gave a similar definition for a set $\mathcal{S}$ consisting only of three vectors. We generalize it to any set of linearly independent vectors. We also make the following trivial remark, which is a special case of [29] Lemma 1.

Lemma 2.1. *If the pair $(\mathbf{u}, \mathbf{v})$ is L-reduced, then*

$$\|\mathbf{u} \pm \mathbf{v}\| \geq \max(\|\mathbf{u}\|, \|\mathbf{v}\|).$$

In higher dimension. The LLL Algorithm [19], can be seen as an extension of Lagrange's algorithm to higher dimensions. Lenstra, Lenstra and Lovász give the following notion of lattice reduction.

Definition 2.5. (LLL-reduction) *We say that* $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d]$ *is a* δ*-LLL reduced basis if the following holds:*

1. $|\mu_{i,j}| \leq \frac{1}{2}$ *(size reduction)*
2. $\forall\, 1 \leq i \leq d$, $\delta \|\mathbf{b}_i^*\|^2 \leq \|\mu_{i+1,i}\mathbf{b}_i^* + \mathbf{b}_{i+1}^*\|^2$ *(Lovász's condition).*

where $\mathbf{B}^*$ *is the GSO of* $\mathbf{B}$*, and the* $\mu_{i,j}$ *coefficients are defined as in Definition 2.2.*

For simplicity, if there exists a $1/4 < \delta < 1$ such that $\mathbf{B}$ is δ-LLL reduced we say that $\mathbf{B}$ is LLL-reduced. Taking as input an arbitrary basis, the LLL Algorithm outputs an LLL-reduced basis in time $\mathsf{poly}(d)$ where d is the dimension of the lattice. Furthermore, these conditions imply that $\mathbf{b}_1$ is a solution to the ASVP$_\gamma$ with $\gamma = \mathcal{O}(2^{\frac{d}{2}})$.

$$\|\mathbf{b}_1\|^2 \leq \left(\frac{2}{(4\delta - 1)^{1/2}}\right)^{d-1} \lambda_1(\Lambda).$$

As such, with the usual choice of $\delta = 3/4$ the LLL Algorithm 1 solves the ASVP$_\gamma$ problem, with a $2^{\frac{d-1}{2}}$ approximation factor.

Algorithm 1 The LLL algorithm

```
Require: a lattice basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_d] \in \mathbb{Z}^{n \times d}$
Ensure: $\delta$-LLL reduced basis of $\mathcal{L}(\mathbf{B})$
  Start : Compute the GSO $\mathbf{B}^* = [\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*]$ of $\mathbf{B}$
  Reduction Step:
  for $i = 2$ to $d$ do
      for $j = i - 1$ down to 1 do
          $\mathbf{b}_i \leftarrow \mathbf{b}_i - c_{i,j}\mathbf{b}_j$ where $c_{i,j} = \lceil \mathbf{b}_i^t \mathbf{b}_j^* / \|\mathbf{b}_j^*\|^2 \rfloor$
      end for
  end for
  Swap Step:
  if $\exists i \in \{1, \ldots, d\}$ s.t. $\delta\|\mathbf{b}_i^*\|^2 > \|\mu_{i+1,i}\mathbf{b}_i^* + \mathbf{b}_{i+1}^*\|^2$ then
      $\mathbf{b}_i \leftrightarrow \mathbf{b}_{i+1}$
      go to Start
  end if
```

In this paper, we use a variant of LLL introduced by Pohst [25], which extends the algorithm when the input is not a basis of a given lattice Λ, but rather a generating family. This variant was later refined by Nguyen and Stehlé [23], where the authors also present a "quadratic variant" of LLL, which they refer to as the L^2 algorithm. The L^2 algorithm also takes into account difficulties related to dealing with floating point arithmetic. More precisely, their algorithm outputs an LLL-reduced basis $\mathbf{B}$ of a lattice of dimension d in time

$$T_{Basis} = \mathcal{O}\left(d^4 n(d + \log M)\log M\right), \tag{2.3}$$

where M is an upper bound on the norm of the vectors in the input basis. The algorithm is said to be "quadratic" as it grows only quadratically with respect to $\log M$ without relying on fast integer-multiplication.

Now, given as input a generating set $\mathcal{S}$ of $m > d$ vectors, [23] variant of Pohst algorithm outputs an LLL-reduced basis in time

$$T_{Set} = \mathcal{O}\left(nd^2(d + \log M)(d^2 \log M + m^2)\right). \tag{2.4}$$

where $M = \mathcal{N}_{\max}(S)$.

Remark 2.1. *The FPLLL implementation [9] of LLL we used in our tests is actually based on [23] variant.*

Since all of these algorithms rely on the original idea of Lenstra, Lenstra and Lovász and compute an LLL-reduced basis from a set of generating vectors, we will indiscriminately call them all LLL reduction in the rest of the paper.

Computational model. Since our complexity analysis uses result from [23, 30], we consider the same bit-complexity model, where all integers we deal with are less than some upper bound $M \geq \mathcal{N}_{\max}(\mathbf{B})$, $\mathbf{B}$ being the input basis. Thus, each integer can be stored using $\mathcal{O}(\log M)$ bits. We also use naive integer multiplication, following [30].

We focus on integral lattice. This means that the vectors we deal with all have integer coefficients. As such, apart from the floating point arithmetic arising in the LLL computation, which is handled by [23, 30], all the operations we perform in our algorithm are on integers, and we do not need to worry about inaccuracies arising from floating point arithmetic. We stress that these methods could be extended to deal with other kind of lattices, but, as we are interested in the cryptographic setting we did not investigate real lattices any further.

3 A new basis reduction algorithm

The idea behind our work comes from the following simple observation. Even if $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_d]$ is a LLL-reduced basis of some d-dimensional lattice Λ, though the $(\mathbf{b}_1, \mathbf{b}_j)$ pairs are L-reduced, this is not the case in general for $(\mathbf{b}_i, \mathbf{b}_j)$ with $1 < i < j \leq d$. In fact, with the classical choice of $\delta = 3/4$, even for relatively small dimensions d we are able to find some pairs $(\mathbf{b}_i, \mathbf{b}_j)$ for which the following $\|\mathbf{b}_i \pm \mathbf{b}_j\| \le \max(\|\mathbf{b}_i\|, \|\mathbf{b}_j\|)$ holds.

For instance, in dimension 40, the average number of such pairs out of 1000 instances is 22. In dimension 200, it grows up to 465. So it is far from being a small amount especially when the dimension grows.

From here, we came up with the following straightforward idea to improve the quality of an LLL-reduced basis. Compute a generating set $\mathcal{S}$ consisting of all the input basis vectors, plus short new ones generated as the difference between two non L-reduced lattice vectors. Our first intuition was to construct $\mathcal{S}$ as

$$\mathcal{S} = \mathbf{B} \cup \{\mathbf{u} = \mathbf{b}_i \pm \mathbf{b}_j : \|\mathbf{u}\| \le \max(\|\mathbf{b}_i\|, \|\mathbf{b}_j\|)\}, \tag{3.5}$$

for all $1 < i < j \le d$, as it seemed quite natural. We call this method the Inflate Procedure. However, we came up an alternative process which provides better results, for a similar runtime.

Once $\mathcal{S}$ is constructed, we sort it by increasing norm and perform an LLL-reduction to obtain an updated basis $\mathbf{B}'$. We restart the whole process with $\mathbf{B}'$ as input. We call this whole algorithm L4.

Termination. The main point of concern is when to terminate the algorithm. The first idea that comes to mind would be to repeat the process until $\mathbf{B}'$ is pairwise L-reduced. However, our experiments show that such an algorithm is likely to never terminate, even in small dimension. In fact, the number of pairs $(\mathbf{b}_i, \mathbf{b}_j)$ that are not L-reduced does not seem to decrease with the number of iterations. Instead, we repeat the whole process while the norm of the first vector of the output basis is strictly smaller than the norm of the first vector of the input basis.

Algorithm 2 L4

```
Require: An LLL-reduced basis $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_d]$ in $\mathbb{Z}^{n \times d}$.
Ensure: An LLL-reduced basis $\mathbf{B}' = [\mathbf{b}'_1, \dots, \mathbf{b}'_d]$ such that $\mathcal{L}(B) = \mathcal{L}(B')$ and $\|\mathbf{b}'_1\| \le \|\mathbf{b}_1\|$.
1: $N_{new} \leftarrow \|\mathbf{b}_1\|^2$
2: do
3:     $N \leftarrow N_{new}$
4:     $\mathcal{S} \leftarrow$ Sample($\mathbf{B}$)
5:     $\mathbf{B} \leftarrow$ LLLReduce($\mathcal{S}$)
6:     $N_{new} \leftarrow \|\mathbf{b}_1\|^2$
7: while $N_{new} < N$
```

A full description of the L4 is given in Algorithm 2. At the end, the basis $\mathbf{B}'$ we obtain is still not pairwise L-reduced. However, we have already improved the quality of the reduction, in the sense that the shortest vector is shorter than the one obtained after a simple LLL-reduction. We discuss this gain in more details in Section 4.

3.1 Construction of $\mathcal{S}$. It is clear that the bottom line of our algorithm is the construction of the generating set $\mathcal{S}$, as the quality of the basis $\mathbf{B}'$ we obtain at the end of an iteration will depend on the vectors in this set. We have the following requirements: (1) $\mathcal{S}$ must be a generating set of our input lattice Λ, (2) $\mathcal{N}_{\max}(\mathcal{S}) = \mathcal{N}_{\max}(\mathbf{B})$, where $\mathbf{B}$ is the input LLL-reduced basis, (3) $\mathcal{N}_{\min}(\mathcal{S}) \le \mathcal{N}_{\min}(\mathbf{B})$, (4) the construction of $\mathcal{S}$ must be fast. Typically we did not want to compute more than $\mathcal{O}(d^2)$ differences, which would be the number of differences required to compute $\mathcal{S}$ as given in Equation 3.5.

We start by initialising $\mathcal{S}$ to $\mathbf{B}$, but then, instead of inflating $\mathcal{S}$ with all $\mathbf{b}_i \pm \mathbf{b}_j$ of norm smaller than $\max(\|\mathbf{b}_i\|, \|\mathbf{b}_j\|)$, we sample new vectors $\mathbf{u} = \mathbf{w} \pm \mathbf{v}$, where $\mathbf{w}$ and $\mathbf{v}$ are drawn uniformly at random from $\mathcal{S}$. We append $\mathbf{u}$ to $\mathcal{S}$ only if its norm is smaller than $\max(\|\mathbf{v}\|, \|\mathbf{w}\|)$. Picking our starting points $\mathbf{w}$ and $\mathbf{v}$ from $\mathcal{S}$ instead of $\mathbf{B}$ allows us to possibly choose newly generated short vectors to perform the reduction, increasing the set of possibilities. We call this step the Sample procedure. It is fully described in Algorithm 3.

More precisely, we fix a constant $0 < \alpha_1 \le 1$, and pick $\alpha_1 d$ starting points $\mathbf{w}$ at random in $\mathcal{S}$. Then, for each of these starting points, we pick $\alpha_2 d$, with $0 < \alpha_2 \le 1$ new random vectors $\mathbf{v}$ and compute the difference between $\mathbf{w}$ and $\mathbf{v}$, before testing if it is worth keeping it. We could have chosen to have a single loop where we pick two random vectors at each of the $\alpha_1\alpha_2 d^2$ iterations, but this leads to similar overall results while being slightly slower so we settled for this method instead.

We chose $\alpha_1 = 1$ and $\alpha_2 = 1/2$, in order to compute roughly the same amount of differences than with the Inflate procedure. We also noticed that increasing α_2 to 1 does not change the overall quality of the output, while it gets significantly slower. Furthermore, we also tried to increase the number of differences to $\mathcal{O}(d^3)$ and this did not improve our basis reduction at all. This is probably due to the fact that, by the Birthday Paradox, the same vectors are picked over again, and thus most of the operations are redundant.

3.2 Analysis of L4. We claim that the L4 algorithm always terminates and furthermore outputs a basis of "better quality" than LLL, in the sense that the first basis vector would be shorter. We also argue that the runtime of our algorithm remains polynomial in the dimension of the lattice, also using a polynomial amount

Algorithm 3 Sample

```
Require: LLL-reduced lattice basis $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_d]$ of a lattice $\Lambda$.
Ensure: $\mathcal{S}$ a set of $m \geq d$ vectors such that $\mathcal{N}_{\max}(\mathcal{S}) = \mathcal{N}_{\max}(\mathbf{B})$ and $\mathcal{L}(\mathcal{S}) = \Lambda$.
    Fix two constants $\alpha_1$ and $\alpha_2$, such that $0 < \alpha_i \leq 1$, for $i \in \{1, 2\}$.
    $\mathcal{S} \leftarrow \{\mathbf{b}_1, \dots, \mathbf{b}_d\}$
    repeat $\alpha_1 d$ times
        $\mathbf{w} \overset{\$}{\leftarrow} \mathcal{S}$
        repeat $\alpha_2 d$ times
            $\mathbf{v} \overset{\$}{\leftarrow} \mathcal{S}$
            $\mathbf{u} \leftarrow \mathbf{w} \pm \mathbf{v}$
            if $0 < \|\mathbf{u}\|^2 \leq \max(\|\mathbf{w}\|^2, \|\mathbf{v}\|^2)$ then
                if $\mathbf{u} \notin \mathcal{S}$ then
                    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{u}\}$
                end if
            end if
        end
    end
    SORT($\mathcal{S}$)                    ▷ $\mathcal{S}$ sorted by increasing norm
    return $\mathcal{S}$
```

of memory. More precisely we show the following:

THEOREM 3.1. (ANALYSIS OF L4) *Given as input an n-by-d basis $\mathbf{B}$ of an integral lattice Λ, such that $\mathcal{N}_{\max}(\mathbf{B}) = M$, the L4 algorithm terminates after a finite number k of iterations and outputs an LLL-reduced basis $\mathbf{B}'$ such that $\mathcal{N}_{\min}(\mathbf{B}') \leq \mathcal{N}_{\min}(\mathbf{B})$.*

Furthermore, its time complexity is given by

$$T = \mathcal{O}(kT_{set}),$$

where $T_{set} = \mathcal{O}(nd^4(d + \log M)(d^2 + \log M))$ is the time complexity required to compute an LLL-reduced basis of Λ from a generating set of $\mathcal{O}(d^2)$ vectors. The memory required is $\mathcal{O}(d^2 n \log M)$.

We start by showing the termination and the correctness of our algorithm.

LEMMA 3.1. *Given as input a basis $\mathbf{B}$ of an integral lattice Λ, the* SAMPLE *procedure returns a generating set $\mathcal{S}$ of Λ.*

Proof. If $\mathbf{B}$ is a basis of Λ, since $\mathcal{S}$ is initialized with the vectors of $\mathbf{B}$, it is clear that $\Lambda \subset \mathcal{L}(\mathcal{S})$. Furthermore, for all vectors $\mathbf{w} \in \mathcal{L}(\mathcal{S})$, we claim that $\mathbf{w} \in \Lambda$.

Let us assume that $\mathcal{S}$ consists of $m \geq d$ vectors $\mathbf{v}_1, \dots, \mathbf{v}_m$. Having $\mathbf{w} \in \mathcal{L}(\mathcal{S})$ means that $\mathbf{w}$ is an integer linear combination of the $\mathbf{v}_j$ vectors. Now, by construction all $\mathbf{v}_j$ are integer linear combinations of the vectors from $\mathbf{B}$. It follows that $\mathbf{w}$ is an integer linear combination of vectors from $\mathbf{B}$. Hence we also have $\mathcal{L}(\mathcal{S}) \subset \Lambda$. □

LEMMA 3.2. (CORRECTNESS) *The L4 Algorithm terminates. Furthermore, given as input an LLL-reduced basis $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_d]$ it outputs an LLL-reduced basis $\mathbf{B}' = [\mathbf{b}'_1, \dots, \mathbf{b}'_d]$, such that $\|\mathbf{b}'_1\| \leq \|\mathbf{b}_1\|$.*

Proof. We call $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_d]$ the input basis, and denote by Λ the lattice generated by $\mathbf{B}$. We also denote by $\mathbf{B}^{(i)} = [\mathbf{b}_1^{(i)}, \dots, \mathbf{b}_d^{(i)}]$ the basis computed during step line 5 of the i-th iteration of Algorithm 2. We assume that $\mathbf{b}_1$ is the shortest vector of $\mathbf{B}$.

From Lemma 3.1, we can show that, if $\mathbf{B}^{(i)}$ is a basis of Λ, then the set $\mathcal{S}^{(i)}$ generated during the SAMPLE procedure is a generating set of Λ. Thus, after processing $\mathcal{S}^{(i)}$ with LLL, we obtain a basis $\mathbf{B}^{(i)}$ of $\mathcal{L}(\mathcal{S}) = \Lambda$. Furthermore, this basis is LLL-reduced. From here, it is clear that if the L4 Algorithm terminates, its output would be an LLL-reduced basis of Λ.

We now show that there is an i_f, such that the algorithm terminates at the end of i_f-th iteration, and that the norm of the first vector $\mathbf{b}_1^{(i_f)}$ of $\mathbf{B}^{(i_f)}$ is smaller than the norm of the first vector $\mathbf{b}_1$ of the input basis. Let us denote by $(N_i)_i$ the sequence such that $N_0 = \|\mathbf{b}_1\|^2$, and for all $i > 0$, $N_i = \left\|\mathbf{b}_1^{(i)}\right\|^2$. We claim that $(N_i)_i$ is a decreasing sequence of integers. Indeed, if we consider the set $\mathcal{S}$ computed during i-th iteration of Algorithm 2, we have

$$N_i \leq \min\left(\|\mathbf{w}\|^2 : \mathbf{w} \in \mathcal{S}\right). \tag{3.6}$$

Indeed, during the LLL-reduction process, $\mathbf{b}_1^{(i)}$ is first initialised at $\mathbf{w}_1$, the first vector of $\mathcal{S}$. Since $\mathcal{S}$ has first been sorted by increasing norm,

$$\|\mathbf{w}_1\|^2 = \min\left(\|\mathbf{w}\|^2 : \mathbf{w} \in \mathcal{S}\right).$$

From here, there are two possible cases. In the first scenario, $\mathbf{b}_1^{(i)}$ is still equal to $\mathbf{w}_1$ at the end of the LLL-reduction step, then Equation 3.6 holds and is an equality. The second possibility is that $\mathbf{b}_1^{(i)}$ has been swapped with $\mathbf{b}_2^{(i)}$ at some point during the LLL-reduction. This mean that Lovász condition has been violated and, at this point,

$$\delta \left\|\mathbf{b}_1^{(i)*}\right\|^2 > \left\|\mathbf{b}_2^{(i)*} + \mu_{2,1}\mathbf{b}_1^{(i)*}\right\|^2$$
$$\delta \left\|\mathbf{b}_1^{(i)}\right\|^2 > \left\|\mathbf{b}_2^{(i)}\right\|^2.$$

With $\delta = 3/4$, we get that $\mathbf{b}_1^{(i)}$ and $\mathbf{b}_2^{(i)}$ are swapped during the LLL-reduction only if $(3/4)\left\|\mathbf{b}_1^{(i)}\right\|^2 > \left\|\mathbf{b}_2^{(i)}\right\|^2$, and thus $\left\|\mathbf{b}_1^{(i)}\right\| > \left\|\mathbf{b}_2^{(i)}\right\|$. In this case, after the LLL-reduction has terminated, we have $\left\|\mathbf{b}_1^{(i)}\right\| < \|\mathbf{w}_1\|$.

We have then proved that $\left\|\mathbf{b}_1^{(i)}\right\| \leq \mathcal{N}_{\min}(\mathcal{S})$. Then, since $\mathbf{b}_1^{(i-1)}$ belongs to $\mathcal{S}$ by construction, $N_i \leq \left\|\mathbf{b}_1^{(i-1)}\right\|^2$ thus $N_i \leq N_{i-1}$.

Since Λ is a integral lattice, $(N_i)_i$ is a decreasing sequence of positive integers, and thus there is an index i_f such that:

$$\begin{cases} N_i < N_{i-1}, & \forall\ 1 \leq i < i_0 \\ N_{i_f} = N_{i_f - 1} \end{cases}$$

It follows that the algorithm terminate after iteration i_f, and the norm $\sqrt{N_{i_f}}$ of the first vector of the output basis is smaller or equal than $\sqrt{N_0}$, the norm of the first vector in the input basis. ∎

We move on to discuss the complexity. We first show the following.

Lemma 3.3. *Given as input an LLL-reduced basis* $\mathbf{B}$ *of an integral lattice* Λ*, the* Sample *procedure outputs a set* $\mathcal{S}$ *of* $\mathcal{O}(d^2)$ *vectors in time* $\mathcal{O}(d^2 n \log^2 M)$ *and memory* $\mathcal{O}(d^2 n \log M)$*, where* M *is an upper bound on* $\mathcal{N}_{\max}(\mathbf{B})$.

Proof. Most of the time spent in this algorithm is in the double for-loop, which takes a total of d^2 iterations. This corresponds to the amount of vectors $\mathbf{w} \pm \mathbf{v}$ computed during this step, which is an upper bound on the number of vectors appended to $\mathcal{S}$. As such, in the end, $\mathcal{S}$ may contain at most $d + d^2$ vectors whose coefficients can each be stored in less than $\log M$ bits, thus $\#\mathcal{S} = \mathcal{O}(d^2)$.

The operations performed inside the loops are memory accesses, and differences of vectors consisting of n integers, as well as norm computations. All of these can be done in time $\mathcal{O}(n \log^2 M)$ with naive multiplication over $\log M$-bit integers. We also need to test whether a vector is already in $\mathcal{S}$, using adapted data structures such as hash tables, this step can be done in $\mathcal{O}(n \log M)$ bit-operations. Hence the time spent in the computation of $\mathcal{S}$ is $\mathcal{O}(d^2 n \log^2 M)$. We then need to sort $\mathcal{S}$ according to the norm. Assuming that, for any vector of $\mathcal{S}$, we also store its norm, finding the right order can be done in $\mathcal{O}(d^2 \log d)$ comparisons between $\log M$-bit integers, plus the time required to copy the vectors in the right place which is $\mathcal{O}(d^2 n \log M)$. All of these lead to a total complexity of $\mathcal{O}(d^2 n \log^2 M)$. Finally, the memory complexity is given by the amount of space needed to store $\mathcal{S}$ which, as mentioned above, consists of $\mathcal{O}(d^2)$ vectors of n integer coefficients. Thus the memory complexity is $\mathcal{O}(d^2 n \log M)$. ∎

The following result is a direct consequence.

Lemma 3.4. *The time complexity of an iteration of the L4 algorithm with input basis* $\mathbf{B} \in \mathbb{Z}^{n \times d}$ *is dominated by the time of the LLL-reduction, which is*

$$T_{set} = \mathcal{O}\left(nd^4(d + \log M)(\log M + d^2)\right) \tag{3.7}$$

where M *is an upper bound on* $\mathcal{N}_{\max}(\mathbf{B})$.

Proof. There are roughly three steps in one iteration of the L4 algorithm. First, compute the set $\mathcal{S}$. Second compute an LLL-reduced basis of $\mathcal{L}(\mathcal{S})$. Third update the norm of the shortest vector. It is clear that the last step is dominated by the other two. From Lemma 3.3, the time complexity of the sample procedure is $\mathcal{O}(d^2 n \log M)$. Then, we estimate the time complexity of the LLL-reduction with input $\mathcal{S}$. Recalling that $\mathcal{S}$ consists of $\mathcal{O}(d^2)$ vectors and, by construction, all of them have norm smaller than $\mathcal{N}_{\max}(\mathbf{B})$, from Lemma 2.4, we have

$$T_{set} = \mathcal{O}\left(nd^2(d + \log M)(d^2 \log M + d^4)\right).$$

After simplification, we get the expression given in Equation 3.7. Since

$$d^2 n \log M = \mathcal{O}\left(nd^4(d + \log M)(\log M + d^2)\right),$$

any iteration of L4 is dominated by the LLL step. ∎

The proof of Theorem 3.1 follows from these previous results.

3.3 Experimental estimation of k.

All our tests were made on lattices from the Darmstadt SVP Challenge generator, which are all full rank lattices. As such our empirical assumptions on the number of iterations in our procedure are made on full rank lattices. We came up with the following conjecture.

Conjecture 1. *Let* Λ *be a* n *dimensional full-rank lattice. Our algorithm terminates after* k *call to an LLL-reduction algorithm with* $k = \mathcal{O}(\log(n))$.

We tested our algorithm on 1000 lattices of dimensions 40 to 200, with an increment of 10. On average, L4 calls the LLL algorithm between 2 and 5 times, with a maximum of 14 calls. There is only a slight increase with the dimension as shown in Figure 1. We therefore make the empiric assumption that the number of times the LLL Algorithm is called is $\mathcal{O}(\log(n))$ where n is the lattice's dimension.

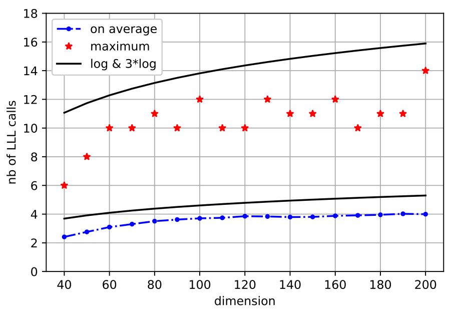

Figure 1: Number of LLL calls

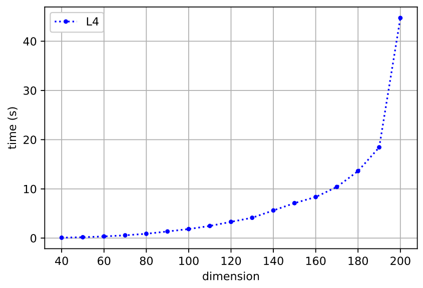

Figure 2: Average runtime of L4

3.4 Experimental runtime. We also tested the runtime of L4 on Computer 1. It is almost instantaneous up to dimension 70, and then slightly increases with the dimension up to 18.44 seconds in dimension 190. Then as shown on Figure 2, there is a significant loss of performance between dimension 190 and 200, the runtime being slightly above 44 seconds in dimension 200. We do not know for certain what causes this sudden change of behaviour. One likely possibility is that the amount of memory handled in this dimension becomes too important and the memory access requires more time. This theory is seconded by the fact that we already had to increase the amount of RAM from 32 GB to 128 GB from dimension 180.

Another difficulty we encountered which might be linked with this sudden drop in performance is that the numerical instabilities in Pohst variant of LLL seem more difficult to handle. Indeed, starting from dimension 190, some of our tests failed due to divisions by zero. In dimension 190, this rarely happens, only 4 times out of 1000 tests, but in dimension 200 this phenomena occurs in about 3.5% of our tests. This may be because the number of vectors in $\mathcal{S}$ that are to be discarded during this phase becomes too high.

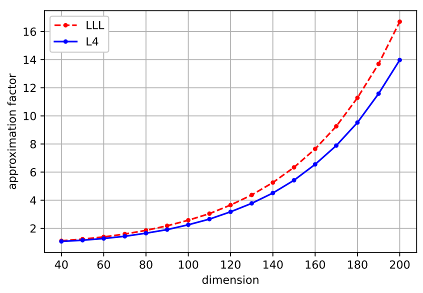

Figure 3: Average approximation factor

4 Norm of the First Vector

We present an empirical estimation of the norm of the first basis vector. All the lattices considered in this section and the next one are full-rank and random in the sense of Goldstein and Mayer [13]. For a given input basis $\mathbf{B}$ of a full-rank lattice Λ we denote by $\mathbf{B}^{L^3} = \left[\mathbf{b}_1^{L^3}, \ldots, \mathbf{b}_n^{L^3}\right]$, and $\mathbf{B}^{L^4} = \left[\mathbf{b}_1^{L^4}, \ldots, \mathbf{b}_n^{L^4}\right]$ the respective outputs of LLL, and L4. We also recall that an estimation of the norm of the shortest vector of Λ denoted by $\mathrm{GH}(\Lambda)$ is given in equation 2.2. To estimate how close we are to finding a solution to the shortest vector problem, we look at the ratio: norm of the first vector of the output basis divided by $\mathrm{GH}(\Lambda)$. We call this value the approximation factor and denote it by $\gamma_{\textsc{Algorithm}}$, or simply γ in the general case. If $\gamma \approx 1$, then $\mathbf{b}_1$ is close to the norm of the shortest vector. Similar to the previous section, we ran tests in dimensions 40, 50, etc. up to 200. For each of these dimensions we tested 1000 random lattices, generated via the Darmstadt SVP Challenge lattice generator [1]. The experiments in this section were all run on Computer 1.

4.1 Experimental results. As argued in Section 3, given as input the same basis $\mathbf{B}$, the outputs of LLL and L4 satisfy $\|\mathbf{b}_1^{L^4}\| \leq \|\mathbf{b}_1^{L^3}\|$. However, we did not discuss how shorter $\mathbf{b}_1^{L^4}$ is. We provide an average comparison between approximation factors γ_{LLL} and γ_{L4}. In dimension 40, γ_{L4} is only 4% smaller than γ_{LLL}, however, as shown in figure 3, this gap improves when the dimension grows. For instance, in dimension 200, γ_{L4} is about 16% shorter than γ_{LLL}, with γ_{LLL} being about 16.7, while γ_{L4} is a bit less than 14.

4.2 Randomizing the algorithm. Although we improved the approximation factor compared to LLL, we figured it remains quite large and wondered if we could obtain better results by re-randomizing the input. This kind of idea is not new, and is used for instance to improve the accuracy of enumeration with pruning and extreme pruning [7, 11].

Given a basis $\mathbf{B}$, our randomization process is heavily inspired by the one used as a routine in the BKZ implementation of FPLLL. An unimodular matrix $\mathbf{U}$ is generated as an upper triangular matrix whose coefficients are 0, 1 or -1 with 1 or -1 on the diagonal. Then, the rows $\mathbf{U}$ are randomly permuted, and a new basis $\mathbf{B}'$ is computed as $\mathbf{BU}$.

Experimentally, we noticed that the number of non-zero coefficients in $\mathbf{U}$ significantly affects the quality of the new basis. If $\mathbf{U}$ is too dense, running L4 after one randomization will likely not lead to better results. If it is too sparse, then it is like performing no randomisation at all. In practice, we noticed that L4 is likely to output a basis of lesser quality, if the density of $\mathbf{U}$ is more than 2%. One may argue that no improvement after a single re-randomization does not mean that no improvements could occur in later re-randomizations. However, we need to keep in mind that our goal is to produce a fast sequential algorithm, so repeating the re-randomization process too much will quickly degrade our runtime.

In our experiments, the density of $\mathbf{U}$ is fixed at 1.3%. We first ran 1000 tests per dimension with a constant number of re-randomization (10, 20 and 50). We call those variants L4-Rand10, -20, -50 respectively. Although we noticed a sensible improvement of the approximation factor between L4 and L4-Rand10, it becomes less significant when the number of randomisation increases. There is barely no improvement at all between L4-Rand20 and L4-Rand50. For instance, in dimension 90, $\gamma_{\text{L4-Rand10}}$ is about 16% smaller than γ_{L4}. In comparison the gain between $\gamma_{\text{L4-Rand50}}$ and $\gamma_{\text{L4-Rand10}}$ is less than 5%. Furthermore, the gap between the approximation factor does not improve when the dimension grows and even seems to get thinner. In dimension 200, $\gamma_{\text{L4-Rand50}}$ is only about 3.4% smaller than $\gamma_{\text{L4-Rand10}}$, while the runtime is increased by a factor of 5, so the gain seems rather small for the cost. As such we deem L4-Rand20 and L4-Rand50 too slow to be really interesting. Instead, we propose to abort the randomization process early, when no improvement is made after a while.

More precisely, if the squared norm of the shortest vector does not decrease after a fixed number k of re-randomization, we end the process. We call this variant L4-Max$k$. We ran tests with k equal to 2 and 4. Interestingly, although L4-Max4 computes about twice more re-randomisations than L4-Max2, it only slightly improves the approximation factor. For instance, in dimension 200, $\gamma_{\text{L4-Max4}}$ is on average 12.27 which is not much better than the 12.66 obtained after running L4-Max2

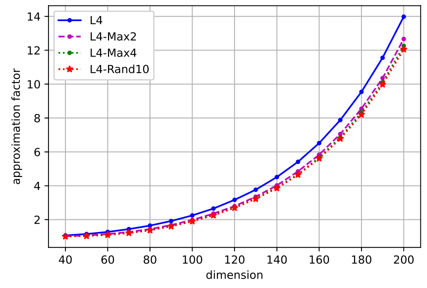

Figure 4: Average approximation factor

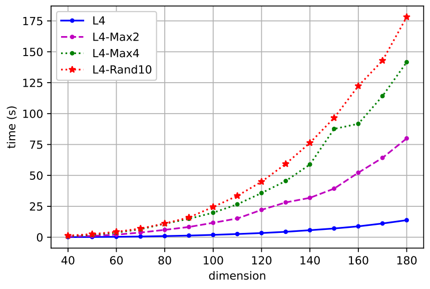

Figure 5: Average runtime

Figure 4 provides the average approximation factor computed for each tested dimension. We can see that, although L4-Max2 and L4-Max4 lead to a better approximation factor than L4, both provide similar results, which is again very close to what is obtained by L4-Rand10. Since L4-Max4 makes about twice more re-randomisation steps, it is significantly slower than L4-Max2, yet as shown on Figure 5, L4-Rand10 is slower than both. Similar to the observation made in section 3.4, we observe a significant drop in the algorithm performance in dimension 200 for all methods. We did not show this dimension on Figure 5 for more clarity.

4.3 In small dimension. According to the Darmstadt Lattice challenge, 1.05GH(Λ) is a good upper bound of the norm of the shortest vector. Any vector whose norm is less than this quantity is considered to be a solution to their SVP challenge. In small dimension, we are able to find a good proportion of such vectors. For instance, in dimension 40, out of 1000 lattices tested, the LLL algorithm finds 161 vectors of norm smaller than 1.05GH(Λ), while L4 finds 355 of them. This number increases to 842 with L4-Max4 and even up to 915 with L4-Rand10. Of course this proportion decreases with the dimension, yet we still get some success up to dimension 70. For instance, in dimension 60, while LLL does not find any such short vector anymore, our L4 still finds 4 of them. With the L4-Max4 algorithm, we find 103 of them, which is a 10% success rate. Yet this significantly drops for higher dimension. In dimension 70, L4-Max4 still recovers 2 of such vectors and L4-Rand10 4, but none are found in dimension 80 and above.

We do not claim this is a feat, as for small dimensions, there are many algorithms which solve SVP very efficiently (i.e., BKZ, enumeration, sieving). However, we thought this result was still worth mentioning.

4.4 Short-comings of L4. We also tried to compare L4 with BKZ with blocksize β, but the results were rather disappointing. For instance, BKZ-12 offers much smaller approximation factors, while being significantly faster. Indeed, in dimension 140, on average out of 1000 tests, $\gamma_{BKZ12} = 2.38$ for a runtime of about 5 seconds, while L4-Max2 is 5 to 6 time slower and leads to an approximation factor $\gamma_{\text{L4-Max2}} = 3.9$. Meanwhile, L4 without randomisation offers a similar runtime than BKZ12 in dimension 140, but the approximation factor is about 1.89 bigger than γ_{BKZ12}, and the gap grows with the dimension. While we could improve the runtime with a better implementation, reducing the approximation factor seems less straightforward.

Modifying L4 to use a BKZ-β algorithm instead of LLL for the lattice reduction part is not likely to work. This is due to the fact that, depending on the value of β, BKZ-β reduced bases are "more orthogonal" than LLL-reduced ones. We notice that the number of pairs $(\mathbf{b}_i, \mathbf{b}_j)$ of basis vectors that are not L-reduced is lesser. This implies that the set $\mathcal{S}$ constructed during each iteration of L4 will likely consists of too few vectors for the algorithm to correctly work. For instance, in dimension 70, there are about 58 non L-reduced pairs after a BKZ-12 reduction, compared to 64 after an LLL reduction. And this gap increases with the dimension. In dimension 180, there are only an average of 154 non L-reduced pair in a BKZ-12 reduced basis, while there are more than 385 in an LLL-reduced one.

5 L4 as a Pre-processing Step for BKZ

Since using L4 as a Post-processing for BKZ is unlikely to work, we choose to use it as a pre-processing instead. At the beginning of the BKZ reduction algorithm, the input basis is first reduced with LLL, in order to facilitate the later enumeration steps performed in each block. We decided to use L4 in this pre-processing step instead of LLL and compare the results provided by both algorithms. We call our variant L4+BKZ. Note that BKZ uses randomizations. To make our comparison fairer, we fixed its seed so that, given an input lattice, the same randomizations will be applied to both BKZ and L4+BKZ. For our experiments, we choose to focus primary on BKZ with blocksize 24, as it is known to provide an interesting tradeoff between runtime and output quality. Indeed, as argued in [10], for blocksize 25 and higher, the runtime of BKZ significantly increases.

Due to technical reasons, we were not able to run our tests on Computer 1, and had to use the less performant Computer 2 instead. It also means that all of our experiments had to be processed sequentially. As such, since BKZ-24 becomes quite slow when the dimension grows, we were not able to make as many experiments as in previous sections. We only ran 100 tests per dimensions, from 40 to 140 with an increment of 10. Although this may be a bit too small to draw strict conclusions, it already helps us figure out the behaviour of L4+BKZ. The results we present in this section are those we obtained with BKZ-24. We also made about 100 experiments per dimension with BKZ-25 up to 90, and the results we obtained were similar (c.f., Appendix A). For each test, we compared the approximation factor γ of the outputs of the usual BKZ-24 with LLL as pre-processing, and L4+BKZ-24. For the record, we also compared the runtime of both algorithms.

5.1 Our experimental results. On average, using L4 as a pre-processing step for BKZ slightly reduces approximation factor. From dimension 60 to 140, $\gamma_{\text{L4+BKZ-24}}$ is about 3% smaller than $\gamma_{\text{BKZ-24}}$, as shown on Figure 6. From dimension 70, and upwards, we also note that the number of lattices for which using L4+BKZ-24 instead of BKZ-24 leads to a smaller γ appears to stabilize between 70 and 80 out of a hundred input lattices. We also noticed a pic in dimension 90, where $\gamma_{\text{L4+BKZ-24}} < \gamma_{\text{BKZ-24}}$ in 87 out of 100 tests. On average the gain in γ was about 4.3%. However, since we only have a few number of tests, this may be due to luck. In dimension 60 and below, the norm of the shortest vector in a BKZ-24 reduced basis mostly remains below $1.05\text{GH}(\Lambda)$, so the comparison is less relevant.

Although L4 is significantly slower than LLL, we noticed that L4+BKZ-24 is not much slower than BKZ-24, as shown of figure 7 which compares runtime of both methods. The performances of L4+BKZ-24 do not seem to worsen with the dimension either. In dimension 140, L4+BKZ-24 was only 5.9% slower, on average. In every dimension, we even get between 5 and 15 lattices out of 100 for which L4+BKZ-24 is faster than BKZ-24. Furthermore, we have at least one instance for which L4+BKZ-24 improves both the approximation factor and the runtime. On average there are about 4.5 of them

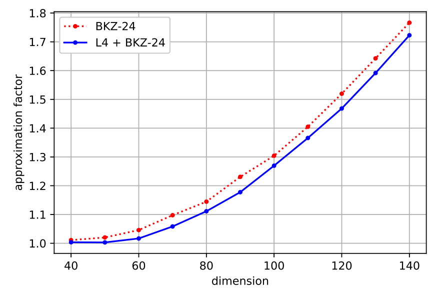

Figure 6: Average approximation factor

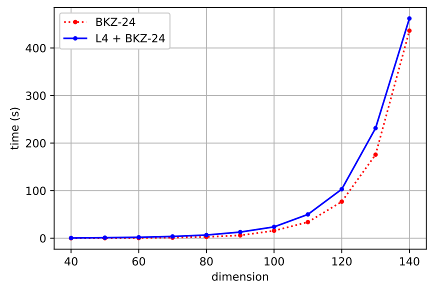

Figure 7: Average runtime

per dimension, and a maximum of 10 in dimension 140. We also compared the runtime of BKZ-24 alone after being given as input an LLL-reduced basis, $\mathbf{B}^{L^3}$, and a L4-reduced one, $\mathbf{B}^{L^4}$, of the same lattice Λ. We noted that from dimension 70 and upwards, BKZ-24 is faster in 35 to 45 % of our tests when taking $\mathbf{B}^{L^4}$ as input, with the exception of dimension 90, where it is only 24%. On the downside, the average runtime of BKZ-24 after L4 is not better than BKZ-24 after an LLL-reduction, and there are even instances where it is slower.

5.2 Other variants. We also ran tests with L4+BKZ-24, where the INFLATE routine quickly described in section 3 is used instead of SAMPLE. As a reminder, at each call, this routine constructs $\mathcal{S}$ as in Equation 3.5. We call INFLATE-L4 the variant of L4 which uses this procedure instead of SAMPLE. Oddly enough, even though L4 and INFLATE-L4 provide similar results in terms of runtime and approximation factors, their behaviour while used as a pre-processing step for BKZ is rather different.

In fact, out of the 100 experiments we ran per dimension, INFLATE-L4+BKZ-24 leads to similar results than BKZ-24. More precisely, although there were some cases where we obtain a lower approximation factor (e.g., for $n = 90$, $\gamma_{\text{INFLATE-L4+BKZ-24}} < \gamma_{\text{BKZ-24}}$ in 50 tests out of 100), the improvement is rather small, both algorithms leading to the same average results. Similarly, while there are a few cases where L4+BKZ-24 with INFLATE is faster than BKZ-24, the average total runtime of both algorithms is the same.

This difference in behaviour between the two variants of L4 can be explained by the following. Although the first vector of both output bases have similar norm, the bases themselves have a different structure. The SAMPLE procedure seems to produce bases that have stronger orthogonality properties. Indeed we notice that the proportion of L-reduced pairs in a basis $\mathbf{B}$ obtained after L4 is higher than after INFLATE-L4. For instance, in dimension 100, out of 1000 tests, the average number of non L-reduced pairs in the output of L4 is 102, while it is 127 for both INFLATE-L4 and LLL. In dimension 200, this gap widens with only 282 pairs that are not L-reduced for L4, while it is about 359 for INFLATE-L4 and 465 for LLL. This stronger orthogonality may explain why BKZ-24 is able to produce shorter vectors when given as input an L4-reduced basis.

5.3 Discussion. Although the gain of using L4 as a pre-processing step instead of LLL may seem scarce, we would like to point out that our runtime comparison was not fair. Indeed, we compared our un-optimised Python implementation of L4, to a code which makes use of the highly optimised C++ FPLLL Library. So there is a good hope that a better implementation would lead to even more interesting results.

More precisely, we think we can improve the runtime for those 35 to 45% of lattices for which BKZ-24 is faster after taking as input an L4-reduced basis. First we could use adequate data structures like hash-tables or stacks to efficiently check if a vector belongs to $\mathcal{S}$. Right now our code uses a very naive membership testing which requires to check all elements in $\mathcal{S}$. This makes the runtime of our SAMPLE implementation cubic in the dimension of the lattice, while it could be only quadratic. Other optimisations can be made, like avoiding in-code conversions from Python array to FPyLLL matrices. Finally, we could use a more performant programming language, like C++.

6 Conclusion

We present a polynomial time lattice reduction algorithm, L4 which improves on the quality of LLL while remaining fairly efficient. When used as a pre-processing step for BKZ-24, it allows us to slightly reduce the norm of the first basis vector. Although the improvement is only about 3% on average, we argue that it is still significant, considering that the vectors returned by BKZ-24 were already quite short in the dimension we tested. On the downside, our hybrid L4+BKZ appears to be slower than BKZ. However, we have reasons to believe that its runtime can be improved with a more optimised implementation. In particular, in our tests, there are about 40% of the lattices for which BKZ is faster when it is given as input an L4-reduce basis. So we hope we can bridge the gap in performance, and even get faster than BKZ for these lattices, offering an alternative to LLL for BKZ pre-processing.

Future work and food for thought. As mentioned earlier, we do not know exactly how to tune the parameters of our L4 to offer the best tradeoff between efficiency and output quality. In particular, the choice of parameters α_1 and α_2 which determine the number of iterations in our Sample procedure is crucial. The choice we made here was rather a naive one, and is probably not optimal. Although our experiment suggests that increasing their value does not seem to improve the overall quality of the output, we need to investigate further to be certain. Indeed, as we noticed when comparing L4 and Inflate-L4, even though two algorithms seem to produce similar results when we consider only the norm of the first vector, their behaviour can widely differ when combined with BKZ.

The question of up to which point the number of iterations inside the Sample procedure can be reduced, also remains open. We ran some preliminary tests for Sample when only $0.25d$ vectors are drawn in the second loop, and the overall approximation factors look similar up to dimension 200. However, we have not yet tested if the results remain interesting when combined with BKZ, or if they degrade similar to Inflate-L4.

Finally, we did not investigate either how L4 behaves when combined with BKZ-β for other values of 24. We did a few experiments for $\beta = 25$ but since this takes a lot of time when processed sequentially, we did not go far in dimension. It could also be interesting to look at smaller blocksizes, for which BKZ is already quite fast (i.e., $\beta \leq 20$), and see if we can improve the approximation factor even further, without getting much slower. On the other hand, it could also be interesting to see if taking as input an L4-reduced basis in BKZ-β for higher values of β can help reduce the runtime of the enumeration routine. If so, it could be worth considering using L4 as a routine inside each block before performing the enumeration for large β.

References

[1] Darmstadt, svp challenge. Available at `https://www.latticechallenge.org/svp-challenge`, 2010.

[2] Dorit Aharonov and Oded Regev. Lattice problems in np $\cap$ conp. *J. ACM*, 52:749–765, 2005.

[3] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 99–108, New York, NY, USA, 1996. ACM.

[4] Miklós Ajtai. Random lattices and a conjectured 0-1 law about their polynomial time computable properties. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 733–742. IEEE, 2002.

[5] Johannes Buchmann, Richard Lindner, Markus Rückert, and Michael Schneider. Explicit hard instances of the shortest vector problem. *IACR Cryptol. ePrint Arch.*, page 333, 2008.

[6] Hao Chen. A measure version of gaussian heuristic. *IACR Cryptol. ePrint Arch.*, 2016:439, 2016.

[7] Yuanmi Chen and Phong Nguyen. Bkz 2.0: Better lattice security estimates. In *Advances in Cryptology – ASIACRYPT 2011*, pages 1–20, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[8] Joseph Louis de Lagrange. Recherches d'arithmétique. In *Nouveaux mémoires de l'Académie royale des sciences et belles-lettres de Berlin*, 1773.

[9] The FPLLL development team. fplll, a lattice reduction library, Version: 5.4.5. Available at `https://github.com/fplll/fplll`, 2023.

[10] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.

[11] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2010.

[12] Oded Goldreich and Shafi Goldwasser. On the limits of nonapproximability of lattice problems. *J. Comput. Syst. Sci.*, 60:540–563, 2000.

[13] Daniel Goldstein and Andrew Mayer. On the equidistribution of hecke points. 15(2):165–189, 2003.

[14] Charles Hermite. Extraits de lettres de m. ch. hermite à m. jacobi sur différents objects de la théorie des nombres. *Journal für die reine und angewandte Mathematik*, 40:261–315, 1850.

[15] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, 1983.

[16] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

[17] Korkine and Zolotarev. Sur les formes quadratiques positives quaternaires. *Mathematische Annalen*, 5:581–583, 1872.

[18] Korkine and Zolotarev. Sur les formes quadratiques. *Mathematische Annalen*, 6:366–389, 1873.

[19] Arjen Lenstra, Hendrik Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[20] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, 2001.

[21] Hermann Minkowski. Ueber geometrie der zahlen. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 1:64–65, 1890.

[22] Phong Q Nguyen. Hermite's constant and lattice algorithms. In *The LLL Algorithm: Survey and Applications*, pages 19–69. Springer, 2009.

[23] Phong Q. Nguyen and Damien Stehlé. Floating-point lll revisited. In *International Conference on the Theory and Application of Cryptographic Techniques*, 2005.

[24] Phong Q Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Transactions on algorithms (TALG)*, 5(4):1–48, 2009.

[25] Michael Pohst. A modification of the LLL reduction algorithm. *J. Symb. Comput.*, 4(1):123–127, 1987.

[26] Claus-Peter Schnorr. Block reduced lattice bases and successive minima. *Combinatorics, Probability and Computing*, 3:507–522, 1994.

[27] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

[28] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2):201–224, 1987.

[29] Igor Semaev. A 3-dimensional lattice reduction algorithm. In *Cryptography and Lattices: International Conference, CaLC 2001 Providence, RI, USA, March 29–30, 2001 Revised Papers*, pages 181–193. Springer, 2001.

[30] Damien Stehlé. Floating-point lll: theoretical and practical aspects. In *The LLL Algorithm: survey and applications*, pages 179–213. Springer, 2009.

[31] Peter van Emde Boas. Another np-complete problem and the complexity of computing short vectors in a lattice. *Tecnical Report, Department of Mathmatics, University of Amsterdam*, 1981.

A Experimental results for L4+BKZ-25

Here are some additional results obtained with L4+BKZ-25 up to dimension 90. We made 99 to 100 tests per dimension and the outcome is similar to what we obtained with L4+BKZ-24 for the same dimensions. On average $\gamma_{\text{L4+BKZ-25}}$ also seem to be about 3% smaller than $\gamma_{\text{BKZ-24}}$, with the best ratio obtained in dimension 90 with $\gamma_{\text{L4+BKZ-25}} \approx 1.15$ while γ_{BKZ25} is slightly above 1.2 as shown in Figure 8. The number of time L4+BKZ-25 improves either the approximation factor or the runtime or both, is also very similar to what we obtain with L4+BKZ-24 (c.f., Table 1). The same can be said about number of times BKZ-25 alone was faster after taking as input an L4-reduce basis: around 40, the worst case being 34 in dimension 80 and the best one 56 in dimension 40.

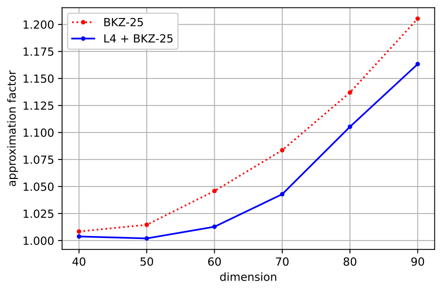

Figure 8: Average approximation factor

dim	# tests	smaller γ	faster	both
40	100	17	15	1
50	99	42	11	2
60	99	70	11	9
70	99	75	9	6
80	100	81	8	5
90	100	81	6	3

Table 1: Number of time L4+BKZ-25 improved on BKZ-25