Rigorous and Efficient Short Lattice Vectors Enumeration

Xavier Pujol[1] and Damien Stehlé[1,2] *

[1] LIP Arénaire, CNRS/INRIA/ENS Lyon/UCBL/Université de Lyon
[2] University of Macquarie/University of Sydney
xavier.pujol@ens-lyon.fr, damien.stehle@gmail.com

Abstract. The Kannan-Fincke-Pohst enumeration algorithm for the shortest and closest lattice vector problems is the keystone of all strong lattice reduction algorithms and their implementations. In the context of the fast developing lattice-based cryptography, the practical security estimates derive from floating-point implementations of these algorithms. However, these implementations behave very unexpectedly and make these security estimates debatable. Among others, numerical stability issues seem to occur and raise doubts on what is actually computed. We give here the first results on the numerical behavior of the floating-point enumeration algorithm. They provide a theoretical and practical framework for the use of floating-point numbers within strong reduction algorithms, which could lead to more sensible hardness estimates.

Keywords. Lattices, SVP, lattice cryptanalysis, numerical stability.

1 Introduction

A *lattice* L is a discrete subgroup of some $\mathbb{R}^n$. It can always represented by a *basis*, i.e., some $d \leq n$ linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in \mathbb{R}^n$ such that $L = \sum \mathbb{Z}\boldsymbol{b}_i$. A given lattice has infinitely many bases as soon as $d \geq 2$. One is most often interested in bases made of rather short/orthogonal vectors, which are generically called reduced. They provide a more tractable description of the lattice. Since a lattice is discrete, it contains a vector of smallest non-zero Euclidean length: this length λ is called the *lattice minimum*. The most famous problem related to lattices is the *Shortest Vector Problem* (SVP), which aims at finding a lattice vector of length λ from an arbitrary basis. SVP is known to be NP-hard under randomized reductions [2]. Another popular lattice problem is the *Closest Vector Problem* (CVP): given a lattice basis and a target vector in $\mathbb{R}^n$, find a lattice vector that is closest to the target. This non-homogeneous version of SVP is

* This work is part of the Australian Research Council Discovery Project on Lattices and their Theta Series.

NP-hard [7]. Since these problems are costly to solve for large dimensions, one is often satisfied with weaker variants. E.g., in γ-SVP one asks for a non-zero lattice vector no longer than $\gamma \cdot \lambda$.

Lattice reduction algorithms range between two extremes. On one side, the LLL algorithm [20] provides a basis with relatively poor properties, in polynomial time. On the opposite, the Hermite-Korkine-Zolotarev (HKZ) reduction provides an excellent basis but requires a huge computational effort. Schnorr [31] was the first to devise hierarchies of algorithms ranging from LLL to HKZ, depending on a parameter k. Schnorr's algorithms make use, in a LLL fashion, of HKZ reductions in projections of sublattices of dimension $O(k)$. When k increases, the cost increases as well, but the quality of the bases improves. The recent hierarchies [9, 10] achieve better trade-offs but follow the same general strategy. In practice, the Schnorr-Euchner BKZ algorithm [32] seems to be the best, at least for small values of k. The HKZ reduction uses the Kannan-Fincke-Pohst (KFP) enumeration of short lattice vectors [19, 8]. KFP may be replaced by the probabilistic algorithm of [4], but the latter seems slower in practice [28].

Lattices appeared for the first time in cryptology at the beginning of the 80's, when the renowned LLL algorithm [20] was used to break knapsack cryptosystems [29]. For many years lattices were mostly used as a cryptanalytic tool [18]. The landscape changed dramatically in the mid-90's with the invention of several lattice-based encryption schemes, among which Ajtai-Dwork [3], NTRU [17] and GGH [13]. Their securities provably/heuristically rely on the hardness of relaxed variants of SVP and CVP. For example, in the GGH/NTRU framework, the hardness of recovering the secret key from the public key is related to SVP and the hardness of maliciously deciphering a message is related to CVP. A recent but very active and promising trend consists in building other cryptographic schemes whose securities provably reduce to the assumed worst-case hardness of Poly(d)-SVP for special lattices (called ideal). This includes hashing [23], signatures [22] and public-key identification [21]. Gentry, Peikert and Vaikuntanathan [12] introduced other elaborate schemes, including a signature and an identity-based cryptosystem. We refer to [24] for more details. Besides cryptology, lattice reduction and in particular KFP is used in many areas, including number theory [6] and communications theory [25, 15], in which the present results may prove useful as well.

Despite the high-speed development of lattice-based cryptography, its practical security remains to be assessed (see [11] for a first step in that direction). Contrary to factorization and discrete logarithm in finite fields and in elliptic curves, the practical limits for solving SVP and CVP and

their relaxed variants are essentially unknown, implying that the practicality of the schemes above is debatable. It could be that the suggested key sizes are below what they should, as what happened to be the case with GGH [26]. They may also be too large and then unnecessarily sacrifice efficiency. No significant computational project has ever been undertaken. The main reason is that the algorithmic facet of lattice reduction remains mysterious. In particular, the theoretically best algorithms [9, 10] seem to remain slower than heuristic ones such as [32], whose practical behaviors are themselves suspicious. Let us discuss NTL's BKZ routine [33] which implements [32] and is the only publicly available such implementation: when the so-called block-size k is around 30, the number of internal calls to SVP in dimension k seems to explode suddenly (although the corresponding quantity decreases with k in the theoretical algorithms); when k increases, BKZ seems to require more precision for the underlying floating-point computations, although the considered bases should become more orthogonal, which implies a better conditioning with respect to numerical computations. The latter raises doubts on what is actually computed and thus on the practical security estimates of lattice-based cryptography.

Classically, to obtain correctness guarantees, the lattices under study should be in $\mathbb{Q}^n$ and the KFP enumeration should rely on a rational arithmetic. However, the rationals may have huge bit-sizes (though polynomial in the bit-size of the input basis). The bit-size of the rationals is a polynomial factor of the overall enumeration cost (between linear and quadratic, depending on the integer arithmetic). Keeping a rational arithmetic would decrease the efficiency of KFP significantly. In practice, e.g., in NTL, these rational numbers are always replaced by small precision floating-point numbers. Finding a small lattice vector corresponds to disclosing an integer linear combination of vectors whose coordinates are small, i.e., for which any coordinate is a cancellation of integer multiples of initial coordinates. However, floating-point computations are notoriously inadequate when cancellations occur since it often implies huge losses of precision (and thus a possibly dramatic growth of relative errors). Moreover, the precision is rather low (usually 53 bits), though the number of operations performed may be exponential with the dimension. If the operations reuse the variables sequentially, then one may run out of precision simply because of the accumulation of the errors. Finally, there is no efficient way to check the optimality of a solution but to re-run the whole algorithm in rational arithmetic: by comparing the length of the output vector with the lattice determinant, one can check that it looks reasonable, but it could be that (much) better solutions have been missed.

In the present paper, we give the first analysis of the influence of floating-point arithmetic within the KFP enumeration algorithm. More precisely, we show that if it is called on an LLL-reduced basis of a lattice made of integer vectors and uses floating-point arithmetic with a precision that is $\Omega(d)$ (the constant being explicit), then it finds the desired solution, i.e., a vector reaching the lattice minimum λ. Moreover, if the lattice is known only approximately (which may be the case for the projected sublattices in BKZ-style algorithms), then it finds a close to optimal solution. Finally, we also prove that the floating-point enumeration involves essentially the same number of arithmetic operations as the rational one. The results hold in a broad context: the technique can be adapted to fixed-point arithmetic, a weak condition is required for the input basis (if the input basis is not LLL-reduced, then the cost of the enumeration would grow dramatically), and the input may not be known exactly. Furthermore, the worst-case precision may be provably and adaptively decreased to a usually much smaller sufficient precision that can be computed efficiently from a given input basis. Double precision seems to suffice for KFP for all computationally tractable dimensions.

For the result to be valid, KFP has to be slightly modified (essentially, the initial upper bound has to be enlarged). The proof relies on a subtle analysis of the floating-point variant with respect to the rational enumeration: because of internal tests whose outcomes may differ due to inaccuracies, the execution of the floating-point variant may not mimic at all the ideal one. After working around that difficulty, the proof reduces to standard error analysis. To obtain a low sufficient precision, we heavily use the LLL-reducedness of the input basis.

Our result complements the Nguyen-Stehlé floating-point LLL [27]. By combining these two results, the use of floating-point arithmetic in all practical lattice algorithms may be made rigorous. Providing tight conditions leading to guarantees for the enumeration algorithm is likely to lead to significantly faster algorithms. Since the possible troubles coming from the use of floating-point arithmetic are better understood, one may work around them in the cheapest valid way rather than using unnecessarily large precisions. Like LLL [34], one may hope to design combinations of reduction algorithms whose arithmetic handling is oblivious to the user, that are guaranteed and as fast as possible. A good understanding of the underlying numerical stability issues provides a firm ground to study other questions. Furthermore, the knowledge of a small sufficient precision for the enumeration algorithm is an invaluable ingredient for hardware-based enumeration: in software, one should not use a precision cruder than the

processor double precision; in hardware, however, the smaller the precision the faster. Overall, the floating-point analysis of the enumeration algorithm is a step towards intense cryptanalytic computations.

ROAD-MAP. In Section 2, we give the necessary background on lattices and floating-point arithmetic. In Section 3, we precisely describe the algorithm under scope and describe our results. We give elements of the proofs in Section 4, the more technical details being postponed to the appendix of the full version. In Section 5, we discuss the practicality of our results.

NOTATIONS. If $x \in \mathbb{R}$, we denote by $\lfloor x \rceil$ its closest integer (if there are two possibilities, the even one is chosen). A variable $\bar{x}$ is supposed to approximate the corresponding x, and we define $\Delta x = |\bar{x} - x|$.

REMARKS. For simplicity, we will only consider SVP. The results can be extended to CVP. Many variables occur in the text. This is due to the combined technicalities of floating-point arithmetic and LLL. This also comes from the will to provide explicit bounds, which is necessary to actually derive rigorous implementations. Here is a heuristic glossary for a first reading: the LLL-parameters $\delta, \eta, \alpha, \rho$ are essentially $1, 1/2, \sqrt{4/3}, \sqrt{3}$; the variables $C_1, C_2, \ldots$ are $\tilde{O}(1)$; the variables ϵ and ϵ' quantify inaccuracies and are negligible, whereas K is close to 1.

2 Reminders on Lattices and Floating-Point Arithmetic

We give some quick reminders on floating-point arithmetic and lattices. For more details, we respectively refer to [16] and [6].

FLOATING-POINT ARITHMETIC. A precision t *floating-point number* is a triple $(s, e, m) \in \{0,1\} \times \mathbb{Z} \times \left(\mathbb{Z} \cap [2^{t-1}, 2^t - 1]\right)$. It represents the real $(-1)^s \cdot m \cdot 2^{e-t+1}$. The *unit in the last place* is $\epsilon = 2^{-t+1}$. If $a \in \mathbb{R}$, we denote by $\diamond(a)$ the floating-point number that is closest to a (the one with an even m if there are two solutions). We have $|a - \diamond(a)| \leq \epsilon/2 \cdot |a|$. If a and b are two floating-point numbers, we define $a \oplus b, a \ominus b$ and $a \otimes b$ by $\diamond(a+b), \diamond(a-b)$ and $\diamond(a \cdot b)$. The *double precision* $t = 53$ is a common choice as $\oplus, \ominus$ and $\otimes$ are implemented at the processor level in most computers. In practice, and for the KFP enumeration in particular, one should use double precision as much as possible. However, asymptotically with respect to the growing lattice dimension d, we will need $t = \Omega(d)$.

GRAM-SCHMIDT ORTHOGONALIZATION. Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ be linearly independent vectors. We define their Gram-Schmidt orthogonalization by $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j<i} \mu_{i,j} \boldsymbol{b}_j^*$ with $\mu_{i,j} = \frac{\langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle}{\|\boldsymbol{b}_j^*\|^2}$ for $i > j$. We define $r_i = \|\boldsymbol{b}_i^*\|^2$.

The $\mu_{i,j}$'s and r_i's. are the *Gram-Schmidt coefficients*. The $\boldsymbol{b}_i^*$'s are pairwise orthogonal. If the $\boldsymbol{b}_i$'s are integral, then the $\mu_{i,j}$'s are rational and can be computed in polynomial time with the formula above.

LLL-REDUCTION. Let $\eta \in [1/2, 1)$ and $\delta \in (\eta^2, 1)$. Consider a lattice basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ and its corresponding $\boldsymbol{b}_i^*$'s and $\mu_{i,j}$'s. The basis is said to be (δ, η)-*LLL-reduced* if for all $i > j$ we have $|\mu_{i,j}| \leq \eta$ and $\delta \|\boldsymbol{b}_{i-1}^*\|^2 \leq \|\boldsymbol{b}_i^* + \mu_{i,i-1}\boldsymbol{b}_{i-1}^*\|^2$. This directly implies that the lengths of the $\boldsymbol{b}_i^*$'s cannot decrease too fast: if $\alpha := (\delta - \eta^2)^{-1/2}$ then $\alpha^2 r_i \geq r_{i-1}$. In this paper, we will further assume that $\delta > \eta^2 + (1+\eta)^{-2}$. This assumption is reasonable, since before starting an enumeration one should always LLL-reduce the lattice with δ close to 1 and η close to $1/2$. Our analysis can be adapted to the general case, but this complicates the exposure for a useless situation. Lenstra, Lenstra and Lovász [20] gave an algorithm that computes an LLL-reduced basis from an arbitrary integral basis in time $O(d^5 n \log^3 B)$ where B is the maximum of the lengths of the input vectors. Using (low precision) floating-point arithmetic for the Gram-Schmidt computations, Nguyen and Stehlé [27] decreased that complexity to $O(d^4 n(d + \log B)\log B)$. Their algorithm requires $\eta > 1/2$. They rely on floating-point approximation to the Gram-Schmidt orthogonalization, which is much cheaper to obtain than computing the exact one. As an intermediate result, they show that if the input basis is LLL-reduced and if the computations are based on the exact Gram matrix (the matrix of the pairwise scalar products of the basis vectors), then this approximation is accurate even with low precision (linear with respect to the dimension).

Theorem 1 ([27]). *Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in \mathbb{Z}^n$ be a (δ, η)-LLL-reduced basis, with $\eta \in [1/2, 1)$ and $\delta \in (\eta^2, 1)$. Let $u \in (0, 1/16)$ and $\rho = (1+\eta+u)(\delta - \eta^2)^{-1/2}$. Let t be such that $C_1 \rho^{2d} \epsilon < u$ where $\epsilon = 2^{-t+1}$ and $C_1 = 32d^2$. Starting from the Gram matrix of the $\boldsymbol{b}_i$'s and using precision t floating-point arithmetic, one can compute some $\bar{r}_i$'s and $\bar{\mu}_{i,j}$'s such that:*

$$\forall i > j, |\bar{\mu}_{i,j} - \mu_{i,j}| \leq C_1 \rho^{2j} \epsilon \quad \text{and} \quad \forall i, |\bar{r}_i - r_i| \leq C_1 \rho^{2i} \epsilon \cdot r_i.$$

3 Floating-Point Lattice Enumeration

The usual method to solve SVP and CVP relies on the KFP enumeration [19, 8]. We refer to [1] for a comprehensive survey. Here we will consider the variant due to Schnorr and Euchner [32] since it is the fastest and the one used in NTL. After describing the algorithm, we explain how to use floating-point arithmetic and finally give our main results.

3.1 The Enumeration Algorithm

The KFP algorithm for SVP takes as input a lattice basis and returns a shortest non-zero lattice vector. For this, it considers some A and finds all solutions $(x_1, \ldots, x_d) \in \mathbb{Z}^d$ to the equation

$$\left\| \sum_{i=1}^{d} x_i \boldsymbol{b}_i \right\|^2 \leq A. \tag{1}$$

If $A \geq \|\boldsymbol{b}_1\|^2$, then the set of solutions is non-trivial and SVP is solved by keeping the best one. Equation (1) is equivalent to

$$\sum_{i=1}^{d} \left(x_i + \sum_{j=i+1}^{d} \mu_{j,i} x_j \right)^2 r_i \leq A. \tag{2}$$

We let $c_i = -\sum_{j=i+1}^{d} \mu_{j,i} x_j$ and perform the change of variable $y_i := x_i - c_i$. This corresponds to applying to $\boldsymbol{x}$ the triangular matrix whose diagonal coefficients are 1 and whose off-diagonal coefficients are the $\mu_{i,j}$'s. Any sequence $(y_i, \ldots, y_d)$ corresponds to a unique sequence $(x_i, \ldots, x_d)$. Equation (2) becomes $\sum_{i=1}^{d} y_i^2 r_i \leq A$, which implies that:

$$\begin{aligned} y_d^2 r_d &\leq A, \\ y_{d-1}^2 r_{d-1} &\leq A - y_d^2 r_d, \\ &\cdots \\ y_1^2 r_1 &\leq A - \sum_{j=2}^{d} y_j^2 r_j. \end{aligned}$$

KFP finds all y_d's satisfying the first equation, then all (y_{d-1}, y_d)'s satisfying the second equation, etc. until it discloses all $(y_1, \ldots, y_d)$'s satisfying the last equation. Let $i < d$. Suppose that $y_{i+1}, \ldots, y_d$ are already set. Then there is a finite number of possibilities for y_i since y_i belongs to a bounded interval and is the fixed shift (by c_i) of the integer variable x_i. The number of possibilities for y_i is $\leq 1 + 2\sqrt{A/r_i}$. This shows that the bigger the r_i's, the faster the enumeration. We will see that big r_i's also help decreasing the required floating-point precision needed for the computations. Overall, KFP consists in trying to build solution vectors $\sum_{i=1}^{d} x_i \boldsymbol{b}_i$ to Equation (1) by successively looking at the projections orthogonally to the spans of $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_i)$ for a decreasing i. For a given choice of $(x_{i+1}, \ldots, x_d)$, the variable x_i belongs to an interval centered in c_i. Its length is $\sqrt{\frac{A - \ell_{i+1}}{r_i}}$, where $\ell_{i+1} := \sum_{j>i} y_j^2 r_j$.

Schnorr and Euchner improved KFP as follows. Suppose $(x_{i+1}, \ldots, x_d)$ is set. Instead of looking at the possible x_i's in a straight increasing fashion, they are chosen from the center of the interval to its borders: the first value is $\lfloor c_i \rceil$, then the integer that is second closest to c_i, etc. This has the effect of sorting the ℓ_i's by increasing order, and thus of maximizing the likelihood of quickly finding a solution to Equation (1). Once a solution is found, the value of A may be decreased, which possibly cuts off many branches of the execution tree. In Figure 1, we give a detailed description of the enumeration algorithm using the Schnorr-Euchner zig-zag path. The vector $\boldsymbol{sol}$ stores the non-zero vector $\boldsymbol{x}$ that is currently thought as minimizing $\| \sum_{i \leq d} x_i \boldsymbol{b}_i \|$. It remains $\mathbf{0}$ as long as no length below $\sqrt{A}$ has been found. The Δx_i's and $\Delta^2 x_i$'s are used to implement the zig-zag path.

Input: A bound A. Approximations $\bar{\mu}_{i,j}$'s and $\bar{r}_i$'s to the Gram-Schmidt coefficients of a possibly unknown basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$.
Output: A coordinate vector $\boldsymbol{x} \in \mathbb{Z}^d \setminus \{\mathbf{0}\}$ such that $\sum_{i=1}^{d} x_i \boldsymbol{b}_i$ is likely to reach the lattice minimum.

1. $\boldsymbol{x} := (1, 0, \ldots, 0)$; $\boldsymbol{\Delta x} := (1, 0, \ldots, 0)$; $\boldsymbol{\Delta^2 x} := (1, -1, \ldots, -1)$; $\boldsymbol{sol} := \mathbf{0}$.
2. $\boldsymbol{c}, \boldsymbol{\ell}, \boldsymbol{y} := \mathbf{0}$.
3. $i := 1$. Repeat
4. $\quad y_i := |x_i - c_i|$; $\ell_i := \ell_{i+1} + y_i^2 r_i$.
5. $\quad$ If $\ell_i \leq A$ and $i = 1$, then $(\boldsymbol{sol}, A) := \texttt{update}(\boldsymbol{sol}, A, \boldsymbol{x}, \ell_1)$.
6. $\quad$ If $\ell_i \leq A$ and $i > 1$, then $i := i - 1$ and
7. $\quad\quad c_i := -\sum_{j=i+1}^{d} x_j \mu_{j,i}$.
8. $\quad\quad x_i := \lfloor c_i \rceil$; $\Delta x_i := 0$; if $c_i < x_i$ then $\Delta^2 x_i := 1$ else $\Delta^2 x_i := -1$.
9. $\quad$ Else if $\ell_i > A$ and $i = d$ return $\boldsymbol{sol}$ and stop.
10. $\quad$ Else $i := i + 1$ and
11. $\quad\quad \Delta^2 x_i := -\Delta^2 x_i$; $\Delta x_i := -\Delta x_i + \Delta^2 x_i$; $x_i := x_i + \Delta x_i$.

Fig. 1. The Schnorr-Euchner variant of the KFP enumeration algorithm.

The algorithm of Figure 1 calls an `update` routine. In the ideal case, i.e., with correct input Gram-Schmidt coefficients and exact computations, we simply take $\texttt{update}_1(\boldsymbol{sol}, A, \boldsymbol{x}, \ell_1) = (\boldsymbol{x}, \ell_1)$. If we use floating-point arithmetic, however, this strategy may lead us to cut off branches of the tree that could contain the minimal non-zero length: if the computed approximation to ℓ_1 under-estimates it and if the lattice minimum is between both values and has not been reached yet, it will be missed. One can avoid this pitfall when floating-point arithmetic is used but the lattice is perfectly known, i.e., the genuine $\boldsymbol{b}_i$'s or the correct Gram-Schmidt quantities are given. In that situation, it is useful to consider $\texttt{update}_2$ defined as follows: $\texttt{update}_2(\boldsymbol{sol}, A, \boldsymbol{x}, \ell_1) = (\boldsymbol{x}, A)$ when $\boldsymbol{sol} = \mathbf{0}$ or $\| \sum_i x_i \boldsymbol{b}_i \| \leq \| \sum_i sol_i \boldsymbol{b}_i \|$ (exactly), and $\texttt{update}_2(\boldsymbol{sol}, A, \boldsymbol{x}, \ell_1) = (\boldsymbol{sol}, A)$ otherwise.

When using floating-point arithmetic, it is crucial to specify the order in which the operations are performed. At Step 4, we will evaluate the term $y_i^2 r_i$ as: $\bar{r}_i \otimes (\bar{y}_i \otimes \bar{y}_i)$. At Step 7, we will evaluate $\sum_{j=i+1} x_j \mu_{j,i}$ as $(x_{i+1} \otimes \bar{\mu}_{i+1,i}) \oplus [(x_{i+2} \otimes \bar{\mu}_{i+2,i}) \oplus [\ldots \oplus (x_d \otimes \bar{\mu}_{d,i}) \ldots]]$. Finally, notice that the x_i's, Δx_i's, $\Delta^2 x_i$'s and sol_i's remain integers.

An iteration of the loop is uniquely determined by the values of i and $(x_i, \ldots, x_d)$ at the beginning of the iteration. We say that the state is $\sigma = (i, [x_i, \ldots, x_d])$. Let $i \leq d$ and $x_i, \ldots, x_d \in \mathbb{Z}$. The floating-point algorithm and the exact algorithm do not necessarily perform the same iterations, and even if they do they may not be performed in the same order. It is thus impossible to compare the values of the variables for a given loop iteration. However, one may compare the values of the variables for a given state of the loop. In both the exact and floating-point variants, the values of the c_i's, y_i's and ℓ_i's do not depend on the iteration, but only on the state. Furthermore, these values are well-defined even if they are not actually computed: they do not depend on the initial bound A, nor on the existence of an iteration with the right state, nor in the order in which the states are visited. Consider a variable of the algorithm. We use the notation v to represent its value at a given state with exact computations and $\bar{v}$ its value at the same state with floating-point computations.

3.2 Main Results

We consider a lattice basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ that is (δ, η)- LLL-reduced with $\eta \in [1/2, 1)$ and $\eta^2 + \frac{1}{(1+\eta)^2} < \delta < 1$. We let $\alpha = \frac{1}{\sqrt{\delta - \eta^2}}$ and $\rho = (1+\eta)\alpha$. The minimum of the lattice spanned by the $\boldsymbol{b}_i$'s is denoted by λ. Below, when using KFP, the basis may not be known. In that case, its Gram-Schmidt coefficients or approximations thereof are known. The former situation may arise if one knows only the Gram matrix of the basis. The latter is typical of BKZ-style algorithms: one tries to reduce a large-dimensional lattice basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ by enumerating short vectors of lattices spanned by the projections of the vectors $\boldsymbol{b}_{i+1}, \ldots, \boldsymbol{b}_{i+k}$ orthogonally to $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_i$, for some i and k; usually, one only knows approximations to the Gram-Schmidt coefficients of the projected k-dimensional basis.

Suppose we use floating-point arithmetic in the enumeration procedure, as described above. We denote by ϵ the unit in the last place and we define $K = 1 + \epsilon/2 \approx 1$. We allow the input Gram-Schmidt coefficients to be incorrect. For this purpose, we define:

$$\kappa = \max\left(\max_{i>j} \frac{\Delta\mu_{i,j}}{\epsilon}, \max_i \frac{\Delta r_i}{r_i \cdot \epsilon}\right).$$

If the Gram-Schmidt coefficients are exactly known and then rounded, we have $\kappa \leq 1$. They can also be computed as mentioned in Theorem 1, in which case we have $\kappa \leq C_1 \rho^{2d}(1+u')^{2d}$ for some small $u' > 0$.

To simplify the theorems below, we introduce some notation. We define $R = (1+\kappa\epsilon) \cdot \max_i r_i$: it bounds all the r_i's as well as the $r_i + \Delta r_i$'s. We also define:

$$C_2 = \frac{\kappa+2}{\alpha-1} + \frac{\kappa+4}{\rho-1}, \qquad C_3 = \frac{2\alpha(2+\kappa+2C_2)}{1+\eta-\alpha},$$
$$\epsilon' = 2\frac{R}{r_1}\left[(1+\kappa)\alpha^{2d} + (2C_2+C_3)\rho^d\right] \cdot \epsilon, \qquad C_4 = C_3\frac{K+d\epsilon}{1-\epsilon'}(2+d\epsilon).$$

The following theorem shows that when some exact knowledge of the lattice is provided then the floating-point enumeration solves SVP, if the precision is $\Omega(d)$ and the initial length upper bound is slightly increased in order to take care of the inaccuracies. In particular, in the most usual case where the r_i's decrease, one can choose $A = r_1 \cdot (1 + (2d + C_3\rho^d)\epsilon)$, which is only slightly larger than r_1. If the r_i's do not decrease, they can still be assumed of the same order of magnitude (up to a factor $2^{O(d)}$), thanks to the LLL-reducedness of the input basis, and the a priori knowledge that larger r_i's will not be used in vectors reaching the minimum.

Theorem 2. *Consider the floating-point KFP algorithm described in Subsection 3.1. Suppose that either the $\mathbf{b}_i$'s are known or that the Gram-Schmidt quantities are correct, and that the* `update`$_2$ *function is used. We assume that $C_2\rho^d \cdot \epsilon \leq 0.01$ and $A \geq (1+2d\epsilon) \cdot \lambda^2 + C_3\rho^d\epsilon \cdot R$. Then the returned coordinates **sol** satisfy $\| \sum_{i \leq d} sol_i \mathbf{b}_i \| = \lambda$.*

In the theorem above, we do not cut off branches of the computation once a short vector has been found: we keep the initial bound A. It is possible to decrease A each time a significantly shorter vector is found. Suppose a vector of exact squared norm $A' < A$ has been found. Then we can set $A = \min(A, A'(1+\epsilon"))$, for a well chosen $\epsilon"$ that can be made explicit. This takes care of possible slight over-estimates of internal ℓ_i's which could erroneously lead to the removal of useful loop iterations. For the sake of simplicity, we do not consider this variant here.

Within BKZ-style algorithms, one may only know approximations to the Gram-Schmidt coefficients of the input basis, making Theorem 2 useless in such situations. Furthermore, due to the input uncertainty, one may not be able to decide which is the shortest between two vectors of close-by lengths: one cannot do better than finding a vector which is not much longer than λ. Of course, if there is a sufficient gap between λ and

the length of any lattice vector different that does not reach the minimum, then an optimal solution will be found. The theorem below shows that finding a close to optimal vector is actually possible.

Theorem 3. *Consider the floating-point KFP algorithm described in Subsection 3.1, with the* $\texttt{update}_1$ *function. Let* $\gamma = \|\sum_i sol_i \mathbf{b}_i\|$ *be the norm of the found solution. If* $A \geq \bar{r}_1$ *and* $\epsilon' < 0.01$*, then:*

$$\lambda^2 \leq \gamma^2 \leq (1 + 4d\epsilon) \cdot \lambda^2 + C_4 \max\left(1, \frac{A}{r_1}\right) \rho^d \epsilon \cdot R.$$

It should be noted that floating-point variants of BKZ cannot solve their internal SVP instantiations exactly: the best they can do is to solve $(1+\epsilon")$-SVP instantiations instead, for some small $\epsilon"$. However, with a small enough $\epsilon"$, this does not change significantly the overall quality of the output bases.

The two results above provide as good as could be expected correctness guarantees to the floating-point enumeration. However, since the algorithm is not the rational one, the complexity analyzes do not hold anymore. The following theorem shows that the overhead of the floating-point enumeration with respect to the rational one is small.

Theorem 4. *Consider the floating-point KFP algorithm described in Subsection 3.1 with either of the update functions and either the knowledge of the basis or the Gram-Schmidt coefficients or only approximations thereof. Let* $\gamma = \|\sum_i sol_i \mathbf{b}_i\|$ *be the norm of the found solution. We suppose that* $\epsilon' < 0.01$*. Then the number of loop iterations is lower than the number of loop iterations of the rational algorithm given the genuine basis and an input bound* $A' = (1 + d\epsilon) \cdot A + C_4 \max\left(1, \frac{A}{r_1}\right) \rho^d \epsilon \cdot R$*.*

As a consequence of Theorems 2 and 4, the cost of Kannan's algorithm [19] can be decreased from $\mathrm{Poly}(n, \log B) \cdot d^{\frac{d}{2e}(1+o(1))}$ (see [14]) to $\left(d^{\frac{d}{2e}} + \mathrm{Poly}(n, \log B)\right) \cdot d^{o(d)}$: it suffices to use rationals everywhere but in the enumerations which should be performed with precision $\Theta(d)$.

4 Error Analysis of the Floating-Point Enumeration

We now turn to the proofs of Theorems 2, 3 and 4. We proceed by proving that the computed lengths $\bar{\ell}_i$ of the projected vectors are accurate. Lemma 1 means that $\bar{\ell}_1$ cannot be much larger than ℓ_1, which suffices for Theorem 3. For the other results, we need the converse: Lemma 2 means

that the true ℓ_i cannot be much larger than the computed one. The proofs of Lemmata 1 and 2 are explained in Subsection 4.2.

As mentioned in Section 3, an $\bar{\ell}_i$ computed by the floating-point algorithm may not correspond to any ℓ_i computed by the rational one with the same bound A, and vice-versa. To be rigorous, we need the following definitions. For $\boldsymbol{x} \in \mathbb{Z}^d$, we let $n(\boldsymbol{x}) = \| \sum_{i=1}^{d} x_i \boldsymbol{b}_i \|^2$ and $\bar{n}(\boldsymbol{x})$ its approximation as would be computed by the enumeration were the state $(1, [x_1, \ldots, x_d])$ visited. We use the notations and hypotheses of Subsection 3.1.

Lemma 1. *Suppose that $C_2 \rho^d \cdot \epsilon < 0.01$. Let $\boldsymbol{x} \in \mathbb{Z}^d$. If $n(\boldsymbol{x}) \leq r_1$, then:*

$$\bar{n}(\boldsymbol{x}) \leq (1 + 2d\epsilon) \cdot n(\boldsymbol{x}) + C_3 \rho^d \epsilon \cdot R.$$

Lemma 2. *Suppose that $\epsilon' < 0.01$. Let $\boldsymbol{x} \in \mathbb{Z}^d$ and $i \leq d$. We consider the state $(i, [x_i, \ldots, x_d])$. Then*

$$\ell_i \leq (1 + d\epsilon) \cdot \bar{\ell}_i + C_3 \max\left(1, \frac{\bar{\ell}_i (K + d\epsilon)}{r_1 (1 - \epsilon')}\right) \rho^d \epsilon \cdot R.$$

4.1 Using Lemmata 1 and 2 to Prove the Theorems

Let us first prove Theorem 2 from Lemma 1. Let $(x_1, \ldots, x_d)$ be the coordinates of a shortest vector. If the state $(1, \boldsymbol{x})$ is considered by the floating-point algorithm with $A \geq (1 + 2d\epsilon) \cdot \lambda^2 + C_3 \rho^d \epsilon \cdot R$, then a shortest vector will be found. Making sure that $(1, \boldsymbol{x})$ is indeed considered is the purpose of the following lemma. It relies on subtle properties of the floating-point model, in particular that the rounding is a non-decreasing function.

Lemma 3. *If one uses the* $\texttt{update}_1$ *function within the enumeration, then all coordinate vectors $\boldsymbol{x}$ such that $\bar{n}(\boldsymbol{x}) \leq A$ will indeed be considered during the execution.*

Proof. Let $\boldsymbol{x} \in \mathbb{Z}^d$ with $\bar{n}(\boldsymbol{x}) \leq A$. We show by induction on decreasing i that $(i, [x_i, \ldots, x_d])$ is considered and that at this moment the test $\bar{\ell}_i \leq A$ is satisfied. Let $i \leq d$. We consider the sequence $(\sigma_1, \ldots, \sigma_\tau)$ of considered states $(i, [X, x_{i+1}, \ldots, x_d])$ with $X \in \mathbb{Z}$. It is non-empty if $i = d$, and it is also non-empty if $i < d$ by induction hypothesis.

The sequence $\left(\bar{\ell}_i(\sigma_t)\right)_t$ is non-decreasing. The first integer $X = x_i(\sigma_1)$ is exactly $\lfloor \bar{c}_i \rceil$. The computation of $x_i(\sigma_t)$ from $x_i(\sigma_{t-1})$ is exact, and the distance between $x_i(\sigma_t)$ and $\bar{c}_i$ is non-decreasing. Since the rounding function is non-decreasing, the sequence $(\bar{y}_i(\sigma_t))_t$ is also non-decreasing. For the same reason, the sequence $\left(\bar{\ell}_i(\sigma_t)\right)_t$ is non-decreasing.

Consider the value $\bar{\ell}$ of $\bar{\ell}_i$ were it computed with $(x_i, \ldots, x_d)$. We have $\bar{\ell} \leq \bar{n}(\boldsymbol{x}) \leq A$. Since $\bar{\ell}_i(\sigma_\tau) > A$, there must exist t such that $x_i(\sigma_t) = x_i$ and the test $\bar{\ell}_i \leq A$ is satisfied for that state σ_t. □

We now prove Theorem 3. If we use $\texttt{update}_2$, the bound A may decrease during the execution, to finally reach a value A_{end}. The final output would have been the same if we had started with $A = A_{end}$. We consider that it is the case, which implies that A is not modified during the execution. Let $\boldsymbol{x} \in \mathbb{Z}^d$ such that $n(\boldsymbol{x}) = \lambda^2$. Lemma 1 implies that $\bar{n}(\boldsymbol{x}) \leq (1 + 2d\epsilon) \cdot \lambda^2 + C_3\rho^d\epsilon \cdot R$. We must have $A \leq (1 + 2d\epsilon) \cdot \lambda^2 + C_3\rho^d\epsilon \cdot R$ since otherwise A would have been decreased after $\boldsymbol{x}$ was found. Applying Lemma 2 with $\boldsymbol{sol}$ and using the above bound on A provides the result.

For Theorem 4, consider a state $(i, [x_i, \ldots, x_d])$ with a successful test $\bar{\ell}_i \leq A$. Lemma 2 gives $\ell_i \leq (1+d\epsilon)\cdot A + C_3 \max\left(1, \frac{A(K+d\epsilon)}{r_1(1-\epsilon')}\right)\rho^d\epsilon \cdot R \leq A'$. Therefore, the exact algorithm with the bound A' would have considered this state and the corresponding test would have been successful as well. Moreover, there are as many failed loop iterations with $i < d$ as successful loop iterations with $i > 1$. This completes the proof.

4.2 Proving Lemmata 1 and 2

The proofs of Lemmata 1 and 2 rely on standard techniques of floating-point error analysis. We simultaneously bound the errors and the variables, which leads us to use an induction on the decreasing index i. Within the induction step, we rely on three basic facts whose proofs are tedious but straightforward. They are given in the appendix of the full version.

Lemma 4. *Suppose that $C_2\rho^d\epsilon \leq 0.01$. Suppose we are at the end of Step 4 of some loop iteration with state $(i, [x_i, \ldots, x_d])$. If there exists a constant $\nu \geq 1$ such that for any $j > i$ we have $y_j \leq \nu\alpha^{j-1}$, then*

$$\Delta c_i \leq C_2\nu\alpha^d(1+\eta)^{d-i}\epsilon \quad \textit{and} \quad \Delta y_i \leq y_i\epsilon/2 + KC_2\nu\alpha^d(1+\eta)^{d-i}\epsilon.$$

Lemma 5. *At Step 4 of the floating-point algorithm, we have:*

$$|(\bar{y}_i \otimes \bar{y}_i) \otimes \bar{r}_i - r_i y_i^2| \leq RK^2[(\kappa+1)y_i^2\epsilon + (2y_i + \Delta y_i)\Delta y_i]$$

Lemma 6. *Suppose that $C_2\rho^d\epsilon \leq 0.01$. Suppose we are at the end of Step 4 of some loop iteration with state $(i, [x_i, \ldots, x_d])$. If there exists a constant $\nu \geq 1$, such that for any $j \geq i$ we have $y_i \leq \nu\alpha^{i-1}$, then:*

$$\Delta\ell_i \leq d\epsilon \cdot \ell_i + C_3\nu^2\rho^d\epsilon \cdot R \quad \textit{and} \quad \Delta\ell_i \leq d\epsilon \cdot \bar{\ell}_i + C_3\nu^2\rho^d\epsilon \cdot R.$$

We can now prove Lemma 1. Let $\boldsymbol{x} \in \mathbb{Z}^d$ such that $n(\boldsymbol{x}) \leq r_1$. Since the basis is LLL-reduced, the y_i's corresponding to $\boldsymbol{w}$ satisfy $y_i \leq \sqrt{n(\boldsymbol{x})/r_i} \leq \sqrt{r_1/r_i} \leq \alpha^{i-1}$. The first part of Lemma 6 with $\nu = 1$ provides the result.

Finally, we prove Lemma 2. Let $\boldsymbol{x} \in \mathbb{Z}^d$ and $i \leq d$. We show by induction on j decreasing from d to i that the bound on Δc_j of Lemma 4 holds and that we have $y_j \leq \nu\alpha^{j-1}$, with $\nu = \max\left(1, \sqrt{\frac{\bar{\ell}_i(K+d\epsilon)}{r_1(1-\epsilon')}}\right)$. Lemma 2 will then follow from the second part of Lemma 6. Let $j \geq i$. By induction, we have $y_k < \nu\alpha^{k-1}$ for any $k > j$, so that the bounds of Lemma 4 hold. It remains to see that $y_j \leq \nu\alpha^{j-1}$. Lemmata 5 and 6 provide:

$$\begin{aligned} r_j y_j^2 \leq \ell_j &\leq \bar{\ell}_j + \Delta\ell_j \leq K\bar{\ell}_j + \Delta\ell_{j+1} + \left|(\bar{y}_j \otimes \bar{y}_j) \otimes \bar{r}_j - r_j y_j^2\right| \\ &\leq K\bar{\ell}_j + d\epsilon\bar{\ell}_j + C_3\nu^2\rho^d\epsilon R + RK^2\left[(\kappa+1)y_j^2\epsilon + (2y_j + \Delta y_j)\Delta y_j\right]. \end{aligned}$$

We use Lemma 4 to bound Δy_j in the equation above. This leads $P(y_j) \leq 0$, where P is the degree-2 polynomial with coefficients:

$$\begin{aligned} P_0 &= -\bar{\ell}_j(K + d\epsilon) - C_3\nu^2 R\rho^d\epsilon - RK^4(C_2\nu\rho^d\epsilon)^2, \\ P_1 &= -2RK^4C_2\nu\alpha^{d-j}(1+\eta)^d\epsilon \quad \text{and} \quad P_2 = r_j - 2RK^3(\kappa+1)\epsilon. \end{aligned}$$

The fact that $\epsilon' < 0.01$ implies that $P_2 > 0$ and thus that y_j is below the positive root of P. It can be checked that $P(\nu\alpha^{j-1}) \geq 0$, which implies that $y_j \leq \nu\alpha^{j-1}$. This completes the proof.

5 Practical Considerations

The algorithm described in Section 3 has been implemented in C++ and is freely distributed within `fplll-3.0` [5]. The code does not use the worst-case bounds above but remains guaranteed, as explained below. We also explain how our results may be used within BKZ-style algorithms.

5.1 Guaranteeing the Computations with Smaller Precision

The worst-case bounds given in Section 3 are very pessimistic for generic instantiations. This is due to the facts that all $|\mu_{i,j}|$'s (resp. r_{i-1}/r_i's) are bounded by their worst-case value η (resp. α^2) and all floating-point errors are considered to be always maximal and in the worst direction. Although they might occur, cases where all these bounds are tight are unlikely. In the worst-case analysis, we also use loose bounds to simplify the technicalities, though they do not modify the terms that are exponential with d. For $(\delta, \eta) = (0.99, 0.51)$, if the Gram-Schmidt coefficients are

correct up to their last bit ($\kappa \leq 1$), the provably sufficient precision for a d-dimensional enumeration is $\approx 0.8 \cdot d$ (when d grows to infinity). To take advantage of the machine instructions, one is tempted to use double precision, i.e., $\epsilon = 2^{-52}$. In that case, the enumeration is guaranteed up to dimension ≈ 45 (for an output relative error $\leq 1\%$).

In practice, one should rather turn the worst-case error analysis into an algorithm. One can use the values the actual Gram-Schmidt coefficients rather than general upper bounds. If they are known approximately, one should take into consideration their intrinsic inaccuracies. The adaptive precision computation uses $O(d^2)$ arithmetic operations: Lemmata 4 and 6 are applied $O(d)$ times each and both perform $O(d)$ operations. This computation is thus dominated by the enumeration. The error computations are themselves performed in floating-point arithmetic, but one should be cautious with the rounding modes: since we try to upper bound a quantity, the default rounding to nearest should be replaced by roundings towards infinities and zero. In the code, we used MPFR [30] for that purpose.

The table below illustrates the above technique. Each entry corresponds to 10 samples of the following experiment. A $(d+1) \times d$ matrix B is sampled: for any i, $B[1,i]$ is a random integer with $100 \cdot d$ bits, $B[i+1,i]$ is 1 and the other entries are 0. The columns of the matrix B are then $(0.99, 0.51)$-LLL-reduced. Then the adaptive precision computation is performed. The precision is computed so that the algorithm is guaranteed to solve 1.01-SVP. One observes that double precision suffices for dimensions up to 90, which is higher than what is currently handleable in practice.

Dimension d	20 30 40 50 60 70 80
Worst-case required precision (Theorem 3)	33 41 49 57 66 74 82
Adaptively computed required precision (worst-case over the samples)	20 25 29 33 38 42 47

5.2 Enumerating within BKZ-style Algorithms

With the floating-point LLL of Nguyen and Stehlé [27] and the present results, one may use floating-point arithmetic within BKZ-style algorithms in a guaranteed way. However, it is not clear yet how to maximize the efficiency while doing this. As a target, double precision should be used as much as possible, since multi-precision arithmetic is significantly slower.

A first solution consists in performing all operations with the same provably sufficient precision, provided by the bounds given in Section 3 after replacing κ by the bounds of Theorem 1 and R by $2\alpha^{2d} \cdot r_1$ (the vectors whose r_i's are $> \alpha^{2d} r_1$ cannot be used to create a vector of minimal

non-zero length). Though the precision remains $O(d)$, it will be fairly large and slow multi-precision arithmetic will be necessary. It can be checked that the required precision can be decreased by a constant factor by noticing that in Theorem 1 the errors on $\mu_{i,j}$ and r_j depends on j.

Another possibility is to use a Gram-Schmidt orthogonalization with very high precision and then use the adaptive precision estimate described above. Double precision is likely to be sufficient for all reasonable values of the hierarchy parameter k, making the computed approximations to the Gram-Schmidt coefficients correct up to relative error $\approx 2^{-53}$. Since the enumerations are likely to dominate the overall cost, it is worth using multi-precision arithmetic to compute accurate Gram-Schmidt coefficients in order to be allowed double precision within the enumerations.

If the Gram-Schmidt computations are not negligible with respect to the enumerations, then one could try using double precision in all computations. This may be done by relying with the following strategy:

- Run the floating-point LLL algorithm with double precision for the Gram-Schmidt computations, with infinite loop detection (see [34]).
- If the double precision seemed to suffice (i.e., the execution terminated without an infinite loop detection), compute a posteriori accuracy bounds as described by Villard in [35].
- Run the adaptive precision computation to see if double precision suffices for the enumeration.

6 Concluding Remarks

We proved strong numerical properties of the KFP enumeration algorithm, which gives a stronger insight about the use of floating-point arithmetic within lattice reduction algorithms. To obtain a full hierarchy of reduction algorithms ranging from LLL to HKZ that efficiently relies on floating-point arithmetic, it only remains to see how to combine our new results with those on floating-point LLL from [27]. It would also be interesting to devise new techniques to decrease the required precision in order to be able to use double precision as often as possible.

However, we answered only one of the two main troubles related to BKZ-style algorithms: it is still unknown how to best use small dimensional lattice enumeration within a large dimensional reduction. It would be desirable to have an algorithm which is theoretically at least as good as the best current one [10], that would beat BKZ in practice and whose behavior would be perfectly understood. Once this will be done, there will remain to mount massive computational projects to assess the limits of

current computers against lattice-based cryptography. It will then make sense to run the enumeration on hardware. Our analysis extends to fixed-point arithmetic, which is the natural arithmetical choice in hardware.

References

1. E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, 2002.
2. M. Ajtai. The shortest vector problem in l_2 is NP-hard for randomized reductions (extended abstract). In *Proc. of the 30th Symposium on the Theory of Computing (STOC 1998)*, pages 284–293. ACM Press, 1998.
3. M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proc. of the 29th Symposium on the Theory of Computing (STOC 1997)*, pages 284–293. ACM Press, 1997.
4. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proc. of the 33rd Symposium on the Theory of Computing (STOC 2001)*, pages 601–610. ACM Press, 2001.
5. D. Cadé, X. Pujol, and D. Stehlé. fplll-3.0, a floating-point LLL implementation. Available at `http://perso.ens-lyon.fr/damien.stehle`.
6. H. Cohen. *A Course in Computational Algebraic Number Theory, 2nd edition.* Springer-Verlag, 1995.
7. P. van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical report 81-04, Mathematisch Instituut, Universiteit van Amsterdam, 1981.
8. U. Fincke and M. Pohst. A procedure for determining algebraic integers of given norm. In *Proc. of EUROCAL*, volume 162 of *Lecture Notes in Computer Science*, pages 194–202. Springer-Verlag, 1983.
9. N. Gama, N. Howgrave-Graham, H. Koy, and P. Nguyen. Rankin's constant and blockwise lattice reduction. In *Proc. of Crypto 2006*, number 4117 in Lecture Notes in Computer Science, pages 112–130. Springer-Verlag, 2006.
10. N. Gama and P. Nguyen. Finding short lattice vectors within Mordell's inequality. In *Proc. of the 40th Symposium on the Theory of Computing (STOC 2008)*, pages 207–216. ACM Press, 2008.
11. N. Gama and P. Nguyen. Predicting lattice reduction. In *Proc. of Eurocrypt 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer-Verlag, 2008.
12. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proc. of the 40th Symposium on the Theory of Computing (STOC 2008)*, pages 197–206. ACM Press, 2008.
13. O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proc. of Crypto 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer-Verlag, 1997.
14. G. Hanrot and D. Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm (extended abstract). In *Proc. of Crypto 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 170–186. Springer-Verlag, 2007.
15. A. Hassibi and S. Boyd. Integer parameter estimation in linear models with applications to GPS. *IEEE Transactions on signal process.*, 46(11):2938–2952, 1998.
16. N. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM Publications, 2002.

17. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: a ring based public key cryptosystem. In *Proc. of the 3rd Algorithmic Number Theory Symposium (ANTS III)*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer-Verlag, 1998.
18. A. Joux and J. Stern. Lattice reduction: a toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998.
19. R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proc. of the 15th Symposium on the Theory of Computing (STOC 1983)*, pages 99–108. ACM Press, 1983.
20. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.
21. V. Lyubashevsky. Lattice-based identification schemes secure under active attacks. In *Proc. of PKC 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 162–179. Springer-Verlag, 2008.
22. V. Lyubashevsky and D. Micciancio. Asymptotically efficient lattice-based digital signatures. In *Proc. of TCC 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 37–54. Springer-Verlag, 2008.
23. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: a modest proposal for FFT hashing. In *Proc. of FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer-Verlag, 2008.
24. D. Micciancio and Regev O. Lattice-based cryptography. In *Proc. of PQCrypto 2008*, Lecture Notes in Computer Science. Springer-Verlag, 2008.
25. W. H. Mow. Maximum likelihood sequence estimation from the lattice viewpoint. *IEEE Transactions on Information Theory*, 40:1591–1600, 1994.
26. P. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto'97. In *Proc. of Crypto 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. Springer-Verlag, 1999.
27. P. Nguyen and D. Stehlé. Floating-point LLL revisited. In *Proc. of Eurocrypt 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 215–233. Springer-Verlag, 2005.
28. P. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. To appear in the Journal of Mathematical Cryptology, 2008.
29. A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In *Proc. of Cryptology and Computational Number Theory*, volume 42 of *Proc. of Symposia in Applied Mathematics*, pages 75–88. American Mathematical Society, 1989.
30. The SPACES Project. MPFR, a LGPL-library for multiple-precision floating-point computations with exact rounding. Available at `http://www.mpfr.org/`.
31. C. P. Schnorr. A hierarchy of polynomial lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.
32. C. P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematics of Programming*, 66:181–199, 1994.
33. V. Shoup. NTL, Number Theory Library. Available at `http://www.shoup.net/`.
34. D. Stehlé. Floating-point LLL: theoretical and practical aspects. In *Proc. of the LLL+25 conference*. To appear.
35. G. Villard. Certification of the QR factor R, and of lattice basis reducedness. In *Proc. of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC'07)*. ACM Press, 2007.