AN EFFICIENT ALGORITHM FOR INTEGER LATTICE REDUCTION*

FRANÇOIS CHARTON†, KRISTIN LAUTER‡, CATHY LI§, AND MARK TYGERT¶

Abstract. A lattice of integers is the collection of all linear combinations of a set of vectors for which all entries of the vectors are integers and all coefficients in the linear combinations are also integers. Lattice reduction refers to the problem of finding a set of vectors in a given lattice such that the collection of all integer linear combinations of this subset is still the entire original lattice and so that the Euclidean norms of the subset are reduced. The present paper proposes simple, efficient iterations for lattice reduction which are guaranteed to reduce the Euclidean norms of the basis vectors (the vectors in the subset) monotonically during every iteration. Each iteration selects the basis vector for which projecting off (with integer coefficients) the components of the other basis vectors along the selected vector minimizes the Euclidean norms of the reduced basis vectors. Each iteration projects off the components along the selected basis vector and efficiently updates all information required for the next iteration to select its best basis vector and perform the associated projections.

Key words. matrix, projection, orthogonalization, optimization, cryptography, number theory

MSC codes. 65F25, 65K10, 11T71

1. Introduction. Lattices of integers are common tools in cryptography and number theory, among other areas, as reviewed by [4], [11], and others. A lattice of integers in m dimensions consists of all linear combinations of a set of n vectors, with all coefficients in the linear combinations being integers and all m entries of each of the n vectors being integers. Thus, the lattice is a collection of infinitely many vectors of integers.

The set of n vectors whose linear combinations form the lattice is known as a "basis" for the lattice. Traditionally the vectors in the basis are required to be linearly independent, but throughout the present paper, the term "basis" will abuse terminology slightly in omitting any requirement to be linearly independent. The term "reduced basis" refers to any unimodular integer linear transformation of a basis. (A unimodular integer linear transformation is multiplication with an invertible $n \times n$ matrix such that all entries of the matrix and its inverse are integers; the entries of the inverse of a matrix of integers are integers if and only if the absolute value of the determinant of the matrix is 1 — this follows straightforwardly from the Cramer Rule for solving systems of linear equations.) The goal of lattice reduction (the topic of the present paper) is to minimize the Euclidean norms of the vectors in the reduced basis.

Finding a nonzero vector in the lattice whose Euclidean norm is least (or nearly least) is a common use of lattice reduction. To see the connection, fix any positive real number p. Any reduced basis which minimizes the sum of the p-th powers of the Euclidean norms of the vectors in the reduced basis must include a shortest vector as one of the basis vectors. Indeed, if no basis vector is the shortest vector, then the sum of the p-th powers of the Euclidean norms can be made smaller by replacing with

*Submitted to the editors DATE.

Funding: This work was funded by Meta Platforms, Inc.

†Meta Platforms, Inc., 6 Rue Ménars 75002, Paris, France (fcharton@meta.com).

‡Meta Platforms, Inc., 1101 Dexter Ave N, Seattle, WA (klauter@meta.com).

§Meta Platforms, Inc., 1101 Dexter Ave N, Seattle, WA (cathyli@meta.com).

¶Meta Platforms, Inc., 1 Facebook Way, Menlo Park, CA (tygert@meta.com).

the shortest vector one of the basis vectors whose projection on the shortest vector is nonzero. "Projection" is defined as follows.

The projection of a column vector v onto a column vector w is $c \cdot w$, with the scalar coefficient c defined to be the real number

$$c = \frac{w^\top v}{w^\top w}, \tag{1.1}$$

where $w^\top$ denotes the transpose of w. Needless to say, $w^\top v$ is the inner product between w and v, while $w^\top w$ is the inner product of w with itself, which is of course the square of the Euclidean norm of w.

Unfortunately, even if all entries of both v and w are integers, subtracting off this projection $c \cdot w$ from v directly would result in a vector, $v - c \cdot w$, whose entries may not be integers. Fortunately, all entries of the vector $v - \mathrm{nint}(c) \cdot w$ would remain integers, and we prove in Subsection 2.2 below that the Euclidean norm of $v - \mathrm{nint}(c) \cdot w$ is less than or equal to the Euclidean norm of v. (Here and throughout the present paper, $\mathrm{nint}(c)$ denotes the result of rounding c to the nearest integer. The reader is welcome to use any convention desired for the definition of $\mathrm{nint}(k+1/2)$, where k is an integer, provided that the convention is used consistently. The proofs in Subsection 2.2 below omit special consideration of these special cases, as the proofs are trivial in such cases.) In fact, if both v and w are in a lattice of integers, then $v - \mathrm{nint}(c) \cdot w$ is in the same lattice of integers and is shorter (or at least no longer) than v. Theorem 2.6 in Subsection 2.2 below elaborates. Mapping the pair v and w to the pair $v - \mathrm{nint}(c) \cdot w$ and w is clearly a unimodular integer linear transformation, as is any composition of such mappings (since the product of unimodular integer linear transformations is just another unimodular integer linear transformation).

Let us denote by a_1^0, a_2^0, ..., a_n^0 the initial basis vectors, each being a column vector of m integers. The present paper proposes an iterative algorithm that reduces the basis from iteration i to iteration $i+1$ from a_1^i, a_2^i, ..., a_n^i to a_1^{i+1}, a_2^{i+1}, ..., a_n^{i+1}, starting with $i = 0$. Each iteration i of the algorithm selects an index k that minimizes the Euclidean norms $\|a_j^i - c_{j,k}^i \cdot a_k^i\|$ for all $j \neq k$, where the scalar projection coefficient $c_{j,k}^i$ is

$$c_{j,k}^i = \mathrm{nint}\left(\frac{(a_k^i)^\top a_j^i}{(a_k^i)^\top (a_k^i)} \right) \tag{1.2}$$

for $j \neq k$ and

$$c_{j,k}^i = 0 \tag{1.3}$$

for $j = k$. Specifically, the index k is chosen to minimize the sum of the p-th powers of the Euclidean norms, that is, k minimizes $\sum_{j=1}^n \|a_j^i - c_{j,k}^i \cdot a_k^i\|^p$. (If $p = 2$, then this sum is the square of the Frobenius norm of the matrix.) Iteration i constructs the matrix for the next iteration as the matrix whose columns are the basis vectors $a_j^{i+1} = a_j^i - c_{j,k}^i \cdot a_k^i$.

Prima facie, considering all possible indices in order to minimize the sum of the p-th powers appears to be computationally expensive. However, the sum being minimized can be evaluated efficiently via the Gram matrix whose entries are $g_{j,k}^i = (a_k^i)^\top a_j^i$, as can the projection coefficients $c_{j,k}^i$ defined in (1.2). Moreover, updating the entries of the Gram matrix from one iteration to the next is also computationally efficient.

The algorithm of the present paper can complement others for lattice reduction, such as the "LLL" algorithm introduced by [7]. Two entire books about the LLL algorithm are those of [10] and [3]. The numerical examples presented in Section 3 below report the results of running the algorithm of [7] in conjunction with the algorithm of the present paper. The implementation of the LLL algorithm used here is significantly more efficient (albeit less robust to worst-case, adversarial examples devised for applications outside cryptography) than the others' reviewed by [13] and [14]; see Subsection 3.3 below for details.[1]

The primary purpose of the algorithm of the present paper is to polish the outputs of other algorithms for lattice reduction; on its own, the algorithm of the present paper tends to get stuck in local minima, with the iterations attaining a fixed-point equilibrium that is far away from optimally minimizing the sum of the p-th powers. The convergence is monotonic, but need not arrive at the optimal minimum when fully converged.

The remainder of the present paper has the following structure: Section 2 describes and analyzes the algorithm in detail. Section 3 illustrates the algorithm via several numerical examples, comparing and combining the proposed algorithm with the classic method of [7]. Section 4 draws some conclusions. Section SM1 of the Supplementary Materials sketches several seemingly natural alternatives that performed poorly in numerical experiments; this supplementary section also points to other authors' variations of the LLL algorithm. Section SM2 of the Supplementary Materials complements the figures of Section 3 with further figures.

2. Methods. This section elaborates the algorithm of the present paper. First, Subsection 2.1 details the algorithm and estimates its computational costs. Then, Subsection 2.2 proves that the algorithm converges monotonically, strictly reducing the sum of the p-th powers of the Euclidean norms of the basis vectors during every iteration (and hence halting after a finite number of iterations).

2.1. Algorithm. This subsection describes the algorithm of the present paper in detail. Pseudocode is available in Algorithm 1.

Consider the n column vectors a_1^0, a_2^0, …, a_n^0, each of size $m \times 1$. The proposed scheme makes no assumption on the relative sizes of m and n; that is, m can be less than, equal to, or greater than n. Calculate the entries of the symmetric square Gram matrix

$$g_{j,k}^0 = (a_k^0)^\top (a_j^0) \tag{2.1}$$

for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$, where $(a_k^0)^\top$ denotes the transpose of a_k^0. (Of course, $(a_k^0)^\top (a_j^0)$ is simply the inner product between a_k^0 and a_j^0.) This costs $\mathcal{O}(mn^2)$ operations.

Iterations, $i = 0, 1, 2, \ldots$, will maintain the relation

$$g_{j,k}^i = (a_k^i)^\top (a_j^i) \tag{2.2}$$

for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$.

Now repeat all of the following steps, again and again, moving from $i = 0$ to $i = 1$ to $i = 2$ and so on:

[1] Permissively licensed open-source codes implementing both the algorithm of the present paper and the classical LLL algorithm of [7] are available at https://github.com/facebookresearch/latticer

Calculate the projection coefficients

$$c^i_{k,k} = 0 \tag{2.3}$$

for $k = 1, 2, \ldots, n$, and

$$c^i_{j,k} = \text{nint}\left(\frac{(a^i_k)^\top a^i_j}{(a^i_k)^\top (a^i_k)}\right) = \text{nint}\left(\frac{g^i_{j,k}}{g^i_{k,k}}\right) \tag{2.4}$$

for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$ with $k \neq j$, where nint denotes the nearest integer. This costs $\mathcal{O}(n^2)$ operations.

The sum of the p-th powers of the Euclidean norms when projecting off the k-th vector is

$$s^i_k = \sum_{j=1}^{n} \|a^i_j - c^i_{j,k}\, a^i_k\|^p = \sum_{j=1}^{n} \left(g^i_{j,j} + (c^i_{j,k})^2\, g^i_{k,k} - 2c^i_{j,k}\, g^i_{j,k}\right)^{p/2} \tag{2.5}$$

for $k = 1, 2, \ldots, n$. This costs $\mathcal{O}(n^2)$ to compute.

Define $\tilde{\imath}$ to be the index such that $s^i_{\tilde{\imath}}$ is minimal, that is,

$$\tilde{\imath} = \operatorname*{argmin}_{1 \leq k \leq n} s^i_k. \tag{2.6}$$

This costs $\mathcal{O}(n)$ to calculate.

Project off the $\tilde{\imath}$-th vector to update every vector

$$a^{i+1}_k = a^i_k - c^i_{k,\tilde{\imath}}\, a^i_{\tilde{\imath}} \tag{2.7}$$

for $k = 1, 2, \ldots, n$, that is,

$$(a^{i+1}_k)_j = (a^i_k)_j - c^i_{k,\tilde{\imath}}\, (a^i_{\tilde{\imath}})_j \tag{2.8}$$

for $j = 1, 2, \ldots, m$, and $k = 1, 2, \ldots, n$, where $(a^i_k)_j$ denotes the j-th entry of a^i_k. This costs $\mathcal{O}(mn)$.

Update the Gram matrix via the relation

$$g^{i+1}_{j,k} = (a^{i+1}_k)^\top (a^{i+1}_j) = (a^i_k - c^i_{k,\tilde{\imath}}\, a^i_{\tilde{\imath}})^\top (a^i_j - c^i_{j,\tilde{\imath}}\, a^i_{\tilde{\imath}}) = g^i_{j,k} + c^i_{j,\tilde{\imath}}\, c^i_{k,\tilde{\imath}}\, g^i_{\tilde{\imath},\tilde{\imath}} - c^i_{j,\tilde{\imath}}\, g^i_{\tilde{\imath},k} - c^i_{k,\tilde{\imath}}\, g^i_{j,\tilde{\imath}} \tag{2.9}$$

for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$. This costs $\mathcal{O}(n^2)$.

The total cost per iteration is $\mathcal{O}(mn + n^2)$ operations. Notice that only the precomputation of the Gram matrix in (2.1) and (2.8) explicitly involve the individual entries of the vectors; all other steps in the iterations involve only the entries of the Gram matrix.

Remark 2.1. Another possibility is to replace the sum in (2.5) with a maximum, replacing (2.5) with

$$s^i_k = \max_{1 \leq j \leq n} \|a^i_j - c^i_{j,k}\, a^i_k\|^2 = \max_{1 \leq j \leq n} \left(g^i_{j,j} + (c^i_{j,k})^2\, g^i_{k,k} - 2c^i_{j,k}\, g^i_{j,k}\right) \tag{2.10}$$

for $k = 1, 2, \ldots, n$. However, using the maximum may fail to force any but the longest vectors in the reduced basis to become shorter.

Remark 2.2. The Gram matrix is symmetric, that is, $g^i_{j,k} = g^i_{k,j}$ for all $i = 0, 1, 2, \ldots$, for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$. Calculating $g^i_{j,k}$ for only $j \le k$ suffices to fill the entire matrix for all $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$. This can save computational costs in (2.1) and (2.9).

Remark 2.3. Cryptography can benefit from reductions to collections of basis vectors that include all the unit basis vectors, each multiplied by the prime order of a finite field, in addition to the other basis vectors. This essentially formalizes the concept of lattice reduction over the finite field. Instead of working directly on a collection of basis vectors $\left(\begin{array}{c|c} A & q \cdot \mathrm{Id} \end{array}\right)$ in this way, where the columns of A form the initial collection of basis vectors, q is the order of the finite field, and Id is the identity matrix, the classical methods for lattice reduction — such as the LLL algorithm of [7] — must operate instead on

$$\left(\begin{array}{c|c} A & q \cdot \mathrm{Id} \\ \hline \mathrm{Id} & 0 \end{array}\right), \tag{2.11}$$

effectively doubling the dimension of the basis vectors.

2.2. Theory. The purpose of this subsection is to state and prove Theorem 2.6, elaborating the facts stated in the fifth paragraph of the introduction, Section 1.

The following lemma is helpful in the proof of the subsequent lemma.

LEMMA 2.4. *Suppose that r is a real number. Then*

$$(\mathrm{nint}(r))^2 - 2\,\mathrm{nint}(r)\,r \le 0. \tag{2.12}$$

Proof. Since the sign of $\mathrm{nint}(r)$ is the same as the sign of r (or 0), the sign of the left-hand side of (2.12) is the sign of r (or 0) times the sign of

$$\mathrm{nint}(r) - 2r. \tag{2.13}$$

Now, the sign of (2.13) is opposite to the sign of r: if $r > 1/2$, then $\mathrm{nint}(r) - 2r \le r + 1/2 - 2r = 1/2 - r < 0$; if $r < -1/2$, then $\mathrm{nint}(r) - 2r \ge r - 1/2 - 2r = -1/2 - r > 0$; if $|r| < 1/2$, then $\mathrm{nint}(r) - 2r = -2r$, whose sign is opposite to r's. Combining these observations yields (2.12). ∎

The following lemma is helpful in the proof of the subsequent theorem.

LEMMA 2.5. *Suppose that $c^i_{j,k}$ is the coefficient defined in (2.4) and $g^i_{j,k}$ is the entry of the Gram matrix defined in (2.2) for $i = 0, 1, 2, \ldots$, $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$. Then*

$$(c^i_{j,k})^2\, g^i_{k,k} - 2c^i_{j,k}\, g^i_{j,k} \le 0 \tag{2.14}$$

for $i = 0, 1, 2, \ldots$, $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$.

Proof. Combining (2.4) and the fact that $g^i_{k,k} = (a^i_k)^\top (a^i_k) \ge 0$ yields that the sign of the left-hand side of (2.14) is the same as the sign of

$$(c^i_{j,k})^2 - 2c^i_{j,k}\,\frac{g^i_{j,k}}{g^i_{k,k}} = (\mathrm{nint}(f^i_{j,k}))^2 - 2\,\mathrm{nint}(f^i_{j,k})\, f^i_{j,k}, \tag{2.15}$$

where

$$f^i_{j,k} = \frac{g^i_{j,k}}{g^i_{k,k}}. \tag{2.16}$$

The lemma follows from (2.12) with $r = f^i_{j,k}$. ∎

The following theorem states that projection using the coefficients defined in (2.4) never increases the Euclidean norm. This implies that the above algorithm never increases the Euclidean norm of any of the basis vectors at any time; the sum of the p-th powers of the Euclidean norms therefore converges monotonically to a local minimum, for every possible (positive) value of p simultaneously.

THEOREM 2.6. *Suppose that $c^i_{j,k}$ is the coefficient defined in (2.4) for $i = 0, 1, 2, \ldots$, $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$. Then the Euclidean norms satisfy*

$$\|a^i_j - c^i_{j,k}\, a^i_k\| \leq \|a^i_j\| \tag{2.17}$$

for $i = 0, 1, 2, \ldots$, $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$.

Proof. According to (2.2), the square of the right-hand side of (2.17) is

$$\|a^i_j\|^2 = g^i_{j,j}. \tag{2.18}$$

The square of the left-hand side of (2.17) is

$$\|a^i_j - c^i_{j,k}\, a^i_k\|^2 = g^i_{j,j} + (c^i_{j,k})^2\, g^i_{k,k} - 2c^i_{j,k}\, g^i_{j,k}. \tag{2.19}$$

Formula (2.14) shows that (2.19) is less than or equal to (2.18). ∎

3. Results and Discussion. This section presents the results of several numerical experiments.[2] Subsection 3.1 describes the figures. Subsection 3.2 constructs the examples whose numerical results the figures report. Subsection 3.3 provides details of the implementation and computer system used. Subsection 3.4 describes the tables for a dead-simple example included due to a reviewer's request. Subsection 3.5 discusses the empirical results.

3.1. Description of the figures. This subsection describes empirical results, illustrating the algorithms via Figures 1–5. Supplementary figures are available in Section SM2 of the Supplementary Materials.

The figures refer to the algorithm of [7] as "LLL." Subsection 3.3 details the especially efficient implementation used here. The figures refer to the algorithm of the present paper as "ours."

The figures report on two kinds of experiments. The first kind runs the LLL algorithm followed by ours, once in each of ten trials. Each of the ten trials permutes the basis vectors at random, with a different random permutation. (The algorithm of the present paper is invariant to the ordering of the basis vectors, whereas the LLL algorithm is sensitive to the ordering.) The points plotted in the figures are the means of the ten trials. So, for example, the times reported in Figure 1 are averages over ten trials. The plotted bars range from the associated minimum over the ten trials to the associated maximum (not displaying the standard deviation that error bars sometimes report). Figures 2 and 4 include such bars ranging from the minimum to the maximum of the ten trials. The figures refer to this single randomized permutation followed by LLL followed by ours as "run once." Figures 1, 2, and 4 concern this first kind of experiment.

The second kind of experiment runs the following sequence ten times in succession: (1) random permutation of the basis vectors followed by (2) LLL followed by (3) the algorithm of the present paper, with each of the first nine times feeding its output

[2] Permissively licensed open-source software that automatically reproduces all results reported in the present paper is available at https://github.com/facebookresearch/latticer

into the input of the next time. Algorithm 2 formalizes this series of ten three-step sequences explicitly. The figures refer to the sequence of ten as "run repeatedly." Figures 3 and 5 concern this second kind of experiment. The first steps (random permutation) affect the LLL algorithm, which is sensitive to the ordering of the basis vectors (while the algorithm of the present paper is invariant to the ordering). The points plotted in Figures 3 and 5 are the final results of the series of ten three-step sequences (whereas the points plotted in Figures 1, 2, and 4 are the means over ten independent trials).

Figure 1 displays the time taken per trial. Figures 2–5 report the reduction either in the Frobenius norm of the matrix of basis vectors or in the minimum of the Euclidean norms of the basis vectors. (The Frobenius norm is the square root of the sum of the squares of the entries of the matrix.) The figures report the reduction due to LLL run on its own, as well as the further reduction due to post-processing the output from LLL via the algorithm of the present paper. Therefore, the full reduction in norm is the product of the reported fractions remaining after reduction, as all runs polish the outputs of the LLL algorithm via ours. The fraction reduced by ours that is reported for the sequence run repeatedly pertains only to the very last run of ours, whereas the fraction reduced by LLL reported for the same sequence run repeatedly pertains to the entire series of ten runs of LLL followed by ours, aside from separating out the very last run of ours. The plots displaying the fraction reduced by the last run of ours that is reported for the sequence run repeatedly feature titles, "reduction in Frob. norm by ours after reps. of ours+LLL" and "reduction in min. norm by ours after reps. of ours+LLL," abbreviating "repetitions" to "reps." In all the figures, the plots on the right display the results of running the algorithm of Section 2 (that is, Algorithm 1) after having run either the LLL algorithm or all of Algorithm 2 except for the final run of the algorithm of Section 2 (which is Algorithm 1).

The horizontal axes report the dimension n of the matrix whose columns are the initial basis vectors being reduced (the matrix is $n \times n$). The figure captions' δ is that used in the so-called "Lovász criterion" of the LLL algorithm. The figure captions' p is the power used in (2.5) for the algorithm of the present paper. The figure captions' q is an integer characterizing the size of the entries of the initial basis vectors prior to reduction; the following subsection provides further details about q.

Section SM2 of the Supplementary Materials presents further figures, reporting results for different values of δ and p. The conclusions that can be drawn from the further figures appear to be broadly consistent with those presented in the present section.

3.2. Description of the examples. This subsection details the particular examples whose numerical results the figures of the previous subsection report.

To construct the initial basis vectors being reduced as related to a finite field whose order is a large prime integer q, the examples consider $q = 2^{13} - 1$ and $q = 2^{31} - 1$ (two well-known Mersenne primes). The matrix whose columns are the initial basis vectors for the figures' examples takes the form

$$A = \left(\begin{array}{c|c} R & q \cdot \mathrm{Id} \\ \hline \mathrm{Id} & 0 \end{array} \right), \tag{3.1}$$

where "Id" denotes the identity matrix, "0" denotes the matrix whose entries are all zeros, and R denotes the pseudorandom matrix whose entries are drawn independent and identically distributed from the uniform distribution over the integers $-(q-1)/2$, $-(q-3)/2$, $\ldots$, $(q-3)/2$, $(q-1)/2$. This special form of matrix effectively views

the entries of R modulo q, as unimodular integer linear transformations acting on A from the right can add any integer multiple of q to any entry of R. The captions to the figures give the values of q considered.

The dimension of the matrix R in (3.1) is $(2\ell) \times \ell$, where $\ell = 2, 4, 8, \ldots, 128$. Hence the dimension of the matrix A in (3.1) whose columns are the initial basis vectors is $n \times n$, with $n = 3\ell$, so that $n = 6, 12, 24, \ldots, 384$. The horizontal axes of the figures give the values of n associated with the corresponding points in the plots.

3.3. Implementation details. This subsection details the implementations (including significant optimizations) used in the reported results.

The implementation of the LLL algorithm of [7] used here is entirely in IEEE standard double-precision arithmetic, with acceleration via basic linear algebra subroutines (BLAS) and enhanced accuracy via re-orthogonalization. BLAS originates from [6], [2], and others, with implementation in Apple's Accelerate Framework of Xcode for MacOS Ventura 13.5 in the experiments reported here (and compiled using Clang LLVM with the highest optimization flag, `-O3`, set). All experiments ran on a 2020 MacBook Pro with a 2.3 GHz quad-core Intel Core i7 processor and 3.733 GHz LPDDR4X SDRAM. For the greatest portability, the implementation calls `random()` to generate random numbers; many compiler distributions implement `random()` via the BSD linear-congruential generator (such compilers include our implementation's defaults of MacOS Clang referenced by `gcc` under Xcode and GNU `gcc` under Linux). The results reported here pertain to this class of pseudorandom numbers.

Re-orthogonalization combats round-off errors, as reviewed by [8]. Namely, all orthogonalization gets repeated until only at most the last digit of any result changes, and then one additional round of orthogonalization occurs (just to be sure).

Furthermore, upon any swapping of basis vectors in the LLL algorithm, the implementation recomputes the swapped orthogonal basis vectors from scratch for numerical stability. Recomputing from scratch incurs extra computational expense, but no more than the computational cost incurred by the size reduction which accompanies every swap.

The combination of re-orthogonalization and recomputing from scratch swapped orthogonal basis vectors may not be sufficient to handle the hardest possible examples constructed purposefully to test resistance to round-off errors, but is sufficient to handle the random examples of interest in cryptography (since cryptosystems necessarily must generate bases at random in order to be secure). Methods that take advantage of floating-point arithmetic while also being able to tackle worst-case adversarial examples are overviewed by [13] and [14]. For the more limited scope of cryptographic applications, however, the enhanced accuracy of re-orthogonalization and recomputing from scratch swapped orthogonal basis vectors enables all computations to leverage BLAS for dramatic accelerations. The implementation of the algorithm of the present paper also takes advantage of this acceleration, primarily for calculating the initial Gram matrix (other steps of the algorithm benefit less from BLAS than LLL does, at least in the implementation reported here).

Figure 1 indicates that both LLL and the algorithm of the present paper have running-times proportional to roughly n^3, with LLL being over an order of magnitude slower. The cost per iteration is proportional to n^2, as discussed in Section 2; the numbers of iterations required appear to be proportional to n for the examples reported here.

3.4. Description of the tables. This subsection describes empirical results from a dead-simple example, illustrating the algorithms via Tables 1 and 2.

To keep everything as simple as possible, the tables report results from processing a single 500×500 matrix whose entries are independent and identically distributed pseudorandom numbers drawn uniformly from the integers 0, 1, 2, ..., $q - 1$, where q is the Mersenne prime specified in Tables 1 and 2. The processing for the tables applies either the LLL algorithm or ours (as indicated in the tables) directly to the matrix, not combining any of the algorithms. Table 1 reports the attained reduction in the Frobenius norm of the matrix; Table 2 reports the attained reduction in the minimum Euclidean norm of any column (that is, of any vector in the reduced basis). The tables also report the parameter δ used in the so-called "Lovász criterion" of the LLL algorithm; for this example, we set the parameter p used in our algorithm to $p = 2$. In all cases, the LLL algorithm reduces the norms significantly less than ours. Thus the LLL algorithm appears to be uniformly and significantly inferior to ours on this random example (however, this example may be too simplistic to be representative of applications to cryptography, despite fulfilling a request from a reviewer).

3.5. Discussion. This subsection discusses the numerical results reported in the figures and tables.

As mentioned in the last paragraph of Subsection 3.3, the cost per iteration of either the LLL algorithm or ours is proportional to n^2. Figure 1 indicates that both LLL and ours run in time proportional to roughly n^3 (with LLL being over an order of magnitude slower), in accord with the numbers of iterations taken to reach equilibrium (that is, a fixed point) being proportional to the dimension n in the experiments of the present section. (Figure SM1 in the Supplementary Materials shows the same, but for $\delta = 1 - 10^{-1}$ rather than $\delta = 1 - 10^{-15}$, where δ is the parameter in the so-called "Lovász criterion" of the LLL algorithm.)

Figures 2–5 indicate that both the LLL algorithm and ours reduce norms as a function of the dimension n similarly for different sizes of the entries of the basis vectors; specifically, changing $q = 2^{13} - 1$ to $q = 2^{31} - 1$ has little effect on how the results vary as a function of n. Repeated runs that could in principle help jump out of local minima appear little more effective than simply taking the best of several single runs (with each run randomizing the order of the basis vectors differently).

In all cases except for the simplistic example from Subsection 3.4, LLL reduces norms far more than ours. The algorithm of the present paper is suitable only for polishing the results of another algorithm (such as LLL); the algorithm of the present paper is very fast and often reduces norms beyond what other algorithms achieve, but is ineffective on its own for anything other than demonstrating that a given basis can be reduced further. Ours appears to be more useful for reducing the Frobenius norm than the minimum of the Euclidean norms of the basis vectors, though the algorithm does reduce both (or at least never increases either, as guaranteed by the theory of Subsection 2.2 and illustrated in all the figures and tables).

Possible applications include the following:

1. The algorithm of the present paper can often give a constructive proof that the result of another algorithm for lattice reduction is not optimal.
2. Some lattice-based cryptosystems can be compromised with reductions in the Euclidean norm of the shortest vector in the basis by $\frac{100}{n}$ percent, where $n = m$ is both the number and dimension of the basis vectors. For example, [12] mentions several such schemes. In conjunction with other methods, the algorithm of the present paper might help attack such cryptosystems.
3. Improvement via lattice reduction of the solution to Diophantine equations and other problems from number theory can be of theoretical interest.

TABLE 1
Frobenius norm of the matrix whose columns are the basis vectors resulting from reducing the columns of a 500×500 *matrix whose entries are independent and identically distributed pseudorandom numbers drawn uniformly from the integers* $0, 1, 2, \ldots, q-1$

δ	q	Frobenius norm before reduction	fraction remaining after running LLL	fraction remaining after running ours
$1-10^{-15}$	$2^{13}-1$	2.36E06	0.805	0.667
$1-10^{-15}$	$2^{31}-1$	6.19E11	0.822	0.667
$1-10^{-1}$	$2^{13}-1$	2.36E06	0.813	0.667
$1-10^{-1}$	$2^{31}-1$	6.19E11	0.815	0.667

TABLE 2
Minimum Euclidean norm of any of the basis vectors resulting from reducing the columns of a 500×500 *matrix whose entries are independent and identically distributed pseudorandom numbers drawn uniformly from the integers* $0, 1, 2, \ldots, q-1$

δ	q	min. Euclidean norm initially	fraction remaining after running LLL	fraction remaining after running ours
$1-10^{-15}$	$2^{13}-1$	1.00E05	0.675	0.658
$1-10^{-15}$	$2^{31}-1$	2.62E10	0.675	0.658
$1-10^{-1}$	$2^{13}-1$	1.00E05	0.689	0.658
$1-10^{-1}$	$2^{31}-1$	2.62E10	0.689	0.658

4. Recent work of [9] accelerated its preprocessing by a factor of 40 using the algorithm of the present paper interleaved with its earlier method for lattice reduction. Similar interleaving with the present paper's implementation of the LLL algorithm could enable further acceleration.

Section SM2 of the Supplementary Materials presents several more figures, with different settings of parameters; all further corroborate the results discussed in the present subsection. And, naturally, the algorithm and its analysis generalize to lattices formed from linear combinations of vectors whose entries are real numbers (not necessarily integers), with the coefficients in the linear combinations being integers. However, the case involving real numbers may pose challenges due to round-off errors.

The dual of a lattice of integers is in general such a lattice of real vectors. The least-possible maximum of the Euclidean norms of the vectors in a basis for the dual lattice yields bounds on the Euclidean norm of the shortest nonzero vector in the original lattice, courtesy of the transference of [1], as highlighted by [12] and others. The algorithm of the present paper can reduce the maximum of the Euclidean norms toward the least possible.

4. Conclusion. The present paper proposes a very simple and efficient scheme for lattice reduction. The scheme reduces the Euclidean norms of the basis vectors monotonically, as guaranteed by rigorous proofs. Fortunately, the algorithm of the present paper runs much faster than even a highly optimized implementation of the most classical baseline, the LLL algorithm of [7]. Unfortunately, the proposed algorithm reduces norms far less than LLL if not used in conjunction with a method such as LLL. The main use of the proposed scheme should therefore be to polish the outputs of another algorithm (such as LLL). On its own, the algorithm of the present paper tends to get stuck in rather shallow local minima, with the iterations reaching

Algorithm 1: Lattice reduction via the Gram matrix and projections

Input: Positive integers m and n, a positive real number p, and column vectors $a_1^0, a_2^0, \ldots, a_n^0$, each of size $m \times 1$.

Output: A positive integer i and column vectors $a_1^i, a_2^i, \ldots, a_n^i$, each $m \times 1$, such that the vectors are a unimodular integer linear transform of the inputs $a_1^0, a_2^0, \ldots, a_n^0$ and $\|a_k^i\| \le \|a_k^0\|$ for all $k = 1, 2, \ldots, n$.

Set $g_{j,k}^0 = (a_k^0)^\top (a_j^0)$ for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$.
Set $i = 0$.
repeat
  Set $c_{k,k}^i = 0$ for $k = 1, 2, \ldots, n$.
  Set $c_{j,k}^i = \mathrm{nint}(g_{j,k}^i / g_{k,k}^i)$ for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$ with $k \ne j$.
  Set $s_k^i = \sum_{j=1}^n \left(g_{j,j}^i + (c_{j,k}^i)^2 \, g_{k,k}^i - 2 c_{j,k}^i \, g_{j,k}^i \right)^{p/2}$ for $k = 1, 2, \ldots, n$.
  Set $\tilde{\imath} = \mathrm{argmin}_{1 \le k \le n} \, s_k^i$.
  Set $(a_k^{i+1})_j = (a_k^i)_j - c_{k,\tilde{\imath}}^i \, (a_{\tilde{\imath}}^i)_j$ for $j = 1, 2, \ldots, m$, and $k = 1, 2, \ldots, n$.
  Set $g_{j,k}^{i+1} = g_{j,k}^i + c_{j,\tilde{\imath}}^i \, c_{k,\tilde{\imath}}^i \, g_{\tilde{\imath},\tilde{\imath}}^i - c_{j,\tilde{\imath}}^i \, g_{\tilde{\imath},k}^i - c_{k,\tilde{\imath}}^i \, g_{j,\tilde{\imath}}^i$ for $j = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n$.
  Increment i by 1.
until $(a_k^i)_j = (a_k^{i-1})_j$ *for all* $j = 1, 2, \ldots, m$, *and all* $k = 1, 2, \ldots, n$.

Algorithm 2: Series of ten sequences, each having the same three steps: (1) random permutation followed by (2) the LLL algorithm of [7] followed by (3) Algorithm 1

Input: A positive integer n, a positive real number p, and column vectors $a_1^0, a_2^0, \ldots, a_n^0$, each of size $n \times 1$.

Output: Column vectors $a_1^{10}, a_2^{10}, \ldots, a_n^{10}$, each $n \times 1$, such that the vectors are a unimodular integer linear transformation of the input vectors $a_1^0, a_2^0, \ldots, a_n^0$.

for *i = 0, 1, 2, …, 9* **do**
  Construct a uniformly random permutation $\sigma_1^i, \sigma_2^i, \ldots, \sigma_n^i$ of the integers $1, 2, \ldots, n$ via the Fisher-Yates-Durstenfeld-Knuth shuffle described in Section 3.4.2 of [5].
  Set $b_k^i = a_{\sigma_k^i}^i$ for $k = 1, 2, \ldots, n$.
  Run the LLL algorithm of [7] (implemented as detailed in Subsection 3.3) on $b_1^i, b_2^i, \ldots, b_n^i$ to obtain vectors $b_1^{i+1}, b_2^{i+1}, \ldots, b_n^{i+1}$, each $n \times 1$, such that $b_1^{i+1}, b_2^{i+1}, \ldots, b_n^{i+1}$ are a unimodular integer linear transformation of the vectors $b_1^i, b_2^i, \ldots, b_n^i$.
  Run Algorithm 1 (with the given values of n and p and with $m = n$) on $b_1^{i+1}, b_2^{i+1}, \ldots, b_n^{i+1}$ to obtain vectors $a_1^{i+1}, a_2^{i+1}, \ldots, a_n^{i+1}$, each $n \times 1$, such that $a_1^{i+1}, a_2^{i+1}, \ldots, a_n^{i+1}$ are a unimodular integer linear transform of $b_1^{i+1}, b_2^{i+1}, \ldots, b_n^{i+1}$ and $\|a_k^{i+1}\| \le \|b_k^{i+1}\|$ for all $k = 1, 2, \ldots, n$.
end

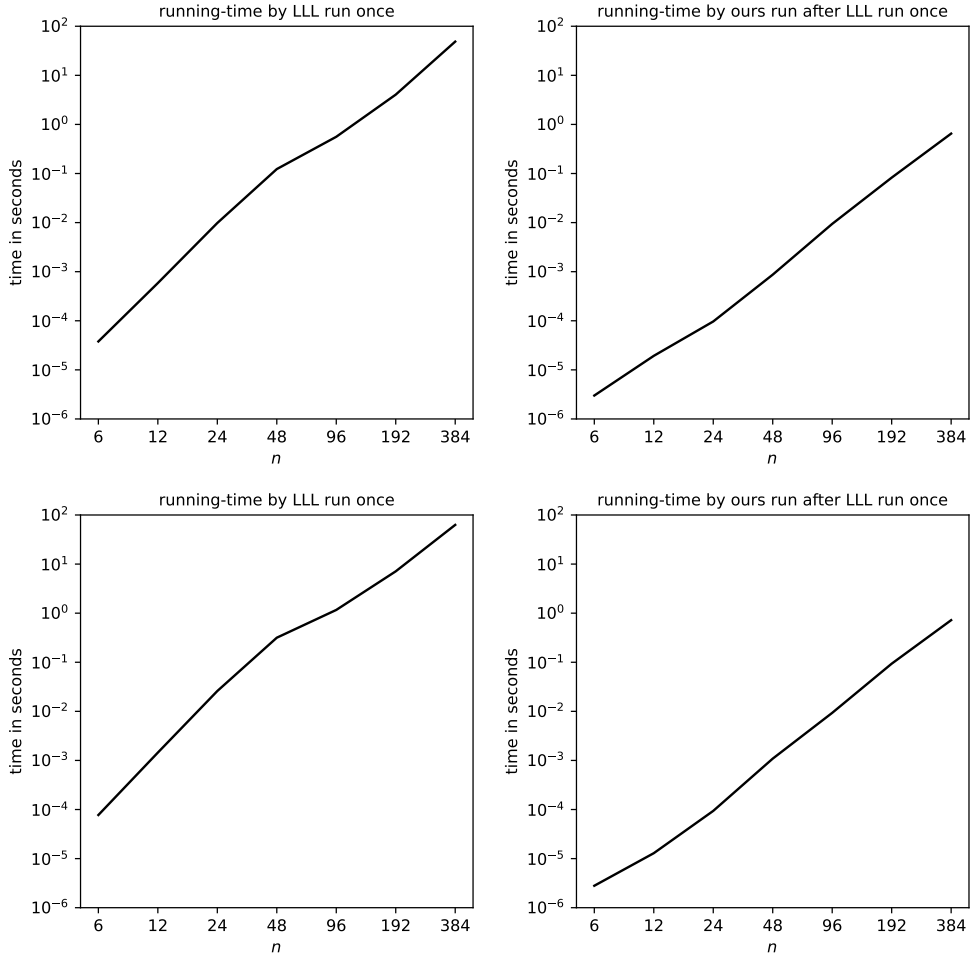

FIG. 1. $\delta = 1 - 10^{-15}$, $p = 2$; *the upper plots are for* $q = 2^{13} - 1$, *the lower plots are for* $q = 2^{31} - 1$

a fixed-point equilibrium that is far away from optimally minimizing the Euclidean norms of the basis vectors. Convergence is monotonic and hence guaranteed, but equilibrium tends to attain without reaching the global optimum. The algorithm of the present paper is extremely efficient computationally, however, so can be suitable for rapidly burnishing the outputs of other algorithms for lattice reduction.

Acknowledgements. We would like to thank our colleagues, Zeyuan Allen-Zhu, Kamalika Chaudhuri, Mingjie Chen, Evrard Garcelon, Matteo Pirotta, Jana Sotakova, and Emily Wenger. We would also like to thank the anonymous reviewers and editor.

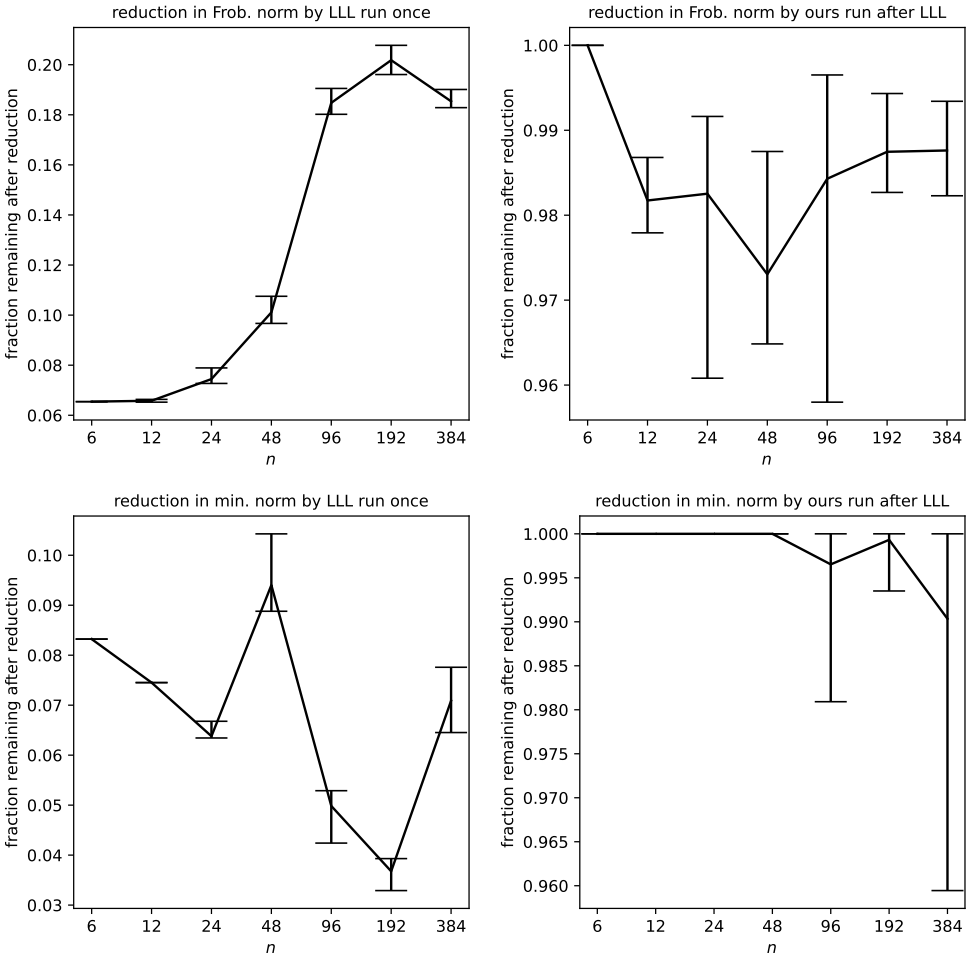

FIG. 2. $\delta = 1 - 10^{-15}$, $p = 2$, $q = 2^{13} - 1$

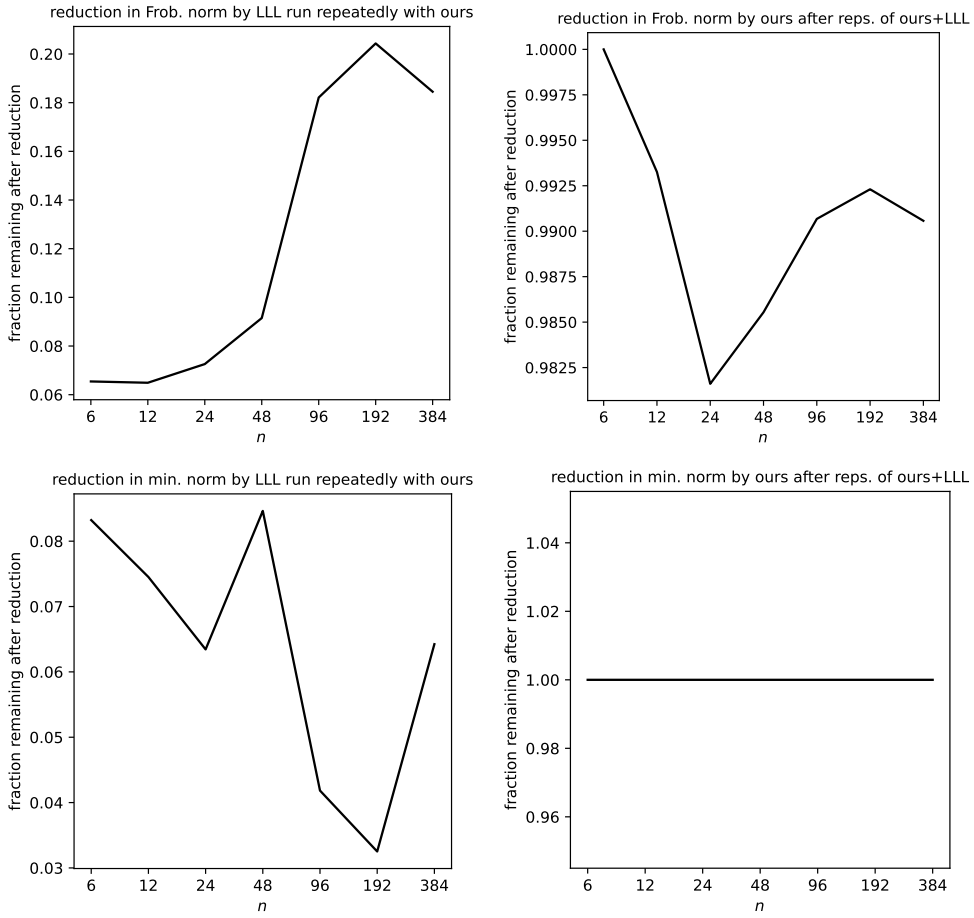

FIG. 3. $\delta = 1 - 10^{-15}$, $p = 2$, $q = 2^{13} - 1$

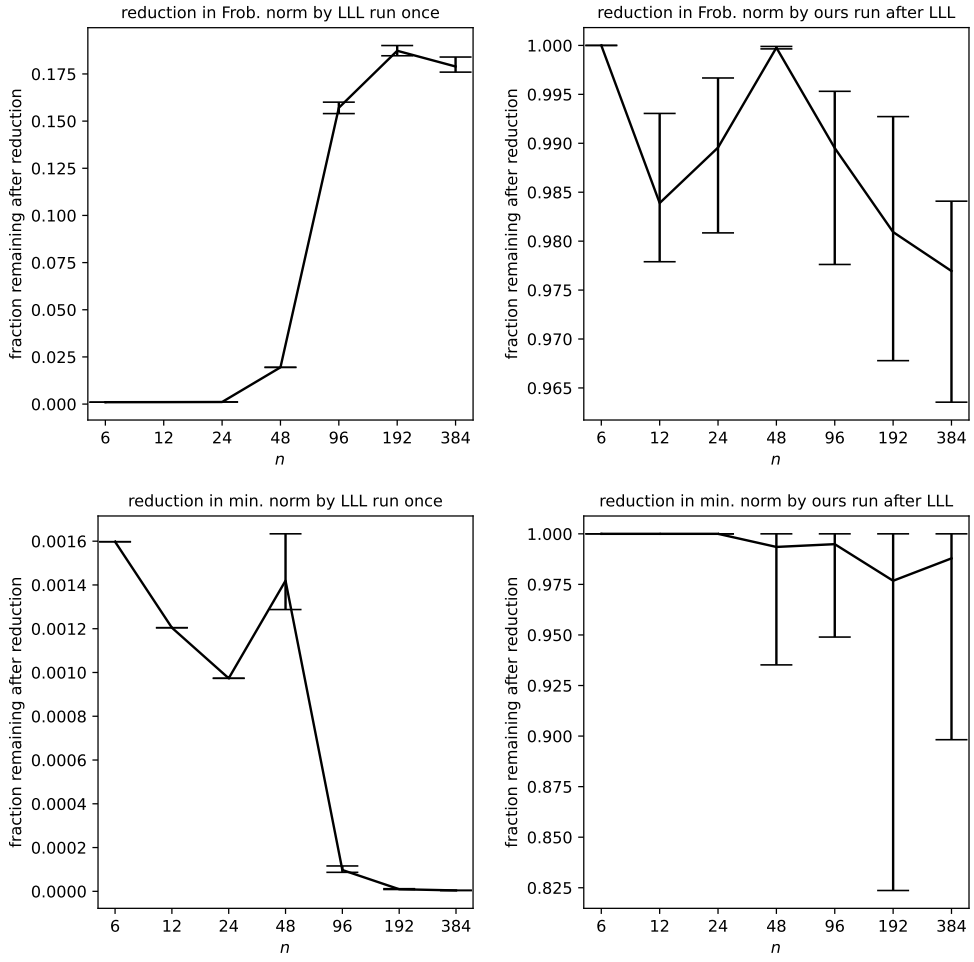

FIG. 4. $\delta = 1 - 10^{-15}$, $p = 2$, $q = 2^{31} - 1$

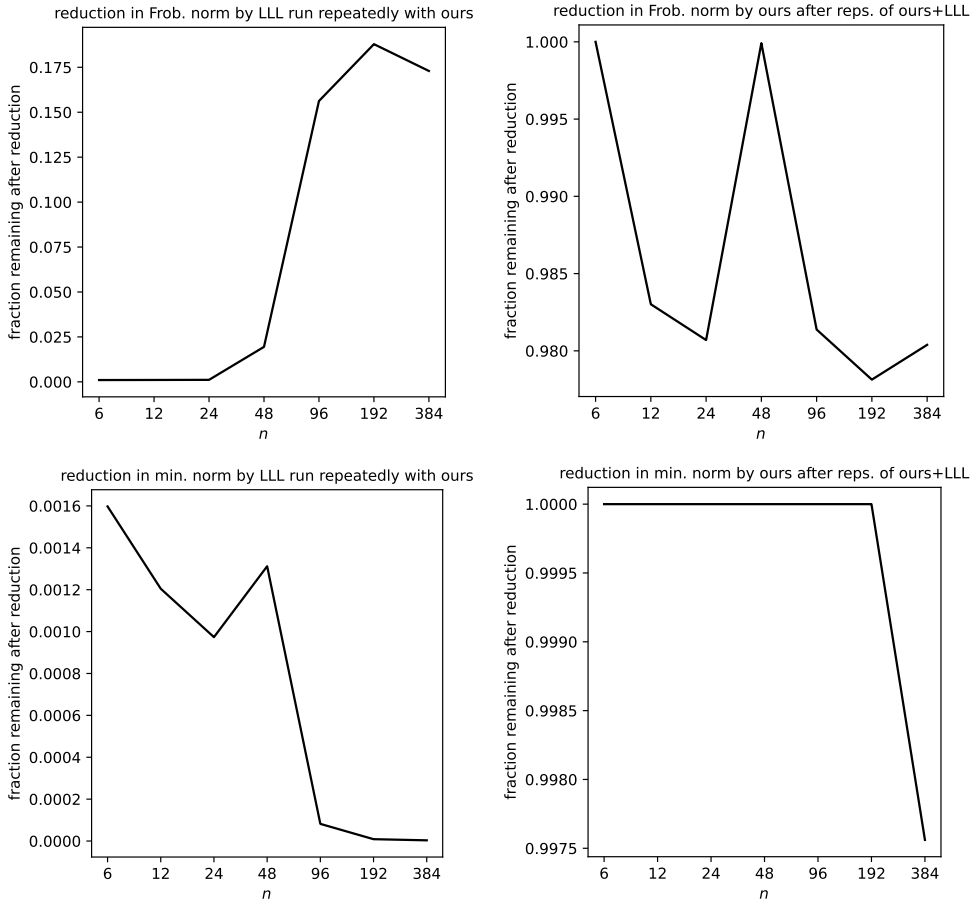

FIG. 5. $\delta = 1 - 10^{-15}$, $p = 2$, $q = 2^{31} - 1$

REFERENCES

[1] W. Banaszczyk, *New bounds in some transference theorems in the geometry of numbers*, Math. Ann., 296 (1993), pp. 625–635.

[2] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, *An updated set of basic linear algebra subprograms (BLAS)*, ACM Trans. Math. Softw., 28 (2002), pp. 135–151.

[3] M. R. Bremner, *Lattice Basis Reduction: An Introduction to the LLL Algorithm and Its Applications*, Pure and Applied Mathematics, Chapman & Hall, CRC Press, 2012.

[4] J. W. S. Cassels, *An Introduction to the Geometry of Numbers*, Classics in Mathematics, Springer, 1997.

[5] D. E. Knuth, *Art of Computer Programming*, vol. 2: Seminumerical Algorithms, Addison-Wesley, 3rd ed., 1998.

[6] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for FORTRAN usage*, ACM Trans. Math. Softw., 5 (1979), pp. 308–323.

[7] A. K. Lenstra, H. W. Lenstra, and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

[8] S. J. Leon, A. Björck, and W. Gander, *Gram-Schmidt orthogonalization: 100 years and more*, Numer. Lin. Algebra Appl., 20 (2013), pp. 492–532.

[9] C. Y. Li, J. Sotáková, E. Wenger, Z. Allen-Zhu, F. Charton, and K. Lauter, *SALSA VERDE: a machine learning attack on learning with errors with sparse small secrets*, Tech. Report 968, Cryptology ePrint Archive, 2023, https://eprint.iacr.org/2023/968.

[10] P. Q. Nguyen and B. Vallée, eds., *The LLL Algorithm: Survey and Applications*, Information Security and Cryptography, Springer, 2010.

[11] C. Peikart, *A decade of lattice cryptography*, Found. Trends Theor. Comput. Sci., 10 (2016), pp. 283–424.

[12] O. Regev, *On lattices, learning with errors, random linear codes, and cryptography*, J. ACM, 56 (2009), pp. 1–40.

[13] C. P. Schnorr and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Math. Program., 66 (1994), pp. 181–199.

[14] D. Stehlé, *Floating-point LLL: theoretical and practical aspects*, in The LLL Algorithm: Survey and Applications, P. Q. Nguyen and B. Vallée, eds., Springer, 2010, pp. 179–213.