

Computing a Lattice Basis Revisited

Jianwei Li, Phong Q. Nguyen

▶ **To cite this version:**

Jianwei Li, Phong Q. Nguyen. Computing a Lattice Basis Revisited. ISSAC '19: International Symposium on Symbolic and Algebraic Computation, Jul 2019, Beijing, China. pp.275-282, 10.1145/3326229.3326265 . hal-03067825

HAL Id: hal-03067825
https://inria.hal.science/hal-03067825

Submitted on 15 Dec 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Computing a Lattice Basis Revisited*

Jianwei Li† Phong Q. Nguyen‡

Abstract

Given $(a, b) \in \mathbb{Z}^2$, Euclid's algorithm outputs the generator $\gcd(a, b)$ of the ideal $a\mathbb{Z} + b\mathbb{Z}$. Computing a lattice basis is a high-dimensional generalization: given $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{Z}^m$, find a $\mathbb{Z}$-basis of the lattice $L = \{\sum_{i=1}^n x_i\mathbf{a}_i, x_i \in \mathbb{Z}\}$ generated by the $\mathbf{a}_i$'s. The fastest algorithms known are HNF algorithms, but are not adapted to all applications, such as when the output should not be much longer than the input. We present an algorithm which extracts such a short basis within the same time as an HNF, by reduction to HNF. We also present an HNF-less algorithm, which reduces to Euclid's extended algorithm and can be generalized to quadratic forms. Both algorithms can extend primitive sets into bases.

Keywords. Lattice algorithms; $\mathbb{Z}$-basis; HNF; XGCD; Integer quadratic forms

1 Introduction

Let $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{Z}^m$ be integer vectors. They generate the set $L = \{\sum_{i=1}^n x_i\mathbf{a}_i, x_i \in \mathbb{Z}\}$ formed by all integral linear combinations of the $\mathbf{a}_i$'s: this set is a subgroup of $\mathbb{Z}^m$, and is therefore an integer lattice. It follows that there are $d \leq \min(m, n)$ linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{Z}^m$ such that $L = \{\sum_{i=1}^d x_i\mathbf{b}_i, x_i \in \mathbb{Z}\}$: such vectors are called a basis of the lattice L. Computing such a $\mathbb{Z}$-basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ from $A = (\mathbf{a}_1, \ldots, \mathbf{a}_n)$ is a classical problem:

- It is a high-dimensional generalization of the gcd problem, for which $(m, n) = (1, 2)$.
- It can be generalized to quadratic forms: transform a given positive semi-definite integral quadratic form into an equivalent positive definite integral quadratic form (see [Coh93, p. 79]). Namely, given a positive semi-definite integral matrix $P \in \mathbb{Z}^{n\times n}$, find a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $UPU^t = \begin{pmatrix} 0_{(n-d)\times(n-d)} & 0_{(n-d)\times d} \\ 0_{d\times(n-d)} & Q \end{pmatrix}$, where $Q \in \mathbb{Z}^{d\times d}$ is positive definite. In particular, $\{\mathbf{x}P\mathbf{x}^t, \mathbf{x} \in \mathbb{Z}^n\} = \{\mathbf{y}Q\mathbf{y}^t, \mathbf{y} \in \mathbb{Z}^d\}$.

It is often used as a subroutine [Poh87, BP87, Ajt96, GPV08]. This is because a basis is required by many tasks, yet many lattices arising in applications are not given by an explicit basis: for instance, given bases of two lattices L_1 and L_2, one might be interested in computing a basis of $L_1 \cap L_2$, which can be reduced to the basis problem by duality [MG02].

It is well-known that computing a basis can be done in polynomial time. The fastest algorithms known do not run in time quasi-linear in the input size: instead, the time is polynomial in the dimension and quasi-linear w.r.t. the size of the entries of the input vectors. Until now, there has only been two kinds of "quasi-linear" algorithms:

- Canonical form algorithms such as Hermite normal form (HNF) and Smith normal form (SNF) algorithms. However, the HNF can be much bigger than the input generators: in fact, most HNFs have entries as large as the determinant (see [Maz11]). On the other hand, input generators might be arbitrarily smaller than the determinant. There is a similar issue with SNF algorithms (see App. A.2).
- Generalizations of LLL to linearly dependent vectors. LLL algorithms [LLL82] are the fastest lattice basis reduction algorithms: given any basis, they compute a so-called LLL-reduced basis [LLL82]. Most can be modified to take as input possibly linearly dependent vectors [Poh87], but are more expensive than HNF algorithms.

However, several applications of basis algorithms further require that the output basis is not much longer than the input generators, making HNF unsuitable, without needing the full power of LLL reduction.

- One example is when given a basis B of a lattice L and a primitive set $(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ of L, one would like to find a basis of L of the form $(\mathbf{p}_1, \ldots, \mathbf{p}_r, *, \ldots, *)$, with no basis vector after $\mathbf{p}_r$ much longer than those of B (see [Sie89, MvTW08]). A useful case is when $L = \mathbb{Z}^n$ and B is the canonical basis, which appears in blockwise lattice reduction algorithms [Sch87, GN08]. In such applications, one cannot tolerate a too large increase, because one needs to call this subroutine many times.
- Another example is the generation of random bases (crucial in pruned enumeration [GNR10]): although there is no natural distribution over bases, it is useful to generate bases in a random way within a prescribed length, which can be done by applying the basis algorithm to sufficiently many random lattice vectors with discrete Gaussian distribution (generated by a sampler [GPV08]). Here, the output basis only depends on the lattice, the Gaussian parameters and the randomness.

*A preliminary version of this work appeared in *Proceedings of ISSAC '19* as [LN19].

†Information Security Group, Royal Holloway, University of London. Email: lijianweithu@sina.com

‡Inria and DIENS, PSL. Email: Phong.Nguyen@inria.fr

In such applications, HNF algorithms do not work, and neither does a combination of HNF+LLL, because LLL's output may still be much longer than the input generators.

Our results. We present fast algorithms to compute short bases. To do so, we first introduce weak notions of reduction:

- We define a partial order on matrices such that if a basis B is lower than a generating set A, its Gram-Schmidt orthogonalization (GSO) B^* is never longer than that of A, which is useful in complexity reductions among lattice problems [CN97, MG02] and in lattice-based cryptographic constructions [GPV08].
- This partial order is not sufficient to force the output B not to be much longer than the input A. To achieve this, it can be strengthened by LLL's so-called size-reduction, in which case $\|B\| \leq \sqrt{d} \times \|A\|$ where d is the rank of B and $\|\cdot\|$ denotes the maximal Euclidean norm of the matrix vectors. We introduce a cheaper but weaker strengthening called *loose reduction*, which guarantees that $\|B\| \leq d \times \|A\|$.

Our first basis algorithm is obtained by adapting and improving the Cai-Nerurkar/ Micciancio-Goldwasser (CNMG) algorithm [CN97, MG02]: this is an algorithmic version of an elementary result proved by Siegel [Sie89, Theorem 18], stating that n linearly independent vectors of an n-rank lattice L can be linearly transformed into a basis of L, using a triangular transformation with diagonal coefficients less than 1 in absolute value. A simple modification leads to a short basis algorithm requiring two HNF computations, but also relying on LLL's size-reduction, therefore runnning in time $O(\max\{m,n\}^{4+\varepsilon} \log^{1+\varepsilon} \|A\|)$ for any $\varepsilon > 0$, where $A \in \mathbb{Z}^{n\times m}$ denotes the input generators. We present a faster variant, which computes a loosely-reduced basis in time $O(\max\{m,n\}^{\theta+1+\varepsilon} \log^{1+\varepsilon} \|A\|)$ where $\theta < 2.3728639$ [Gal14] is the matrix multiplication exponent, *i.e.* as fast as computing A's GSO [Sto96a, Cor. 10] or HNF [Sto00, Chap. 6].

Historically, lattice algorithms were introduced in terms of quadratic forms. However, HNF-based algorithms usually cannot be generalized to quadratic forms because there is no HNF analogue for integral quadratic forms. Thus, we also present a direct "quasi-linear" basis algorithm which does not rely on the HNF, but has the same guarantees on the quality of the output basis: it has a worse running time, but is a simple generalization of Euclid's extended algorithm (XGCD). Unlike the first algorithm, it only uses the inner products of the input vectors, and can therefore be generalized to quadratic forms: it transforms a positive semi-definite integral quadratic form into an equivalent positive definite integral quadratic form. Generalizations of LLL [Poh87] can also perform the latter task, but a proof that the corresponding unimodular transformation can be computed in polynomial time seems to be missing in the literature.

Both algorithms are reasonably simple to implement and do not require floating-point arithmetic.

Table 1 summarizes the worst-case bit-complexity of the fastest basis algorithms, with or without fast integer arithmetic and/or fast matrix multiplication. Without fast integer arithmetic and without fast matrix multiplication, the worst-case complexity of our first algorithm is essentially $O(mnd^3 \log^2 \|A\|)$, which is that of size reduction.

Table 1: Comparison of quasi-linear basis algorithms: $A \in \mathbb{Z}^{n\times m}$ is the input of rank d and B is the output basis; $\det(L(A))$ denotes the determinant of the lattice generated by A; a pair (x, y) means that the complexity is $O(\max\{m,n\}^x \log^y \|A\|)$.

Algorithm	Time complexity with/without fast integer arithmetic		$\|B^*\| \leq \|A^*\|$	Upper bound on $\|B\|$
	With	Without		
Storjohann HNF algorithm [Sto00]	$(\theta+1+\varepsilon, 1+\varepsilon)$	$(4+\varepsilon, 2)$	No	$\det(L(A))$
Storjohann SNF algorithm [Sto00]	$(\theta+1+\varepsilon, 1+\varepsilon)$	$(4+\varepsilon, 2)$	No	$nd^{4d+5} \times \|A\|^{4d+2}$
LLL-variant [HPS11] with modification	$(8+\varepsilon, 1+\varepsilon)$	$(9+\varepsilon, 2+\varepsilon)$	Yes	$\sqrt{d} \times \|A\|$
$\widetilde{L^1}$ [NSV11] with floating-point arithmetic	$(5+\varepsilon, 1+\varepsilon)$	$(6+\varepsilon, 2+\varepsilon)$	No	$\xi^d \times \|A\|$ $(\xi > 1)$
First HNF [Sto00] then NS-algorithm [NS16]	$(5+\varepsilon, 1+\varepsilon)$	$(7+\varepsilon, 2+\varepsilon)$	No	$\sqrt{d} \times \det(L(A))$
Modified CNMG algorithm [CN97, MG02]	$(4+\varepsilon, 1+\varepsilon)$	$(5, 2)$	Yes	$\sqrt{d} \times \|A\|$
Our XGCD-based basis algorithm	$(5+\varepsilon, 1+\varepsilon)$	$(6, 2)$	Yes	$\sqrt{d} \times \|A\|$
Our HNF-based basis algorithm	$(\theta+1+\varepsilon, 1+\varepsilon)$	$(\max\{\theta+2, 4+\varepsilon\}, 2)$	Yes	$d \times \|A\|$

Overview. Let $A = (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \mathbb{Z}^{n\times m}$ of rank d be the input generators to both basis algorithms. We use row-representation of both vectors and matrices.

Our first algorithm is based on identifying a special class of short bases ("lower" than A) for lattices of the form $L(A) = \mathbb{Z}^n A$ and $\overline{L(A)} = \mathrm{span}(L(A)) \cap \mathbb{Z}^m$. These bases are of the form CA, where C is the dual basis of a well-chosen HNF. Because of the HNF structure, the integer matrix CA can be efficiently computed without explicitly computing the rational matrix C, and further loosely reduced. We thus obtain short bases of $L(A)$ and $\overline{L(A)}$ in the same time as HNF. For $\overline{L(A)}$, a single HNF suffices but for $L(A)$ itself, we need two, because the required HNF cannot be computed directly from A.

Our second algorithm is a "quasi-linear" variant of the Li-Nguyen basis algorithm [LN14, Alg. B.1]. It is an incremental algorithm, which reduces the problem to the setting where all input vectors (barring the last one) are linearly independent. Then the projections of the last two vectors orthogonally to the first $n-2$ vectors generate a one-rank lattice, whose basis can be found by Euclid's XGCD algorithm. By repeating this process quadratically many times, one finally obtains a basis. We use LLL's size reduction to guarantee that the final basis is "short" and that all intermediate entries remain polynomially bounded.

ROADMAP. Sect. 2 recalls background on lattices. Sect. 3 presents our HNF-based basis algorithm, and clarifies its relation with the CNMG algorithm. Sect. 4 presents our XGCD-based direct algorithm and its quadratic form variant. App. A revisits previous basis algorithms. App. B-E provide missing proofs. App. F describes an application on matrix triangularization.

2 Background

We use row-representation of both vectors and matrices throughout this paper: bold lower case letters and upper case letters denote row vectors and matrices, respectively. The ring of $n \times m$ matrices with coefficients in the ring $\mathbb{A}$ is denoted by $\mathbb{A}^{n\times m}$, and we identify $\mathbb{A}^m$ with $\mathbb{A}^{1\times m}$. The $n \times n$ identity matrix and $s \times t$ zero matrix are denoted by I_n and $0_{s\times t}$, respectively. For a matrix $A = (a_{i,j}) = (\mathbf{a}_1, \ldots, \mathbf{a}_n)$ of n rows, we let $\|A\|_\infty = \max_{i,j} |a_{i,j}|$ and $\|A\| = \max\{\|\mathbf{a}_1\|, \ldots, \|\mathbf{a}_n\|\}$, where $\|\cdot\|$ is the Euclidean norm. We denote the set $\mathbb{Z}^n A \triangleq \mathbb{Z}\mathbf{a}_1 + \cdots + \mathbb{Z}\mathbf{a}_n$ by $L(A)$ or $L(\mathbf{a}_1, \ldots, \mathbf{a}_n)$. The size of an object is the length of its binary representation. The notation $\log(\cdot)$ stands for the base 2. Let $\lceil\cdot\rfloor$ denote the nearest integer rounding such that $\lceil x\rfloor = x - \frac{1}{2}$ if $x \in \mathbb{Z} + \frac{1}{2}$.

Complexity model. Let $\mathcal{M}(t)$ denote the number of bit operations required to multiply two t-bit integers. Following [HM91, SL96], let $\mathcal{B}(t)$ bound the number of bit operations both to execute Euclid's extended algorithm (abbreviated XGCD) on two t-bit integers and to apply Chinese Remainder Theorem with all input integers including moduli less than t-bit.

With *fast integer arithmetic*, $\mathcal{M}(t) = O(t(\log t)\log\log t)$ [HvdH19] and $\mathcal{B}(t) = O(\mathcal{M}(t)\log t)$ [AHU74]. Without fast integer arithmetic, $\mathcal{M}(t) = O(t^2)$ and $\mathcal{B}(t) = O(t^2)$.

Let θ denote the matrix multiplication exponent over rings. Standard matrix multiplication has $\theta = 3$, while fast matrix multiplication allows $\theta < 2.3728639$ [Gal14].

2.1 Lattices

A *lattice* is a discrete subgroup of $\mathbb{R}^n$, whose inner product is $\langle\cdot,\cdot\rangle$.

Span and orthogonality. We let $\mathrm{span}(\cdot)$ denote the linear span of a set or row vectors in parentheses. If L is a lattice, its rank $\mathrm{rank}(L)$ is the dimension of $\mathrm{span}(L)$. For any subspace S of $\mathbb{R}^n$, its *orthogonal complement* is the subspace $S^\perp = \{\mathbf{x} \in \mathbb{R}^n : \langle\mathbf{x}, \mathbf{y}\rangle = 0 \text{ for all } \mathbf{y} \in S\}$. Then $(S^\perp)^\perp = S$.

Sublattices and primitive set. Let Λ and L be lattices in $\mathbb{R}^n$. Λ is a *sublattice* of L if $\Lambda \subseteq L$. A sublattice Λ is *full-rank* if $\mathrm{rank}(\Lambda) = \mathrm{rank}(L)$. It is *pure* if $\Lambda = L \bigcap \mathrm{span}(\Lambda)$, or equivalently, any basis of Λ can be extended to a basis of L. Linearly independent vectors $\mathbf{p}_1, \ldots, \mathbf{p}_r \in L$ form a *primitive set* for L if $L(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ is a pure sublattice of L. In particular, $\mathbf{p}_1, \ldots, \mathbf{p}_r \in \mathbb{Z}^n$ form a primitive set for $\mathbb{Z}^n$ iff they can be extended to a unimodular matrix.

Special lattices. For any set $S \subseteq \mathbb{R}^n$, we define the integer lattice spanned by S as $\overline{S} = \mathbb{Z}^n \bigcap \mathrm{span}(S)$.

For any lattice L in $\mathbb{Z}^n$, its *orthogonal lattice* is $L^\perp = \{\mathbf{x} \in \mathbb{Z}^n : \langle\mathbf{x}, \mathbf{y}\rangle = 0 \text{ for all } \mathbf{y} \in L\} = \mathbb{Z}^n \bigcap \mathrm{span}(L)^\perp$. It is an $(n - \mathrm{rank}(L))$-rank pure sublattice of $\mathbb{Z}^n$. Note that $L \subseteq (L^\perp)^\perp = \overline{L}$. For any $n \times m$ real matrix A, its *kernel lattice* is $\ker_\mathbb{Z}(A) = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x}A = \mathbf{0}\} = \mathbb{Z}^n \bigcap \mathrm{span}(A^t)^\perp$. It is a pure sublattice of $\mathbb{Z}^n$ and $\ker_\mathbb{Z}(A)^\perp = \mathbb{Z}^n \bigcap \mathrm{span}(A^t) = \overline{L(A^t)}$.

Lemma 2.1. *Let A be an $n \times m$ real matrix of rank d. Then the set $L(A)$ is a lattice iff the kernel lattice $\ker_\mathbb{Z}(A)$ has rank $(n - d)$.*

Proof. Note that $L(A) \simeq \mathbb{Z}^n/\ker_\mathbb{Z}(A)$ as groups. If $L(A)$ is a lattice, its rank is d, therefore the rank of $\ker_\mathbb{Z}(A)$ is $n - d$. Reciprocally, if the pure sublattice $\ker_\mathbb{Z}(A)$ of $\mathbb{Z}^n$ has rank $n - d$, there is a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of $\mathbb{Z}^n$ such that $(\mathbf{b}_1, \ldots, \mathbf{b}_{n-d})$ is a basis of $\ker_\mathbb{Z}(A)$. A has rank d, so $C = BA$ too: the first $n - d$ rows of C are zero, so its last d rows $\mathbf{c}_{n-d+1}, \ldots, \mathbf{c}_n$ are linearly independent. Hence, $L(A) = L(\mathbf{c}_{n-d+1}, \ldots, \mathbf{c}_n)$ is a lattice. □

Duality. For any lattice L, its *dual lattice* is $L^\times = \{\mathbf{x} \in \mathrm{span}(L) : \langle\mathbf{x}, \mathbf{y}\rangle \in \mathbb{Z} \text{ for all } \mathbf{y} \in L\}$. If B is a basis of L, then $L^\times$ has basis $B^\times \triangleq (BB^t)^{-1}B$, called the *dual basis* of B.

Lemma 2.2 (Integral dual)**.** *Let $B \in \mathbb{Z}^{d\times m}$ be a lattice basis and $W = (\mathbf{w}_1, \ldots, \mathbf{w}_d) = \det(BB^t)B^\times$. Then W is a $d \times m$ integer matrix of rank d such that $\|\mathbf{w}_i\| \le 2^{i-1} \times \|B\|^{2d-1}$ for $i = 1, \ldots, d$. Moreover, W can be computed deterministically in $O(md^{\theta-1} \cdot \mathcal{B}(d\log\|B\|))$ bit operations with operands of size $O(d\log\|B\|)$.*

HNF. An $n \times m$ integer matrix $H = (h_{i,j})$ of rank d is in *row-Hermite normal form* (abbr. HNF) if

- there exist indices $1 \le i_1 < i_2 < \cdots < i_d \le m$ s.t. for $j = 1, \ldots, d$, $h_{j,i_j} > 0$, $h_{j,k} = 0$ if $k < i_j$ and $0 \le h_{\ell,i_j} < h_{j,i_j}$ if $\ell < j$;
- the bottom $n - d$ rows of H are zero.

For any $n \times m$ integer matrix A, there is an $n \times n$ unimodular matrix U such that UA is in HNF; UA is called the HNF of A. The nonzero rows of UA form the *HNF-basis* of the lattice $L(A)$, which is uniquely determined by $L(A)$. If H is the HNF-basis of an integer lattice L, then $\|H\| \le \det(L)$.

Rounding matrices. We call t-matrix any lower-triangular matrix $T = (t_{i,j}) \in \mathbb{R}^{d\times d}$ s.t. $0 < |t_{i,i}| \le 1$ for $i = 1, \ldots, d$. We round such a matrix as follows: $\mathsf{Round}(T) = (r_{i,j}) \in \mathbb{Z}^{d\times d}$ such that for all $1 \le i, j \le d$, $r_{i,j} = 0$ if $i = j$ and $r_{i,j} = \lceil t_{i,j}\rfloor$ otherwise. Then $\mathsf{Round}(T)$ is a strictly lower triangular matrix s.t. $\|T - \mathsf{Round}(T)\|_\infty \le 1$.

Integral quadratic forms. Two integral quadratic forms f and g represented by symmetric matrices $P \in \mathbb{Z}^{n\times n}$ and $Q \in \mathbb{Z}^{d\times d}$ for $n \geq d$ are *equivalent* if there exists a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $UPU^t = \begin{pmatrix} 0_{(n-d)\times(n-d)} & 0_{(n-d)\times d} \\ 0_{d\times(n-d)} & Q \end{pmatrix}$.

2.2 Gram-Schmidt orthogonalization

Let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{R}^{n\times m}$. Lattice algorithms usually consider the orthogonal projections $\pi_i : \mathbb{R}^m \to \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^\perp$ for $i = 1, \ldots, n$. The *Gram-Schmidt orthogonalization* (GSO) of B is $B^* = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ where $\mathbf{b}_i^* = \pi_i(\mathbf{b}_i)$. Then: $\mathbf{b}_1^* = \mathbf{b}_1$ and $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$ for $i = 2, \ldots, n$, where $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$ if $\mathbf{b}_j^* \neq \mathbf{0}$ and 0 otherwise. For completeness, let $\mu_{i,i} = 1$ and $\mu_{i,j} = 0$ for $i < j$.

We call *fingerprint* of B the binary string $\varepsilon_1\varepsilon_2\cdots\varepsilon_n \in \{0,1\}^n$ which we write as $(\varepsilon_1\varepsilon_2\cdots\varepsilon_n)_{\mathcal{F}}$, where $\varepsilon_i = 1$ iff $\mathbf{b}_i^* \neq 0$. If d is the rank of B, there are exactly d indices $i_1 < i_2 < \cdots < i_d$ such that $\varepsilon_{i_j} \neq 0$. Then $\underline{B} = (\mathbf{b}_{i_1}, \mathbf{b}_{i_2}, \ldots, \mathbf{b}_{i_d})$ is the *support* of B. The $\mu_{i,j}$'s and $\|\mathbf{b}_i^*\|^2$'s actually only depend on the Gram matrix $\mathrm{Gram}(B) \triangleq BB^t$, and thus define the GSO of a positive semi-definite symmetric matrix P, giving a decomposition $P = \mu\Delta\mu^t$, where $\mu = (\mu_{i,j})_{1\leq i,j\leq n}$ is unit lower triangular and $\Delta = \mathrm{Diag}(\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2)$ is non-negative diagonal. Accordingly, we can define the fingerprint of P.

Integral GSO. If $\mathrm{Gram}(B)$ is integral, then the $\mu_{i,j}$'s and $\|\mathbf{b}_i^*\|^2$'s are rational by [LLL82, (1.27) and (1.29)]. To avoid rational arithmetic, one uses the so-called integral GSO [dW87, Coh93]. Let $d_0 = 1$ and:

$$d_i = \begin{cases} d_{i-1}\|\mathbf{b}_i^*\|^2 & \text{if } \mathbf{b}_i^* \neq \mathbf{0}, \\ d_{i-1} & \text{if } \mathbf{b}_i^* = \mathbf{0}, \end{cases} \qquad \lambda_{i,j} = d_j\mu_{i,j} \text{ for } i = 1, \ldots, n \text{ and } j = 1, \ldots, i-1.$$

Let $L_i = L(\varepsilon_1\mathbf{b}_1, \ldots, \varepsilon_i\mathbf{b}_i)$ for $i = 1, \ldots, n$, then $d_i = 1$ if $L_i = \{\mathbf{0}\}$ and $d_i = \prod_{1\leq j\leq i, \varepsilon_j=1} \|\mathbf{b}_j^*\|^2 = \det(L_i)^2 \in \mathbb{Z}$ otherwise. For $1 \leq j < i \leq n$, we have $\lambda_{i,j} = \langle \mathbf{b}_i, d_{j-1}\mathbf{b}_j^* \rangle \in \mathbb{Z}$, since $d_{j-1}\mathbf{b}_j^* \in L_j$ (see [LLL82, (1.28)] and [Coh93, §2.6.3]).

The integral GSO and fingerprint of a positive semi-definite integral quadratic form can be computed using Algorithm 9 in Appendix A.1.

Size reduction. B is *size-reduced* if $|\mu_{i,j}| \leq \frac{1}{2}$ for all $1 \leq j < i \leq n$. The single vector $\mathbf{b}_i$ is *size-reduced* (w.r.t. B) if $|\mu_{i,j}| \leq \frac{1}{2}$ for $1 \leq j < i$. Then $\|\mathbf{b}_i\| \leq \sqrt{i} \times \|B^*\|$. Clearly, this definition can be extended to a positive semi-definite quadratic form. It is known that one can size reduce an integer matrix $B \in \mathbb{Z}^{n\times m}$ in time $O(mn^{3+\varepsilon}\log^{1+\varepsilon}\|B\|)$.

Partial order. It will be convenient to compare two integer matrices $A = (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \mathbb{Z}^{n\times m}$ and $B = (\mathbf{b}_1, \ldots, \mathbf{b}_{n'}) \in \mathbb{Z}^{n'\times m}$ with the same number of columns m and rank d: often, B is a transformation of A based on elementary operations. We say that B is *lower than* A (abbreviated $B \preceq A$) if the following conditions hold:

1. Shifted supports: $n' - j_k \leq n - i_k$ for all $k = 1, \ldots, d$, where $\underline{A} = (\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \ldots, \mathbf{a}_{i_d})$ and $\underline{B} = (\mathbf{b}_{j_1}, \mathbf{b}_{j_2}, \ldots, \mathbf{b}_{j_d})$,
2. Sublattice inclusion: $L(A) \subseteq L(B)$ and $L(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{i_k}) \subseteq L(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{j_k})$ for $k = 1, \ldots, d$;
3. Distance decrease: $\|\mathbf{b}_{j_k}^*\| \leq \|\mathbf{a}_{i_k}^*\|$ for $k = 1, \ldots, d$.

The partial order is clearly reflexive ($B \preceq B$) and transitive: if $C \preceq B$ and $B \preceq A$, then $C \preceq A$. If $B \preceq A$, then obviously $\|B^*\| \leq \|A^*\|$.

Sect. 3 will use the following properties of the HNF:

Lemma 2.3. *Let L be an integer lattice with HNF-basis $H \in \mathbb{Z}^{d\times n}$ and $A \in \mathbb{Z}^{n\times m}$ be an integer matrix such that $L(A^t)$ is a full-rank sublattice of L. Then*

1. *$H^\times A \in \mathbb{Z}^{d\times m}$ has rank d and $A = H^t(H^\times A)$;*
2. *A and H^t have the same fingerprint;*
3. *$\underline{H^t} \in \mathbb{Z}^{d\times d}$ is lower-triangular with strictly positive diagonal s.t. $H^\times A = (\underline{H^t})^{-1}\,\underline{A}$ and $H^\times A \preceq A$.*
4. *$\|(\underline{H^t})^{-1}\,\underline{A}\|_\infty \leq 2^{d-1} \times \|\underline{A}\|_\infty$ and $\|\underline{H^t}\|_\infty \leq \|\underline{A}\|^d$.*

3 Fast HNF-based basis algorithms

We show how to efficiently reduce the basis problem to two well-chosen HNF computations: given as input an $n \times m$ integer matrix A of rank d, one can output a basis B of $L(A)$ within $O(mn(d^{\theta-2} + \log d) \cdot \mathcal{B}(d(\log n + \log\|A\|)))$ bit operations (*i.e.* $O(mnd^{\theta-1+\varepsilon}(\log n + \log\|A\|)^{1+\varepsilon})$ with fast integer arithmetic) such that $\|B^*\| \leq \|A^*\|$ and $\|B\| \leq d \times \|A\|$.

This algorithm is an optimization of the algorithmic version of Siegel's basis result [Sie89], namely the Cai-Nerurkar/Micciancio-Goldwasser algorithm [CN97, MG02]. We observe in Sect. 3.1 that one can transform it into a basis algorithm running in $O(mnd(\frac{\log d\|A\|}{\log\|A\|}) \cdot \mathcal{B}(d\log\|A\|))$ bit operations: it requires two HNF computations, solving two triangular systems, two GSO and a full size-reduction. We speed up this basis algorithm using two ideas:

- Loose reduction (Sect. 3.2), a cheaper variant of size-reduction, tailored to basis computations and not relying on GSO: roughly speaking, it replaces Babai's nearest plane algorithm [Bab86] by Babai's rounding-off.
- Th. 3.5, which provides special bases in lattices of the form $L(A)$ and $\overline{L(A)}$: these bases are lower than A, and are of the form CA where $C^\times$ is some well-chosen HNF.

Sect. 3.4 is devoted to our HNF-based basis algorithm, which finds a loosely-reduced basis of $L(A)$ using its HNF and duality. An extended version in Sect. 3.5 recovers a unimodular transformation.

3.1 The CNMG algorithm

Cai-Nerurkar [CN97] and Micciancio-Goldwasser [MG02] independently showed that any set of linearly independent lattice vectors can be efficiently transformed into a "short" basis:

Theorem 3.1 ([CN97, Lemma 3] and [MG02, Lemma 7.1])**.** *There is a poly-time algorithm which given as input a basis B_0 of a d-rank lattice L and a linear independent set $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_d\} \subseteq L$ s.t. $\|\mathbf{s}_1\| \leq \|\mathbf{s}_2\| \leq \cdots \leq \|\mathbf{s}_d\|$, outputs a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ of L such that for $i = 1, \ldots, d$, $\|\mathbf{b}_i^*\| \leq \|\mathbf{s}_i^*\|$, $\|\mathbf{b}_i\| \leq \sqrt{i} \times \|\mathbf{s}_i\|$ and $span(\mathbf{b}_1, \ldots, \mathbf{b}_i) = span(\mathbf{s}_1, \ldots, \mathbf{s}_i)$.*

Proof. We recall the algorithmic proof in [CN97, MG02]. There is a unique nonsingular matrix $M \in \mathbb{Z}^{d \times d}$ such that $S = MB_0$. Let $U \in \mathbb{Z}^{d \times d}$ be a unimodular matrix such that $T = UM^{-1}$ is a t-matrix. Then $C = UB_0 = (\mathbf{c}_1, \ldots, \mathbf{c}_d)$ is a basis of L such that $C = TS$, $\|\mathbf{c}_i^*\| \leq \|\mathbf{s}_i^*\|$ and $\text{span}(\mathbf{c}_1, \ldots, \mathbf{c}_i) = \text{span}(\mathbf{s}_1, \ldots, \mathbf{s}_i)$ for $i = 1, \ldots, d$. C can be size-reduced into another basis $B = VC$ of L where $V \in \mathbb{Z}^{d \times d}$ is unit lower triangular. □

An earlier version of Th. 3.1 appeared in [Ajt96, Lemma 1]. We observe that a simple modification of the proof implies a basis algorithm, namely Alg. 1:

Algorithm 1 A CNMG-based basis algorithm

Input: A matrix $A \in \mathbb{Z}^{n \times m}$ whose rows might be linearly dependent.
Output: A size-reduced basis of the lattice $L(A)$ generated by the rows of A.

1: Find the support $S \in \mathbb{Z}^{d \times m}$ of A using its GSO
2: Compute the HNF-basis $E \in \mathbb{Z}^{d \times m}$ of $L(A)$ by applying Storjohann's HNF Algorithm [Sto00, Chap. 6] to A
3: Find a nonsingular matrix $M \in \mathbb{Z}^{d \times d}$ s.t. $S = ME$ //Since $\underline{S^t} = \underline{E^t} M^t$, $\det(M)$ divides $\det(\underline{S^t})$
4: Compute the HNF F of M^t using Storjohann-Labahn's HNF algorithm with mod $|\det(\underline{S^t})|$ arithmetic [SL96] //$(F^t)^{-1} = UM^{-1}$ is a t-matrix for some unimodular matrix $U \in \mathbb{Z}^{d \times d}$
5: Find a matrix B s.t. $S = F^t B$ //$B = UE \in \mathbb{Z}^{d \times m}$ is a basis of $L(A)$
6: Compute the integral GSO of Gram(B) using Alg. 9
7: Size-reduce B
8: **return** B

Theorem 3.2. *Given as input an $n \times m$ integer matrix A, Alg. 1 outputs a size-reduced basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ of the lattice $L(A)$ such that for $i = 1, \ldots, d$, $\|\mathbf{b}_i^*\| \leq \|\mathbf{s}_i^*\|$, $\|\mathbf{b}_i\| \leq \sqrt{i} \times \|S\|$ and $span(\mathbf{b}_1, \ldots, \mathbf{b}_i) = span(\mathbf{s}_1, \ldots, \mathbf{s}_i)$, where $S = (\mathbf{s}_1, \ldots, \mathbf{s}_d)$ is the support of A. It requires $O(mnd(\frac{\log d\|A\|}{\log \|A\|}) \cdot \mathcal{B}(d \log \|A\|))$ bit operations and runs on integers of size $O(d(\log d + \log \|A\|))$ during execution.*

The execution of size-reduction at Step 7 dominates the global cost of Alg. 1. Our basis algorithm replaces it by a rounding-off operation "$B \leftarrow B - \texttt{Round}((F^t)^{-1}) \times S$", which allows to output a similarly good basis in time $O(\max\{m, n\}^{\theta+1+\varepsilon} \log^{1+\varepsilon} \|A\|)$. Since the HNF F has provable magnitude: $\|F\| \leq \|A\|^d$, the main issue is to cheaply round the inverse $(F^t)^{-1}$ from F. This is the goal of the following loose reduction.

3.2 Loose reduction

If B is a lattice basis $\preceq A$, $\|B\|$ can still be arbitrarily larger than $\|A\|$. To prevent this problem, one can use size-reduction. We introduce *loose reduction*, a cheaper alternative: we say that a basis B of a d-rank lattice is *A-reduced* if $B \preceq A$ and there exists a t-matrix $T = (t_{k,\ell}) \in \mathbb{Q}^{d \times d}$ s.t. $B = T\underline{A}$ and $|t_{k,\ell}| \leq 1/2$ for all $1 \leq \ell < k \leq d$. Geometrically, this means that for each k, $\mathbf{b}_k - t_{k,k}\mathbf{a}_{i_k}$ is inside the parallelepiped $\mathcal{P} = \{\sum_{\ell=1}^{k-1} x_\ell \mathbf{a}_{i_\ell} : |x_\ell| \leq 1/2\}$ spanned by $\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \ldots, \mathbf{a}_{i_{k-1}}$, which is reminiscent of Babai's rounding-off algorithm [Bab86]. Loose reduction guarantees that $\|B\| \leq d \times \|A\|$:

Lemma 3.3. *Let B be an A-reduced basis of a d-rank lattice L.*

1. *$span(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k) = span(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{i_k})$ and $\|\mathbf{b}_k^*\| \leq \|\mathbf{a}_{i_k}^*\|$ for $k = 1, \ldots, d$;*
2. *$\|\mathbf{b}_k\| \leq k \times \|A\|$ and $\|\mathbf{b}_k\|_\infty \leq k \times \|A\|_\infty$ for $k = 1, \ldots, d$;*
3. *If $1 \leq r < d$ and $(\mathbf{a}_1, \ldots, \mathbf{a}_r)$ is a primitive set of L, then $(\mathbf{a}_1, \ldots, \mathbf{a}_r, \mathbf{b}_{r+1}, \ldots, \mathbf{b}_d)$ is also an A-reduced basis of L.*

Proof. Items 1 and 2 are straightforward. For Item 3, let $(\mathbf{a}_1, \ldots, \mathbf{a}_r)$ be a primitive set of L. Then $(\mathbf{a}_1, \ldots, \mathbf{a}_r)$ is a basis of the pure sublattice $\Lambda = \text{span}(\mathbf{a}_1, \ldots, \mathbf{a}_r) \cap L$ of L, and $i_k = k$ for $1 \leq k \leq r$. By Item 1, $\text{span}(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k) = \text{span}(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k)$ for $1 \leq k \leq r$. Thus, $(\mathbf{a}_1, \ldots, \mathbf{a}_r)$ and $(\mathbf{b}_1, \ldots, \mathbf{b}_r)$ are two bases of Λ. Therefore, $C = (\mathbf{a}_1, \ldots, \mathbf{a}_r, \mathbf{b}_{r+1}, \ldots, \mathbf{b}_d)$ is a basis of L because B is. It can be checked that C is an A-reduced basis of $L(B)$, because B is. □

The definition of loose reduction can easily be explained. If $B \preceq A$ and B has full-rank d, there exists a unique t-matrix $T \in \mathbb{Q}^{d\times d}$ such that $B = T\underline{A}$. For any strictly lower triangular matrix R in $\mathbb{Z}^{d\times d}$, then $B - R\underline{A} = (T - R)\underline{A}$ is $\preceq B$ and is a basis of $L(B)$. By taking $R = \mathsf{Round}(T)$, $B - R\underline{A} = (T - R)\underline{A}$ gives an A-reduced basis.

We thus obtain Alg. 2, which loosely reduces a basis $B \preceq A$ without GSO (unlike size-reduction). The main difficulty is to cheaply find the triangular representation $B = T\underline{A}$, which is done using the integral dual basis of $\underline{A}$. Then $(T - \mathsf{Round}(T))\underline{A}$ is A-reduced.

Algorithm 2 Integral loose-reduction procedure

Input: The support $\underline{A} \in \mathbb{Z}^{d\times m}$ of an integer matrix $A \in \mathbb{Z}^{n\times m}$ and a lattice basis $B \in \mathbb{Z}^{d\times m}$ where $B \preceq A$.
Output: B becomes A-reduced without changing $L(B)$.

$\delta \leftarrow \det(\underline{A} \cdot \underline{A}^t)$
$W \leftarrow \delta(\underline{A} \cdot \underline{A}^t)^{-1}\underline{A}$ //W is the integral dual basis of $\underline{A}$
$S \leftarrow BW^t$ //$B = \frac{1}{\delta}S\underline{A}$ and $S \in \mathbb{Z}^{d\times d}$ s.t. S/δ is a t-matrix.
$R \leftarrow \mathsf{Round}(S/\delta)$
$B \leftarrow B - R\underline{A}$ //We A-reduce B into $(S/\delta - \mathsf{Round}(S/\delta))\underline{A}$
return B

Theorem 3.4. *Let $A \in \mathbb{Z}^{n\times m}$ and $B \in \mathbb{Z}^{d\times m}$ be two integer matrices of rank d such that $B \preceq A$. Given as input $\underline{A}$ and B, Alg. 2 outputs an A-reduced basis C of the lattice $L(B)$ such that $C \preceq B$ and $C^* = B^*$. It requires $O(md^{\theta-1} \cdot \mathcal{B}(\log \alpha))$ bit operations and uses integers of size $O(\log \alpha)$ where $\alpha = \max\{\|\underline{A}\|^d, \|B\|\}$.*

It is known that size-reducing B requires $O(md^2 \cdot \mathcal{M}(d \log \|B\|))$ bit operations [NV10, p. 43]. With fast matrix multiplication, our loose-reduction is asymptotically faster than size reduction in the typical case $\|\underline{A}\| \in O(\|B\|)$.

3.3 Short bases of special lattices

The key result underlying our first basis algorithm gives special bases of $L(A)$ and $\overline{L(A)}$ which are both lower than A, as follows:

Theorem 3.5. *Let A be an integer matrix. Let E and F be respectively the HNF-bases of $L(A^t)$ and $ker_{\mathbb{Z}}(A)^{\perp} = \overline{L(A^t)}$. Then:*

1. *$E^{\times}A$ is a basis of $ker_{\mathbb{Z}}(A^t)^{\perp} = \overline{L(A)}$ such that $E^{\times}A \preceq A$.*
2. *$F^{\times}A$ is a basis of $L(A)$ such that $F^{\times}A \preceq A$.*

Th. 3.5 is just a corollary of Lemma 2.3 with the following result, which shows how to map any generating set onto a lattice basis, by generalizing unimodular matrices:

Lemma 3.6. *Let A be an $n \times m$ real matrix whose rows generate a lattice of rank d. Let L_A denote the orthogonal projection of $\mathbb{Z}^n$ onto the space spanned by the rows of A^t.*

1. *L_A is a lattice of rank d in $\mathbb{Q}^n$ such that $L_A = (ker_{\mathbb{Z}}(A)^{\perp})^{\times}$.*
2. *If B is a basis of $L(A)$, then $B^{\times}A^t$ is a basis of $ker_{\mathbb{Z}}(A)^{\perp}$, equivalently, $(B^{\times}A^t)^{\times}$ is a basis of L_A.*
3. *If C is a basis of L_A, then CA is a basis of $L(A)$. Equivalently, $\mathbb{Z}^nA = L_AA$.*

Proof. We show Item 1. Since $\ker_{\mathbb{Z}}(A)^{\perp}$ is a d-rank pure sublattice of $\mathbb{Z}^n$ (by Lemma 2.1), $(\ker_{\mathbb{Z}}(A)^{\perp})^{\times}$ is a d-rank lattice in $\mathbb{Q}^n$. It suffices to prove $L_A = (\ker_{\mathbb{Z}}(A)^{\perp})^{\times}$: since $\ker_{\mathbb{Z}}(A)^{\perp} = \mathbb{Z}^n \bigcap \mathrm{span}(A^t)$ and $(\mathbb{Z}^n)^{\times} = \mathbb{Z}^n$, it follows from [Mar02, Prop. 1.3.4] that $(\ker_{\mathbb{Z}}(A)^{\perp})^{\times}$ is the projection of $(\mathbb{Z}^n)^{\times}$ onto the space $\mathrm{span}(A^t)$, namely L_A.

We show Item 2. There is a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ s.t. $UA = \begin{pmatrix} 0_{(n-d)\times m} \\ B \end{pmatrix}$. Let $P \in \mathbb{Z}^{d\times n}$ denote the last d rows of $U^{\times}$. Then P is a primitive set for $\mathbb{Z}^n$ s.t. $A = P^tB$. Thus, $P = B^{\times}A^t$. This implies $\mathrm{span}(A^t) = \mathrm{span}(P)$ and hence $\ker_{\mathbb{Z}}(A)^{\perp} = \mathbb{Z}^n \bigcap \mathrm{span}(P) = L(P)$. Then P is a basis of $\ker_{\mathbb{Z}}(A)^{\perp}$. This proves Item 2.

To show Item 3, we prove $\mathbb{Z}^nA = L_AA$. For any $\mathbf{x} \in \mathbb{Z}^n$, we can decompose it into $\mathbf{x} = \mathbf{y} + \mathbf{z}$ s.t. $\mathbf{y} \in L_A$ and $\mathbf{z} \in \mathrm{span}(A^t)^{\perp}$. Conversely, for any $\mathbf{y} \in L_A$, there exist $\mathbf{x} \in \mathbb{Z}^n$ and $\mathbf{z} \in \mathrm{span}(A^t)^{\perp}$ s.t. $\mathbf{x} = \mathbf{y} + \mathbf{z}$. Since $\mathbf{z}A = \mathbf{0}$ implies $\mathbf{x}A = \mathbf{y}A$, it follows that $\mathbb{Z}^nA = L_AA$.

It remains to explain the equivalence in Item 3. Since CA is full row rank and $L_A = \mathbb{Z}^dC$ implies $L_AA = \mathbb{Z}^d(CA)$, it follows that $\mathbb{Z}^nA = \mathbb{Z}^d(CA)$ iff CA is a basis of $L(A)$. This proves Item 3. □

Note that if A is a lattice basis, $L_A = \mathbb{Z}^n$: any basis of L_A is indeed a unimodular matrix. However, Item 3 is not a characterization of $L(A)$'s bases: for instance, if $A = (2, 4)^t$ and $C = (1, 0)$, then CA is a basis of $L(A)$ but C is not a basis of $L_A = L((\frac{1}{5}, \frac{2}{5}))$.

From a computational point of view, the integer lattice $\ker_{\mathbb{Z}}(A)^{\perp}$ is more convenient than the rational lattice L_A. To use Th. 3.5 algorithmically, we need to A-reduce bases of the form $H^{\times}A$, given an integer matrix A and the HNF H of a lattice such that $H^{\times}A \preceq A$. This is done by Alg. 3 (analyzed in Th. 3.7), which computes $H^{\times}A$ without explicitly

computing $H^\times$: compute $H^\times A$ by solving a triangular system $\underline{A} = \underline{H^t}X$ (because $\underline{A} = \underline{H^t}(H^\times A)$ by Lemma 2.3.3) and then A-reduce $H^\times A$.

Algorithm 3 Computing an A-reduced basis of $L(H^\times A)$

Input: The HNF-basis $H \in \mathbb{Z}^{d\times n}$ of an integer lattice L and an integer matrix $A \in \mathbb{Z}^{n\times m}$ such that $L(A^t)$ is a full-rank sublattice of L.
Output: An A-reduced basis of the integer lattice $L(H^\times A)$.

1: Find the indices $i_1 < \cdots < i_d$ corresponding to H^t's support. By Lemma 2.3, they also define the support $\underline{A}$.
2: Find a matrix B such that $\underline{A} = \underline{H^t}B$, using the fact that $\underline{H^t}$ is square lower-triangular //By Lemma 2.3, $B = H^\times A \in \mathbb{Z}^{d\times m}$ has rank d and $B \preceq A$
3: A-reduce B using Alg. 2
4: **return** B

Theorem 3.7. *Let L be an integer lattice with HNF-basis $H \in \mathbb{Z}^{d\times n}$ and $A \in \mathbb{Z}^{n\times m}$ be an integer matrix such that $L(A^t)$ is a full-rank sublattice of L. Given as input H and A, Alg. 3 outputs an A-reduced basis $B \in \mathbb{Z}^{d\times m}$ of the lattice $L(H^\times A)$ such that $B \preceq H^\times A$. It requires $O(md^{\theta-1}(\frac{\log d\|A\|}{\log\|A\|}) \cdot \mathcal{B}(d\log\|A\|))$ bit operations and runs on integers of size $O(d\log\|A\|)$ during execution.*

3.4 An HNF-based basis algorithm

The main result is the following:

Theorem 3.8. *Given as input an $n\times m$ integer matrix A of rank d, one can compute A-reduced bases of $\overline{L(A)}$ and $L(A)$ in $O(mn(d^{\theta-2} + \log d) \cdot \mathcal{B}(d(\log n + \log\|A\|)))$ bit operations, using operands of size $O(d(\log n + \log\|A\|))$. For $L(A)$, this is done by Alg. 4.*

For $\overline{L(A)}$, this follows directly from Th. 3.5 and Th. 3.7 (Alg. 3), by calling once Storjohann's HNF algorithm [Sto00, Chap. 6] and Alg. 3. For $L(A)$, Th. 3.5 requires the HNF of $\ker_\mathbb{Z}(A)^\perp = \overline{L(A^t)}$. But that HNF can be efficiently computed from any short basis of $\overline{L(A^t)}$. Thus, Alg. 4 first computes the HNF of $L(A)$, which is used to find an A^t-reduced basis of $\ker_\mathbb{Z}(A)^\perp$ and therefore the HNF of $\ker_\mathbb{Z}(A)^\perp$, which is eventually used to find an A-reduced basis of $L(A)$.

Algorithm 4 An HNF-based basis algorithm

Input: A matrix $A \in \mathbb{Z}^{n\times m}$ whose rows might be linearly dependent.
Output: An A-reduced basis of the lattice $L(A)$ generated by the rows of A.

1: Compute the HNF-basis $E \in \mathbb{Z}^{d\times m}$ of $L(A)$ by applying Storjohann's HNF algorithm [Sto00, Chap. 6] to A
2: Find an A^t-reduced basis $C \in \mathbb{Z}^{d\times n}$ of $\ker_\mathbb{Z}(A)^\perp$ by applying Alg. 3 to E and A^t //Here $C \preceq E^\times A^t \preceq A^t$
3: Compute the HNF-basis $F \in \mathbb{Z}^{d\times n}$ of $\ker_\mathbb{Z}(A)^\perp$ by applying Storjohann's HNF algorithm [Sto00, Chap. 6] to C
4: Find an A-reduced basis $B \in \mathbb{Z}^{d\times m}$ of $L(A)$ by applying Alg. 3 to F and A //Here $B \preceq F^\times A \preceq A$
5: **return** B

The output basis is A-reduced and therefore has all the properties of Lemma 3.3.
Alg. 4 is a variant of Alg. 1, with the following differences:

- Unlike Alg. 1, Alg. 4 does not rely on GSO.
- The HNF F of Alg. 4 is a basis of $\ker_\mathbb{Z}(A)^\perp = L(E^\times A^t) \subseteq \mathbb{Z}^n$, but the HNF F of Alg. 1 is a basis of $L(M^t) = L(E^\times S^t) \subseteq \mathbb{Z}^d$.
- Alg. 4 runs Storjohann's HNF algorithm [Sto00] on a "loosely-reduced" basis C of $\ker_\mathbb{Z}(A)^\perp$, but Alg. 1 calls Storjohann-Labahn's HNF algorithm "with mod determinant arithmetic" [SL96] on M^t.

As an application, Alg. 4 can extend a primitive set into a lattice basis:

Corollary 3.9. *Let $B_0 \in \mathbb{Z}^{d\times m}$ be a basis of a lattice L and $S = (\mathbf{s}_1, \ldots, \mathbf{s}_r)$ be a linear independent set such that $L(S) \subseteq L$. Given as input the row-concatenation A of S and B_0, Alg. 4 outputs an A-reduced basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ of L within $O(md(d^{\theta-2} + \log d) \cdot \mathcal{B}(d(\log d + \log\|A\|)))$ bit operations. Moreover, if S is a primitive set for L, then $(\mathbf{s}_1, \ldots, \mathbf{s}_r, \mathbf{b}_{r+1}, \ldots, \mathbf{b}_d)$ is also an A-reduced basis of L.*

If the input S is the same as in Th. 3.1, then Cor. 3.9 provides a basis B with the same properties as Th. 3.1, with a better bit-complexity upper bound, except that $\|\mathbf{b}_i\| \leq i \times \|\mathbf{s}_i\|$ instead of $\|\mathbf{b}_i\| \leq \sqrt{i} \times \|\mathbf{s}_i\|$.

3.5 The extended version

Some applications (such as integer programming [AvH14], linear Diophantine equations [Ili89b] and matrix gcds [SL95]) require a unimodular transform corresponding to the output basis. Alg. 5 efficiently computes a unimodular transformation for any basis algorithm. It calls Storjohann's HNF algorithm [Sto00, Chap. 6] as a subroutine, which recovers a unimodular transformation for the HNF computation.

Algorithm 5 Computing a unimodular transformation

Input: A matrix $A \in \mathbb{Z}^{n\times m}$ and a basis $B \in \mathbb{Z}^{d\times m}$ of the lattice $L(A)$.
Output: A unimodular transformation between A and B.

1: Compute the HNF-basis $E \in \mathbb{Z}^{d\times m}$ of A and a unimodular matrix $V \in \mathbb{Z}^{n\times n}$ such that $VA = \begin{pmatrix} 0_{(n-d)\times m} \\ E \end{pmatrix}$ using Storjohann's HNF algorithm [Sto00, Chap. 6]
2: Find the unimodular matrix $W \in \mathbb{Z}^{d\times d}$ such that $B = WE$ by solving a triangular system $\underline{B^t} = \underline{E^t}X$
3: $U \leftarrow \begin{pmatrix} I_{n-d} & \\ & W \end{pmatrix} V$ //Note that $UA = \begin{pmatrix} 0_{(n-d)\times m} \\ B \end{pmatrix}$
4: **return** U

Combining Alg. 4 with Alg. 5, we obtain the following:

Theorem 3.10. *There is a deterministic algorithm which, given as input an $n \times m$ integer matrix A, outputs an A-reduced basis B of the lattice $L(A)$ and a unimodular matrix $U = (\mathbf{u}_1, \ldots, \mathbf{u}_n) \in \mathbb{Z}^{n\times n}$ s.t.:*

1. $\mathbf{u}_i A = \mathbf{0}$ *for* $i = 1, \ldots, n-d$ *and* $(\mathbf{u}_{n-d+1}, \ldots, \mathbf{u}_n)A = B$;
2. $\|\mathbf{u}_i\|_\infty \leq d^{d+1}\|A\|_\infty^{2d}$ *for* $i = 1, \ldots, n-d$ *and* $\|\mathbf{u}_j\|_\infty \leq 2^d d^{d/2+1}\|A\|_\infty^{d+1}$ *for* $j = n-d+1, \ldots, n$,

where d is the rank of A. It requires $O(mn(d^{\theta-2}\log\frac{2n}{d} + \log n)\cdot\mathcal{B}(d(\log n + \log\|A\|)))$ bit operations and uses integers of size $O(d(\log n + \log\|A\|))$.

4 An XGCD-based basis algorithm

Sect. 3 showed that the basis problem can be reduced to HNF computations, which classically reduces to XGCD (see, *e.g.*, [KB79, HM91]). Here, we directly reduce the basis problem to XGCD using the GSO: given as input an $n \times m$ integer matrix A of rank d, we output a basis B of $L(A)$ within $O((m+n)(n-d+1)d^2(\frac{\log d\|A\|}{\log\|A\|})\cdot\mathcal{B}(d\log\|A\|))$ bit operations such that $B \leq A$ and $\|B\| \leq \sqrt{d}\times\|A\|$. This algorithm has quasi-linear complexity $O(mn(n-d+1)d^{2+\varepsilon}(\log\|A\|)^\varepsilon(\log d + \log\|A\|))$ with fast integer arithmetic.

It can be generalized to quadratic forms: given as input a positive semi-definite integral matrix $P \in \mathbb{Z}^{n\times n}$, it outputs a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ within $O(n^2(n-d+1)d^{\theta-1}\cdot\mathcal{B}(d(\log d + \log\|P\|_\infty)))$ bit operations such that $UPU^t = \begin{pmatrix} 0_{(n-d)\times(n-d)} & 0_{(n-d)\times d} \\ 0_{d\times(n-d)} & Q \end{pmatrix}$ and $\|U\|_\infty \leq (2d^2\times\|P\|_\infty^{2.5})^{d(n-d)}$, where $Q \in \mathbb{Z}^{d\times d}$ is positive definite and $\|Q\|_\infty \leq d\times\|P\|_\infty$.

Both algorithms can be viewed as a quasi-linear variant of the Li-Nguyen basis algorithm presented as a subroutine in [LN14], which has super-quadratic complexity (see Appendix A.3). Its main idea is to iteratively transform the fingerprint of the input matrix into $(0^{n-d}1^d)_\mathcal{F}$, in which case the first $n-d$ vectors are zero, and the remaining d vectors necessarily form a basis of the lattice. To do so, it moves all the 0's of the fingerprint to the front, by applying sufficiently many times an elementary operation. To make the algorithm quasi-linear, we regroup many consecutive elementary operations using a single XGCD which we call Euclid-swap.

4.1 Euclid-swap

We introduce the *Euclid-swap* operation (Alg. 6), which transforms any occurrence of 10 in the fingerprint into 01 in one shot using an XGCD on suitable integers related to the GSO: the output matrix is lower than the input matrix, so it never increases the lengths of the Gram-Schmidt vectors. If 10 occurs at indice $(i-1, i)$, then $\pi_{i-1}(\mathbf{b}_{i-1}) = \mathbf{b}^*_{i-1}$ and $\pi_{i-1}(\mathbf{b}_i)$ generate a one-rank lattice, for which Euclid's algorithm can compute a basis. More precisely, $\pi_{i-1}(\mathbf{b}_i) = \mu_{i,i-1}\mathbf{b}^*_{i-1} = (\lambda_{i,i-1}/d_{i-1})\mathbf{b}^*_{i-1}$, therefore $\pi_{i-1}(d_{i-1}\mathbf{b}_i - \lambda_{i,i-1}\mathbf{b}_{i-1}) = 0$. So if we replace $\mathbf{b}_{i-1}$ by $d_{i-1}\mathbf{b}_i - \lambda_{i,i-1}\mathbf{b}_{i-1}$, we make $\mathbf{b}^*_{i-1} = 0$. To update $(\mathbf{b}_{i-1}, \mathbf{b}_i)$ and preserve the lattice, we use XGCD on $\lambda_{i,i-1}$ and d_{i-1} to obtain a suitable unimodular transform (see Alg. 6) replacing 10 in the fingerprint into 01.

Theorem 4.1. *Given as input an index i and an integer matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{n\times m}$ of rank d whose fingerprint satisfies $\varepsilon_{i-1}\varepsilon_i = 10$, Alg. 6 outputs a matrix $C = (\mathbf{b}_1, \ldots, \mathbf{b}_{i-2}, \mathbf{c}_{i-1}, \mathbf{c}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{n\times m}$ such that $C \leq B$, $L(C) = L(B)$, and the fingerprint of C is such that $\varepsilon_{i-1}\varepsilon_i = 01$. It requires $O(nd\cdot\mathcal{B}(d\log\|B\|) + md\cdot\mathcal{M}(\log\|B\|))$ bit operations and runs on integers of size $O(d\log\|B\|)$.*

The Euclid-swap operation can be adapted to integral quadratic forms. The only difference is Step 2: instead of performing operations directly on vectors, we apply the corresponding unimodular transform to the quadratic form: First, extract the unimodular transformation: $U \leftarrow \begin{pmatrix} I_{i-2} & & & \\ & -\lambda_{i,i-1}/g & d_{i-1}/g & \\ & s & t & \\ & & & I_{n-i} \end{pmatrix}$; Secondly, transform the associated matrix: $P \leftarrow UPU^t$. This adapted Euclid-swap procedure requires $O(nd\cdot\mathcal{B}(d\log\|P\|_\infty))$ bit operations and uses integers of size $O(d\log\|P\|_\infty)$, where d is the rank of $P \in \mathbb{Z}^{n\times n}$.

Algorithm 6 Integral Euclid-swap

Input: An integer matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{n \times m}$ with integral GSO d_j's and $\lambda_{j,k}$'s and whose fingerprint is such that $\varepsilon_{i-1}\varepsilon_i = 10$.
Output: B is modified so that $L(B)$ is preserved and $\varepsilon_{i-1}\varepsilon_i = 01$. The d_j's and $\lambda_{j,k}$'s are updated accordingly.

1: Find integers (s, t, g) using XGCD such that $sd_{i-1} + t\lambda_{i,i-1} = g = \gcd(d_{i-1}, \lambda_{i,i-1})$ with $|s| \leq \max\{1, |\lambda_{i,i-1}/g|\}$ and $|t| \leq |d_{i-1}/g|$
2: $(\mathbf{b}_{i-1}, \mathbf{b}_i) \leftarrow \begin{pmatrix} -\lambda_{i,i-1}/g & d_{i-1}/g \\ s & t \end{pmatrix} (\mathbf{b}_{i-1}, \mathbf{b}_i)$
3: **for** $k = 1$ to $i - 2$ **do**
4: $\begin{pmatrix} \lambda_{i-1,k} \\ \lambda_{i,k} \end{pmatrix} \leftarrow \begin{pmatrix} -\lambda_{i,i-1}/g & d_{i-1}/g \\ s & t \end{pmatrix} \begin{pmatrix} \lambda_{i-1,k} \\ \lambda_{i,k} \end{pmatrix}$ //We update the d_j's and $\lambda_{j,k}$'s at Steps 3-12
5: **end for**
6: **for** $\ell = i + 1$ to n **do**
7: $d_\ell \leftarrow d_\ell g^2 / d_{i-1}^2$; $\lambda_{\ell,i} \leftarrow \lambda_{\ell,i-1} g / d_{i-1}$; $\lambda_{\ell,i-1} \leftarrow 0$
8: **end for**
9: **for** $u = i + 2$ to n **do**
10: **for** $v = i + 1$ to $u - 1$ **do** $\lambda_{u,v} \leftarrow \lambda_{u,v} g^2 / d_{i-1}^2$
11: **end for**
12: $d_i \leftarrow g^2 / d_{i-1}$; $d_{i-1} \leftarrow d_{i-2}$; $\lambda_{i,i-1} \leftarrow 0$
13: **return** $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$, the d_j's and $\lambda_{j,k}$'s.

4.2 An XGCD-based basis algorithm

Our XGCD-based basis algorithm is Alg. 7, whose technical ideas are summarized below:

- A single Euclid-swap replaces many consecutive swaps of the Li-Nguyen basis algorithm [LN14].
- To avoid "intermediate entries explosion", we use size reduction like LLL [LLL82].

Algorithm 7 The XGCD-based basis algorithm

Input: An integer matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{n \times m}$ such that $\mathbf{b}_1 \neq \mathbf{0}$. The rows of B might be linearly dependent.
Output: A basis of the lattice $L(B)$ generated by the rows of B.

1: Compute the d_i's, $\lambda_{i,k}$'s and ε_i's of Gram(B) using Alg. 9
2: $z \leftarrow 0$; $j \leftarrow 2$
3: **while** $j \leq n$ **do**
4: **if** $\varepsilon_j = 1$ **then**
5: $j \leftarrow j + 1$
6: **else**
7: **for** $i = j$ downto $z + 2$ **do**
8: Euclid-swap $(\mathbf{b}_{i-1}, \mathbf{b}_i)$ using Alg. 6
9: Size-reduce $\mathbf{b}_{i-1}$ and $\mathbf{b}_i$ w.r.t. $(\mathbf{b}_{z+1}, \ldots, \mathbf{b}_i)$ using Alg. 10
10: **end for**
11: $j \leftarrow j + 1$; $z \leftarrow z + 1$
12: **end if**
13: **end while**
14: **return** $(\mathbf{b}_{z+1}, \ldots, \mathbf{b}_n)$

Our main result on Alg. 7 is the following:

Theorem 4.2. *Given as input an $n \times m$ integer matrix A of rank d, Alg. 7 outputs a basis B of the lattice $L(A)$ such that $B \preceq A$ and $\|B\| \leq \sqrt{d} \times \|A\|$. It requires $O((m+n)(n-d+1)d^2(\frac{\log d\|A\|}{\log \|A\|}) \cdot \mathcal{B}(d \log \|A\|))$ bit operations and its operands have size $O(d(\log d + \log \|A\|))$.*

We have $B \preceq A$ by transitivity: each Euclid-swap can only lower the matrix, and the final Step 14 too, because it removes front zero vectors.

4.3 Generalization to quadratic forms

The quadratic-form analogue of the basis problem is to transform a positive semi-definite integral quadratic form into an equivalent positive definite integral quadratic form (see, *e.g.*, [Kit93, Coh93, Sch09]). However, very few algorithms for this natural problem are known: perhaps the most famous solution is a generalization of LLL [Poh87], but the full complexity analysis of the variant with unimodular transformation seems to be missing in the literature. HNF-based basis algorithms are not trivial to generalize to integral quadratic forms, since quadratic forms have no HNF analogue.

- Any positive semi-definite integral matrix $P \in \mathbb{Z}^{n \times n}$ has a Cholesky decomposition $P = RR^t$, but the matrix R is in general real, so may not have an HNF-basis.

- Even if there exists an integral decomposition $P = MM^t$ for some integer matrix M, it may not be possible to extract such an M in polynomial time.

Our XGCD-based basis algorithm can tackle this problem, because it only uses the inner products of the input vectors, which gives rise to Alg. 8. However, its polynomial-time complexity is not straightforward: we prove it by analyzing consecutive Euclid-swaps at Steps 7-9.

Algorithm 8 An XGCD-based quadratic form algorithm

Input: A positive semi-definite integral matrix $P \in \mathbb{Z}^{n\times n}$.

Output: A unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that $UPU^t = \begin{pmatrix} 0_{(n-d)\times(n-d)} & 0_{(n-d)\times d} \\ 0_{d\times(n-d)} & Q \end{pmatrix}$, where $Q \in \mathbb{Z}^{d\times d}$ is positive definite.

1: Compute the integral GSO and the fingerprints ε_i's of P
2: $z \leftarrow 0$; $j \leftarrow 2$; $U \leftarrow$ the $n \times n$ identity matrix
3: **while** $j \leq n$ **do**
4: **if** $\varepsilon_j = 1$ **then**
5: $j \leftarrow j + 1$
6: **else**
7: **for** $i = j$ downto $z + 2$ **do**
8: Euclid-swap P w.r.t. $(i - 1, i)$ using the quadratic form version of Alg. 6, and update U accordingly
9: **end for**
10: **for** $i = z + 2$ to j **do**
11: Size-reduce P w.r.t. index i using the quadratic form version of Alg. 10, and update U accordingly
12: **end for**
13: $j \leftarrow j + 1$; $z \leftarrow z + 1$
14: **end if**
15: **end while**
16: **return** U

Our main result on Alg. 8 is as follows:

Theorem 4.3. *Given as input a positive semi-definite integral quadratic form $f(\mathbf{x}) = \mathbf{x}P\mathbf{x}^t$ with matrix $P \in \mathbb{Z}^{n\times n}$, Alg. 8 outputs a unimodular matrix $U \in \mathbb{Z}^{n\times n}$ such that*

$$UPU^t = \begin{pmatrix} 0_{(n-d)\times(n-d)} & 0_{(n-d)\times d} \\ 0_{d\times(n-d)} & Q \end{pmatrix} \text{ and } \|U\|_\infty \leq (2d^2 \times \|P\|_\infty^{2.5})^{d(n-d)},$$

where $Q \in \mathbb{Z}^{d\times d}$ is positive definite and $\|Q\|_\infty \leq d \times \|P\|_\infty$. It requires $O(n^2(n - d + 1)d^{\theta-1} \cdot \mathcal{B}(d(\log d + \log \|P\|_\infty)))$ bit operations and runs on integers of size $O(d(n - d + 1)(\log d + \log \|P\|_\infty))$ during execution.

As an application, Alg. 8 can be used to compute a short basis of any associated lattice of integral quadratic forms: given as input a generator matrix $A \in \mathbb{R}^{n\times m}$ of a real lattice L satisfying $\langle \mathbf{u}, \mathbf{v} \rangle \in \mathbb{Z}$ for any $\mathbf{u}, \mathbf{v} \in L$, one can run Alg. 8 on AA^t to output a unimodular transformation U, then the nonzero rows of UA form a basis B of L such that $\|B^*\| \leq \|A^*\|$ and $\|B\| \leq \sqrt{\text{rank}(L)} \times \|A\|$.

References

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. 3

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proc. of STOC '96*, pages 99–108. ACM, 1996. 1, 5

[AvH14] K. Aardal and F. von Heymann. On the structure of reduced kernel lattice bases. *Mathematics of Operations Research*, 39(3):823–840, 2014. 7

[Bab86] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. 4, 5

[Bla66] W. A. Blankinship. Algorithm 287: Matrix triangulation with integer arithmetic. *Communications of the ACM*, 9(7):513, 1966. 21

[BP87] J. Buchmann and M. Pohst. Computing a lattice basis from a system of generating vectors. In *Proc. of EUROCAL '87*, volume 378, pages 54–63. Springer, 1987. 1

[Bra71] G. H. Bradley. Algorithms for Hermite and Smith normal matrices and linear Diophantine equations. *Mathematics of Computation*, 25:897–907, 1971. 21, 22

[CC82] T.-W. J. Chou and G. E. Collins. Algorithms for the solution of systems of linear Diophantine equations. *SIAM Journal of Computing*, 11(4):687–708, 1982. 21, 22

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, third edition, 2009. 14

[CN97] J.-Y. Cai and A. Nerurkar. An improved worst-case to average-case connection for lattice problems. In *Proc. of FOCS '97*, pages 468–477. IEEE, 1997. 2, 4, 5

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory. Volume 138 of Graduate Texts in Mathematics*. Springer, 1993. 1, 4, 9, 12, 16

[DKT87] P. D. Domich, R. Kannan, and L. E. Trotter. Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research*, 12(1):50–59, 1987. 21, 22

[dW87] B. M. M. de Weger. Solving exponential Diophantine equations using lattice basis reduction algorithms. *Journal of Number Theory*, 26(3):325–367, 1987. 4, 13, 16

[Gal14] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. of ISSAC '14*, pages 296–303. ACM, 2014. 2, 3

[GHGN06] N. Gama, N. Howgrave-Graham, and P. Q. Nguyen. Symplectic lattice reduction and NTRU. In *Proc. of EUROCRYPT '06*, volume 4004 of *LNCS*, pages 233–253. Springer, 2006. 13

[Gie95] M. Giesbrecht. Fast computation of the Smith normal form of an integer matrix. In *Proc. of ISSAC '95*, pages 110–118. ACM, 1995. 22

[Gie96] M. Giesbrecht. Probabilistic computation of the Smith normal form of a sparse integer matrix. In *Proc. of ANTS-II*, volume 1122 of *LNCS*, pages 173–186. Springer, 1996. 22

[GL96] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, third edition, 1996. 14

[GN08] N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *Proc. of STOC '08*, pages 207–216. ACM, 2008. 1

[GNR10] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *Proc. of EUROCRYPT '10*, volume 6110 of *LNCS*, pages 257–278. Springer, 2010. 1

[GPV08] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proc. of STOC '08*, pages 197–206. ACM, 2008. 1, 2

[HJLS89] J. Håstad, B. Just, J. C. Lagarias, and C. P. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM Journal on Computing*, 18(5):859–881, 1989. 15

[HM91] J. L. Hafner and K. S. McCurley. Asymptotically fast triangularization of matrices over rings. *SIAM Journal on Computing*, 20(6):1068–1083, 1991. 3, 8, 21, 22

[HM97] G. Havas and B. S. Majewski. Integer matrix diagonalization. *Journal of Symbolic Computation*, 24:399–408, 1997. 22

[HMM98] G. Havas, B. S. Majewski, and K. R. Matthews. Extended gcd and Hermite normal form algorithms via lattice basis reduction. *Experimental Mathematics*, 7(2):125–136, 1998. 21, 22

[HPS11] G. Hanrot, X. Pujol, and D. Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Proc. of CRYPTO '11*, volume 6841 of *LNCS*, pages 447–464. Springer, 2011. 2, 15

[HvdH19] D. Harvey and J. van der Hoeven. Faster integer multiplication using plain vanilla FFT primes. *Mathematics of Computation*, 88(315):501–514, 2019. 3

[Ili89a] C. S. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix. *SIAM Journal of Computing*, 18(4):658–669, 1989. 21, 22

[Ili89b] C. S. Iliopoulos. Worst-case complexity bounds on algorithms for computing the canonical structure of infinite abelian groups and solving systems of linear Diophantine equations. *SIAM Journal on Computing*, 18(4):670–678, 1989. 7

[KB79] R. Kannan and A. Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8(4):499–507, 1979. 8, 15, 21, 22

[Kit93] Y. Kitaoka. *Arithmetic of Quadratic Forms*. Cambridge Tracts in Mathematics. Cambridge University Press, 1993. 9

[Knu98] D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison-Wesley, third edition, 1998. 12

[KV04] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13:91–130, 2004. 17

[LLL82] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:366–389, 1982. 1, 4, 9, 12, 15

[LN14] J. Li and P. Q. Nguyen. Approximating the densest sublattice from Rankin's inequality. *LMS Journal of Computation and Mathematics*, 17(Special Issue A):92–111, 2014. Contributed to ANTS-XI, 2014. 2, 8, 9, 15, 16

[LN19] J. Li and P. Q. Nguyen. Computing a lattice basis revisited. In *Proc. of ISSAC '19*, pages 275–282. ACM, 2019. 1

[Mar02] J. Martinet. *Perfect Lattices in Euclidean Spaces*. Springer, 2002. 6

[Maz11] G. Maze. Natural density distribution of Hermite normal forms of integer matrices. *Journal of Number Theory*, 131(12):2398–2408, 2011. 1

[MG02] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*, vol-

ume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 2002. 1, 2, 4, 5

[MvTW08] S. S. Magliveras, T. van Trung, and W. Wei. Primitive sets in a lattice. *Australasian Journal of Combinatorics*, 40:173–186, 2008. 1

[MW01] D. Micciancio and B. Warinschi. A linear space algorithm for computing the Hermite normal form. In *Proc. of ISSAC '01*, pages 231–236. ACM, 2001. 21, 22

[NS09] P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009. 15, 16

[NS16] A. Neumaier and D. Stehlé. Faster LLL-type reduction of lattice bases. In *Proc. of ISSAC '16*, pages 373–380. ACM, 2016. 2, 15

[NSV11] A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity. In *Proc. of STOC '11*, pages 403–412. ACM, 2011. 2, 15

[NV10] P. Q. Nguyen and B. Vallée, editors. *The LLL Algorithm: Survey and Applications*. Information Security and Cryptography. Springer, New York, 2010. 6

[Poh87] M. Pohst. A modification of the LLL reduction algorithm. *Journal of Symbolic Computation*, 4(1):123–127, 1987. 1, 2, 9, 15

[Sch87] C. P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987. 1

[Sch09] A. Schürmann. *Computational geometry of positive definite quadratic forms: Polyhedral reduction theories, algorithms, and applications*. American Mathematical Society, 2009. 9

[Sie89] C. L. Siegel. *Lectures on the Geometry of Numbers*. Springer, New York, 1989. 1, 2, 4

[SL95] A. Storjohann and G. Labahn. Preconditioning of rectangular polynomial matrices for efficient Hermite normal form computation. In *Proc. of ISSAC '95*, pages 119–125. ACM, 1995. 7

[SL96] A. Storjohann and G. Labahn. Asymptotically fast computation of Hermite normal forms of integer matrices. In *Proc. of ISSAC '96*, pages 259–266. ACM, 1996. 3, 5, 7, 15, 17, 21, 22

[Sto96a] A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical report 249, Department of Computer Science, ETH Zürich, Switzerland, 1996. 2

[Sto96b] A. Storjohann. Near optimal algorithms for computing Smith normal forms of integer matrices. In *Proc. of ISSAC '96*, pages 267–274. ACM, 1996. 22

[Sto98] A. Storjohann. Computing Hermite and Smith normal forms of triangular integer matrices. *Linear Algebra and its Applications*, 282:25–45, 1998. 22

[Sto00] A. Storjohann. *Algorithms for Matrix Canonical Forms*. Ph.D. thesis, ETH Zürich, 2000. 2, 5, 7, 8, 15, 17, 18, 21, 22

[Vil03] G. Villard. Exact computations on polynomial and integer matrices. 2003. Aailable at `http://www.ens-lyon.fr/gvillard`. 17

A Previous basis algorithms

A.1 Preliminaries

Our time complexity analysis for basis algorithms often uses an elementary fact for integer multiplication:

Fact A.1 (See, *e.g.*, [Knu98, §4.3.3]). *For $s, t \in \mathbb{Z}^+$, $\mathcal{M}(st) = O(s^2\mathcal{M}(t))$ via truncations. If $s > t$, an s-bit integer and a t-bit integer can be multiplied in $O(\frac{s}{t}\mathcal{M}(t))$ bit operations.*

SNF. An $n \times m$ integer matrix S of rank d is in *Smith normal form* (abbreviated SNF) if it is of the form $S = \mathrm{diag}(s_1, \ldots, s_d, 0, \ldots, 0)$ such that s_i divides s_{i+1} for $i = 1, \ldots, d-1$.

Computing GSO. An elementary procedure can compute the integral GSO and fingerprint of a positive semi-definite integral quadratic form, namely Algorithm 9. Let $P \in \mathbb{Z}^{n\times n}$ be the input quadratic form, which can be thought as $\mathrm{Gram}(B)$ for some matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$.

We additionally let $\lambda_{i,i} = d_{i-1}\langle \mathbf{b}_i, \mathbf{b}_i^* \rangle$ for $i = 1, \ldots, n$, then $\lambda_{i,i} = d_i$ if $\varepsilon_i = 1$ and $\lambda_{i,i} = 0$ otherwise.

We explain the recursive formulae on S at Step 7 for indices (i, j). For $k = 1, \ldots, j-1$, let $S_k = d_k\langle \mathbf{b}_i, \sum_{\ell=1}^{k} \mu_{j,\ell}\mathbf{b}_\ell^* \rangle$ and then $S_k \in \mathbb{Z}$ (see [LLL82, p. 523] and [Coh93, §2.6.3]). Then $S_1 = \lambda_{i,1}\lambda_{j,1}$ and $S_k = (d_k S_{k-1} + \lambda_{i,k}\lambda_{j,k})/d_{k-1}$ for $k = 2, \ldots, j-1$, in particular, $S_k = S_{k-1}$ if $\varepsilon_k = 0$. We have $\lambda_{i,j} = d_{j-1}\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle = \langle \mathbf{b}_i, \mathbf{b}_j \rangle d_{j-1} - S_{j-1}$ for all $1 \le j \le i \le n$.

The *main properties* of Algorithm 9 which we use in this paper are as follows:

- Given as input a positive semi-definite integral matrix $P \in \mathbb{Z}^{n\times n}$ of rank d, Algorithm 9 computes the integral GSO and fingerprint of P within $O(nd^2 \cdot \mathcal{M}(d \log \|P\|_\infty))$ bit operations and runs on integers of size $O(d \log \|P\|_\infty)$.
- Given as input an integer matrix $B \in \mathbb{Z}^{n\times m}$ of rank d, Algorithm 9 computes the integral GSO and fingerprint of $\mathrm{Gram}(B)$ within $O(mnd \cdot \mathcal{M}(\log \|B\|) + nd^2 \cdot \mathcal{M}(d \log \|B\|))$ bit operations and runs on integers of size $O(d \log \|B\|)$.

Algorithm 9 Computing the integral GSO and fingerprint of an integral quadratic form (adapted from [GHGN06, Alg. 1])

Input: A positive semi-definite integral matrix $P = (p_{i,j}) \in \mathbb{Z}^{n\times n}$, such as the Gram matrix $(\langle \mathbf{b}_i, \mathbf{b}_j\rangle)_{1\le i,j\le n}$ of $(\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{n\times m}$.
Output: The integral GSO d_i's and $\lambda_{i,j}$'s, the fingerprints ε_i's, and the rank d of P.

$d_0 \leftarrow 1$
for $i = 1$ to n **do**
  $\lambda_{i,1} \leftarrow p_{i,1}$
  for $j = 2$ to i **do**
    if $j < i$ and $\varepsilon_j = 0$ **then**
      $\lambda_{i,j} \leftarrow 0$
    else
      $S \leftarrow \lambda_{i,1}\lambda_{j,1}$
      for $k = 2$ to $j-1$ **do**
        if $\varepsilon_k = 1$ **then** $S \leftarrow (d_k S + \lambda_{i,k}\lambda_{j,k})/d_{k-1}$
      end for
      $\lambda_{i,j} \leftarrow p_{i,j} d_{j-1} - S$
    end if
  end for
  if $\lambda_{i,i} = 0$ **then** $d_i \leftarrow d_{i-1}$, $\varepsilon_i \leftarrow 0$; **else** $d_i \leftarrow \lambda_{i,i}$, $\varepsilon_i \leftarrow 1$
end for
$d \leftarrow \sum_{i=1}^{n} \varepsilon_i$
return all the d_i's, $\lambda_{i,j}$'s, ε_i's and d.

- The integral GSO d_i's and $\lambda_{i,j}$'s of Gram(B) satisfy: $d_i \le \|B^*\|^{2\times\min\{i,d\}}$ and $|\lambda_{i,j}| \le \|B\| \cdot \|B^*\|^{2\times\min\{j,d\}-1}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, i-1$.

Size-reduction procedures. Algorithm 10 is a procedure for size-reducing a single vector in terms of integral GSO, since $\mu_{i,j} = \lambda_{i,j}/d_j$ for all $j < i$. It has a quadratic form version. The only difference is Step 6: instead of performing operations directly on vectors, we apply the corresponding unimodular transform to the quadratic form:

- First, extract the unimodular transformation: $U \leftarrow \begin{pmatrix} I_{i-1} & & \\ -\mathbf{c} & 1 & \\ & & I_{n-i} \end{pmatrix}$, where $\mathbf{c} = (x_1, \ldots, x_{i-1})$;
- Secondly, transform the associated matrix: $P \leftarrow UPU^t$.

Algorithm 10 Integral size-reduction procedure (adapted from [dW87, Table 1, Step (C)])

Input: An index i and an integer matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{n\times m}$ with integral GSO d_j's and $\lambda_{j,k}$'s.
Output: The vector $\mathbf{b}_i$ becomes size-reduced w.r.t. B, and update $\{\lambda_{i,j}\}_{i>j\ge 1}$ accordingly.

for $j = i-1$ downto 1 **do**
  $x_j \leftarrow \lceil \frac{\lambda_{i,j}}{d_j} \rfloor$
  for $k = 1$ to $j-1$ **do** $\lambda_{i,k} \leftarrow \lambda_{i,k} - x_j\lambda_{j,k}$
  $\lambda_{i,j} \leftarrow \lambda_{i,j} - x_j d_j$
end for
$\mathbf{b}_i \leftarrow \mathbf{b}_i - \sum_{j=1}^{i-1} x_j \mathbf{b}_j$
return $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$, the d_j's and $\lambda_{j,k}$'s

When the size-reduction process occurs in the basis algorithms, the quantities $\|\mathbf{b}_i\|$, $\max_{1\le j<i}\|\mathbf{b}_j\|$ and $\max_{1\le k<i}\|\mathbf{b}_k^*\|$ have different magnitudes. This inspires the following useful lemma.

Lemma A.2. *Let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{Z}^{n\times m}$ and $i \in [2, n]$. Let $\alpha = \max_{1\le k\le j<i} |\langle \mathbf{b}_j, \mathbf{b}_k^*\rangle|$ and $\beta = \max_{1\le k<i} |\langle \mathbf{b}_i, \mathbf{b}_k^*\rangle|$. Given as input i and B with integral GSO d_j's and $\lambda_{j,k}$'s, Algorithm 10 satisfies the following:*

1. *Algorithm 10 takes $O(i(m+i))$ arithmetic operations, in which only integers occur.*
2. *The integers x_j's at Step 2 satisfy $|x_j| \le (2\alpha)^{i-j-1}(\alpha+\beta)\frac{d_{j-1}}{d_{i-1}}$ for $j = 1, \ldots, i-1$.*
3. *Let $\lambda_{i,k}^{(j)}$ denote $\lambda_{i,k}$ right after Step 3 for index j. Then $|\lambda_{i,k}^{(j)}| \le d_{k-1}(\beta + \alpha(i-j)\max_{j\le s<i}|x_s|)$ for $1 \le k < j < i$.*

Proof. Item 1 is obvious. If some $\mathbf{b}_j^*$ is zero, then $\lambda_{i,j} = d_{j-1}\langle \mathbf{b}_i, \mathbf{b}_j^*\rangle = 0$ implies $x_j = 0$. Without loss of generality, assume that $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ are linearly independent throughout the proof below.

To show Item 2, let $\mathbf{b}_i^{(i)} = \mathbf{b}_i$ and $\mathbf{b}_i^{(j)} = \mathbf{b}_i - \sum_{s=j}^{i-1} x_s \mathbf{b}_s$ for $j = i-1, \ldots, 1$. Then $x_j = \left\lceil \frac{\langle \mathbf{b}_i^{(j+1)}, \mathbf{b}_j^*\rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^*\rangle} \right\rfloor$ and

$$|x_j| \le \left| \frac{\langle \mathbf{b}_i^{(j+1)}, \mathbf{b}_j^*\rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^*\rangle} \right| + \frac{1}{2} = \frac{|\langle \mathbf{b}_i^{(j+1)}, \mathbf{b}_j^*\rangle|}{\|\mathbf{b}_j^*\|^2} + \frac{1}{2} \quad \text{for} \quad j = i-1, \ldots, 1. \tag{A.1}$$

Note that $\langle \mathbf{b}_i^{(j+1)}, \mathbf{b}_j^* \rangle = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle - \sum_{s=j+1}^{i-1} x_s \langle \mathbf{b}_s, \mathbf{b}_j^* \rangle$ implies $|\langle \mathbf{b}_i^{(j+1)}, \mathbf{b}_j^* \rangle| \le \beta + \alpha \sum_{s=j+1}^{i-1} |x_s|$, then (A.1) yields

$$|x_j| \le \frac{\beta + \alpha \sum_{s=j+1}^{i-1} |x_s|}{\|\mathbf{b}_j^*\|^2} + \frac{1}{2} \quad \text{for} \quad j = i-1, \ldots, 1. \tag{A.2}$$

We now prove Item 2 by backward induction on j. Initially, (A.1) implies $|x_{i-1}| \le \frac{\beta}{\|\mathbf{b}_{i-1}^*\|^2} + \frac{1}{2} \le (\alpha+\beta)\frac{d_{i-2}}{d_{i-1}}$. Assume that $x_{i-1}, x_{i-2}, \ldots, x_{j+1}$ have the desired bounds for some $j \in [1, i-2]$. Since $\alpha^{i-s-1} d_{s-1} \le \alpha^{i-j-2} d_j$ for $j+1 \le s \le i-1$, by the induction hypothesis, we have

$$\sum_{s=j+1}^{i-1} |x_s| \le \sum_{s=j+1}^{i-1} 2^{i-s-1} \alpha^{i-j-2} (\alpha+\beta) \frac{d_j}{d_{i-1}} = (2^{i-j-1} - 1) \alpha^{i-j-2} (\alpha+\beta) \frac{d_j}{d_{i-1}}. \tag{A.3}$$

Since $\frac{\beta}{\|\mathbf{b}_j^*\|^2} + \frac{1}{2} \le \alpha^{i-j-1}(\alpha+\beta)\frac{d_{j-1}}{d_{i-1}}$, (A.2) and (A.3) imply $|x_j| \le (2\alpha)^{i-j-1}(\alpha+\beta)\frac{d_{j-1}}{d_{i-1}}$. This proves Item 2.

It remains to show Item 3. Note that $\lambda_{i,k}^{(j)} = d_{k-1}\langle \mathbf{b}_i^{(j)}, \mathbf{b}_k^* \rangle = d_{k-1}(\langle \mathbf{b}_i, \mathbf{b}_k^* \rangle - \sum_{s=j}^{i-1} x_s \langle \mathbf{b}_s, \mathbf{b}_k^* \rangle)$, this implies $|\lambda_{i,k}^{(j)}| \le d_{k-1}(\beta + \alpha \sum_{s=j}^{i-1} |x_s|)$. This proves Item 3 and hence the lemma. □

Solving triangular systems. Basis algorithms relying on HNF need to solve triangular systems.

Under the notation of Lemma 2.3, both triangular systems $A = H^t X$ and $\underline{A} = \underline{H^t} X$ have the same solution $H^\times A$: but $H^\times A$ is easier to compute from the latter system by divide-and-conquer (see the lemma below), because $\underline{H^t}$ is square lower-triangular with non-zero diagonal.

Lemma A.3. *Let $A \in \mathbb{Z}^{d\times m}$ and $T \in \mathbb{Z}^{d\times d}$ be two integer matrices of rank d where T is lower-triangular such that $T^{-1}A$ is an integer matrix. Assume $\alpha = \lceil \log \|T^{-1}A\|_\infty \rfloor + 1 \le \lceil \log \|T\|_\infty \rfloor + 1 = \beta$. Then the system $A = TX$ can be solved within $O(md \cdot \mathcal{M}(\beta) + md^{\theta-1}(\log d)\frac{\beta}{\alpha} \cdot \mathcal{M}(\alpha))$ bit operations with integer arithmetic.*

Proof. We recall the classical divide-and-conquer approach for solving $A = TX$ (see, *e.g.*, [CLRS09, GL96]). Since $A = TX$ is equivalent to $\begin{pmatrix} A \\ 0_{i\times m} \end{pmatrix} = \begin{pmatrix} T & \\ & I_i \end{pmatrix} \begin{pmatrix} X \\ 0_{i\times m} \end{pmatrix}$ for $\forall\ i \ge 0$, we may assume that $d = 2^k$ for some positive integer k.

We show how to reduce the system $A = TX$ to two subsystems of half the size using a block decomposition. Indeed, we partition T into four $d/2 \times d/2$ subblocks and partition each of A and X into two $d/2 \times m$ subblocks:

$$T = \begin{pmatrix} T_1 & \\ T_3 & T_2 \end{pmatrix}, \quad A = \begin{pmatrix} A_1 \\ B_1 \end{pmatrix} \text{ and } X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}.$$

Then $A = TX$ implies $A_1 = T_1 X_1$ and $B_1 - T_3 X_1 = T_2 X_2$. We can solve $A = TX$ as follows:

- Step 1: solve the subsystem $A_1 = T_1 X_1$;
- Step 2: compute $A_2 \leftarrow B_1 - T_3 X_1$;
- Step 3: solve the subsystem $A_2 = T_2 X_2$.

We continue to partition each of systems $A_1 = T_1 X_1$ and $A_2 = T_2 X_2$ into two smaller subsystems, until the solutions to subsystems are vectors instead of matrices. This recurrence derives a recursive divide-and-conquer procedure for solving $A = TX$ with integer arithmetic.

Let $T(d)$ denote the number of bit operations for solving $A = TX$ using this procedure. We claim that

$$T(d) \le 2T(d/2) + cm(d/2)^{\theta-1} t \tag{A.4}$$

for some absolute constant $c > 0$, where $t = \frac{\beta}{\alpha} \cdot \mathcal{M}(\alpha)$. Indeed, the total cost for Step 1 and Step 3 is $2T(d/2)$, and Step 2 takes $O(m(d/2)^{\theta-1}t)$ bit operations, which imply (A.4). Iterating (A.4), we have

$$\begin{aligned} T(d) &\le 2T(\frac{d}{2}) + cm(\frac{d}{2})^{\theta-1} t \\ &\le 2^2 T(\frac{d}{2^2}) + 2cm(\frac{d}{2^2})^{\theta-1} t + cm(\frac{d}{2})^{\theta-1} t \\ &\vdots \\ &\le 2^k T(\frac{d}{2^k}) + \sum_{j=1}^{k} 2^{j-1} cm(\frac{d}{2^j})^{\theta-1} t = dm \cdot \mathcal{M}(\beta) + \frac{1}{2} cmd^{\theta-1} t \sum_{j=1}^{k} 2^{j(2-\theta)}. \end{aligned}$$

Since $\theta \ge 2$ implies $\sum_{j=1}^{k} 2^{j(2-\theta)} \le k$, we have $T(d) \in O(md \cdot \mathcal{M}(\beta) + md^{\theta-1} t \log d)$. This completes the proof. □

A.2 Basis algorithms based on the HNF, SNF and LLL computations

Consider an integer matrix $A \in \mathbb{Z}^{n\times m}$ of rank d with $\alpha = (\sqrt{d}\|A\|_\infty)^d$ and $\beta = \log\|A\|$. Many algorithms for HNF, SNF and LLL can be used or adapted as basis algorithms which, given as input A, output a basis B of the lattice $L(A)$.

Basis algorithms related to HNF or SNF. Storjohann's HNF algorithm [Sto00, Chap. 6] is the best one in terms of both the output guarantees and the runtime among the known algorithms for HNF and SNF: given A, it outputs the HNF-basis H of $L(A)$ such that $\|H\| \le \det(L(A))$, in time $O(mnd^{\theta-1}\log\|A\|_\infty \cdot \mathcal{M}(\log\log\alpha) + mn\log d \cdot \mathcal{B}(\log\alpha))$.

No SNF-based basis algorithm known outperforms Storjohann's HNF algorithm. Indeed, there exist two unimodular matrices $U \in \mathbb{Z}^{n\times n}$ and $V \in \mathbb{Z}^{m\times m}$ such that UAV is in SNF. Let W be the first d rows of U. Then WA is a basis of $L(A)$. The best Storjohann's SNF algorithm [Sto00, Chapter 8] known can find W in time $O(mnd^{\theta-1}(\log mn)(\log\|A\|_\infty)\cdot \mathcal{M}(\log\log\alpha) + mn(\log mn)\cdot\mathcal{B}(\log\alpha))$ such that $\|W\|_\infty \le d^{2d+5}\alpha^4\|A\|_\infty$.

Besides, most of the known HNF and SNF algorithms work with input matrices which have full column rank or even are nonsingular (*e.g.*, [KB79, SL96]). Some of them can be adapted as basis algorithms. For instance, let M be a maximal column independent system of A. Let U be an $n \times n$ unimodular matrix such that UM is in HNF of rank d. Then the first d rows of UA form a basis of $L(A)$, because the bottom $n-d$ rows of UA are zero.

LLL-based basis algorithms. The asymptotically fastest basis algorithm among the known LLL-type algorithms is a modified version of the Novocin *et al.* $\widetilde{\mathrm{L}}^1$ algorithm whose Step 1 performs an HNF computation (see [NSV11, Fig. 2]): it outputs a basis B of $L(A)$ such that $\|B^*\| \le \|B\| \le \xi^d \times \|A\|$ for some constant $\xi > 1$ (depending on input parameters); with fast integer arithmetic, the cost is $O(mn^{4+\varepsilon}\beta + mn^{\theta+\varepsilon}\beta^{1+\varepsilon})$ [NSV11, Theorem 7]; without it, we deduce that the cost is $O(mn^5(\log n)^2(n+\beta^2) + mn^4\beta^2\log\beta)$; $\widetilde{\mathrm{L}}^1$ inherently involves floating-point arithmetic (see [NSV11, Theorem 2]).

The Hanrot *et al.* LLL-type algorithm [HPS11] (with modification) outputs a basis B of $L(A)$ such that $\|B^*\| \le \|A^*\|$ and $\|B\| \le \sqrt{d} \times \|A\|$, but requires $O(mn^4(\log n + \log\beta)(n^2 + \log\beta)\cdot\mathcal{M}(n\beta)$ bit operations.

There is another quasi-linear LLL-type algorithm devised by Neumaier and Stehlé [NS16] (abbreviated $\widehat{\mathrm{L}}^1$): it reduces a basis $B \in \mathbb{Z}^{d\times d}$ into a size-reduced basis $(\mathbf{c}_1,\ldots,\mathbf{c}_d)$ of $L(B)$ such that $\|\mathbf{c}_1\| \le \xi(4/3)^{(d-1)/4}\det(L(B))^{1/d}$ for some constant $\xi > 1$, within time $O(\rho(\log d\gamma)^{4\rho}d^3\cdot\mathcal{M}(d\gamma)$ where $\rho = \lceil\sqrt{\frac{\ln d}{\ln(\phi\ln 8d\gamma)}}\rceil$ for some constant $\phi > 0$ and $\gamma = \log\|B\|$. The $\widehat{\mathrm{L}}^1$ algorithm can work as a basis algorithm using the standard modification: first call an HNF algorithm on A to output the HNF-basis H of $L(A)$ such that $\|H\| \le \det(L(A))$ and then run the $\widehat{\mathrm{L}}^1$-reduction on H. The resulting basis algorithm outputs a size-reduced basis B of $L(A)$ such that $\|B^*\| \le \det(L(A))$ and $\|B\| \le \sqrt{d}\times\det(L(A))$, whose $\widehat{\mathrm{L}}^1$-reduction process requires $O(\rho'(\log d\gamma')^{4\rho'}md^2\cdot\mathcal{M}(d\gamma')$ bit operations. Here, $\rho' = \lceil\sqrt{\frac{\ln d}{\ln(\phi'\ln 8d^2\gamma')}}\rceil$ for some constant $\phi' > 0$ and $\gamma' = \log\det(L(A))$.

Finally, we remind of that Pohst's MLLL algorithm [Poh87] is the first modification of LLL [LLL82] for computing a basis B from the input generators A such that $\|B^*\| \le \|A^*\|$ and $\|B\| \le \sqrt{d}\times\|A\|$. A similar algorithm appeared in [HJLS89, p. 864]. Both algorithms run in time $O(mn^2d\beta\cdot\mathcal{M}(d\beta))$ [HJLS89, Theorem 2.1]. Their floating-point variant has quadratic complexity $O(m(d+\beta)(d^2\beta + n(n-d))\cdot\mathcal{M}(d))$ (see [NS09, §6]).

A.3 The Li-Nguyen basis algorithm

Li and Nguyen [LN14] simplified Pohst's MLLL algorithm [Poh87] into Alg. 11 by ignoring Lovász's conditions.

Algorithm 11 The Li-Nguyen basis algorithm [LN14, Algorithm B.1]

Input: An integer matrix $B = (\mathbf{b}_1,\ldots,\mathbf{b}_n) \in \mathbb{Z}^{n\times m}$ such that $\mathbf{b}_1 \neq \mathbf{0}$. The rows of B might be linearly dependent.
Output: A basis of the lattice $L(B)$ generated by the rows of B.

$z \leftarrow 0$; $j \leftarrow 2$
while $j \le n$ **do**
  if $\mathbf{b}_j^* \neq \mathbf{0}$ **then**
    $j \leftarrow j+1$
  else
    Size-reduce $\mathbf{b}_j$ w.r.t. the previous vectors $(\mathbf{b}_{z+1},\ldots,\mathbf{b}_{j-1})$
    if $\mathbf{b}_j = 0$ **then**
      $(\mathbf{b}_{z+1},\mathbf{b}_{z+2},\ldots,\mathbf{b}_j) \leftarrow (\mathbf{b}_j,\mathbf{b}_{z+1},\mathbf{b}_{z+2},\ldots,\mathbf{b}_{j-1})$ //We move one more zero-vector $\mathbf{b}_j$ to the front and shift the rest
      $j \leftarrow j+1$; $z \leftarrow z+1$
    else
      Swap $\mathbf{b}_{j-1}$ and $\mathbf{b}_j$
      $j \leftarrow \max\{z+2, j-1\}$
    end if
  end if
end while
return $(\mathbf{b}_{z+1},\ldots,\mathbf{b}_n)$

Theorem A.4 ([LN14, Theorem B.1]). *Given as input an $n \times m$ integer matrix A, Algorithm 11 runs in time polynomial in the size of A, and outputs a basis B of the lattice $L(A)$ such that $\|B^*\| \leq \|A^*\|$ and $\|B\| \leq \sqrt{\mathrm{rank}(L)} \times \|A\|$.*

The correctness of the algorithm follows from the following loop invariants [LN14]:

1. $j \geq z + 2$;
2. $\|B^*\|$ never increases;
3. $B_{[1,j-1]}$ has fingerprint $(0^z 1^{j-z-1})_{\mathcal{F}}$ outside of Step 8 and Step 11.

Let f_i be the product of the i first nonzero $\|\mathbf{b}_k^*\|^2$. It was shown in [LN14] that the number of `while` loop iterations is at most $\log(\|A\|^{2\mathrm{rank}(L)^2} 2^{n^2})$, by considering the potential $D = D_L \times D_R$ (as in [Coh93, § 2.6.4] and [NS09, § 6]) where

$$D_L = \prod_{i=1}^{\mathrm{rank}(L)} f_i \in \mathbb{Z} \quad \text{and} \quad D_R = \prod_{i \leq n, \|\mathbf{b}_i^*\| = 0} 2^i \in \mathbb{Z},$$

Algorithm 11 has super-quadratic complexity. To estimate the runtime, we refine Algorithm 11 as follows: call Algorithm 9 to compute initially the integral GSO; apply Algorithm 10 to Step 6; call the swap step in the integral LLL algorithm [dW87, Table 1] to do Step 11, including updating the integral GSO; it is trivial to update the integral GSO at Step 8. To check whether $\mathbf{b}_j^*$ is zero or not, it needs to update the fingerprints.

Theorem A.5. *Assume that Algorithm 11 is refined as above. Given as input an $n \times m$ integer matrix A, Algorithm 11 requires $O((md + n)d(n - d + \log \|A\|)\frac{\log d\|A\|}{\log \|A\|} \cdot \mathcal{M}(d \log \|A\|))$ bit operations and runs on integers of size $O(d(\log d + \log \|A\|))$ during execution, where d is the rank of A.*

Proof. We first improve previous estimate on the number of `while` loop iterations. Consider a simpler potential

$$D = d_n \times D_R \text{ where } d_n = \prod_{i \leq n, \|\mathbf{b}_i^*\| \neq 0} \|\mathbf{b}_i^*\|^2 \in \mathbb{Z} \text{ and } D_R = \prod_{i \leq n, \|\mathbf{b}_i^*\| = 0} 2^{2i} \in \mathbb{Z}.$$

Note that D is a positive integer: initially, $D \leq \|A\|^{2d} 2^{(n+d+1)(n-d)}$. Since size-reduction preserves the $\mathbf{b}_i^*$'s, the only operations which can change D are Step 8 and Step 11.

Step 8 does not change d_n, but decreases D_R by a multiplicative factor ≥ 4. Indeed, since $j \geq z + 2$, at least one nonzero vector is moved and a zero vector is inserted at index $z + 1$.

Step 11 changes $\mathbf{b}_{j-1}^*$ and $\mathbf{b}_j^*$, but preserves the other $\mathbf{b}_i^*$'s. There are two cases:

- If $\mu_{j,j-1} \neq 0$, then the new $\mathbf{b}_{j-1}^*$ is $\mu_{j,j-1}\mathbf{b}_{j-1}^*$ and the new $\mathbf{b}_j^*$ is zero. Thus, d_n decreases by a multiplicative factor ≥ 4 (because $0 < \mu_{j,j-1}^2 \leq 1/4$ by size-reduction) and D_R remains.
- Otherwise $\mu_{j,j-1} = 0$, then the new $\mathbf{b}_{j-1}^*$ is zero and the new $\mathbf{b}_j^*$ is $\mathbf{b}_{j-1}^*$. Thus, it preserves d_n and decreases D_R by a multiplicative factor 4.

It follows that each Step 5 decreases D by a multiplicative factor ≥ 4. Since $D \geq D_R \geq 2^{(n-d+1)(n-d)}$ throughout Algorithm 11, the number of `while` loop iterations is at most $\log_4(\|A\|^{2d} 2^{2d(n-d)})$.

Next, consider the operands. The refinement ensures that only integers occur during Algorithm 11. By Invariant 2, we always have $\max_{1 \leq i \leq n} \|\mathbf{b}_i\| \leq \sqrt{d} \times \|A\|$ due to the size-reduction at Step 6. Then the integral GSO d_k's and $\lambda_{i,k}$'s have size $O(d \log \|A\|)$ except Step 6, and the occurring integers including intermediate $\{\lambda_{j,k}\}_{j>k\geq z+1}$ during Step 6 have size $O(d(\log d + \log \|A\|))$ by Lemma A.2. Therefore, Algorithm 11 uses operands of size $O(d(\log d + \log \|A\|))$.

It remains to bound the cost. Step 6 costs $O(md\frac{\log d\|A\|}{\log \|A\|} \cdot \mathcal{M}(d \log \|A\|))$ by Lemma A.2, in which the worst-case multiplications involve operands of bit-sizes $O(d \log d\|A\|)$ and $O(d \log \|A\|)$. The cost of Step 11 is $O(n \cdot \mathcal{M}(d \log \|A\|))$. Step 8 does not involve integer multiplications. Since Step 5 occurs at most $d(n - d + \log \|A\|)$ times, we conclude that Algorithm 11 requires $O((md + n)d(n - d + \log \|A\|)\frac{\log d\|A\|}{\log \|A\|} \cdot \mathcal{M}(d \log \|A\|))$ bit operations: it bounds the cost of Algorithm 9 as well, since $d(n - d) \geq n - 1$ for $1 \leq d \leq n - 1$. This completes the proof. □

B Proofs of Section 2

The proof of Lemma 2.2 relies on the following claim.

Claim B.1. *Let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_d) \in \mathbb{R}^{d \times m}$ be a lattice basis with GSO matrices $D = Diag(\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_d^*\|)$ and $\mu = (\mu_{i,j})_{1 \leq i,j \leq d}$. Then the inverse $\mu^{-1} = (\eta_{i,j})_{1 \leq i,j \leq d}$ is a unit lower triangular matrix with*

$$|\eta_{i,j}| \leq \frac{2^{i-j-1}\|B\|^{i-j}}{\prod_{k=j}^{i-1} \|\mathbf{b}_k^*\|} \quad \text{for } 1 \leq j < i \leq d.$$

Proof. It is classical that μ is unit lower triangular and so is μ^{-1}. Since $\mu\mu^{-1} = I_d$ implies $\sum_{k=j}^{i} \mu_{i,k}\eta_{k,j} = 0$ for $1 \leq j < i \leq n$, it can be checked by induction on $l = i - j$ that

$$\eta_{i,j} = -\mu_{i,j} + \sum_{j<t<i} \mu_{i,t}\mu_{t,j} - \sum_{j<s<t<i} \mu_{i,t}\mu_{t,s}\mu_{s,j} + \cdots + (-1)^{i-j}\mu_{i,i-1}\mu_{i-1,i-2}\cdots\mu_{j+2,j+1}\mu_{j+1,j} \text{ for } 1 \leq j < i \leq d. \quad \text{(B.1)}$$

Note that $\max\{1, |\mu_{s,t}|\} \le \frac{\|B\|}{\|\mathbf{b}_t^*\|}$ for $1 \le t \le s \le d$. For indices $j = j_0 < j_1 < \cdots < j_\ell < i$, we have

$$|\mu_{i,j_\ell}\mu_{j_\ell,j_{\ell-1}}\cdots\mu_{j_2,j_1}\mu_{j_1,j}| \le \frac{\|B\|^{\ell+1}}{\prod_{k=0}^{\ell}\|\mathbf{b}_{j_k}^*\|} \le \frac{\|B\|^{i-j}}{\prod_{k=j}^{i-1}\|\mathbf{b}_k^*\|} \quad \text{for } 1 \le \ell < i-j \le d-1.$$

Applying this to the right-hand terms of Eq. (B.1), we obtain

$$|\eta_{i,j}| \le \left(\binom{i-j-1}{0} + \binom{i-j-1}{1} + \cdots + \binom{i-j-1}{i-j-1}\right)\frac{\|B\|^{i-j}}{\prod_{k=j}^{i-1}\|\mathbf{b}_k^*\|} = \frac{2^{i-j-1}\|B\|^{i-j}}{\prod_{k=j}^{i-1}\|\mathbf{b}_k^*\|}$$

for $1 \le j < i \le d$, where $\binom{m}{n} = \frac{m!}{n!(m-n)!}$ denotes the binomial coefficient. This completes the proof. □

Proof of Lemma 2.2. First, $W = \det(BB^t)(BB^t)^{-1}B \in \mathbb{Z}^{d\times m}$ has rank d, because $\det(BB^t)(BB^t)^{-1} \in \mathbb{Z}^{d\times d}$ is nonsingular.

We next bound the $\|\mathbf{w}_i\|$'s. Consider the Gram-Schmidt decomposition $B = \mu DQ$, where $Q = (\frac{\mathbf{b}_1^*}{\|\mathbf{b}_1^*\|}, \ldots, \frac{\mathbf{b}_d^*}{\|\mathbf{b}_d^*\|})$ is an orthogonal set, $D = \mathrm{Diag}(\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_d^*\|)$, and $\mu = (\mu_{i,j})_{1\le i,j\le d}$ is a unit lower triangular matrix. Then $\mu^{-1} = (\eta_{i,j})_{1\le i,j\le d}$ is unit lower triangular, $\delta \triangleq \det(BB^t) = \prod_{k=1}^{d}\|\mathbf{b}_k^*\|^2$ and $B^\times = \mu^{-1}D^{-1}Q$. This implies

$$W = \det(BB^t)B^\times = \delta\mu^{-1}(\frac{\mathbf{b}_1^*}{\|\mathbf{b}_1^*\|^2}, \ldots, \frac{\mathbf{b}_d^*}{\|\mathbf{b}_d^*\|^2}) \text{ with } \mathbf{w}_i = \delta(\eta_{i,1}\frac{\mathbf{b}_1^*}{\|\mathbf{b}_1^*\|^2} + \cdots + \eta_{i,i}\frac{\mathbf{b}_i^*}{\|\mathbf{b}_i^*\|^2}) \text{ for } i = 1, \ldots, d.$$

We have $\frac{\delta|\eta_{i,i}|}{\|\mathbf{b}_i^*\|} \le \|B\|^{2d-1}$ for $i = 1, \ldots, d$. Claim B.1 implies $\frac{\delta|\eta_{i,j}|}{\|\mathbf{b}_j^*\|} \le 2^{i-j-1}\|B\|^{i-j}(\frac{\delta}{(\prod_{k=j}^{i-1}\|\mathbf{b}_k^*\|)\|\mathbf{b}_j^*\|}) \le 2^{i-j-1}\|B\|^{2d-1}$ for $1 \le j < i \le d$. It follows that $\|\mathbf{w}_i\| \le \sum_{j=1}^{i}\frac{\delta|\eta_{i,j}|}{\|\mathbf{b}_j^*\|} \le 2^{i-1}\|B\|^{2d-1}$ for $i = 1, \ldots, d$.

It remains to analyze the complexity. Since $BB^t \in \mathbb{Z}^{d\times d}$ is nonsingular and $W = \det(BB^t)(BB^t)^{-1}B$, it is folklore to compute W deterministically in time $O(md^{\theta-1}\cdot\mathcal{B}(d\log\|B\|))$: we can efficiently compute $\det(BB^t)$ and $(BB^t)^{-1}$ by calculating their residues modulo small primes (say, $\le 2^d\|B\|^{2d}$) and then recovering the final results using Chinese Remainder Theorem; this approach is classical (see, *e.g.*, [KV04, Vil03]). The operands during the computation have size $O(d\log\|B\|)$. This completes the proof. □

Proof of Lemma 2.3. Let $H = (h_{i,j})$ and let $i_1 < \cdots < i_d$ be indices corresponding to H^t's support. Note that for $j = 1, \ldots, d$, $h_{j,i_j} > 0$ and $h_{j,k} = 0$ if $k < i_j$. Then $\underline{H^t} \in \mathbb{Z}^{d\times d}$ is lower-triangular with positive diagonal entries.

We show Item 1. Note that A has rank d and there is a matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_d) \in \mathbb{Z}^{d\times m}$ of rank d such that $A^t = B^tH$, then $H^\times H^t = I_d$ implies $H^\times A = B$. This proves Item 1.

We show Item 2. Let $A = (\mathbf{a}_1, \ldots, \mathbf{a}_n)$. Since $A = H^tB$, we have:

- $\mathbf{a}_\ell = \mathbf{0}$ for $\ell < i_1$ and $\mathbf{a}_{i_1} = h_{1,i_1}\mathbf{b}_1$;
- For $j = 2, \ldots, d$, $\mathbf{a}_s = h_{1,s}\mathbf{b}_1 + \cdots + h_{j-1,s}\mathbf{b}_{j-1}$ over $i_{j-1} < s < i_j$ and $\mathbf{a}_{i_j} = h_{1,i_j}\mathbf{b}_1 + \cdots + h_{j-1,i_j}\mathbf{b}_{j-1} + h_{j,i_j}\mathbf{b}_j$.

It can be checked by induction on j that $\mathbf{a}_{i_j}^* = h_{j,i_j}\mathbf{b}_j^*$ for $j = 1, \ldots, d$. Then the indices $i_1 < \cdots < i_d$ also correspond to A's support. This proves Item 2.

Item 3 holds. Indeed, $A = H^tB$ implies $\underline{A} = \underline{H^t}B$. Then $H^\times A = B = (\underline{H^t})^{-1}\underline{A}$. We verify $H^\times A \preceq A$ as follows:

- The equality $A = H^t(H^\times A)$ implies the sublattice inclusion;
- Since $(\underline{H^t})^{-1}$ is a t-matrix, the equality $H^\times A = (\underline{H^t})^{-1}\underline{A}$ implies the distance decrease.
- The shifted supports are trivial, because $H^\times A \in \mathbb{Z}^{d\times m}$ has rank d.

We show Item 4. Note that $\mathbf{b}_j = (\mathbf{a}_{i_j} - \sum_{k=1}^{j-1}h_{k,i_j}\mathbf{b}_k)/h_{j,i_j}$ implies $\|\mathbf{b}_j\|_\infty \le \|\mathbf{a}_{i_j}\|_\infty + \sum_{k=1}^{j-1}\|\mathbf{b}_k\|_\infty$ for $j = 2, \ldots, d$. It can be shown by induction on j that $\|\mathbf{b}_j\|_\infty \le 2^{j-1}\times\|\underline{A}\|_\infty$ for $j = 1, \ldots, d$. Let M be a maximal column independent system of $\underline{A}$. Then $M \in \mathbb{Z}^{d\times d}$ is nonsingular. Note that $(\underline{H^t})^{-1}\underline{A} \in \mathbb{Z}^{d\times m}$ implies $V \triangleq (\underline{H^t})^{-1}M \in \mathbb{Z}^{d\times d}$, then $\det(M) = \det(\underline{H^t})\times\det(V)$. Thus, $\|\underline{H^t}\|_\infty \le \det(\underline{H^t}) \le \det(M) \le \|\underline{A}\|^d$. This proves Item 4 and hence the lemma. □

C Proofs of Section 3

Proof of Theorem 3.2. Since the matrix E at Step 2 is the HNF-basis of $L(A)$ and the support S at Step 1 is a linearly independent set such that $L(S) \subseteq L(A)$, the correctness of the algorithm follows from the proof of Theorem 3.1.

Algorithm 1 runs on integers of size $O(d(\log d + \log\|A\|))$. Indeed, $\|E\| \le \det(L(A)) \le \|S\|^d$ and $|\det(M)| \le |\det(\underline{S^t})| \le \|S\|^d$. Note that $M = (E^\times S^t)^t$, Lemma 2.3 implies $\|M\|_\infty \le 2^{d-1}\times\|S\|_\infty$. Consider Step 5: $S = F^tB$ implies $\|F\| \le \det(F) \le \|S\|^d$ and $\|B\| \le 2^{d-1}\times\|S\|$. Since $\|B^*\| \le \|S^*\|$, all integral GSO of Gram(B) at Step 6 have size $O(d\log\|S\|)$. Yet the occurring integral GSO of Gram(B) at Step 7 have size $O(d(\log d + \log\|S\|))$ by Lemma A.2.

We bound the cost. Step 1 and Step 6 totally cost $O(mnd\cdot\mathcal{M}(d\log\|A\|))$. From [Sto00, Chap. 6], Step 2 costs $O(mnd^{\theta-1}\log\|A\|_\infty\cdot\mathcal{M}(\log\log\alpha) + mn(\log d)\cdot\mathcal{B}(\log\alpha))$, where $\alpha = (\sqrt{d}\|A\|_\infty)^d$; by [SL96, Theorem 5], Step 4 costs $O(d^\theta\cdot\mathcal{B}(\log\gamma))$ where $\gamma = \max\{\|M\|_\infty, |\det(\underline{S^t})|\}$. Step 3 and Step 5 solve two triangular systems $\underline{S^t} = \underline{E^t}X$ and $S = F^tX$, respectively: by Lemma A.3, they totally cost $O(md\cdot\mathcal{M}(d\log\|S\|) + md^{\theta-1}(\log d)\frac{d\log\|S\|}{d+\log\|S\|}\cdot\mathcal{M}(d+\log\|S\|))$.

By Lemma A.2, Step 7 costs $O(md^2(\frac{\log d\|S\|}{\log\|S\|}) \cdot \mathcal{M}(d\log\|S\|))$. It follows that Algorithm 1 requires $O(mnd(\frac{\log d\|A\|}{\log\|A\|}) \cdot \mathcal{B}(d\log\|A\|))$ bit operations. This completes the proof. □

Proof of Theorem 3.4. We first show correctness of Algorithm 2. Let $T = S/\delta$. Then $B = T\underline{A}$ and $C = (T - \mathsf{Round}(T))\underline{A}$. It follows that $C^* = B^*$, $C \preceq B \preceq A$ and C is A-reduced. This proves the correctness.

We analyze the complexity. Algorithm 2 runs on integers of size $O(\log\alpha)$: indeed, Cauchy-Schwartz's inequality and Lemma 2.2 imply that $\|R\|_\infty \le \|S\|_\infty \le \|B\| \cdot \|W\| \le 2^{d-1}\|\underline{A}\|^{2d-1} \cdot \|B\|$. Then Step 3 takes $O(md^{\theta-1} \cdot \mathcal{M}(\log\alpha))$ bit operations, which bounds the cost of other steps as well by Lemma 2.2. This proves Theorem 3.4. □

Proof of Theorem 3.7. We first show correctness of Algorithm 3. By Lemma 2.3.3, the matrix B appearing at Step 2 is $H^\times A \preceq A$, which justifies Step 3. By Theorem 3.4, the output matrix $B \in \mathbb{Z}^{d\times m}$ is an A-reduced basis of $L(H^\times A)$ such that $B \preceq H^\times A$. This proves the correctness.

It remains to analyze the complexity. By Lemma 2.3.4, $\log\|(\underline{H^t})^{-1}\,\underline{A}\|_\infty \in O(d + \log\|A\|)$ and $\log\|\underline{H^t}\|_\infty \in O(d\log\|A\|)$. Note that $\frac{d(\log d)\log\|A\|}{d+\log\|A\|} \cdot \mathcal{M}(d + \log\|A\|) \in \frac{\log d\|A\|}{\log\|A\|} \cdot \mathcal{B}(d\log\|A\|)$, it follows from Lemma A.3 that Step 2 takes $O(md^{\theta-1}(\frac{\log d\|A\|}{\log\|A\|}) \cdot \mathcal{B}(d\log\|A\|))$ bit operations and runs on integers of size $O(d\log\|A\|)$, which bound Step 3 as well by Theorem 3.4. This completes the proof. □

Proof of Theorem 3.8. Given as input A, Algorithm 4 computes an A-reduced basis of $L(A)$. Its correctness follows from Theorems 3.5 and 3.7. It remains to analyze the complexity. From [Sto00, Chap. 6], Step 1 and Step 3 totally cost $O(mnd^{\theta-1}\log\max\{\|A\|_\infty, \|C\|_\infty\} \cdot \mathcal{M}(\log\log\alpha) + mn\log d \cdot \mathcal{B}(\log\alpha))$ and run on integers of size $O(\log\alpha)$, where $\alpha = (\sqrt{d} \times \max\{\|A\|_\infty, \|C\|_\infty\})^d$. By Theorem 3.7, Step 2 and Step 4 totally cost $O((m+n)d^{\theta-1}(\frac{\log d\beta}{\log\beta}) \cdot \mathcal{B}(d\log\beta))$ and run on integers of size $O(d\log\beta)$, where $\beta = \max\{\|A\|, \|A^t\|\}$. Since $\|C\|_\infty \le d\|A^t\|_\infty \le d\|A\|$ (by Lemma 3.3.2) and $\|A^t\| \le \sqrt{n}\|A\|$, we conclude that Algorithm 4 requires $O(mn(d^{\theta-2} + \log d) \cdot \mathcal{B}(d(\log n + \log\|A\|)))$ bit operations and runs on integers of size $O(d(\log n + \log\|A\|))$ during execution.

Note that if given as input A^t, the first two steps of Algorithm 4 actually compute an A-reduced basis of $\ker_{\mathbb{Z}}(A^t)^\perp = \overline{L(A)}$. This completes the proof. □

Proof of Corollary 3.9. Since $L(A) = L$, the claim follows from Theorem 3.8 and Lemma 3.3.3. □

Proof of Theorem 3.10. The combination of Algorithm 4 and Algorithm 5 is as desired. From Theorem 3.8, the only issue is to discuss the correctness and complexity of Algorithm 5, which is done below.

It is obvious that the output matrix U of Algorithm 5 is unimodular and satisfies Item 1.

We verify Item 2. Let $\alpha = (\sqrt{d}\|A\|_\infty)^d$. Let V_1 and V_2 denote the first $(n-d)$ rows and bottom d rows of matrix V at Step 1 of Algorithm 5, respectively. It follows from [Sto00, Chap. 6] that $\|V_1\|_\infty \le d\alpha^2$ and $\|V_2\|_\infty \le \alpha$. Let W_i denote the ith column of matrix W at Step 2 of Algorithm 5 for $i = 1, \ldots, d$. Since $W = (E^\times B^t)^t$, Lemma 2.3 implies $\|W_i\|_\infty \le 2^{i-1}\|B\|_\infty$ for $i = 1, \ldots, d$. Combining with Lemma 3.3.2, we have $\|WV_2\|_\infty \le \sum_{i=1}^d 2^{i-1}\|B\|_\infty\alpha < 2^d d\|A\|_\infty\alpha$. Note that $U = \begin{pmatrix} V_1 \\ WV_2 \end{pmatrix}$, this proves Item 2.

It remains to analyze the complexity. It follows from [Sto00, Chap. 6] that Step 1 of Algorithm 5 runs on integers of size $O(\log\alpha)$ and takes $O(mnd^{\theta-1}(\log 2n/d)\log\|A\|_\infty \cdot \mathcal{M}(\log\log\alpha) + mn(\log n) \cdot \mathcal{B}(\log\alpha))$ bit operations. By Lemma A.3, Algorithm 5 needs $O(mn(d^{\theta-2}\log\frac{2n}{d} + \log n) \cdot \mathcal{B}(d(\log n + \log\|A\|)))$ bit operations and runs on integers of size $O(d(\log n + \log\|A\|))$. Combining with Theorem 3.8, this proves Theorem 3.10. □

D Proofs of Sections 4.1 and 4.2

Proof of Theorem 4.1. The correctness of Algorithm 6 follows from the two facts below: the transformation at Step 2 is unimodular; the new $\mathbf{b}^*_{i-1}$ is $\mathbf{0}$ and the new $\mathbf{b}^*_i$ is $\frac{g}{d_{i-1}}\mathbf{b}^*_{i-1}$, while the other $\mathbf{b}^*_k$'s remain unchanged.

We analyze the complexity. Since the d_j's and $\lambda_{j,k}$'s have size $O(d\log\|B\|)$ during the algorithm, this implies: Steps 1 and 2 totally cost $O(\mathcal{B}(d\log\|B\|) + md \cdot \mathcal{M}(\log\|B\|))$; Steps 3-12 need $O(nd)$ arithmetic operations, then the cost is $O(nd \cdot \mathcal{M}(d\log\|B\|))$. The conclusion follows. □

Proof of Theorem 4.2. We first show correctness of Algorithm 7. To do so, note that the following loop invariants hold at Steps 3-13, which connect the input matrix A with current matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$:

1. $2 \le j - z \le d + 1$;
2. $L(A) = L(B)$;
3. $B \preceq A$;
4. $(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1})$ has fingerprint $(0^z 1^{j-z-1})_{\mathcal{F}}$ outside of Steps 7-10.

Invariants 1 and 2 are obvious. Invariant 3 holds by transitivity: each Euclid-swap can only lower the matrix, and size-reduction does not change the partial order. Invariant 4 can be shown by induction on (z, j) using the fact: only Step 8 changes the $\mathbf{b}^*_k$'s, which transforms $(\mathbf{b}^*_{i-1}, \mathbf{b}^*_i) = (\mathbf{b}^*_{i-1}, \mathbf{0})$ into $(\mathbf{0}, c\mathbf{b}^*_{i-1})$ for some $c \in (0, 1]$ and preserves the other $\mathbf{b}^*_k$'s.

Now, if the algorithm terminates, then $j = n + 1$: it follows from Invariants 2-4 that the output vectors $\mathbf{b}_{z+1}, \ldots, \mathbf{b}_n$ form a basis of $L(A)$ such that $(\mathbf{b}_{z+1}, \ldots, \mathbf{b}_n) \preceq A$.

Next, consider the operands. Algorithm 7 runs on integers. Due to the size-reduction at Step 9 and $\|B^*\| \le \|A^*\|$ (by Invariant 3), we always have $\|B\| \le \sqrt{d} \times \|A\|$ except "in" Steps 8-9. Then

- Step 8 outputs the new $\mathbf{b}_{i-1}$ and $\mathbf{b}_i$ with norm $\leq 2d\|A\|^{2d+1}$;
- The integral GSO d_k's and $\lambda_{k,l}$'s have size $O(d \log \|A\|)$ except Step 9;
- All occurring integers during Step 9 have size $O(d(\log d + \log \|A\|))$ by Lemma A.2.

It follows that all occurring integers during Algorithm 7 have size $O(d(\log d + \log \|A\|))$.

We bound the cost. Each Step 8 costs $O(nd \cdot \mathcal{B}(d \log \|A\|) + md\frac{\log d\|A\|}{\log \|A\|} \cdot \mathcal{M}(\log \|A\|))$, since its input vectors have norm $\leq \sqrt{d} \times \|A\|$ and input integral GSO have size $O(d \log \|A\|)$. Each Step 9 costs $O(md\frac{\log d\|A\|}{\log \|A\|} \cdot \mathcal{M}(d \log \|A\|))$ by Lemma A.2, in which the worst-case multiplications involve operands of bit-sizes $O(d \log d\|A\|)$ and $O(d \log \|A\|)$. Since Steps 8-9 occur at most $d(n-d)$ times, we conclude that Algorithm 7 requires $O(m(n-d+1)d^2(\frac{\log d\|A\|}{\log \|A\|}) \cdot \mathcal{M}(d \log \|A\|) + n(n-d)d^2 \cdot \mathcal{B}(d \log \|A\|))$ bit operations. This proves Theorem 4.2. □

E Proof of Theorem 4.3

We follow the notation of Algorithm 8 and additionally use the notation below through this section:

- Let P and P^{cur} denote the input and current positive semi-definite integral matrix during execution.
- Let $\|P\|_*$ denote the invariant $\|B^*\|^2$ for any decomposition $P = BB^t$. Then $\|P\|_* \leq \|P\|_\infty$.
- Let $U = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ denote the current unimodular transformation during Algorithm 8.

We first illustrate three technical lemmas below, which ensures correctness and efficiency of Algorithm 8.

Lemma E.1. *The following loop invariants hold during Steps 3-15 of Algorithm 8:*

1. $2 \leq j - z \leq d + 1$;
2. $\|P^{cur}\|_* \leq \|P\|_*$;
3. $\|P^{cur}\|_\infty \leq d \times \|P\|_\infty$ *right after Step 12;*
4. P^{cur} *has fingerprint* $(0^z 1^{j-z-1} 0*)_{\mathcal{F}}$ *right before Step 7;*
5. U *is a unimodular matrix such that* $UPU^t = P^{cur}$;
6. $span(\mathbf{u}_1, \ldots, \mathbf{u}_j) \subseteq span(\mathbf{e}_1, \ldots, \mathbf{e}_j)$ *and* $\mathbf{u}_i = \mathbf{e}_i$ *for* $i = j+1, \ldots, n$, *where* $(\mathbf{e}_1, \ldots, \mathbf{e}_n) = I_n$.

Proof. Items 1-6 can be checked by induction on z. □

We quantize consecutive Euclid-swaps during Algorithm 8 as follows.

Lemma E.2. *Using the notation of Algorithm 8. Let* $\Phi(z, i) \in \mathbb{Z}^{n \times n}$ *denote the corresponding unimodular transformation of Euclid-swap w.r.t. index pair* $(i-1, i)$ *when Steps 7-9 occur w.r.t. index z. Then*

1. *The corresponding unimodular transformation for consecutive Euclid-swaps at Steps 7-9 w.r.t. index z is*

$$\Phi(z, z+2) \cdot \Phi(z, z+3) \cdot \cdots \cdot \Phi(z, j) = \begin{pmatrix} I_z & & \\ & \phi(z) & \\ & & I_{n-j} \end{pmatrix}$$

for some $(j-z) \times (j-z)$ *unimodular matrix* $\phi(z)$ *with*

$$\max\{\|\Phi(z, z+2)\|_\infty, \ldots, \|\Phi(z, j)\|_\infty, \|\phi(z)\|_\infty\} \leq (d \cdot \|P\|_* \cdot \|P\|_\infty)^{d/2}.$$

2. *The matrix* P^{cur} *during Steps 7-9 always has magnitude:* $\|P^{cur}\|_\infty \leq d^{d+3} \times \|P\|_\infty^{2d+1}$.
3. *All integral GSO occurring during these steps have size* $O(d(\log d + \log \|P\|_\infty))$.

Algorithm 8 allows to explicitly trace the evolution of current unimodular transformation:

Lemma E.3. *Let* $P^{(z)}$ *and* $U^{(z)}$ *denote respectively the current positive semi-definite matrix and current unimodular transformation right before Step 7 w.r.t. index z. Let* $V^{(z+1)} = \begin{pmatrix} I_z & & \\ & W^{(z+1)} & \\ & & I_{n-j} \end{pmatrix} \in \mathbb{Z}^{n \times n}$ *be the corresponding unimodular transformation of Steps 7-12 w.r.t. index z such that* $V^{(z+1)} P^{(z)} (V^{(z+1)})^t = P^{(z+1)}$. *For* $z = 0, \ldots, n-d-1$, *we have:*

$$U^{(z+1)} = V^{(z+1)} U^{(z)} \text{ with } \|V^{(z+1)}\|_\infty \leq d^{2d-1}(2\|P\|_\infty^{2.5})^d \text{ and } \|U^{(z+1)}\|_\infty \leq (2d^2 \times \|P\|_\infty^{2.5})^d \|U^{(z)}\|_\infty.$$

We will prove Lemmas E.2 and E.3 in Appendices E.1 and E.2, respectively.

Theorem 4.3 immediately follows from Lemmas E.1-E.3.

Proof of Theorem 4.3. Lemma E.1 ensures correctness of Algorithm 8. The main issue is to analyze the complexity. From Lemma E.3 together with its notation, since $U^{(0)} = I_n$, we have

$$\|U^{(z+1)}\|_\infty \le (2d^2 \times \|P\|_\infty^{2.5})^{d(z+1)} \text{ for } z = 0, \dots, n-d-1.$$

Then Algorithm 8 eventually outputs a unimodular matrix $U = U^{(n-d)}$ with $\|U\|_\infty \le (2d^2 \times \|P\|_\infty^{2.5})^{d(n-d)}$. Combining with Lemmas E.2 and A.2, Items 2-3 of Lemma E.1 ensures that all entries of P^{cur} and all integral GSO occurring during the computation have size $O(d(\log d + \log \|P\|_\infty))$. It follows that Steps 7-12 w.r.t. index z transform $U^{(z)}$ into $U^{(z+1)}$ within $O((z+1)nd^{\theta-1} \cdot \mathcal{B}(d(\log d + \log \|P\|_\infty)))$ bit operations using matrix multiplication and run on integers of size $O((z+1)d(\log d + \log \|P\|_\infty))$. Since the computation of unimodular transformation dominates the global cost of Algorithm 8, it totally takes $O(n(n-d+1)(d^2 + d^{\theta-1}(n-d)) \cdot \mathcal{B}(d(\log d + \log \|P\|_\infty)))$ bit operations and runs on integers of size $O(d(n-d+1)(\log d + \log \|P\|_\infty))$ during execution. This completes the proof of Theorem 4.3. □

E.1 Proof of Lemma E.2

We now show Lemma E.2. For simplicity, we tackle the case $(z, j) = (0, d+1)$ in the language of lattices. Consider a matrix $A = (\mathbf{a}_1, \dots, \mathbf{a}_{d+1}) \in \mathbb{R}^{(d+1)\times m}$ such that $\text{Gram}(A) \in \mathbb{Z}^{(d+1)\times(d+1)}$ has fingerprint $(1^d 0)_{\mathcal{F}}$. The argument of Lemma E.2 is reduced to the easier setting for analyzing the unimodular transformation of the following procedure:

Algorithm 12 Consecutive Euclid-swaps

1: **for** $i = d+1$ downto 2 **do**
2: Euclid-swap $(\mathbf{a}_{i-1}, \mathbf{a}_i)$ using Algorithm 6
3: **end for**

Let $A_i = (\mathbf{a}_1, \dots, \mathbf{a}_{i-1}, \mathbf{a}_{d+1}, \mathbf{a}_{i+1}, \dots, \mathbf{a}_d)$ for $i = 1, \dots, d$. Define the integers:

$$y_{d+1} = \det(\underline{A} \cdot \underline{A}^t), \quad x_i = \det(\underline{A} \underline{A}_i^t) \text{ and } y_i = \gcd(y_{d+1}, x_d, \dots, x_i) \text{ for } i = d, \dots, 1.$$

Such integers can be bounded well: $|x_i| \le \sqrt{\det(\underline{A} \cdot \underline{A}^t) \det(A_i A_i^t)} \le (\|A^*\| \cdot \|A\|)^d$ for $i = 1, \dots, d$ and $1 \le y_1 \le \cdots \le y_{d+1} \le \|A^*\|^{2d}$. By Cramer's rule, $(\frac{x_1}{y_{d+1}}, \dots, \frac{x_d}{y_{d+1}})$ is the unique solution to linear system $\mathbf{x}\underline{A} = \mathbf{a}_{d+1}$:

$$\mathbf{a}_{d+1} = \frac{x_d}{y_{d+1}}\mathbf{a}_d + \frac{x_{d-1}}{y_{d+1}}\mathbf{a}_{d-1} + \cdots + \frac{x_1}{y_{d+1}}\mathbf{a}_1.$$

Then $\mathbf{b}_i \triangleq \frac{x_{i-1}}{y_i}\mathbf{a}_{i-1} + \cdots + \frac{x_1}{y_i}\mathbf{a}_1 = \frac{1}{y_i}(y_{d+1}\mathbf{a}_{d+1} - x_d\mathbf{a}_d - \cdots - x_i\mathbf{a}_i)$ with $\|\mathbf{b}_i\| \le d\|A^*\|^d\|A\|^{d+1}$ for $i = 2, \dots, d+1$.

Let d_i's and $\lambda_{i,k}$'s be the integral GSO of Gram(A). Let $\lambda_{i,k}^{(\text{b})} = d_{k-1}\langle \mathbf{a}_k^*, \mathbf{b}_i \rangle$ for $1 \le k < i \le d+1$. Note that the $\lambda_{i,k}^{(\text{b})}$'s are integers, we find integers (s_i, t_i, g_{i-1}) using XGCD such that $s_i d_{i-1} + t_i \lambda_{i,i-1}^{(\text{b})} = g_{i-1} = \gcd(d_{i-1}, \lambda_{i,i-1}^{(\text{b})})$ with $|s_i| \le \max\{1, |\lambda_{i,i-1}^{(\text{b})}|/g_{i-1}\}$ and $|t_i| \le d_{i-1}/g_{i-1}$ for $i = d+1, \dots, 2$. There are close relations among such scalars:

Claim E.4. *Under the above notation. For $i = d+1, d, \dots, 2$, we have:*

1. $s_i y_i + t_i x_{i-1} = \gcd(x_{i-1}, y_i) = y_{i-1}$ *with* $|s_i| \le \max\{1, |x_{i-1}|/y_{i-1}\}$ *and* $|t_i| \le y_i/y_{i-1}$;
2. $y_{i-1} = y_i g_{i-1}/d_{i-1}$ *and* $x_{i-1} = y_i \lambda_{i,i-1}^{(b)}/d_{i-1}$, *where* $y_{d+1} = d_d$;
3. $\lambda_{i-1,k}^{(b)} = -(\lambda_{i,i-1}^{(b)}/g_{i-1})\lambda_{i-1,k} + (d_{i-1}/g_{i-1})\lambda_{i,k}^{(b)}$ *with* $|\lambda_{i-1,k}^{(b)}| \le d\|A^*\|^{3d-3}\|A\|^{d+1}$ *for* $1 \le k \le i-2$.

Such relations are based on an elementary fact related to XGCD:

Fact E.5. *Let a, b, c and d be integers such that $ac \ne 0$ and $\frac{b}{a} = \frac{d}{c}$. If the integers x and y satisfy $xa + yb = \gcd(a, b)$, then $xc + yd = \gcd(c, d)$ with $\frac{a}{\gcd(a,b)} = \frac{c}{\gcd(c,d)}$ and $\frac{b}{\gcd(a,b)} = \frac{d}{\gcd(c,d)}$.*

Proof of Claim E.4. Items 1 and 2 follow from Fact E.5, since $\frac{x_{i-1}}{y_i} = \frac{\langle \mathbf{a}_{i-1}^*, \mathbf{b}_i \rangle}{\langle \mathbf{a}_{i-1}^*, \mathbf{a}_{i-1}^* \rangle} = \frac{\lambda_{i,i-1}^{(\text{b})}}{d_{i-1}}$. Then Item 2 implies

$$\mathbf{b}_{i-1} = -\frac{x_{i-1}}{y_{i-1}}\mathbf{a}_{i-1} + \frac{y_i}{y_{i-1}}\mathbf{b}_i = -\frac{\lambda_{i,i-1}^{(\text{b})}}{g_{i-1}}\mathbf{a}_{i-1} + \frac{d_{i-1}}{g_{i-1}}\mathbf{b}_i.$$

Let $k \in [1, i-2]$. Since $\lambda_{i-1,k}^{(\text{b})} = d_{k-1}\langle \mathbf{a}_k^*, \mathbf{b}_{i-1} \rangle$, this implies $\lambda_{i-1,k}^{(\text{b})} = -(\lambda_{i,i-1}^{(\text{b})}/g_{i-1})\lambda_{i-1,k} + (d_{i-1}/g_{i-1})\lambda_{i,k}^{(\text{b})}$ with $|\lambda_{i-1,k}^{(\text{b})}| \le d_{k-1} \cdot \|\mathbf{a}_k^*\| \cdot \|\mathbf{b}_{i-1}\|$. Then $|\lambda_{i-1,k}^{(\text{b})}| \le d\|A^*\|^{3d-3}\|A\|^{d+1}$. This proves Item 3 and hence Claim E.4. □

By Claim E.4, it can be checked that Procedure 12 in fact transforms vector pair $(\mathbf{a}_{i-1}, \mathbf{b}_i)$ into $(\mathbf{b}_{i-1}, s_i\mathbf{a}_{i-1} + t_i\mathbf{b}_i)$ in turn over $i = d+1, d, \ldots, 2$, using the corresponding unimodular transformation:

$$U_i \triangleq \begin{pmatrix} I_{i-2} & & & \\ & -\lambda^{(\mathrm{b})}_{i,i-1}/g_{i-1} & d_{i-1}/g_{i-1} & \\ & s_i & t_i & \\ & & & I_{d+1-i} \end{pmatrix} = \begin{pmatrix} I_{i-2} & & & \\ & -x_{i-1}/y_{i-1} & y_i/y_{i-1} & \\ & s_i & t_i & \\ & & & I_{d+1-i} \end{pmatrix}.$$

The product of the U_i's is the unimodular transformation for consecutive Euclid-swaps of Procedure 12:

$$\phi \triangleq U_2 \cdot U_3 \cdot \cdots \cdot U_{d+1} = \begin{pmatrix} -\frac{x_1}{y_1} & -\frac{x_2}{y_1} & -\frac{x_3}{y_1} & \cdots & \cdots & -\frac{x_{d-1}}{y_1} & -\frac{x_d}{y_1} & \frac{y_{d+1}}{y_1} \\ s_2 & -\frac{t_2x_2}{y_2} & -\frac{t_2x_3}{y_2} & \cdots & \cdots & -\frac{t_2x_{d-1}}{y_2} & -\frac{t_2x_d}{y_2} & \frac{t_2y_{d+1}}{y_2} \\ 0 & s_3 & -\frac{t_3x_3}{y_3} & \cdots & \cdots & -\frac{t_3x_{d-1}}{y_3} & -\frac{t_3x_d}{y_3} & \frac{t_3y_{d+1}}{y_3} \\ \vdots & \ddots & \ddots & \cdots & \cdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \cdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & s_{d-1} & -\frac{t_{d-1}x_{d-1}}{y_{d-1}} & -\frac{t_{d-1}x_d}{y_{d-1}} & \frac{t_{d-1}y_{d+1}}{y_{d-1}} \\ 0 & 0 & \cdots & \cdots & 0 & s_d & -\frac{t_dx_d}{y_d} & \frac{t_dy_{d+1}}{y_d} \\ 0 & 0 & 0 & \cdots & \cdots & 0 & s_{d+1} & t_{d+1} \end{pmatrix}.$$

We have $\max\{\|U_2\|_\infty, \ldots, \|U_{d+1}\|_\infty, \|\phi\|_\infty\} \leq (\|A^*\| \cdot \|A\|)^d$ and $\|\phi A\| \leq d \cdot \|A^*\|^d \cdot \|A\|^{d+1}$. Since the $\lambda^{(\mathrm{b})}_{i-1,k}$'s are actually intermediate integral GSO occurring during execution, Procedure 12 runs on integers of size $O(d \log \|A\|)$.

Applying the same analysis to a single execition of Steps 7-9 of Algorithm 8, then Lemma E.2 follows.

E.2 Proof of Lemma E.3

Proof of Lemma E.3. Consider Steps 7-12 w.r.t. index z. The equality $U^{(z+1)} = V^{(z+1)}U^{(z)}$ obviously holds.

Let $\Phi(z) \in \mathbb{Z}^{n\times n}$ and $\Psi(z) \in \mathbb{Z}^{n\times n}$ denote the corresponding unimodular transformations for consecutive Euclid-swaps of Steps 7-9 and for consecutive size-reductions of Steps 10-12, respectively.Then

$$\Phi(z) = \begin{pmatrix} I_z & & \\ & \phi(z) & \\ & & I_{n-j} \end{pmatrix} \text{ and } \Psi(z) = \begin{pmatrix} I_{z+1} & & \\ & \psi(z) & \\ & & I_{n-j} \end{pmatrix},$$

where $\phi(z) \in \mathbb{Z}^{(j-z)\times(j-z)}$ is a unimodular matrix and $\psi(z) \in \mathbb{Z}^{(j-z-1)\times(j-z-1)}$ is a unit lower triangular matrix.

We have $V^{(z+1)} = \Psi(z) \times \Phi(z)$ and $W^{(z+1)} = \begin{pmatrix} 1 & \\ & \psi(z) \end{pmatrix} \phi(z)$ with $\|\phi(z)\|_\infty \leq (d \cdot \|P\|_* \cdot \|P\|_\infty)^{d/2}$ (by Lemma E.2) .

By Item 2 of both Lemma A.2 and Lemma E.1, we deduce that

$$\|W^{(z+1)}\|_\infty \leq ((j-z-2)\|\psi(z)\|_\infty + 1) \times \|\phi(z)\|_\infty \leq 2^{d-2} d^{\frac{3}{2}(d+1)} \|P\|_*^{2d-1} \|P\|_\infty^{\frac{1}{2}(d+1)}.$$

By Item 6 of Lemma E.1, we have $\|U^{(z+1)}\|_\infty \leq d\|W^{(z+1)}\|_\infty \cdot \|U^{(z)}\|_\infty$. Then $\|U^{(z+1)}\|_\infty \leq (2d^2 \times \|P\|_\infty^{2.5})^d \|U^{(z)}\|_\infty$. This completes the proof of Lemma E.3. □

F Application: triangularizing integer matrices

Matrix triangularization is a fundamental theme in linear algebra: given an $n \times n$ nonsingular integer matrix B_0, find a lower triangular basis of the lattice $L(B_0)$. It has many applications, for instance, which can be further reduced to the HNF or SNF. It has many polynomial-time algorithms [Bla66, Bra71, KB79, CC82, DKT87, Ili89a, HM91, SL96, HMM98, Sto00, MW01]:

- For input matrices of large determinant, the previously fastest triangularization algorithm is Storjohanns HNF algorithm [Sto00, Chap. 6]. Given as input B_0, it requires $O(n^{\theta+1} \log \|B_0\|_\infty \cdot \mathcal{M}(\log\log\beta) + n^2 \log n \cdot \mathcal{B}(\log\beta))$ bit operations, where $\beta = (\sqrt{n}\|B_0\|_\infty)^n$.
- For input matrices of small determinant, the previously fastest algorithm of Hafner and McCurley [HM91] requires $O(n^\theta \cdot \mathcal{B}(\log\delta) + n^2 \cdot \mathcal{B}(\log\|B_0\|_\infty))$ bit operations if given as input B_0 and $\delta = |\det(B_0)|$.

However, the description of previous triangularization algorithms is not so simple. In this section, we present a simple triangularization algorithm by adapting our XGCD-based basis algorithm.

For simplicity, we say that a matrix is $\mathfrak{p}$-*triangular* if it is lower triangular with positive diagonal entries; for $1 \leq i \leq j \leq 2n$, a matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_{2n}) \in \mathbb{Z}^{2n\times n}$ of rank n is $\{i,j\}$-$\mathfrak{p}$-*triangular* if $\mathbf{b}_1 = \cdots = \mathbf{b}_{i-1} = \mathbf{b}_j^* = \mathbf{0}$ and $(\mathbf{b}_i, \ldots, \mathbf{b}_{j-1}, \mathbf{b}_{j+1}, \ldots, \mathbf{b}_{i+n})$ is $\mathfrak{p}$-triangular. Then $(\mathbf{b}_i, \ldots, \mathbf{b}_{j-1}, \mathbf{b}_{j+1}, \ldots, \mathbf{b}_{i+n})$ is the support of B. The ordered set $\{i,j\}$ measures the *level* of triangularization.

Both Algorithm 4 and Algorithm 7 can do matrix triangularization. Let $B_0 \in \mathbb{Z}^{n\times n}$ such that $\delta = |\det(B_0)| > 0$: simply feed the row-concatenation of δI_n and B_0 to the algorithm, which returns a lower triangular basis of $L(B_0)$. We observe that the current matrix during Algorithm 7 in this case always has a striangular structure: it allows to update the vectors without GSO and to perform mod δ arithmetic instead of size-reduction.

This gives rise to Algorithm 13. It calls Algorithm 14 as a unique subroutine, which is tailored for triangularization by applying mod δ arithmetic to Euclid-swap. The main ideas of Algorithm 13 are as follows:

- Step 4 transforms the level of triangularization from $\{i, j\}$ into $\{i, j-1\}$, so that a single execution of Steps 3-5 can line down the $\mathfrak{p}$-triangular subblock formed by the support;
- It performs mod δ arithmetic to prevent coefficient explosion, as usual in many HNF algorithms [DKT87, SL96, MW01].

Algorithm 13 Triangularizing an integer matrix

Input: An nonsingular integer matrix $B_0 \in \mathbb{Z}^{n\times n}$ and the positive integer $\delta = |\det(B_0)|$.
Output: A $\mathfrak{p}$-triangular basis of the lattice $L(B_0)$.

1: $B = (\mathbf{b}_1, \ldots, \mathbf{b}_{2n}) \leftarrow$ the row-concatenation of δI_n and B_0
2: **for** $i = 1$ to n **do**
3: **for** $j = i + n$ downto $i + 1$ **do**
4: Euclid-swap $(\mathbf{b}_{j-1}, \mathbf{b}_j)$ using Algorithm 14
5: **end for**
6: **end for**
7: **return** $(\mathbf{b}_{n+1}, \ldots, \mathbf{b}_{2n})$

Algorithm 14 Integral Euclid-swap procedure for triangularization

Input: The integer δ and a $\{i, j\}$-$\mathfrak{p}$-triangular matrix $B = (b_{s,t}) = (\mathbf{b}_1, \ldots, \mathbf{b}_{2n}) \in \mathbb{Z}^{2n\times n}$ where $j \in [i + 1, i + n]$.
Output: B becomes $\{i, j-1\}$-$\mathfrak{p}$-triangular.

1: **if** $b_{j,j-i} = 0$ **then**
2: Swap $\mathbf{b}_{j-1}$ and $\mathbf{b}_j$
3: **else**
4: Find integers (x, y, g) using XGCD such that $x \cdot b_{j-1,j-i} + y \cdot b_{j,j-i} = g = \gcd(b_{j-1,j-i}, b_{j,j-i})$ with $|x| \le |b_{j,j-i}|/g$ and $|y| \le |b_{j-1,j-i}|/g$
5: **for** $k = 1$ to $j - i - 1$ **do**
6: $\begin{pmatrix} b_{j-1,k} \\ b_{j,k} \end{pmatrix} \leftarrow \begin{pmatrix} -b_{j,j-i}/g & b_{j-1,j-i}/g \\ x & y \end{pmatrix} \begin{pmatrix} b_{j-1,k} \\ b_{j,k} \end{pmatrix} \pmod{\delta}$ //We use mod δ arithmetic
7: **end for**
8: $b_{j-1,j-i} \leftarrow 0$; $b_{j,j-i} \leftarrow g$
9: **end if**
10: **return** B

Algorithm 13 is simple to admit excellent performance in practice. Its main result is as follows:

Theorem F.1. *Given as input an nonsingular integer matrix $B_0 \in \mathbb{Z}^{n\times n}$ and a positive integer $\delta = |\det(B_0)|$, Algorithm 13 outputs a $\mathfrak{p}$-triangular basis B of the lattice $L(B_0)$ such that $\|B\|_\infty \le \delta$. It requires $O(n^3 \cdot \mathcal{M}(\log\delta) + n^2 \cdot \mathcal{B}(\log\beta))$ bit operations and runs on integers of size $O(\log\beta)$ during execution, where $\beta = \max\{\delta, \|B_0\|_\infty\}$.*

Combining with Storjohann's algorithms [Sto98] for computing the HNF (resp. SNF) of triangular integer matrices, Algorithm 13 implies an algorithm for computing the HNF (resp. SNF) of matrix $B_0 \in \mathbb{Z}^{n\times n}$ with $\delta = |\det(B_0)| > 0$ within time $O(n^3 \cdot \mathcal{M}(\log\delta) + n^2 \cdot \mathcal{B}(\log\beta))$. This scheme is simple to implement and different from previous algorithms [Bra71, KB79, CC82, DKT87, Ili89a, HM91, SL96, HMM98, Sto00, MW01, Gie95, Gie96, Sto96b, HM97].

Also, Algorithm 13 can be modified to triangularize an $m \times n$ integer matrix B_0 of rank n as long as the positive integer δ is a multiple of $\det(L(B_0))$: it requires $O(mn^2 \cdot \mathcal{M}(\log\delta) + mn \cdot \mathcal{B}(\log\beta))$ bit operations.

Theorem F.1 follows from Lemmas F.2 and F.3, which give the key properties of Algorithm 13.

Lemma F.2. *The following loop invariants hold for Algorithm 13:*

1. *B is $\{i, j-1\}$-$\mathfrak{p}$-triangular right after Step 4;*
2. *$B \le A$ and $L(A) = L(B)$, where A is the row-concatenation of δI_n and B_0.*

Proof. We show Item 1 by induction on i. Note that A is $\{1, n + 1\}$-$\mathfrak{p}$-triangular, assume that B is $\{i, j\}$-$\mathfrak{p}$-triangular right before Step 4 for indices (i, j) where $i \in [1, n-1]$ and $j \in [i + 2, i + n]$. Then Step 4 transforms the level of triangularization from $\{i, j\}$ into $\{i, j-1\}$. When j decreases to $i + 1$, B becomes $\{i, i\}$-$\mathfrak{p}$-triangular right after Step 4 for indices $(i, i + 1)$, or equivalently, B is $\{i', j'\}$-$\mathfrak{p}$-triangular right before Step 4 for new indices $(i', j') = (i + 1, i + 1 + n)$. Item 1 follows.

We show Item 2. It holds initially since $A \preceq A$ and $L(A) = L(A)$. Assume that Item 2 holds right before Step 4 for indices (i, j). Since the input matrix B to Step 4 is $\{i, j\}$-$\mathfrak{p}$-triangular, the partial order "$\preceq$" implies $L(\delta\mathbf{e}_1, \ldots, \delta\mathbf{e}_{j-i-1}) \subseteq L(\mathbf{b}_1, \ldots, \mathbf{b}_{j-2})$, where $(\mathbf{e}_1, \ldots, \mathbf{e}_n) = I_n$. Hence, the sublattice $L(\mathbf{b}_1, \ldots, \mathbf{b}_j)$ remains unchanged under mod δ arithmetic of Algorithm 14. Then $B \preceq A$ and $L(A) = L(B)$ hold throughout Step 4. This proves Item 2 and hence the lemma. □

The mod δ arithmetic in Algorithm 14 implies the following property:

Lemma F.3. *During the execution of Algorithm 13, any vector $\mathbf{b}_k$ satisfies:*

$$\|\mathbf{b}_k\|_\infty \leq \begin{cases} \delta & \text{if it has been modified by Steps 3-8 of Algorithm 14,} \\ \max\{\delta, \|B_0\|_\infty\} & \text{otherwise.} \end{cases}$$

Proof of Theorem F.1. We first show correctness of Algorithm 13. Let A be the row-concatenation of δI_n and B_0. We always have $L(B) = L(B_0)$ throughout the algorithm: indeed, since $L(\delta I_n) \subseteq L(B_0)$, we have $L(A) = L(B_0)$; note that $L(A) = L(B)$ by Lemma F.2.2, then $L(B) = L(B_0)$.

Now if the algorithm terminates, then $i = n$ and $j = n + 1$: By Lemma F.2.1, the final $(\mathbf{b}_1, \ldots, \mathbf{b}_{2n})$ is $\{n, n\}$-$\mathfrak{p}$-triangular, that is, $\mathbf{b}_1 = \cdots = \mathbf{b}_n = \mathbf{0}$ and $(\mathbf{b}_{n+1}, \ldots, \mathbf{b}_{2n})$ is $\mathfrak{p}$-triangular; then the output returned by Step 7 is a $\mathfrak{p}$-triangular basis of $L(B_0)$.

We analyze the complexity. Algorithm 13 runs on integers. By Lemma F.3, all occurring integers during execution have size $O(\log \beta)$. For each index i, Steps 3-5 cost $O((n \cdot \mathcal{M}(\log \beta) + \mathcal{B}(\log \beta)) + (n - 1)(n \cdot \mathcal{M}(\log \delta) + \mathcal{B}(\log \delta)))$. Then Algorithm 13 requires $O(n^3 \cdot \mathcal{M}(\log \delta) + n^2 \cdot \mathcal{B}(\log \beta))$ bit operations. This completes the proof. □