Vector Rational Number Reconstruction

Curtis Bright and Arne Storjohann
University of Waterloo, Ontario, Canada N2L 3G1

January 15, 2010

Abstract

The final step of some algebraic algorithms is to reconstruct the common denominator d of a collection of rational numbers $(n_i/d)_{1\leq i\leq n}$ from their images $(a_i)_{1\leq i\leq n}$ mod M, subject to a condition such as $0 < d \leq N$ and $|n_i| \leq N$ for a given magnitude bound N. Applying elementwise rational number reconstruction requires that $M \in \Omega(N^2)$. We present an algorithm, based on lattice basis reduction, which can perform the reconstruction efficiently even when the modulus satisfies a considerably smaller magnitude bound $M \in \Omega(N^{1+1/c})$ for c a small constant, for example $2 \leq c \leq 5$. Assuming $c \in O(1)$ the cost of the approach is $O(n(\log M)^3)$ bit operations using the original LLL lattice reduction algorithm, but is reduced to $O(n(\log M)^2)$ bit operations by incorporating the L^2 variant [15]. As an application, we give a robust method for reconstructing the rational solution vector of a linear system from its image, such as obtained by a solver using p-adic lifting.

1 Introduction

A rational number reconstruction of an integer $a \in \mathbb{Z}$ with respect to a positive modulus $M \in \mathbb{Z}_{>0}$ is a signed fraction $n/d \in \mathbb{Q}$ with $\gcd(n, d) = 1$, $\gcd(d, M) = 1$ and such that $a \equiv n/d \pmod M$. In general, there may be multiple possibilities, for example $a \equiv n_1/d_1 \equiv n_2/d_2 \pmod M$ with $(n_1, d_1) \neq (n_2, d_2)$. The problem has been very well studied, and the set of all possible reconstructions is linked to the intermediate results of the extended Euclidean algorithm and the continued fraction expansion of a/M, see for example [9, Theorem 5.1] or the books [7, 17].

Uniqueness of the reconstruction, if it exists, can be ensured by stipulating bounds for the magnitudes of the output integers n and d. In addition to a and M, the simplest version of the problem takes as input a bound $N < M$, and asks for output a pair of integers (d, n) such that

$$da \equiv n \pmod{M}, \qquad |n| \leq N, \qquad 0 < d \leq N. \tag{1}$$

Note that if (d, n) is a solution to (1) that satisfies $\gcd(d, M) = 1$ then $a \equiv n/d \pmod{M}$. If the bound $M > 2N^2$ is satisfied, then the solution to (1) with $\gcd(n, d) = 1$ and $\gcd(d, M) = 1$ is unique, if it exists, and can be computed effectively using the well known approach based on the extended Euclidean algorithm.

Rational number reconstruction is an essential tool in many algorithms that employ a homomorphic imaging scheme to avoid intermediate expression swell, to allow for a simple coarse grain parallelization, or to facilitate an output sensitive approach. Often, the final step of these algorithms is to reconstruct the common denominator $d \in \mathbb{Z}_{>0}$ of a collection of rational numbers $(n_i/d)_{1 \leq i \leq n}$ from their images $(a_i)_{1 \leq i \leq n}$ modulo M. The images modulo M are typically computed by combining multiple smaller images, either using Chinese remaindering ($M = p_1 p_2 \cdots p_m$) or a variation of Newton-Hensel lifting ($M = p^m$). The cost of an algorithm that uses a homomorphic imaging scheme is highly correlated to m, the number of smaller images computed, which is directly related to the bitlength of the modulus M. Ideally, just enough smaller images are computed to allow reconstruction of the common denominator d. If N is an upper bound for both d and $\max_i |n_i|$, elementwise rational reconstruction can be applied but requires that $M > 2N^2$ to ensure success.

This paper gives a deterministic algorithm for efficiently computing the common denominator d that for some applications requires about half as many image computations as the standard approach. Our specification of the vector version of the problem differs slightly from the scalar case shown in (1). The vector rational reconstruction problem takes as input a vector $\boldsymbol{a} \in \mathbb{Z}^n$ of images modulo M, and asks for a pair $(d, \boldsymbol{n}) \in (\mathbb{Z}, \mathbb{Z}^n)$ such that

$$d\boldsymbol{a} \equiv \boldsymbol{n} \pmod{M}, \qquad \left\| \left[\, d \mid \boldsymbol{n} \,\right] \right\|_2 \leq N. \tag{2}$$

Here, we use a common bound for d and $\boldsymbol{n}$ based on the 2-norm because this is a more natural condition for the algorithm we will present, which is based on integer lattice basis reduction. In particular, the problem of computing solutions to (2) is equivalent to finding short row vectors in the lattice

$$\left[\begin{array}{c|c} & M\boldsymbol{I}_{n\times n} \\ \hline 1 & \boldsymbol{n} \end{array}\right] \in \mathbb{Z}^{(n+1)\times(n+1)}. \tag{3}$$

The algorithm we give actually computes a complete "generating set" for (2), that is, a set $(d_i, \boldsymbol{n}_i)_{1\le i\le c}$ corresponding to linearly independent vectors $\left[\, d \mid \boldsymbol{n} \,\right]_{1\le i\le c}$ such that every solution of (2) can be expressed as a $\mathbb{Z}$-linear combination of the members of the generating set. On the one hand, from the scalar case, we know that a sufficient condition to ensure the existence of a generating set of dimension zero (no nonzero solution) or one (a unique minimal denominator solution) is that the modulus M be large enough to satisfy $M > 2N^2$. On the other hand, if c is an integer such that $M > 2^{(c+1)/2}N^{1+1/c}$ is satisfied, we prove that the generating set returned by our algorithm will contain at most c vectors. The generating set produced will be LLL reduced, so the 2-norm of the first vector will be at most $2^{(c-1)/2}$ times that of the shortest nonzero vector which solves (2).

To apply lattice reduction directly to a lattice with row dimension $n+1$ would be prohibitively expensive in terms of n when n is large (below we give an example application with $n = 10,000$). Instead, to obtain a cost estimate that is linear in n, we use an iterative approach which adds columns to the work lattice one by one, while keeping the row dimension bounded by $c+1$ by removing vectors which provably can't contribute to a solution of (2). Using a variation of the standard LLL algorithm and assuming $c = O(1)$, the cost is $O(n(\log M)^3)$ bit operations, which is linear in n but still cubic in $\log M$. By incorporating the L^2 algorithm [15] the cost is reduced further to $O(n(\log M)^2)$ bit operations.

The approach is particularly well suited to applications where it is known *a priori* that that there can exist at most one linearly independent solution to (2). In Section 5 we consider the problem of solving a nonsingular integer linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, which has exactly one rational solution vector with common denominator a factor of $\det \boldsymbol{A}$. As a concrete example, coming from [2], suppose $\boldsymbol{A}$ and $\boldsymbol{b}$ have dimension $10,000$ and are filled randomly with single decimal digit integers. The common denominator and the numerators of a typical solution vector for such a system have about 24,044 decimal digits, or magnitude about $\beta = 10^{24044}$. To apply elementwise rational reconstruction requires $M > 2N^2$ with $N \ge \beta$ to be satisfied, so M needs to have length about 48088 decimal digits. But by choosing the parameter $c = 5$, the vector algorithm requires only that $M > 2^{(c+1)/2}N^{1+1/c}$ with $N \geq \sqrt{n+1}\beta$ in order to succeed, so M need have only about 28,856 decimal digits. The method we propose is robust in the sense that β need not be known beforehand; if a reconstruction is attempted with N too small, FAIL will be reported, but if N is sufficiently large the algorithm will guarantee to return the correct reconstruction.

Related work For the scalar version of the problem, much work has focused on decreasing the running time from $O((\log M)^2)$ to nearly linear in $\log M$ by incorporating fast integer multiplication. For a a survey of work in this direction we refer to [13]. An algorithm that doesn't need a priori bounds for the numerator and denominator is described in [14].

Now consider the vector version of the problem. Two approaches are proposed in [10]. The first is a heuristic randomized algorithm to recover a solution d and $\boldsymbol{n}$ that satisfies $d\|\boldsymbol{n}\|_\infty \in O(M)$. The second, based on the good simultaneous Diophantine approximation algorithm in [11], can be applied to find a solution even when $\left\|\left[\, d \mid \boldsymbol{n} \,\right]\right\|_2 \in o(N^2)$ but the algorithm performs lattice reduction directly on $n+1$ dimensional lattices similar to (3) and seems to be expensive when n is large.

The recent preprint [18] gives a very general algorithm that supports improved asymptotic complexity bounds for finding vectors shorter than a given target length in generalized knapsack-type lattices. The algorithm there also adds information to the lattice gradually, while removing rows that are provably too large to contribute to a solution vector. Applied to the lattice shown in (3), a special boundary case of the general lattice shape they consider, it follows from their analysis that the dimension of the work lattice and final generating set will be bounded by $O(1)$ provided that $M \in \Omega(N^2)$. As we noted above, if $M > 2N^2$ then the dimension of the solution space for this special case of the knapsack lattice is known to be at most one from the studies of the scalar rational reconstruction problem, and the unique solution, if it exists, can found using the elementwise rational number reconstruction approach.

An efficient algorithm for the rational function version of the vector reconstruction problem is given in [16].

Organization In Section 2 we illustrate the main ideas of the algorithm with a worked example. In Section 3 we establish our notation and recall the required facts about lattices and the LLL lattice basis reduction algorithm. Section 4 presents algorithm for vector rational reconstruction, proves its correctness, and develops a variant with significantly improved running time. In Section 5 we show how the vector rational reconstruction algorithm can be incorporated into an algorithm for solving nonsingular linear systems to save on the number of required image computations.

2 Outline of the algorithm

First note that the problem of finding solutions to (2) is identical to the problem of finding short vectors, with respect to the 2-norm, in the lattice generated by the rows of the following matrix:

$$\begin{bmatrix} & & & & M \\ & & & \iddots & \\ & & M & & \\ & M & & & \\ 1 & a_1 & a_2 & \cdots & a_n \end{bmatrix} \in \mathbb{Z}^{(n+1)\times(n+1)}. \tag{4}$$

The first n rows of the matrix can be used to reduce modulo M the last n entries of any vector in the lattice; in particular, a vector obtained by multiplying the last row by d. The first entry of such a vector will still be d and the ith entry for $2 \le i \le n+1$ will be congruent to $da_{i-1} \pmod{M}$.

For example, consider the input lattice

$$\boldsymbol{L} = \begin{bmatrix} & & & & & 195967 \\ & & & & 195967 & \\ & & & 195967 & & \\ & & 195967 & & & \\ & 195967 & & & & \\ 1 & -23677 & -49539 & 74089 & -21989 & 63531 \end{bmatrix}$$

where our target length is $N = 10000$.

In general, finding short lattice vectors is a difficult problem, but is facilitated by lattice basis reduction. The LLL Algorithm is guaranteed to return a basis with first vector at most $2^{(n-1)/2}$ times longer than the shortest nonzero vector in an n-dimensional lattice. For example, running LLL on $\boldsymbol{L}$

yields the LLL-reduced basis matrix

$$\boldsymbol{L}' = \left[\begin{array}{cccccc} -3137 & 3256 & 2012 & -331 & -891 & 1692 \\ \hline -3600 & -8445 & 10430 & -9313 & -10268 & -18111 \\ -4047 & -7044 & 10092 & -8673 & 20465 & -1253 \\ 241 & -23114 & 15088 & 22452 & -8240 & 25545 \\ 28082 & 18517 & 15535 & -14341 & -3081 & -6026 \\ -11836 & 8162 & 10340 & 34921 & 17628 & -27537 \end{array}\right],$$

from which the first vector immediately gives a solution. In fact, we can easily show that the other vectors are too large to contribute to a vector shorter than N by using the fact that any lattice vector which includes the final vector must be at least as large as the final vector in the Gram–Schmidt orthogonalization (GSO) of $\boldsymbol{L}'$. The GSO of $\boldsymbol{L}'$ is computed as a side effect of the lattice reduction, and for this example we can check that the last vector in the GSO of $\boldsymbol{L}'$ has norm $> N$, therefore we know the last vector of $\boldsymbol{L}'$ cannot contribute to a vector shorter than N. Removing the last vector, we repeat the magnitude check on the new last vector of the GSO, and continue in this way until the first vector is the only one remaining. This process is formalized in Section 3.1.

However, when n is large it is infeasible to reduce the entire lattice at once. Fortunately, the structure of the lattice permits an iterative approach. For example, consider reducing only the lower-left 2×2 submatrix of $\boldsymbol{L}$:

$$\begin{bmatrix} 0 & 195967 \\ 1 & -23677 \end{bmatrix} \xrightarrow{\text{LLL}} \begin{bmatrix} -389 & -96 \\ -149 & 467 \end{bmatrix}$$

Knowing the reduction of the lower-left 2×2 submatrix can help us reduce the lower-left 3×3 submatrix of $\boldsymbol{L}$. We could have kept track of the third column of $\boldsymbol{L}$ while doing the above reduction, but this isn't required because it is clear that during the reduction the 3rd column will always be a_2 times the first column. Then the lattice generated by the lower-left 3×3 submatrix of $\boldsymbol{L}$ has the following basis, which we again reduce:

$$\left[\begin{array}{cc|c} 0 & 0 & 195967 \\ \hline -389 & -96 & 19270671 \\ -149 & 467 & 7381311 \end{array}\right] \xrightarrow{\text{LLL}} \begin{bmatrix} -538 & 371 & 470 \\ 91 & 1030 & -808 \\ 27089 & 13738 & 20045 \end{bmatrix}$$

Now, the final GSO vector of the reduced matrix has norm larger than N, so we can safely discard the last row, and repeat the same augmentation process to find a sublattice which contains all short vectors in the lattice generated by the lower-left 4×4 submatrix of $\boldsymbol{L}$.

The main effort in Section 4 is to show that if $M > 2^{(c+1)/2}N^{1+1/c}$ for $c \in \mathbb{Z}_{>0}$ then the process described above of adding columns and removing final rows of the lattice will keep the row dimension of the lattice bounded by $c+1$. For $c = O(1)$ this leads to a cost estimate that is linear in n.

3 Preliminaries

For a $k \times n$ matrix $\boldsymbol{L}$ we let $\boldsymbol{L}_S$ be the rows of $\boldsymbol{L}$ which have indices in $S \subseteq \{1, \ldots, k\}$, let $\boldsymbol{L}_R^{\mathrm{T}}$ be the columns of $\boldsymbol{L}$ which have indices in $R \subseteq \{1, \ldots, n\}$, and let $\boldsymbol{L}_{S,R}$ denote $(\boldsymbol{L}_S)_R^{\mathrm{T}}$. We simply write i for $\{i\}$ and $1..i$ for $\{1, \ldots, i\}$. When not used with a subscript, $\boldsymbol{L}^{\mathrm{T}}$ denotes the transpose of $\boldsymbol{L}$.

A subscript on a row vector will always refer to entrywise selection, and the norm of a row vector will refer to the 2-norm, $\|\boldsymbol{x}\| \coloneqq \sqrt{\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}}$.

Vectors are denoted by lower-case bold variables and matrices by upper-case or greek bold variables, with the boldface dropped when referring to individual entries. The zero vector is denoted $\mathbf{0}$ (with the dimension clear from context).

The $\operatorname{rem}_M(x)$ function returns the reduction of $x \pmod M$ in the symmetric range, and applies elementwise to vectors and matrices.

3.1 Lattices

A *point lattice* is a discrete additive subgroup of $\mathbb{R}^n$. The elements of the lattice generated by the rank k matrix $\boldsymbol{L} \in \mathbb{Z}^{k \times n}$ are given by

$$\mathcal{L}(\boldsymbol{L}) \coloneqq \left\{ \sum_{i=1}^{k} r_i \boldsymbol{L}_i : r_i \in \mathbb{Z} \right\},$$

and $\boldsymbol{L}$ is a *basis* of $\mathcal{L}(\boldsymbol{L})$. If $\mathcal{L}(\boldsymbol{S}) \subseteq \mathcal{L}(\boldsymbol{L})$ then $\mathcal{L}(\boldsymbol{S})$ is known as a *sublattice* of $\mathcal{L}(\boldsymbol{L})$; this occurs if and only if there exists an integer matrix $\boldsymbol{B}$ such that $\boldsymbol{S} = \boldsymbol{B}\boldsymbol{L}$. The *volume* of a lattice is independent of the choice of basis and given by $\operatorname{vol} \boldsymbol{L} \coloneqq \sqrt{\det(\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}})}$.

The set of vectors in $\mathcal{L}(\boldsymbol{L})$ shorter than some bound N is denoted

$$\mathcal{L}_N(\boldsymbol{L}) \coloneqq \{\, \boldsymbol{b} \in \mathcal{L}(\boldsymbol{L}) : \|\boldsymbol{b}\| \leq N \,\}.$$

A *generating lattice* of $\mathcal{L}_N(\boldsymbol{L})$ is a sublattice of $\mathcal{L}(\boldsymbol{L})$ which contains all elements of $\mathcal{L}_N(\boldsymbol{L})$; the basis of a generating lattice is known as a *generating matrix*. $\boldsymbol{S}$ is a generating matrix of $\mathcal{L}_N(\boldsymbol{L})$ when it consists of linearly independent rows from $\mathcal{L}(\boldsymbol{L})$ such that any $\boldsymbol{b} \in \mathcal{L}_N(\boldsymbol{L})$ can be written as a $\mathbb{Z}$-linear combination of row vectors in $\boldsymbol{S}$; equivalently,

$$\mathcal{L}_N(\boldsymbol{L}) \subset \mathcal{L}(\boldsymbol{S}) \subseteq \mathcal{L}(\boldsymbol{L}).$$

For example, any basis of $\mathcal{L}(\boldsymbol{L})$ is always a generating matrix of $\mathcal{L}_N(\boldsymbol{L})$ for any N. However, when N is small we might hope to find a generating matrix with fewer than k rows.

3.2 Gram–Schmidt orthogonalization

For a lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$, let $\boldsymbol{L}^*$ be the associated Gram–Schmidt orthogonal $\mathbb{R}$-basis with change-of-basis matrix $\boldsymbol{\mu}$, i.e.,

$$\begin{bmatrix} \boldsymbol{L}_1 \\ \boldsymbol{L}_2 \\ \vdots \\ \boldsymbol{L}_k \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \mu_{2,1} & 1 & & \\ \vdots & & \ddots & \\ \mu_{k,1} & \mu_{k,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1^* \\ \boldsymbol{L}_2^* \\ \vdots \\ \boldsymbol{L}_k^* \end{bmatrix}$$

with $\mu_{i,j} = \langle \boldsymbol{L}_i, \boldsymbol{L}_j^* \rangle / \|\boldsymbol{L}_j^*\|^2$. Then we define the GSO $(\boldsymbol{\mu}, \boldsymbol{d}) \in (\mathbb{Q}^{k\times k}, \mathbb{Z}^{k+1})$ of a basis $\boldsymbol{L}$ to satisfy

$$\boldsymbol{L} = \boldsymbol{\mu}\boldsymbol{L}^* \qquad \text{and} \qquad d_i = \prod_{j=1}^{i} \|\boldsymbol{L}_j^*\|^2,$$

with $d_0 = 1$, so $d_i/d_{i-1} = \|\boldsymbol{L}_i^*\|^2$. Note that $d_k = (\operatorname{vol} \boldsymbol{L})^2$ and in general $d_i = (\operatorname{vol} \boldsymbol{L}_{1..i})^2$.

3.3 LLL reduction

Let $(\boldsymbol{\mu}, \boldsymbol{d})$ be the GSO of a lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$. The GSO is *size-reduced* if $\|\boldsymbol{\mu} - \boldsymbol{I}_{k\times k}\|_{\max} \le \frac{1}{2}$ and *2-reduced* if $d_i d_{i-2} \ge (\frac{3}{4} - \mu_{i,i-1}^2) d_{i-1}^2$ for $1 < i \le k$. A GSO is *LLL-reduced* if it is both size-reduced and 2-reduced, and a basis is LLL-reduced if its corresponding GSO is LLL-reduced.

Given a lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$, the *lattice basis reduction* problem is to compute an LLL-reduced basis $\boldsymbol{L}'$ such that $\mathcal{L}(\boldsymbol{L}) = \mathcal{L}(\boldsymbol{L}')$. Assuming

$k = 0(1)$, the algorithm from [12] accomplishes this in $O(n(\log B)^3)$ bit operations, where $\max_i \|\boldsymbol{L}_i\| \leq B$. A well known and important feature of the LLL algorithm, that we will exploit, is that the sequence of unimodular row operations required to reduce a given lattice can be determined strictly from the GSO $(\boldsymbol{\mu}, \boldsymbol{d})$. Consider Algorithm 1, which only includes the specification of the input and output. If $(\boldsymbol{\mu}, \boldsymbol{d})$ is a GSO such that $\boldsymbol{L} = \boldsymbol{\mu}\boldsymbol{L}^*$ for a lattice $\boldsymbol{L}$ with row dimension k and arbitrary column dimension, we could LLL reduce $\boldsymbol{L}$ inplace by calling $\mathsf{InPlaceLLL}(k, \boldsymbol{\mu}, \boldsymbol{d}, \boldsymbol{L})$. But we could also initialize a matrix $\boldsymbol{U} = \boldsymbol{I}_k$, call $\mathsf{InPlaceLLL}(k, \boldsymbol{\mu}, \boldsymbol{d}, \boldsymbol{U})$ to capture all required unimodular transformations in $\boldsymbol{U}$, and then compute the reduced lattice as $\boldsymbol{L} \leftarrow \boldsymbol{U}\boldsymbol{L}$.

We will make use of the following theorems about LLL reduction [12].

Theorem 1. *An LLL-reduced basis $\boldsymbol{L} \in \mathbb{Z}^{k \times n}$ satisfies the following properties:*

1. $\|\boldsymbol{L}_i^*\| \leq 2^{(j-i)/2} \|\boldsymbol{L}_j^*\|$ *for* $i \leq j$
2. $\max_i \|\boldsymbol{L}_i\| \leq 2^{(k-1)/2} \|\boldsymbol{L}_k^*\|$
3. $(\operatorname{vol} \boldsymbol{L})^{1/k} / 2^{(k-1)/4} \leq \|\boldsymbol{L}_k^*\|$

Theorem 2. *During the execution of Algorithm 1:*

1. $\max_\ell \|\boldsymbol{L}_\ell^*\|$ *is never increased*
2. d_ℓ *is never increased (for all* ℓ*)*
3. $d_i' < \frac{3}{4} d_i$ *during step 5*

Algorithm 1 The $\mathsf{InPlaceLLL}(k, \boldsymbol{\mu}, \boldsymbol{d}, \boldsymbol{U})$ lattice basis reduction algorithm.

Input: The GSO $(\boldsymbol{\mu}, \boldsymbol{d}) \in (\mathbb{Q}^{k \times k}, \mathbb{Z}^{k+1})$ of some lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k \times *}$. A matrix $\boldsymbol{U} \in \mathbb{Z}^{k \times *}$.
Output: Use LLL [12] to update $(\boldsymbol{\mu}, \boldsymbol{d})$ to be 2-reduced and size-reduced and apply all unimodular row operations to $\boldsymbol{U}$.

Given a lattice $\boldsymbol{L} \in \mathbb{Z}^{k \times n}$, and assuming $k = O(1)$, the L^2 algorithm from [15] can be used to compute an LLL reduced basis in only $O(n(\log B)^2)$ bit operations, where $\max_i \|\boldsymbol{L}_i\| \leq B$. We will make use of the inplace variant of L^2 shown in Algorithm 2.

Algorithm 2 The $\mathsf{InPlaceL}^2(k, \boldsymbol{G}, \boldsymbol{U})$ lattice basis reduction algorithm.

Input: The Gramian $\boldsymbol{G} = \boldsymbol{L}\boldsymbol{L}^T \in \mathbb{Z}^{k\times k}$ of some lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times *}$. A matrix $\boldsymbol{U} \in \mathbb{Z}^{k\times *}$.

Output: Use L^2 [15] to update $\boldsymbol{G}$ to be 2-reduced and size-reduced and apply all unimodular row operations to $\boldsymbol{U}$.

4 The vector rational reconstruction algorithm

We will be concerned with the lattice generated by

$$\boldsymbol{\Lambda}_{\boldsymbol{a}}^{M} := \begin{bmatrix} & & & & M \\ & & & \iddots & \\ & & M & & \\ & M & & & \\ 1 & a_1 & a_2 & \cdots & a_n \end{bmatrix} \in \mathbb{Z}^{(n+1)\times(n+1)},$$

where $\boldsymbol{a} \in \mathbb{Z}^{1\times n}$ and $M \in \mathbb{Z}_{>0}$. When $\boldsymbol{a}$ and M are clear from context, $\boldsymbol{\Lambda}_{\boldsymbol{a}}^{M}$ will simply be denoted $\boldsymbol{\Lambda}$. In this section we present an algorithm which computes an LLL-reduced generating matrix for $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^{M})$ with at most c vectors, where $c \geq 1$ is a small constant such that $M > 2^{(c+1)/2}N^{1+1/c}$ is satisfied.

We will present three versions of the algorithm. In Section 4.1 we present a basic version, prove its correctness and prove the running time is $O(n^2(\log B)^2)$ bit operations provided $c = O(1)$. In Section 4.2 we adapt the algorithm so that one needs to only keep track of first column of the work lattice, thus reducing the cost to $O(n(\log B)^3)$ bit operations. Finally, in Section 4.3 we show how to use L^2 instead of LLL in order to reduce the cost to $O(n(\log M)^2)$ bit operations.

4.1 The basic algorithm

The basic algorithm pseudocode is given as Algorithm 3. We will prove the assertions after step 5 hold, from which the correctness of the algorithm follows.

Lemma 1. *If $\boldsymbol{L} \in \mathbb{Z}^{k\times n}$ has GSO $(\boldsymbol{\mu}, \boldsymbol{d})$ then $\boldsymbol{L}_{1..i}$ has GSO $(\boldsymbol{\mu}_{1..i,1..i}, \boldsymbol{d}_{0..i})$ for $1 \leq i \leq k$, and if $\boldsymbol{L}$ is LLL-reduced then so is $\boldsymbol{L}_{1..i}$.*

Algorithm 3 The basic $\mathsf{VecRecon}(n, \boldsymbol{a}, M, N, c)$ generating matrix algorithm.

Input: $\boldsymbol{a} \in \mathbb{Z}_M^{1\times n}$ and $N,\, c \in \mathbb{Z}_{>0}$ with $M > 2^{(c+1)/2}N^{1+1/c}$.
Output: An LLL-reduced generating matrix $\boldsymbol{S} \in \mathbb{Z}^{k\times(n+1)}$ of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^M)$ with $k \le c$.

```
// Note $\boldsymbol{L} \in \mathbb{Z}^{k\times(\ell+1)}$, $\boldsymbol{d} \in \mathbb{Z}^{k+1}$ and $\boldsymbol{\mu} \in \mathbb{Q}^{k\times k}$ throughout (with $\ell$ starting
   at 0).

1. [Initialization]
   $k := 1$; $L_{1,1} := 1$; $d_0 := 1$; $d_1 := 1$; $\mu_{1,1} := 1$;

2. [Iterative lattice augmentation]
   for $\ell := 1$ to $n$ do

     3. [Add new vector to generating matrix]
        $k := k+1$; $\boldsymbol{\mu} := \left[\begin{array}{c|c} 1 & \boldsymbol{0} \\ \hline \boldsymbol{L}_1^{\mathrm{T}} a_\ell/M & \boldsymbol{\mu} \end{array}\right]$; $\boldsymbol{d} := \left[\begin{array}{c} 1 \\ \hline M^2\boldsymbol{d} \end{array}\right]$; $\boldsymbol{L} := \left[\begin{array}{c|c} \boldsymbol{0} & M \\ \hline \boldsymbol{L} & \boldsymbol{L}_1^{\mathrm{T}} a_\ell \end{array}\right]$;

     4. [LLL reduction]
        $\mathsf{InPlaceLLL}(k, \boldsymbol{\mu}, \boldsymbol{d}, \boldsymbol{L})$;

     5. [Remove superfluous vectors from generating matrix]
        Set $k$ to be the maximal value such that $d_k/d_{k-1} \le N^2$.
        If no such $k$ exists, return the unique element of $\mathbb{Z}^{0\times(n+1)}$.
        $\boldsymbol{L} := \boldsymbol{L}_{1..k}$; $\boldsymbol{\mu} := \boldsymbol{\mu}_{1..k,1..k}$; $\boldsymbol{d} := \boldsymbol{d}_{0..k}$;

   assert  A. $(\boldsymbol{\mu}, \boldsymbol{d})$ is LLL-reduced
           B. $\boldsymbol{L}$ is a generating matrix of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..\ell}}^M)$ with GSO $(\boldsymbol{\mu}, \boldsymbol{d})$
           C. $k \le c$

6. [Return generating matrix]
   return $\boldsymbol{S} := \boldsymbol{L}$;
```

Proof. Follows immediately from definitions and $(\boldsymbol{L}_{1..i})^* = \boldsymbol{L}_{1..i}^*$. □

Corollary 1. *Assertion A after step 5 of Algorithm 3 holds.*

Proof. The GSO $(\boldsymbol{\mu}, \boldsymbol{d})$ is LLL-reduced during step 4 and truncated during step 5, so it remains LLL-reduced after step 5. □

Lemma 2. *If $\boldsymbol{S} \in \mathbb{Z}^{k\times n}$ is a generating matrix for $\mathcal{L}_N(\boldsymbol{L})$ and $\|\boldsymbol{S}_k^*\| > N$*

then $\boldsymbol{S}_{1..k-1}$ is also a generating matrix for $\mathcal{L}_N(\boldsymbol{L})$.

Proof. Clearly $\mathcal{L}(\boldsymbol{S}_{1..k-1}) \subseteq \mathcal{L}(\boldsymbol{S})$, and $\mathcal{L}(\boldsymbol{S}) \subseteq \mathcal{L}(\boldsymbol{L})$ since $\boldsymbol{S}$ is a generating matrix of $\mathcal{L}_N(\boldsymbol{L})$, so $\mathcal{L}(\boldsymbol{S}_{1..k-1})$ is a sublattice of $\mathcal{L}(\boldsymbol{L})$. It remains to show that $\mathcal{L}_N(\boldsymbol{L}) \subset \mathcal{L}(\boldsymbol{S}_{1..k-1})$, i.e., if $\boldsymbol{b} \in \mathcal{L}_N(\boldsymbol{L})$ then $\boldsymbol{b}$ can be written as a $\mathbb{Z}$-linear combination of the rows of $\boldsymbol{S}_{1..k-1}$.

Since $\mathcal{L}_N(\boldsymbol{L}) \subset \mathcal{L}(\boldsymbol{S})$, we can write any $\boldsymbol{b} \in \mathcal{L}_N(\boldsymbol{L})$ in the form $\boldsymbol{b} = \sum_{i=1}^{k} r_i \boldsymbol{S}_i$ for some $\boldsymbol{r} \in \mathbb{Z}^{1\times k}$. Rewriting using the Gram–Schmidt decomposition, we have $\boldsymbol{b} = r_k \boldsymbol{S}_k^* + \sum_{i=1}^{k-1} s_i \boldsymbol{S}_i^*$ for $\boldsymbol{s} = \boldsymbol{r}\boldsymbol{\mu} \in \mathbb{Q}^{1\times k}$. Since the $\boldsymbol{S}_i^*$ are orthogonal,

$$\|\boldsymbol{b}\|^2 = r_k^2 \|\boldsymbol{S}_k^*\|^2 + \sum_{i=1}^{k-1} s_i^2 \|\boldsymbol{S}_i^*\|^2 \geq r_k^2 \|\boldsymbol{S}_k^*\|^2,$$

so if $r_k^2 \neq 0$ then $\|\boldsymbol{b}\| \geq \|\boldsymbol{S}_k^*\| > N$. Thus $\boldsymbol{b} \in \mathcal{L}_N(\boldsymbol{L})$ must be in $\mathcal{L}(\boldsymbol{S}_{1..k-1})$. □

In the following proofs about a specific step of the computation, let $\boldsymbol{L}'$ be the *new* value of $\boldsymbol{L}$ at the conclusion of the step (and similarly for the other quantities k, ℓ, $\boldsymbol{\mu}$, $\boldsymbol{d}$).

Proposition 1. *At the completion of steps 3, 4 and 5 in Algorithm 3,*

$$\boldsymbol{L} \text{ is a generating matrix of } \mathcal{L}_N\big(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell}}\big) \text{ with GSO } (\boldsymbol{\mu}, \boldsymbol{d}). \qquad (*)$$

Proof. After step 1 we have $\boldsymbol{L} = \boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..0}} = [\,1\,]$, so $\boldsymbol{L}$ is a generating matrix of $\mathcal{L}(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell}})$. Also, $(\boldsymbol{\mu}, \boldsymbol{d}) = ([\,1\,], [\,{}^{1}_{1}\,])$ so $\boldsymbol{L}$ has GSO $(\boldsymbol{\mu}, \boldsymbol{d})$ and $(*)$ is satisfied as the loop is entered for the first time.

Now we show that if $(*)$ holds at the beginning step 2, it also holds at the conclusion of step 3, and if it holds at the beginning of steps 4 or 5 it also holds at their conclusion. Denote $\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell}}$ by $\boldsymbol{\Lambda}$ and $\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell'}}$ by $\boldsymbol{\Lambda}'$.

- Steps 2–3: For these steps $\ell' = \ell + 1$ and $\boldsymbol{L}$ is updated such that

$$\boldsymbol{L}' = \left[\begin{array}{c|c} \boldsymbol{0} & M \\ \hline \boldsymbol{L} & \boldsymbol{L}_1^{\mathrm{T}} a_{\ell'} \end{array}\right] \qquad \text{and} \qquad \boldsymbol{L}'^* = \left[\begin{array}{c|c} \boldsymbol{0} & M \\ \hline \boldsymbol{L}^* & \boldsymbol{0} \end{array}\right].$$

  Letting $(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{d}})$ denote the GSO of $\boldsymbol{L}'$, these imply:

$$\begin{aligned} \tilde{d}_i &= M^2 \textstyle\prod_{j=1}^{i-1} \|\boldsymbol{L}_j^*\|^2 && \text{for } i \geq 1 \\ \tilde{\mu}_{i,1} &= L_{i,1} a_{\ell'}/M && \text{for } i \geq 2 \\ \tilde{\mu}_{i,j} &= \langle \boldsymbol{L}_{i-1}, \boldsymbol{L}_{j-1}^* \rangle / \|\boldsymbol{L}_{j-1}^*\|^2 && \text{for } i,\, j \geq 2 \end{aligned}$$

Assuming that $(\boldsymbol{\mu}, \boldsymbol{d})$ was the GSO of $\boldsymbol{L}$ these yield the same formulae used to update $(\boldsymbol{\mu}, \boldsymbol{d})$. Thus $(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{d}}) = (\boldsymbol{\mu}', \boldsymbol{d}')$ is the GSO of $\boldsymbol{L}'$.

Assuming that $\mathcal{L}(\boldsymbol{L})$ was a sublattice of $\mathcal{L}(\boldsymbol{\Lambda})$, there is some $\boldsymbol{B} \in \mathbb{Z}^{k\times(\ell+1)}$ such that $\boldsymbol{L} = \boldsymbol{B\Lambda}$. Then $\mathcal{L}(\boldsymbol{L}')$ is a sublattice of $\mathcal{L}(\boldsymbol{\Lambda}')$, since

$$\boldsymbol{L}' = \left[\begin{array}{c|c} \mathbf{0} & M \\ \hline \boldsymbol{B\Lambda} & \boldsymbol{B\Lambda}_1^{\mathrm{T}} a_{\ell'} \end{array}\right] = \left[\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & \boldsymbol{B} \end{array}\right] \boldsymbol{\Lambda}'.$$

Now let $\boldsymbol{b} = \boldsymbol{r\Lambda}'$ for some $\boldsymbol{r} \in \mathbb{Z}^{1\times(\ell'+1)}$ be an arbitrary element of $\mathcal{L}(\boldsymbol{\Lambda}')$. But if $\boldsymbol{b} \in \mathcal{L}_N(\boldsymbol{\Lambda}')$ then $\boldsymbol{b}_{1..\ell} \in \mathcal{L}_N(\boldsymbol{\Lambda})$, which is in $\mathcal{L}(\boldsymbol{L})$ by assumption, so there exists some $\boldsymbol{s} \in \mathbb{Z}^{1\times k}$ such that $\boldsymbol{b}_{1..\ell} = \boldsymbol{sL}$. Then

$$\begin{aligned} \boldsymbol{b} &= \boldsymbol{r\Lambda}' \\ &= \boldsymbol{r} \left[\begin{array}{c|c} \mathbf{0} & M \\ \hline \boldsymbol{\Lambda} & \boldsymbol{\Lambda}_1^{\mathrm{T}} a_{\ell'} \end{array}\right] \\ &= \left[\begin{array}{c|c} r_1 & 1 \end{array}\right] \left[\begin{array}{c|c} \mathbf{0} & M \\ \hline \boldsymbol{sL} & \boldsymbol{sL}_1^{\mathrm{T}} a_{\ell'} \end{array}\right] \\ &= \left[\begin{array}{c|c} r_1 & \boldsymbol{s} \end{array}\right] \boldsymbol{L}' \in \mathcal{L}(\boldsymbol{L}'), \end{aligned}$$

showing $\mathcal{L}_N(\boldsymbol{\Lambda}') \subset \mathcal{L}(\boldsymbol{L}')$.

- Step 4: If $(*)$ holds at the start of step 4, then it necessarily holds at its conclusion since InPlaceLLL ensures $\mathcal{L}(\boldsymbol{L}') = \mathcal{L}(\boldsymbol{L})$ and updates $(\boldsymbol{\mu}', \boldsymbol{d}')$ to be the GSO of $\boldsymbol{L}'$.

- Step 5: By Lemma 1, if $(\boldsymbol{\mu}, \boldsymbol{d})$ was the GSO of $\boldsymbol{L}$ then $(\boldsymbol{\mu}', \boldsymbol{d}')$ is the GSO of $\boldsymbol{L}'$.

  Also, by repeated application of Lemma 2, if $\boldsymbol{L}$ was a generating set of $\mathcal{L}_N(\boldsymbol{\Lambda})$ then $\boldsymbol{L}'$ is also a generating set of $\mathcal{L}_N(\boldsymbol{\Lambda})$.

By induction, $(*)$ always holds after any step during the loop. □

Corollary 2. *Assertion B after step 5 of Algorithm 3 holds.*

We now take Proposition 1 as a given, so in particular $d_i = \prod_{j=1}^{i} \|\boldsymbol{L}_j^*\|^2$ after each step.

Proposition 2. *After every step during Algorithm 3, $M^{2(j-1)} \leq d_j \leq M^{2j}$ for $0 \leq j \leq k$.*

Proof. First, we show inductively that $\|\boldsymbol{L}_i^*\| \leq M$ for $1 \leq i \leq k$ at the completion of every step. This clearly this holds after step 1, and if it holds at the beginning of step 2:

- It holds after step 3 since $\|\boldsymbol{L}_1'^*\| = M$ and $\|\boldsymbol{L}_i'^*\| = \|\boldsymbol{L}_{i-1}'^*\|$ for $i > 1$.
- It holds after step 4 since $\max_i \|\boldsymbol{L}_i'^*\| \leq \max_i \|\boldsymbol{L}_i^*\|$ by Theorem 2.1.
- It holds after step 5 since $\boldsymbol{L}_{1..k'}'^* = \boldsymbol{L}_{1..k'}^*$.

Next, we show inductively that $M^{k-1} \leq \operatorname{vol} \boldsymbol{L}$ at the completion of every step. This clearly this holds after step 1, and if it holds at the beginning of step 2:

- It holds after step 3 since $M^{k'-1} = M^k \leq M \operatorname{vol} \boldsymbol{L} = \operatorname{vol} \boldsymbol{L}'$.
- It holds after step 4 since $\operatorname{vol} \boldsymbol{L}' = \operatorname{vol} \boldsymbol{L}$ and $k' = k$.
- It holds after step 5 since $M^{k-1} \leq \operatorname{vol} \boldsymbol{L} \leq M^{k-k'} \operatorname{vol} \boldsymbol{L}'$ (using $\|\boldsymbol{L}_i^*\| \leq M$ for $k' < i \leq k$). Dividing by $M^{k-k'}$ yields the desired inequality $M^{k'-1} \leq \operatorname{vol} \boldsymbol{L}'$.

The upper bound $d_j \leq M^{2j}$ follows by multiplying $\|\boldsymbol{L}_i^*\|^2 \leq M^2$ for $1 \leq i \leq j$. The lower bound $M^{2(j-1)} \leq d_j$ follows by multiplying $M^{2(k-1)} \leq (\operatorname{vol} \boldsymbol{L})^2$ by $M^{-2} \leq \|\boldsymbol{L}_i^*\|^{-2}$ for $j < i \leq k$. □

Proposition 3. *If $k = c + 1$ at the start of step 5 of Algorithm 3 then at least one vector is discarded during that step.*

Proof. By Assertion A, Proposition 2, and the algorithm's required bounds on M:

$$\begin{aligned} \|\boldsymbol{L}_k^*\| &\geq (\operatorname{vol} \boldsymbol{L})^{1/k}/2^{(k-1)/4} && \text{(Theorem 1.3)} \\ &\geq M^{(k-1)/k}/2^{(k-1)/4} && (\operatorname{vol} \boldsymbol{L} \geq M^{k-1}) \\ &> N && (M > 2^{(c+1)/4} N^{1+1/c}) \end{aligned}$$

Thus $d_k/d_{k-1} > N^2$, so k is decreased during step 5. □

Corollary 3. *Assertion C after step 5 of Algorithm 3 holds.*

Proof. $k \leq c$ holds after step 1, and if it holds before step 2 then $k \leq c + 1$ at the beginning of step 5 and $k \leq c$ at the end of step 5 by Proposition 3. Assertion C follows by induction. □

This concludes the proof of correctness. We now examine the runtime of each step of Algorithm 3. Steps 1 and 6 are clearly $O(1)$ and step 2 is executed at most n times. Steps 3 and 5 require $O(c)$ arithmetic operations and step 4 runs InPlaceLLL on lattice with at most $c+1$ vectors, though we need bounds on the bitlength of the numbers used to compute the actual number of bit operations required in these steps.

Proposition 4. *At the conclusion of the following steps during Algorithm 3 we have:*

- *Step 3:* $\max_i \|\boldsymbol{L}_i\| \leq \sqrt{a_\ell^2+1}M$
- *Step 4:* $\max_i \|\boldsymbol{L}_i\| \leq \sqrt{k}M$
- *Step 5:* $\max_i \|\boldsymbol{L}_i\| < M/2$

Proof. After step 4, $\|\boldsymbol{L}_i\|^2 = \sum_{j=1}^{i} \mu_{i,j}^2 \|\boldsymbol{L}_j^*\|^2 \leq iM^2$ since an LLL-reduced basis satisfies $\mu_{i,j}^2 \leq 1$, and $\|\boldsymbol{L}_j^*\| \leq M$ from Proposition 2.

After step 5, by Theorem 1.2, $\|\boldsymbol{L}_i\| \leq 2^{(k-1)/2}\|\boldsymbol{L}_k^*\| \leq 2^{(c-1)/2}N < M/2$ by the algorithm's bounds on M and since $\|\boldsymbol{L}_k^*\| \leq N$ by choice of k during step 5.

Let $\boldsymbol{L}'$ be the value of $\boldsymbol{L}$ after step 3. The first time step 3 is executed we have $\|\boldsymbol{L}_1'\| = M$ and $\|\boldsymbol{L}_2'\| = \sqrt{a_1^2+1}$, so the bound clearly holds in this case. On subsequent executions, we have $\|\boldsymbol{L}_1'\| = M$, and for $i > 1$, $\|\boldsymbol{L}_i'\|^2 = \|\boldsymbol{L}_{i-1}\|^2 + L_{i-1,1}^2 a_{\ell'}^2 < M^2(1+a_{\ell'}^2)$ by the bound following step 5. ☐

Now we consider the running time of Algorithm 3. Since we are only concerned with $\boldsymbol{a} \pmod M$, it is reasonable to require that a_i be in the symmetric range, in which case we have $\max_i \|\boldsymbol{L}_i\| \leq M^2$ at the end of step 3, and can take $B = M^2$ as a bound for the length of vectors in the input lattice. Assume that $c = O(1)$. Then every execution of step 4 runs in $O(\ell(\log M)^3)$ bit operations, which dominates the loop cost. (Steps 3 and 5 work with integers of bitlength $O(c \log M)$ by Propositions 2 and 4.) Since the loop runs $O(n)$ times, and using $\ell \leq n$, the total cost is $O(n^2(\log M)^3)$ bit operations. We obtain the following result.

Theorem 3. *Algorithm 3 returns an LLL-reduced generating matrix* $\boldsymbol{S} \in \mathbb{Z}^{k \times (n+1)}$ *of* $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^M)$ *with* $k \leq c$. *When* $\boldsymbol{a} \in \mathbb{Z}_M^{1 \times n}$ *is given in the symmetric range and* $c = O(1)$, *the running time is* $O(n^2(\log M)^3)$ *bit operations.*

4.2 A refined algorithm

Due to the special form of the lattices under consideration, the running time of InPlaceLLL in Algorithm 3 may be improved on. It is unnecessary to keep track of the entire lattice basis $\boldsymbol{L}$; we show now that $\boldsymbol{L}$ is uniquely determined by its first column.

Lemma 3. *Any $\boldsymbol{L} \in \mathbb{Z}^{k\times(\ell+1)}$ with $\mathcal{L}(\boldsymbol{L}) \subseteq \mathcal{L}(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell}})$ is of the form*

$$\left[\, \boldsymbol{L}_1^{\mathrm{T}} \;\middle|\; \operatorname{rem}_M(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{a}_{1..\ell}) + M\boldsymbol{R} \,\right]$$

for some $\boldsymbol{R} \in \mathbb{Z}^{k\times\ell}$.

Proof. Since $\mathcal{L}(\boldsymbol{L})$ is a sublattice of $\mathcal{L}(\boldsymbol{\Lambda})$ there exists a $\boldsymbol{B} \in \mathbb{Z}^{k\times(\ell+1)}$ such that

$$\begin{aligned}
\boldsymbol{L} &= \boldsymbol{B}\boldsymbol{\Lambda} \\
&= \boldsymbol{B}_{\ell+1}^{\mathrm{T}}\boldsymbol{\Lambda}_{\ell+1} + \boldsymbol{B}_{1..\ell}^{\mathrm{T}}\boldsymbol{\Lambda}_{1..\ell} \\
&= \left[\, \boldsymbol{B}_{\ell+1}^{\mathrm{T}} \;\middle|\; \boldsymbol{B}_{\ell+1}^{\mathrm{T}}\boldsymbol{a}_{1..\ell} + \boldsymbol{B}_{1..\ell}^{\mathrm{T}}\boldsymbol{\Lambda}_{1..\ell,2..\ell+1} \,\right],
\end{aligned}$$

so $\boldsymbol{B}_{\ell+1}^{\mathrm{T}} = \boldsymbol{L}_1^{\mathrm{T}}$. The result follows since M divides every entry of $\boldsymbol{\Lambda}_{1..\ell,2..\ell+1}$ and $\operatorname{rem}_M(\boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{a}_{1..\ell}) = \boldsymbol{L}_1^{\mathrm{T}}\boldsymbol{a}_{1..\ell} + M\boldsymbol{Q}$ for some $\boldsymbol{Q} \in \mathbb{Z}^{k\times\ell}$. □

Corollary 4. *At the conclusion of step 5 of Algorithm 3, when $\boldsymbol{L}$ is expressed in the form from Lemma 3, $\boldsymbol{R}$ is the zero matrix.*

Proof. If $\boldsymbol{R}$ was not the zero matrix then some entry of $\boldsymbol{L}$ is not in the symmetric range $\pmod M$. In which case there would be an entry $|L_{i,j}| \geq M/2$, so $\|\boldsymbol{L}_i\| \geq M/2$, in contradiction to Proposition 4. □

This shows that we can reconstruct all the entries of $\boldsymbol{L}$ from just $\boldsymbol{L}_1^{\mathrm{T}}$ at the conclusion of the algorithm. Furthermore, the entries of $\boldsymbol{L}_{2..\ell+1}^{\mathrm{T}}$ are not needed during the computation since the lattice reduction only depends on the GSO $(\boldsymbol{\mu}, \boldsymbol{d})$, not on $\boldsymbol{L}$ itself. Algorithm 4 modifies Algorithm 3 to only keep track of $\tilde{\boldsymbol{L}} \coloneqq \boldsymbol{L}_1^{\mathrm{T}}$. The other optimization in Algorithm 4 is use the L^2 algorithm to perform the lattice reduction. Instead of taking as input and updating the exact GSO $(\boldsymbol{\mu}, \boldsymbol{d})$, the inplace L^2 algorithm takes as input and updates the exact Gramian $\boldsymbol{G} \coloneqq \boldsymbol{L}\boldsymbol{L}^T$. Step 2 of Algorithm 4 now updates the exact Gramian $\boldsymbol{G}$ instead of the GSO $(\boldsymbol{\mu}, \boldsymbol{d})$.

Assume $c = O(1)$. Then step 4 of Algorithm 4 now executes in $O((\log M)^2)$ bit operations, which again dominates the loop. Since the loop runs $O(n)$

times, the total cost is $O(n(\log M)^2)$ bit operations. Step 6 requires $O(nc)$ arithmetic operations, all on integers of bitlength $O(\log M)$. This give the following result.

Algorithm 4 The $\mathsf{VecRecon}(n, \boldsymbol{a}, M, N, c)$ generating matrix algorithm using L^2.

Input: $\boldsymbol{a} \in \mathbb{Z}_M^{1\times n}$ and $N,\, c \in \mathbb{Z}_{>0}$ with $M > 2^{(c+1)/2} N^{1+1/c}$.
Output: An LLL-reduced generating matrix $\boldsymbol{S} \in \mathbb{Z}^{k\times(n+1)}$ of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^M)$ with $k \le c$.

```
// Note $\tilde{\boldsymbol{L}} \in \mathbb{Z}^{k\times 1}$, $\boldsymbol{G} \in \mathbb{Z}^{k\times k}$ throughout.

1. [Initialization]
   $k := 1$; $\tilde{L}_1 := 1$; $G_{1,1} := 1$;

2. [Iterative lattice augmentation]
   for $\ell := 1$ to $n$ do

      3. [Add new vector to generating matrix]
         $k := k + 1$; $\boldsymbol{G} := \left[\begin{array}{c|c} 0 & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{G} \end{array}\right] + \left[\begin{array}{c|c} M^2 & M a_\ell \tilde{\boldsymbol{L}}^{\mathrm{T}} \\ \hline M a_\ell \tilde{\boldsymbol{L}} & a_\ell^2 \tilde{\boldsymbol{L}} \tilde{\boldsymbol{L}}^{\mathrm{T}} \end{array}\right]$; $\tilde{\boldsymbol{L}} := \left[\begin{array}{c} 0 \\ \hline \tilde{\boldsymbol{L}} \end{array}\right]$;

      4. [L$^2$ reduction]
         $\mathsf{InPlaceL}^2(k, \boldsymbol{G}, \tilde{\boldsymbol{L}})$;

      5. [Remove superfluous vectors from generating matrix]
         Compute $\boldsymbol{d}$ by fraction-free Gaussian elimination on $\boldsymbol{G}$.
         Set $k$ to be the maximal value such that $d_k/d_{k-1} \le N^2$.
         If no such $k$ exists, return the unique element of $\mathbb{Z}^{0\times(n+1)}$.
         $\tilde{\boldsymbol{L}} := \tilde{\boldsymbol{L}}_{1..k}$; $\boldsymbol{G} := \boldsymbol{G}_{1..k,1..k}$;

   assert A. $\left[\, \tilde{\boldsymbol{L}} \,\middle|\, \mathrm{rem}_M(\tilde{\boldsymbol{L}} \boldsymbol{a}_{1..\ell}) \,\right]$ is a generating matrix of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..\ell}}^M)$ with
             Gramian $\boldsymbol{G}$
          B. $k \le c$

6. [Complete generating matrix]
   return $\boldsymbol{S} := \left[\, \tilde{\boldsymbol{L}} \,\middle|\, \mathrm{rem}_M(\tilde{\boldsymbol{L}} \boldsymbol{a}) \,\right]$;
```

Theorem 4. *Algorithm 4 returns an LLL-reduced generating matrix* $\boldsymbol{S} \in \mathbb{Z}^{k\times(n+1)}$ *of* $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}}^M)$ *with* $k \le c$. *When* $\boldsymbol{a} \in \mathbb{Z}_M^{1\times n}$ *is given in the symmetric* *range and $c = O(1)$, the running time is $O(n(\log M)^2)$ bit operations.*

5 Application to linear system solving

In this section let $\boldsymbol{A} \in \mathbb{Z}^{n\times n}$ be a nonsingular matrix and $\boldsymbol{b} \in \mathbb{Z}^n$ be a vector such that $\|[\,\boldsymbol{A} \mid \boldsymbol{b}\,]\|_{\max} \leq B$. Consider the problem of computing $\boldsymbol{x} \in \mathbb{Q}^n$ such that $\boldsymbol{Ax} = \boldsymbol{b}$, using for example Dixon's algorithm [5]. This requires reconstructing the solution $\boldsymbol{x}$ from its modular image $\operatorname{rem}_M(\boldsymbol{x})$, where $M = p^m$ for some prime $p \nmid \det(\boldsymbol{A})$ and $m \in \mathbb{Z}_{>0}$ is large enough that the reconstruction is unique.

We can use p-adic lifting to recover the image vector $\boldsymbol{a} = \operatorname{rem}_M(\boldsymbol{x})$ for $m = 2, 3, \ldots$. The cost of the lifting phase of the solver is directly related to the number of lifting steps m, which dictates the precision of the image. Highly optimized implementations of p-adic lifting [3, 4, 6, 8] employ an output sensitive approach to the rational reconstruction of the vector $\boldsymbol{x}$ from $\boldsymbol{a}$ in order to avoid computing more images than required. As m increases, the algorithm periodically attempt to perform a rational reconstruction of the current image vector. The attempted rational reconstruction should either return the unique minimal denominator solution or FAIL. When FAIL is returned more lifting steps are performed before another rational reconstruction is attempted.

In the following let $\boldsymbol{v} \in \mathbb{Z}^n$ and $d \in \mathbb{Z}$. Suppose $(d, \boldsymbol{v})$ is such that $\boldsymbol{a} = \operatorname{rem}_M(\boldsymbol{v}/d)$, that is, $\boldsymbol{Av} \equiv d\boldsymbol{b} \pmod M$. To check if $\boldsymbol{Av} = d\boldsymbol{b}$, that is, if $\boldsymbol{v}/d$ is the actual solution of the system $\boldsymbol{Ax} = \boldsymbol{b}$, we could directly check if $\boldsymbol{Av} = d\boldsymbol{b}$ by performing a matrix vector product and scalar vector product. However, this direct check is too expensive. The following idea of Cabay [1] can be used to avoid the direct check, requiring us to only check some magnitude bounds.

Lemma 4. *If $\|\boldsymbol{v}\|_\infty < M/(2nB)$, $|d| < M/(2B)$ and $\boldsymbol{Av} \equiv d\boldsymbol{b} \pmod M$ then $\boldsymbol{x} = \boldsymbol{v}/d$.*

Proof. Note that $\|\boldsymbol{Av}\|_\infty \leq nB\|\boldsymbol{v}\|_\infty$ and $\|d\boldsymbol{b}\|_\infty \leq B|d|$, so by the given bounds $\|\boldsymbol{Av}\|_\infty < M/2$ and $\|d\boldsymbol{b}\|_\infty < M/2$. Every integer absolutely bounded by $M/2$ falls into a distinct congruence class modulo M, so since the components of $\boldsymbol{Av}$ and $d\boldsymbol{b}$ are in this range and componentwise they share the same congruence classes, $\boldsymbol{Av} = d\boldsymbol{b}$, and the result follows. $\square$

Algorithm 5 shows how Lemma 4 can be combined with the elementwise rational reconstruction approach to get an output sensitive algorithm for the reconstruction of $\boldsymbol{x}$ from its image $\boldsymbol{a}$. This algorithm does not take the size bound N as a parameter, but calculates an N such that there will be at most one lowest-terms reconstruction that is guaranteed by Lemma 4 to be the unique solution vector if it exists. Note that the elementwise approach with $N = D$ requires us to choose N to satisfy $M > 2N^2$. Thus, for the algorithm to succeed, we need $M \in \Omega(\beta^2)$, where β is the maximum of the magnitudes of the denominator and numerators of the actual solution vector of the system.

Algorithm 5 An output sensitive $\mathsf{LinSolRecon}(n, \boldsymbol{a}, M, B)$ using scalar reconstruction.

Input: The image $\boldsymbol{a} \in \mathbb{Z}_M^n$ of the solution of the linear system $\boldsymbol{Ax} = \boldsymbol{b}$, and $B \in \mathbb{Z}_{>0}$, an upper bound on the magnitude of the entries of $\boldsymbol{A}$ and $\boldsymbol{b}$.
Output: Either the solution $\boldsymbol{x} \in \mathbb{Q}^n$ or FAIL.

```
// Need $M > 2N^2$ and $M > 2nBN$.

1. [Set an acceptable size bound]
   $N := \lfloor \min\big(\sqrt{M/2}, M/(2nB)\big) \rfloor$;

2. [Simultaneous rational reconstruction]
   $d := 1$;
   for $i := 1$ to $n$ do

      3. [Entrywise rational reconstruction]
         $d := d \cdot \mathsf{RatRecon}(\mathrm{rem}_M(da_i), M, N, \lfloor N/d \rfloor)$;
         If the call to RatRecon returns FAIL then return FAIL.

4. [Check reconstruction]
   If $\|\mathrm{rem}_M(d\boldsymbol{a})\|_\infty > N$ then return FAIL.

5. [Return solution]
   return $\mathrm{rem}_M(d\boldsymbol{a})/d$;
```

Algorithm 6 shows how Lemma 4 can be combined with $\mathsf{VecRecon}$ instead to get an output sensitive algorithm for the reconstruction. For this algorithm we need only $M > 2^{(c+1)/2}N^{1+1/c}$ to satisfy the precondition of $\mathsf{VecRecon}$. On the one hand, Lemma 5 shows that Algorithm 6 will never return an incorrect answer. On the other hand, Lemma 6 shows that the algorithm will succeed for $M \in \Omega(\sqrt{n}\beta^{1+1/c})$.

Algorithm 6 An output sensitive $\mathsf{LinSolRecon}(n, \boldsymbol{a}, M, B, c)$ using vector reconstruction.

Input: The image $\boldsymbol{a} \in \mathbb{Z}_M^n$ of the solution of the linear system $\boldsymbol{Ax} = \boldsymbol{b}$, and $B \in \mathbb{Z}_{>0}$, an upper bound on the magnitude of the entries of $\boldsymbol{A}$ and $\boldsymbol{b}$. Also, a parameter $c \in \mathbb{Z}_{>0}$ controlling the maximum lattice dimension to use in $\mathsf{VecRecon}$.

Output: Either the solution $\boldsymbol{x} \in \mathbb{Q}^n$ or FAIL.

```
    // Need $M > 2^{(c+1)/2}N^{1+1/c}$ and $M > 2^{(c+1)/2}nBN$.

    1. [Set an acceptable size bound]
       $N := \lfloor \min\big(M^{c/(c+1)}/2^{c/2}, M/(2^{(c+1)/2}nB)\big) \rfloor$;

    2. [Vector rational reconstruction]
       $\boldsymbol{S} := \mathsf{VecRecon}(n, \boldsymbol{a}, M, N, c) \in \mathbb{Z}^{k\times(n+1)}$;
       If $k = 0$ then **return** FAIL.

**assert** $k = 1$

    3. [Return solution]
       **return** $\boldsymbol{S}_{2..n+1}/S_1$;
```

Lemma 5. *If Algorithm 6 does not return FAIL then the output is the correct solution $\boldsymbol{x}$.*

Proof. First, note that every entry of $\boldsymbol{S}$ is absolutely bounded by $M/(2nB)$:

$$\begin{aligned}
\|\boldsymbol{S}\|_{\max} &\le \max_i \|\boldsymbol{S}_i\| && \text{(Norm comparison)}\\
&\le 2^{(k-1)/2}\|\boldsymbol{S}_k^*\| && \text{(Theorem 1.2)}\\
&\le 2^{(c-1)/2}N && \text{(Choice of } k \text{ in } \mathsf{VecRecon})\\
&< M/(2nB) && (M > 2^{(c+1)/2}nBN)
\end{aligned}$$

Then we can apply Lemma 4 on any row i of $\boldsymbol{S}$, since $\boldsymbol{A}(\boldsymbol{S}_{i,2..n+1})^{\mathrm{T}} \equiv S_{i,1}\boldsymbol{b}$ (mod M) by construction of $\boldsymbol{S}$. Therefore every row of $\boldsymbol{S}$ yields a solution $\boldsymbol{x} = \boldsymbol{S}_{i,2..n+1}/S_{i,1}$, but since there is only one solution and the rows of $\boldsymbol{S}$ are linearly independent, $\boldsymbol{S}$ can have at most one row. Assuming the algorithm did not return FAIL, we have $\boldsymbol{x} = \boldsymbol{S}_{2..n+1}/S_1$, as required. $\square$

Lemma 6. *Let β be the maximum of the magnitudes of the denominator and numerators of the unique rational solution vector of the system. If $M > 2^{(c+1)/2}(\sqrt{n+1}\beta)^{1+1/c}$ then Algorithm 6 will not return FAIL.*

Proof. Let $\mathrm{denom}(\boldsymbol{x})$ denote a function which returns the minimal $d \in \mathbb{Z}_{>0}$ such that $d\boldsymbol{x} \in \mathbb{Z}^n$, and let $\mathrm{numer}(\boldsymbol{x}) := \mathrm{denom}(\boldsymbol{x}) \cdot \boldsymbol{x}$. Then

$$\left\| \left[\, \mathrm{denom}(\boldsymbol{x}) \;\middle|\; \mathrm{numer}(\boldsymbol{x}) \,\right] \right\|^2 \leq (n+1)\,\beta^2.$$

Therefore, if $\sqrt{n+1}\beta \leq N$, then VecRecon is guaranteed to find a generating lattice which includes $\left[\, \mathrm{denom}(\boldsymbol{x}) \;\middle|\; \mathrm{numer}(\boldsymbol{x}) \,\right]$. $\square$

The running time of Algorithm 6 is simply that of VecRecon, which by Proposition 4 is $O(n(\log M)^2)$ bit operations. Table 1 shows the reduction in required bitlength of $\log M$ by comparing some minimal bounds on $\log M$ for Algorithms 5 and 6 to succeed.

Table 1: The value of $\log M$ required to guarantee Algorithms 5 and 6 return a solution.

n	B	Alg. 5	Alg. 6 $c = 2$	Alg. 6 $c = 3$	Alg. 6 $c = 4$	Alg. 6 $c = 5$
200	1	1060	799	711	667	641
400	1	2397	1802	1603	1503	1443
800	1	5348	4016	3570	3348	3214
1600	1	11805	8859	7875	7384	7089
Alg. 6/Alg. 5			$\approx 75\%$	$\approx 67\%$	$\approx 63\%$	$\approx 60\%$

References

[1] S. Cabay. Exact solution of linear systems. In *Proc. Second Symp. on Symbolic and Algebraic Manipulation*, pages 248—253, 1971.

[2] Z. Chen. A BLAS based C library for exact linear algebra on integer matrices. Master's thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2005.

[3] Z. Chen and A. Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '05*, pages 92–99. ACM Press, New York, 2005.

[4] W. Cook and D. Steffy. Solving very sparse rational systems of equations. Technical report, School of Industrial and Systems Engineering, Georgia Institute of Technology, 2009. Available at `http://www2.isye.gatech.edu/%7Edsteffy/papers/OSLifting.pdf`.

[5] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numer. Math.*, 40:137–141, 1982.

[6] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. LinBox: A generic library for exact linear algebra. In A. J. Cohen and N. Gao, X.-S. andl Takayama, editors, *Proc. First Internat. Congress Math. Software ICMS 2002, Beijing, China*, pages 40–50, Singapore, 2002. World Scientific.

[7] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2 edition, 2003.

[8] P. Giorgi. *Arithmetic and algorithmic in exact linear algebra for the LinBox library*. PhD thesis, École normale supérieure de Lyon, LIP, Lyon, France, December 2004.

[9] E. Kaltofen and H. Rolletschek. Computing greatest common divisors and factorizations in quadratic number fields. *Math. Comput.*, 53(188), 1989.

[10] E. Kaltofen and Z. Yang. On exact and approximate interpolation of sparse rational functions. In J. P. May, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '07*, pages 203–210. ACM Press, New York, 2007.

[11] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal of Computing*, 14(1):196–209, 1985.

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

[13] D. Lichtblau. Half-GCD and fast rational recovery. In M. Kauers, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '05*, pages 231–236. ACM Press, New York, 2005. Extended version available at `http://library.wolfram.com/infocenter/Conferences/7534/HGCD_and_planar_lattices.pdf`.

[14] M. Monagan. Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. In J. Gutierrez, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '04*, pages 243–249. ACM Press, New York, 2004.

[15] P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal of Computing*, 39(3), 2009.

[16] Z. Olesh and A. Storjohann. The vector rational function reconstruction problem. In I. Kotsireas and E. Zima, editors, *Proc. of the Waterloo Workshop on Computer Algebra: devoted to the 60th birthday of Sergei Abramov (WWCA-2006)*, pages 137–149. World Scientific, 2006.

[17] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2 edition, 2005.

[18] M. van Hoeij and A. Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. To appear in *Proc. of the 9th Latin American Theoretical Informatics Symposium: LATIN 2010*.