Vector Rational Number Reconstruction

By

Curtis Bright

A research paper presented to the

University of Waterloo

In partial fulfillment of the requirements for the degree of

Master of Mathematics

in

Computational Mathematics

Supervisors: Arne Storjohann and Kevin G. Hare

Waterloo, Ontario, Canada

Vector Rational Number Reconstruction

August 31, 2009

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 $(a_i)_{1\leq i\leq n}$ modulo M, subject to the condition $0 < d \leq N$ and $|n_i| \leq N$ for a given magnitude bound N. Using elementwise rational reconstruction requires that $M > 2N^2$. We present an algorithm, based on lattice basis reduction, which can perform the reconstruction for problem instances such as linear system solving even when the modulus is of considerably smaller magnitude, $M > 2^{(c+1)/2}N^{1+1/c}$ for c a small constant. The cost of the new algorithm is $O(nc^3(\log M)^3)$ bit operations. This extends the work done in [9] for rational function reconstruction.

1 Introduction

Many algorithms in computer algebra that compute with rational numbers 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 ($M = p_1p_2\cdots p_m$) or a variation of Newton-Hensel lifting ($M = p^m$).

The overall cost of an algorithm that uses a homomorphic imaging scheme depends on m, the number of images computed, which is directly related to the bitlength of the modulus M. Ideally, just enough images are computed to allow reconstruction of the common denominator d. 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.

The remainder of this introduction is divided into three parts. First we recall the standard scalar rational number reconstruction problem. Then we recall how the simultaneous version of the problem mentioned above can be solved using an algorithm for the scalar case. Finally, we define the vector rational reconstruction problem and give an outline of our algorithm that solves the problem.

1.1 Rational Number Reconstruction

A *rational number reconstruction* of an integer $a \in \mathbb{Z}$ with respect to a modulus $M > 0$ is a rational number $n/d \in \mathbb{Q}$ (with coprime n, $d \in \mathbb{Z}$) such that

$$a \equiv n/d \pmod{M}. \tag{1}$$

The set of solutions to (1) are linked to the extended Euclidean algorithm and the continued fraction expansion of a/M, see for example [11]. In general, the rational reconstruction of an integer is not unique, but uniqueness can be ensured in certain cases by stipulating bounds for the magnitude of n and d. In addition to a and M, the *rational number reconstruction problem* takes as input the integer bounds N, $D > 0$. A solution to the problem is a pair of integers (d, n) such that

$$da \equiv n \pmod{M}, \qquad |n| \leq N, \qquad 0 < d \leq D. \tag{2}$$

Note that if (d, n) is a solution to (2) that satisfies $\gcd(d, M) = 1$ then $a \equiv n/d \pmod{M}$. For convenience, we prefer to define a solution to the problem using the slightly more general condition $da \equiv n$ rather than $a \equiv n/d$.

As mentioned above, (2) may not have a unique solution in general. For example, consider the bounds $N = D = 5$. If $a \equiv 10 \pmod{45}$, then $(5, 5)$ and $(4, -5)$ are linearly independent solutions to (2), though the first does not yield a rational reconstruction. If $a \equiv 9 \pmod{41}$ then $(5, 4)$ and $(4, -5)$ are linearly independent solutions which both yield rational reconstructions.

However, if the condition $M > 2ND$ is satified, and (2) is solvable, then there is a minimal solution (d, n) such that every solution of (2) is of the form $(\alpha d, \alpha n)$ for $\alpha \in \mathbb{Z}_{>0}$. The minimal solution may be found by running the extended Euclidean algorithm on M and $0 \leq a < M$, producing the sequence of equations $Ms_i + at_i = r_i$. Then the minimal solution is $(|t_k|, (-1)^{k+1} r_k)$, where k is the minimal index such that $r_k \leq N$ (see [10] Theorem 4.9).

For example, if $N = D = 5$ and $a \equiv 26 \pmod{51}$ then the extended Euclidean algorithm on $(51, 26)$ gives:

i	s_i	t_i	r_i
0	1	0	51
1	0	1	26
2	1	-1	25
3	-1	2	1
4	26	-51	0

In this case $k = 3$ and the minimal solution is $(2, 1)$. All solutions of (2) are given by $(2\alpha, \alpha)$ for $1 \leq \alpha \leq 2$, so a has the unique 'short' rational number reconstruction of $1/2$.

1.2 Simultaneous Rational Number Reconstruction

The *simultaneous rational number reconstruction problem* takes as input integers a_i for $1 \leq i \leq n$. A solution to the problem is an integer d together with integers n_i for $1 \leq i \leq n$

such that

$$da_i \equiv n_i \pmod{M} \quad \text{and} \quad |n_i| \le N \quad \text{for} \quad 1 \le i \le n, \qquad 0 < d \le D. \tag{3}$$

If M is sufficiently large, the simultaneous version of the problem may be solved by considering it as n instances of the rational number reconstruction problem and then checking if a common denominator exists for which the bounds are satisfied. Similar to the scalar version of the problem, if $M > 2ND$ and (3) is solvable then there exists a minimal d that generates all solutions. For $M > 2ND$, let $\mathsf{RatRecon}(a, M, N, D)$ denote a function which returns the $d > 0$ such that $(d, \mathrm{rem}_M(da))$ is the minimal solution to (2), or returns FAIL in case no solution of (2) exists.

The following code fragment shows how to solve an instance of the simultaneous rational number reconstruction problem in case $M > 2ND$ is satisfied.

```
$d := 1$;
**for** $i := 1$ to $n$ **do**
    // If the call to RatRecon returns FAIL then abort.
    $d := d \cdot \mathsf{RatRecon}(\mathrm{rem}_M(da_i), M, N, \lfloor D/d \rfloor)$;
```

Upon completion of the code fragment, if none of the calls to RatRecon returned FAIL, and if $\mathrm{rem}_M(da_i) \le N$ for $1 \le i \le n$, then d is the desired minimal common denominator. Note that the approach described above requires that $M > 2ND$ (a precondition for RatRecon). Otherwise this method cannot necessarily be used, even if there is in fact a unique solution, since *entrywise* there may well not be a unique solution. In general, when $M \le 2ND$ there may be multiple linearly independent solutions to (3).

In the next section we define close variant of the simultaneous rational number reconstruction problem and give an overview of our algorithm to efficiently compute a generating set of all solutions even when M is too small to guarantee a unique solution.

1.3 Vector Rational Number Reconstruction

The *vector rational number reconstruction problem* takes as input a dimension $n \in \mathbb{Z}_{>0}$, a modulus $M \in \mathbb{Z}_{>0}$, a single image integer vector $\boldsymbol{a} \in \mathbb{Z}_M^n$, and a size bound $N > 0$. A solution to the problem is a pair $(d, \boldsymbol{n}) \in (\mathbb{Z}, \mathbb{Z}^n)$ such that

$$d\boldsymbol{a} \equiv \boldsymbol{n} \pmod{M}, \qquad \left\| \left[\, d \mid \boldsymbol{n} \,\right] \right\|_2 \le N. \tag{4}$$

Here, we use the 2-norm instead of the ∞-norm because this is a more natural condition for the algorithm we will present. Also, instead of separate bounds for d and the entries of $\boldsymbol{n}$, we use a common bound for the vector $\left[\, d \mid \boldsymbol{n} \,\right]$.

In this paper we present an algorithm that computes solutions to (4). Our algorithm actually computes a complete "generating set" for (4), that is, a set of linearly independent vectors $(d, \boldsymbol{n})$ to $d\boldsymbol{a} \equiv \boldsymbol{n} \pmod{M}$ such that every solution of (4) 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 for the generating set to have dimension zero (no nonzero solution) or one (a unique minimal denominator solution) is that $M > 2N^2$. On the other hand, if c is a small 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. If $\mathsf{M}(x)$ denotes the cost of multiplication of integers of bitlength x, the algorithm requires

$$O(n(c^3 \log M)\,\mathsf{M}(\log M)) = O(nc^3(\log M)^3) \tag{5}$$

bit operations.

For inputs $\boldsymbol{a}$ coming from applications such as linear system solving, the solution space of (4) will be unique even for M considerably smaller than $2N^2$. For example, the problem instance with bound $N = 10^4$ and the vector of residues

$$\boldsymbol{a} = \begin{bmatrix} -23677 & -49539 & 74089 & -21989 & 63531 \end{bmatrix} \in \mathbb{Z}^5_{195967} \tag{6}$$

has a solution space of dimension one, which yields the unique lowest-terms vector reconstruction

$$\boldsymbol{n}/d = \begin{bmatrix} -3256 & -2012 & 331 & 891 & -1692 \end{bmatrix} / \, 3137,$$

although $M = 195967$ is easily smaller than $2N^2 = 2 \cdot 10^8$.

Considering (5), our algorithm can be used to compute the unique solution efficiently provided $M > 2^{(c+1)/2}N^{1+1/c}$ is satisfied for a small constant c. As a concrete example consider $N = 10^{10000}$. For $c = 5$, M needs to have about 12000 decimal digits to ensure that $M > 2^{(c+1)/2}N^{1+1/c}$ is satisfied. But for $M > 2N^2$ to be satisfied M needs to have length about 20000 decimal digits.

Now we give an outline of our approach. First note that the problem of finding solutions to (4) 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:

$$\begin{bmatrix} & & & & M \\ & & & \iddots & \\ & & M & & \\ & M & & & \\ 1 & a_1 & a_2 & \cdots & a_n \end{bmatrix} \in \mathbb{Z}^{(n+1)\times(n+1)}. \tag{7}$$

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 problem instance from (6), which gives rise to the lattice with basis matrix

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

The vectors in this lattice of norm less than or equal to N give all solutions of (4).

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 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 $M/4 > N$, 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 run LLL on the entire lattice, for two reasons. Firstly, the fudge factor $2^{(n-1)/2}$ becomes too large to guarantee that short enough vectors will be found. Secondly, the process of LLL reduction is much too costly. Assuming the a_i are given in the symmetric range the cost of running LLL on lattices of the form (7) is $O(n^6(\log M)^3)$ bit operations.

It is unnecessary to immediately reduce the entire lattice, however. The structure of the lattice permits a kind of iterative reduction. For example, consider reducing only the lower-left 2×2 submatrix of $\boldsymbol{L}$:

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

We can now use this to 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 contribution of this paper is to show that if $M > 2^{(c+1)/2}N^{1+1/c}$, for $c \in \mathbb{Z}_{>0}$ a small constant, 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.

The rest of this paper is organized as follows. In Section 2 we establish our notation and recall the required facts about lattices and the LLL algorithm. In Section 3 we state our basic algorithm for vector rational reconstruction, prove its correctness, and do a simple cost analysis. In Section 4 we make several improvements to the algorithm which significantly improves its running time. In Section 5 we show how the vector rational reconstruction problem is useful for solving nonsingular linear systems.

2 Preliminaries

2.1 Notation

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, $\|\boldsymbol{x}\| := \sqrt{\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}}$.

Vectors are denoted by lower-case bold variables and matrices by upper-case or greek bold variables, though the boldface is dropped when refering to individual entries. We denote the zero vector by $\boldsymbol{0}$ (the dimension will be clear from context).

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

2.2 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

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

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 $\operatorname{vol} \boldsymbol{L} := \sqrt{\det(\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}})}$.

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

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

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.

2.3 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.,

$$\begin{bmatrix} \boldsymbol{L}_1 \\ \boldsymbol{L}_2 \\ \vdots \\ \boldsymbol{L}_k \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \mu_{2,1} & 1 & & \\ \vdots & \vdots & \ddots & \\ \mu_{k,1} & \mu_{k,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1^* \\ \boldsymbol{L}_2^* \\ \vdots \\ \boldsymbol{L}_k^* \end{bmatrix} \quad \text{with} \quad \mu_{i,j} = \frac{\langle \boldsymbol{L}_i, \boldsymbol{L}_j^* \rangle}{\|\boldsymbol{L}_j^*\|^2}.$$

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 $d_i = (\operatorname{vol} \boldsymbol{L}_{1..i})^2$. Also, the d_i are useful as denominators for $\boldsymbol{L}^*$ and $\boldsymbol{\mu}$:

$$\begin{aligned} d_{i-1}\boldsymbol{L}_i^* &\in \mathcal{L}(\boldsymbol{L}) \subseteq \mathbb{Z}^{1\times n} \\ d_i \boldsymbol{\mu}_i^{\mathrm{T}} &\in \mathbb{Z}^{k\times 1} \end{aligned}$$

2.4 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 $\|\boldsymbol{\mu} - \boldsymbol{I}_{k\times k}\|_{\max} \leq \frac{1}{2}$ and *2-reduced* if $d_i d_{i-2} \geq (\frac{3}{4} - \mu_{i,i-1}^2) d_{i-1}^2$ for $1 < i \leq k$. 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 $\mathcal{L}(\boldsymbol{L}) = \mathcal{L}(\boldsymbol{L}')$; the algorithm from [6] accomplishes this in

$$O(nk^3 \log B\, \mathsf{M}(k \log B)) = O(nk^5 (\log B)^3)$$

bit operations, where $\max_i \|\boldsymbol{L}_i\| \leq B$. We restate this algorithm as Algorithm 1.

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

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

1. $\|\boldsymbol{L}_i^*\| \le 2^{(j-i)/2}\|\boldsymbol{L}_j^*\|$ *for* $i \le j$

2. $\max_i\|\boldsymbol{L}_i\| \le 2^{(k-1)/2}\|\boldsymbol{L}_k^*\|$

3. $(\operatorname{vol}\boldsymbol{L})^{1/k}/2^{(k-1)/4} \le \|\boldsymbol{L}_k^*\|$

Theorem 2. *During the running of Algorithm 2:*

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*

3 The Basic VecRecon 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 $\boldsymbol{\Lambda}$. 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 $c \ge 1$ is a small constant such that $M > 2^{(c+1)/2}N^{1+1/c}$ is satisfied. The basic algorithm pseudocode is given as Algorithm 3. We now prove the assertions after step 5 hold, from which the correctness of the algorithm follows.

The rest of the section is devoted to proving the following theorem. In Section 3.1 we prove correctness and in Section 3.2 we give a crude cost analysis.

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* $k \le c$*. When* $\boldsymbol{a} \in \mathbb{Z}_M^{1\times n}$ *is given in the symmetric range, the running time is* $O(n^2c^5(\log M)^3)$ *bit operations.*

3.1 Algorithm Correctness

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* $1 \le i \le k$*, and if* $\boldsymbol{L}$ *is LLL-reduced then so is* $\boldsymbol{L}_{1..i}$*.*

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. $\square$

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 $\mathcal{L}(\boldsymbol{S}_{1..k-1})$. $\square$

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

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

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, $M^{2(j-1)} \le d_j \le M^{2j}$ for $0 \le j \le k$.*

Proof. First, we show inductively that $\|\boldsymbol{L}_i^*\| \le M$ for $1 \le i \le k$ 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 $\max_i \|\boldsymbol{L}_i'^*\| \le \max_i \|\boldsymbol{L}_i^*\|$ by Theorem 2.1.

- It holds after step 5 since $\boldsymbol{L}_{1..k'}'^* = \boldsymbol{L}_{1..k'}^*$.

Next, we show inductively that $M^{k-1} \le \operatorname{vol} \boldsymbol{L}$ 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 $M^{k'-1} = M^k \le M \operatorname{vol} \boldsymbol{L} = \operatorname{vol} \boldsymbol{L}'$.

- It holds after step 4 since $\operatorname{vol} \boldsymbol{L}' = \operatorname{vol} \boldsymbol{L}$ and $k' = k$.

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

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. □

3.2 Runtime Analysis

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. □

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 $\max_i \|\boldsymbol{L}_i\| \le M^2$ at the end of step 3, and can take $B = M^2$ for our LLL bitlength bound.

Then every execution of step 4 runs in $O(\ell c^3 \log M \,\mathsf{M}(c \log M)) = O(nc^5 (\log M)^3)$ bit operations, which dominates the loop cost. Step 3 works with integers of bitlength $\log M$, and so requires $O(c\,\mathsf{M}(\log M)) = O(c(\log M)^2)$ bit operations. Step 5 works with the d_i (of bitlength $c \log M$ by Proposition 2) and so requires $O(c\,\mathsf{M}(c \log M)) = O(c^3(\log M)^2)$ bit operations.

Since the loop runs $O(n)$ times, the total cost is $O(n^2 c^5 (\log M)^3)$ bit operations.

4 A Refined **VecRecon** Algorithm

Due to the special form of the lattices under consideration, the running time of InPlaceLLL in Algorithm 3 may be improved on. Both the number of swaps and the bitlengths of $\mu_{i,j}$ and $\|\boldsymbol{L}_i^*\|^2$ are less than those required in the general case. Additionally, it is unnecessary to keep track of the entire lattice basis $\boldsymbol{L}$; ultimately $\boldsymbol{L}$ is uniquely determined by its first column.

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, the running time is $O(nc^3(\log M)^3)$ bit operations.*

4.1 Algorithm Correctness

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

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

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

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

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 $\mathrm{rem}_M(\boldsymbol{L}_1^{\mathrm{T}} \boldsymbol{a}_{1..\ell}) = \boldsymbol{L}_1^{\mathrm{T}} \boldsymbol{a}_{1..\ell} + M\boldsymbol{Q}$ 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 $(\mathrm{mod}\ M)$. In which case there would be an entry $|L_{i,j}| \ge M/2$, so $\|\boldsymbol{L}_i\| \ge M/2$, 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. Also, note that the entries of $\boldsymbol{L}_{2..\ell+1}^{\mathrm{T}}$ are otherwise irrelevant to the computation: The execution flow of InPlaceLLL only depends on the GSO $(\boldsymbol{\mu}, \boldsymbol{d})$, not on $\boldsymbol{L}$ itself. Therefore during Algorithm 4 we only keep track of $\tilde{\boldsymbol{L}} := \boldsymbol{L}_1^{\mathrm{T}}$, and pass $\tilde{\boldsymbol{L}}$ to InPlaceLLL, which still does the proper updates to $\tilde{\boldsymbol{L}}$ in place of of $\boldsymbol{L}$. This reduces the cost of size-reduction from $O(\ell k)$ arithmetic operations to $O(k^2)$.

Lemma 4. *Any $\boldsymbol{L} \in \mathbb{Z}^{k\times(\ell+1)}$ with $\mathcal{L}(\boldsymbol{L}) \subseteq \mathcal{L}(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell}})$ satisfies $M^{2(k-1)} \mid (\operatorname{vol} \boldsymbol{L})^2$.*

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 $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{\Lambda}$. Using the Cauchy-Binet formula twice,

$$\begin{aligned}(\operatorname{vol} \boldsymbol{L})^2 &= \det(\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}) \\ &= \sum_S \det((\boldsymbol{B}\boldsymbol{\Lambda})_S^{\mathrm{T}})^2 \\ &= \sum_S \det(\boldsymbol{B}\boldsymbol{\Lambda}_S^{\mathrm{T}})^2 \\ &= \sum_S \Big(\sum_R \det(\boldsymbol{B}_R^{\mathrm{T}}) \det(\boldsymbol{\Lambda}_{R,S})\Big)^2\end{aligned}$$

where the sums range over all subsets of $\{1, \ldots, \ell+1\}$ with k elements. For all such subsets, $M^{k-1} \mid \det(\boldsymbol{\Lambda}_{R,S})$, so every term in the outer sum has a factor of $M^{2(k-1)}$. □

Corollary 5. *$M^{2(j-1)} \mid d_j$ for $1 \leq j \leq k$ at the completion of every step in Algorithm 3.*

Proof. Follows by Lemma 4 since $\mathcal{L}(\boldsymbol{L}_{1..j}) \subseteq \mathcal{L}(\boldsymbol{L}) \subseteq \mathcal{L}(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell}})$ and $(\operatorname{vol} \boldsymbol{L}_{1..j})^2 = d_j$. □

This suggests that we need only store $\tilde{d}_j := d_j/M^{2(j-1)}$ instead of the explicit d_j like we do in Algorithm 3. This improvement is taken into account in Algorithm 4. Note that the computations during LLL do not depend on the $\|\boldsymbol{L}_j^*\|^2$ directly, but on the *relative* values of $\|\boldsymbol{L}_j^*\|^2$ and $\|\boldsymbol{L}_{j-1}^*\|^2$ for $j \geq 2$. Since

$$\frac{\|\boldsymbol{L}_j^*\|^2}{\|\boldsymbol{L}_{j-1}^*\|^2} = \frac{d_j/d_{j-1}}{d_{j-1}/d_{j-2}} = \frac{d_j d_{j-2}}{d_{j-1}^2} = \frac{\tilde{d}_j \tilde{d}_{j-2}}{\tilde{d}_{j-1}^2},$$

LLL may be called during Algorithm 4 with $\tilde{\boldsymbol{d}}$ rather than $\boldsymbol{d}$. In this case we have $\tilde{d}_0 = d_0/M^{2(-1)} = M^2$.

Proposition 5. *When $\boldsymbol{L} \in \mathbb{Z}^{k\times(\ell+1)}$ and $\mathcal{L}(\boldsymbol{L}) \subseteq \mathcal{L}(\boldsymbol{\Lambda}^M_{\boldsymbol{a}_{1..\ell}})$ the $\tilde{d}_i$s may be used as denominators for $\boldsymbol{L}^*$ and $\boldsymbol{\mu}$:*

$$\|\boldsymbol{L}_i^*\|^2 = M^2 \tilde{d}_i/\tilde{d}_{i-1} \tag{8}$$

$$\tilde{d}_{i-1}\boldsymbol{L}_i^* \in \mathcal{L}(\boldsymbol{L}) \subseteq \mathbb{Z}^{1\times(\ell+1)} \tag{9}$$

$$M^2 \tilde{d}_i \boldsymbol{\mu}_i^{\mathrm{T}} \in \mathbb{Z}^{k\times 1} \tag{10}$$

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 $\boldsymbol{L} = \boldsymbol{B}\boldsymbol{\Lambda}$.

For (8),

$$\|\boldsymbol{L}_i^*\|^2 = \frac{d_i}{d_{i-1}} = \frac{d_i/M^{2(i-1)}}{d_{i-1}/M^{2(i-1)}} = M^2\frac{\tilde{d}_i}{\tilde{d}_{i-1}}.$$

For (9), let $\boldsymbol{\lambda}$ be the inverse of $\boldsymbol{\mu}$, i.e., $\boldsymbol{\lambda}\boldsymbol{L} = \boldsymbol{L}^*$. Then $\boldsymbol{\lambda}\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}} = \boldsymbol{L}^*\boldsymbol{L}^{\mathrm{T}}$ is an upper triangular matrix, since $\langle \boldsymbol{L}_i^*, \boldsymbol{L}_j\rangle = 0$ for $i > j$. Selecting out row i and columns $1..i-1$ from this matrix we have $\boldsymbol{\lambda}_i\boldsymbol{L}(\boldsymbol{L}_{1..i-1})^{\mathrm{T}} = \boldsymbol{0}$. Noting that $\boldsymbol{\lambda}$ is unit lower triangular,

$$\boldsymbol{\lambda}_{i,1..i-1}\boldsymbol{L}_{1..i-1}(\boldsymbol{L}_{1..i-1})^{\mathrm{T}} = -\boldsymbol{L}_i(\boldsymbol{L}_{1..i-1})^{\mathrm{T}}.$$

Since $\boldsymbol{\lambda}_{i,1..i-1}$ is the solution of this linear system, by Cramer's rule,

$$|\lambda_{i,j}| = \frac{|\det(\boldsymbol{L}_{(1..i)\backslash j}(\boldsymbol{L}_{1..i-1})^{\mathrm{T}})|}{\det(\boldsymbol{L}_{1..i-1}(\boldsymbol{L}_{1..i-1})^{\mathrm{T}})} = \frac{|\det(\boldsymbol{B}_{(1..i)\backslash j}\boldsymbol{\Lambda}\boldsymbol{\Lambda}^{\mathrm{T}}(\boldsymbol{B}_{1..i-1})^{\mathrm{T}})|}{\tilde{d}_{i-1}M^{2(i-2)}}.$$

Using Cauchy-Binet twice like in Lemma 4, $M^{2(i-2)}$ divides the numerator. Thus $\tilde{d}_{i-1}\boldsymbol{\lambda}_i \in \mathbb{Z}^{1\times k}$, so $\tilde{d}_{i-1}\boldsymbol{L}_i^* = \tilde{d}_{i-1}\boldsymbol{\lambda}_i\boldsymbol{L} \in \mathcal{L}(\boldsymbol{L})$.

For (10), we can show $M^2\tilde{d}_i$ works as a denominator using (8) and (9):

$$\left(M^2\tilde{d}_i\right)\boldsymbol{\mu}_i^{\mathrm{T}} = \left(\tilde{d}_{i-1}\|\boldsymbol{L}_i^*\|^2\right)\frac{\boldsymbol{L}(\boldsymbol{L}_i^*)^{\mathrm{T}}}{\|\boldsymbol{L}_i^*\|^2} = \boldsymbol{L}\left(\tilde{d}_{i-1}\boldsymbol{L}_i^*\right)^{\mathrm{T}} \in \mathbb{Z}^{k\times 1}. \qquad \square$$

4.2 Runtime Analysis

Steps 1 is clearly $O(1)$ and step 2 is executed at most n times. Steps 3 and 5 require $O(c)$ arithmetic operations and step 6 requires $O(nc)$ arithmetic operations, all on integers of bitlength $O(\log M)$. Step 4 runs InPlaceLLL on lattice with at most $c+1$ vectors, but because of the special lattice form, all computations in LLL can be done with integers of bitlength $O(\log M)$. There are $O(c\log M)$ swaps and $O(c^2)$ arithmetic operators required during size reduction. Therefore Step 4 of Algorithm 4 dominates, and requires $O(c^3(\log M)^3)$ bit operations every iteration. Since the loop runs $O(n)$ times, the total cost is $O(nc^3(\log M)^3)$ 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 $\left\|\left[\,\boldsymbol{A} \mid \boldsymbol{b}\,\right]\right\|_{\max} \le B$. Consider the problem of computing $\boldsymbol{x} \in \mathbb{Q}^n$ such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, using for example Dixon's algorithm [3]. This requires reconstructing the solution $\boldsymbol{x}$ from its modular image $\mathrm{rem}_M(\boldsymbol{x})$, 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 $\boldsymbol{a} = \mathrm{rem}_M(\boldsymbol{x})$ for $m = 2, 3, \ldots$. The cost of the lifting is directly related to the number of lifting steps m, which dictates the precision of the image. Highly optimized implementations of p-adic lifting [2, 4, 5] 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 will 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 $\boldsymbol{a} = \operatorname{rem}_M(\boldsymbol{v}/d)$, i.e., $\boldsymbol{A}\boldsymbol{v} \equiv d\boldsymbol{b} \pmod{M}$. To check if $\boldsymbol{A}\boldsymbol{v} = d\boldsymbol{b}$, that is, if $\boldsymbol{v}/d$ is the actual solution of the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, we could directly check if $\boldsymbol{A}\boldsymbol{v} = d\boldsymbol{b}$ 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 5. *If $\|\boldsymbol{v}\|_\infty < M/(2nB)$, $|d| < M/(2B)$ and $\boldsymbol{A}\boldsymbol{v} \equiv d\boldsymbol{b} \pmod{M}$ then $\boldsymbol{x} = \boldsymbol{v}/d$.*

Proof. Note that $\|\boldsymbol{A}\boldsymbol{v}\|_\infty \leq nB\|\boldsymbol{v}\|_\infty$ and $\|d\boldsymbol{b}\|_\infty \leq B|d|$, so by the given bounds $\|\boldsymbol{A}\boldsymbol{v}\|_\infty < M/2$ 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 $\boldsymbol{A}\boldsymbol{v}$ and $d\boldsymbol{b}$ are in this range and componentwise they share the same congruence classes, $\boldsymbol{A}\boldsymbol{v} = d\boldsymbol{b}$, and the result follows. $\square$

Algorithm 5 shows how Lemma 5 is combined with the simultaneous rational reconstruction algorithm described in Section 1.2 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, and if a reconstruction exists it yields the actual solution $\boldsymbol{x}$.

The iterative reconstruction algorithm given in Section 1.2 with $N = D$ requires $M > 2N^2$, and this ensures there is at most one minimal reconstruction. To guarantee that any reconstruction returned actually yields a solution we take $M > 2nBN$. Since the entries of any reconstruction returned by the algorithm are bounded in magnitude by N, by Lemma 5 if $N < M/(2nB)$ then the rational reconstruction is the actual solution.

Algorithm 6 shows how Lemma 5 is combined with VecRecon instead to get an output sensitive algorithm for the reconstruction. In this case we need $M > 2^{(c+1)/2}N^{1+1/c}$ to satisfy the precondition of VecRecon, and $M > 2^{(c+1)/2}nBN$ to ensure the output is the real solution.

Lemma 6. *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 $M/(2nB)$:

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

Then we can use Lemma 5 on any row i of $\boldsymbol{S}$, since $\boldsymbol{A}(\boldsymbol{S}_{i,2..n+1})^{\mathrm{T}} \equiv S_{i,1}\boldsymbol{b} \pmod{M}$ by contruction 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. □

Lemma 7. *If $M > 2^{(c+1)/2}(\sqrt{n+1}\, n^{n/2}B^n)^{1+1/c}$ then Algorithm 6 will not return FAIL.*

Proof. Let 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 numer($\boldsymbol{x}$) $:=$ denom($\boldsymbol{x}$) $\cdot\, \boldsymbol{x}$. By Hadamard's bound and Cramer's rule, denom($\boldsymbol{x}$) and the entries of numer($\boldsymbol{x}$) are bounded in absolute value by $n^{n/2}B^n$, from which it follows that

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

Therefore if $\sqrt{n+1}n^{n/2}B^n \le N$ then $\mathsf{VecRecon}$ is guaranteed to find a generating lattice which includes $\left[\, \operatorname{denom}(\boldsymbol{x}) \mid \operatorname{numer}(\boldsymbol{x}) \,\right]$. In fact, it is straightforward to see that this condition holds for the value of N set in step 1 when $M > 2^{(c+1)/2}(\sqrt{n+1}\, n^{n/2}B^n)^{1+1/c}$. □

The running time of Algorithm 6 is simply that of $\mathsf{VecRecon}$, so we have proved the following theorem.

Theorem 5. *Algorithm 6 works correctly as stated. If M satisfies*

$$M > 2^{(c+1)/2}\left(\sqrt{n+1}\, n^{n/2}B^n\right)^{1+1/c}$$

then the algorithm will not return fail. The cost of the algorithm is

$$O(nc^3(\log M)^3) = O(nc^3(c + n\log(nB))^3)$$

bit operations.

Note that we require $M > 2n^nB^{2n}$ to guarantee Algorithm 5 does not return FAIL, so Algorithm 6 allows a smaller value of M to work when n and B are large but c is small. The following table compares the minimum values of $\log M$ which work for Algorithm 5 and Algorithm 6 for various c values.

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	1060.36	799.76	711.36	667.34	641.06
400	1	2397.28	1802.97	1603.11	1503.35	1443.63
800	1	5348.38	4016.82	3570.97	3348.22	3214.70
1600	1	11805.11	8859.88	7875.91	7384.10	7089.16
Alg. 6/Alg. 5			$\approx 75\%$	$\approx 67\%$	$\approx 63\%$	$\approx 60\%$

Algorithm 5 requires that $\log M$ be approximately equal to twice $\log\left(\left\| \left[\, \operatorname{denom}(\boldsymbol{x}) \mid \operatorname{numer}(\boldsymbol{x}) \,\right] \right\|\right)$ in order to succeed. Algorithm 6 requires that $\log M$ be only about $1 + \frac{1}{c}$ times $\log\left(\left\| \left[\, \operatorname{denom}(\boldsymbol{x}) \mid \operatorname{numer}(\boldsymbol{x}) \,\right] \right\|\right)$ in order to succeed.

References

[1] S. Cabay. Exact Solution of Linear Equations. *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation*, 392–398. 1971.

[2] Z. Chen and A. Storjohann. A BLAS Based C Library for Exact Linear Algebra on Integer Matrices. *Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation*, 92–99. 2005.

[3] J. D. Dixon. Exact Solution of Linear Equations Using P-Adic Expansions. *Numerische Mathematik*, 40:137–141. 1982.

[4] 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. *Mathematical Software: Proceedings of the First International Congress of Mathematical Software*, 40–50. 2002.

[5] P. Giorgi. *Arithmétique et algorithmique en algèbre linéaire exacte pour la bibliothéque LinBox*, PhD Thesis. 2004.

[6] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:513–534. 1982.

[7] M. Monagan. Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, 243–249. 2004.

[8] P. Nguyen and D. Stehlé. An LLL Algorithm with Quadratic Complexity. *SIAM Journal on Computing*, 39:874–903. 2009.

[9] Z. Olesh and A. Storjohann. The Vector Rational Function Reconstruction Problem. *Proceedings of the Waterloo Workshop on Computer Algebra 2006*, 137–149. 2007.

[10] V. Shoup. *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press. 2008.

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

Algorithm 1 The $\mathsf{LLL}(k, \boldsymbol{L})$ lattice basis reduction algorithm.

Input: A lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times k'}$.
Output: Update $\boldsymbol{L}$ to be an LLL-reduced basis of the input lattice.

```
1. [Compute Gram-Schmidt orthogonalization]
   $(\boldsymbol{\mu}, \boldsymbol{d}) := \mathsf{GSO}(\boldsymbol{L})$;

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

Algorithm 2 The $\mathsf{InPlaceLLL}(k, \boldsymbol{\mu}, \boldsymbol{d}, \boldsymbol{L})$ lattice basis reduction algorithm.

Input: The GSO $(\boldsymbol{\mu}, \boldsymbol{d}) \in (\mathbb{Q}^{k\times k}, \mathbb{Z}^{k+1})$ of a lattice basis $\boldsymbol{L} \in \mathbb{Z}^{k\times k'}$.
Output: Update $\boldsymbol{L}$ to be a basis of the input lattice, with LLL-reduced GSO $(\boldsymbol{\mu}, \boldsymbol{d})$.

```
1. [Continually increment $i$ by 1 until $i > k$]
   **for** $i := 2$ to $k$ **do**

      2. [Size reduce vector $i$ against vectors $1, 2, \ldots, i-1$]
         **for** $j := i-1$ to $1$ **do**

            3. [Size reduce vector $i$ against vector $j$]
               $r := \lceil \mu_{i,j} - \frac{1}{2} \rceil$;
               $\boldsymbol{L}_i := \boldsymbol{L}_i - r\boldsymbol{L}_j$;
               $\boldsymbol{\mu}_i := \boldsymbol{\mu}_i - r\boldsymbol{\mu}_j$;

      4. [Check Lovász condition]
         **if** $d_i d_{i-2} < \left(\frac{3}{4} - \mu_{i,i-1}^2\right) d_{i-1}^2$ **then**

            5. [Swap vectors $i-1$ and $i$]
               $m := \mu_{i,i-1}$;
               $d'_{i-1} := d_i d_{i-2}/d_{i-1} + m^2 d_{i-1}$;
               swap$(\boldsymbol{L}_{i-1}, \boldsymbol{L}_i)$;
               swap$(\boldsymbol{\mu}_{i-1}, \boldsymbol{\mu}_i)$;
               swap$(\boldsymbol{\mu}_{i-1}^{\mathrm{T}}, \boldsymbol{\mu}_i^{\mathrm{T}})$;
               $\boldsymbol{\mu}_i^{\mathrm{T}} := \boldsymbol{\mu}_i^{\mathrm{T}} - m\boldsymbol{\mu}_{i-1}^{\mathrm{T}}$;
               $\boldsymbol{\mu}_{i-1}^{\mathrm{T}} := \boldsymbol{\mu}_{i-1}^{\mathrm{T}} + m(d_{i-1}/d'_{i-1})\boldsymbol{\mu}_i^{\mathrm{T}}$;
               $d_{i-1} := d'_{i-1}$;
               $i := \max(i-2, 1)$;

   **assert** $\boldsymbol{L}_{1..i}$ has LLL-reduced GSO $(\boldsymbol{\mu}_{1..i,1..i}, \boldsymbol{d}_{0..i})$
```

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 $\boldsymbol{S} := \boldsymbol{L}$;
```

Algorithm 4 A refined $\mathsf{VecRecon}(n, \boldsymbol{a}, M, N, c)$ 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 $\tilde{\boldsymbol{L}} \in \mathbb{Z}^{k\times 1}$, $\tilde{\boldsymbol{d}} \in \mathbb{Z}^{k+1}$ and $\boldsymbol{\mu} \in \mathbb{Q}^{k\times k}$ throughout.

1. [Initialization]
   $k := 1$; $\tilde{L}_1 := 1$; $\tilde{d}_0 := M^2$; $\tilde{d}_1 := 1$; $\mu_{1,1} := 1$;

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

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

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

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

   assert A. $(\boldsymbol{\mu}, \boldsymbol{d})$ is LLL-reduced, where $\boldsymbol{d} := \mathrm{diag}_{0\le i\le k}(M^{2(i-1)})\tilde{\boldsymbol{d}}$
          B. $\left[\,\tilde{\boldsymbol{L}} \,\middle|\, \mathrm{rem}_M(\tilde{\boldsymbol{L}}\boldsymbol{a}_{1..\ell})\,\right]$ is a generating matrix of $\mathcal{L}_N(\boldsymbol{\Lambda}_{\boldsymbol{a}_{1..\ell}}^M)$ with GSO $(\boldsymbol{\mu}, \boldsymbol{d})$
          C. $k \le c$

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

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]
   $N \coloneqq \lfloor \min(\sqrt{M/2}, M/(2nB)) \rfloor$;
2. [Simultaneous rational reconstruction]
   $d \coloneqq 1$;
   for $i \coloneqq 1$ to $n$ do
      3. [Entrywise rational reconstruction]
         $d \coloneqq d \cdot \mathsf{RatRecon}(\mathrm{rem}_M(da_i), M, N, \lfloor N/d \rfloor)$;
         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 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]
      $N \coloneqq \lfloor \min(M^{c/(c+1)}/2^{c/2}, M/(2^{(c+1)/2} nB)) \rfloor$;
   2. [Vector rational reconstruction]
      $\boldsymbol{S} \coloneqq \mathsf{VecRecon}(n, \boldsymbol{a}, M, N, c) \in \mathbb{Z}^{k \times (n+1)}$;
      If $k = 0$ then return FAIL.
assert $k = 1$
   3. [Return solution]
      return $\boldsymbol{S}_{2..n+1}/S_1$;
```