A Modification of the LLL Reduction Algorithm

M. POHST

Mathematisches Institut, Universität Düsseldorf, 4000 Düsseldorf, FRG

The reduction algorithm of Lenstra et al. (1982) is modified in a way that the input vectors can be linearly dependent. The output consists of a basis of the lattice generated by the input vectors as well as non-trivial linear combinations of O by the input vectors if those are linearly dependent.

One of the disadvantages of the original LLL algorithm (Lenstra et al., 1982) is that the input must consist of linearly independent vectors of a lattice. Hence that algorithm cannot be applied directly to such tasks as determining a lattice basis from a set of generating vectors. But such tasks frequently occur in practice, for example, in connection with the computation of fundamental units of algebraic number fields. In this paper we show how a slight modification of the original algorithm suffices to make it applicable also to vectors of a lattice which are not necessarily linearly independent.

First steps in that direction were already done in Lenstra et al. (1982) and by Odlyzko (1984), Buchmann & Pethö (1987) and others. Those authors, however, did not alter the algorithm itself but rather applied it to modified lattices. We outline their approach since it helps the understanding of how the new algorithm operates.

In the following, \(\Lambda\) denotes a \(k\)-dimensional lattice in the Euclidean space \(\mathbb{R}^n = \mathbb{R}^{n \times 1}\). We consider the problem how to compute a basis of \(\Lambda_g := \mathbb{Z}a_1 + \ldots + \mathbb{Z}a_g\) from given vectors \(a_1, \ldots, a_g\) \((g \in \mathbb{Z}^{>0})\) of \(\Lambda\). It can be solved by an application of the LLL algorithm to a suitable lattice \(\hat{\Lambda} \subset \mathbb{R}^{g+n}\) as follows. Let

\[
\hat{b}_i = (\hat{e}_i, 2^i a_i)^T \quad (1 \leq i \leq g),
\]

where \(\hat{e}_i\) denotes the \(i\)th unit vector of \(\mathbb{R}^g\) and \(\lambda \in \mathbb{Z}^{>0}\) is a constant to be specified later. Clearly, the \(\hat{b}_i\) are linearly independent, and we set

\[
\hat{\Lambda} := \bigoplus_{i=1}^g \mathbb{Z} \hat{b}_i.
\]

An application of the LLL algorithm to the \(\hat{b}_i\) yields a basis \(\hat{e}_1, \ldots, \hat{e}_g\) of \(\hat{\Lambda}\) such that \(\|\hat{e}_i\| \leq 2^{g-1} M\) for

\[
M := \min \{\|\hat{x}\| \mid \hat{x} \in \hat{\Lambda}, \hat{x} \neq \hat{0}\}.
\]

If there is a non-trivial linear combination

\[
\sum_{i=1}^g m_i a_i = 0 \quad (m_i \in \mathbb{Z})
\]

and \(\lambda\) is large enough, then \(\hat{e}_i\) will have zeros in its last \(n\) coordinates \(\hat{e}_{1, \ldots, 1, g+n}\). Namely, let \(|m_i| \leq B\) for a (minimal) potential linear combination

\[
\sum_{i=1}^g m_i a_i = 0
\]
and \[ 0 < M_1 \leq \min \{ \|x\|^2 \mid x \in \Lambda_g, x \neq 0 \}. \]

Since \( \hat{c}_1 \) is of the form \( \hat{c}_1 = (\hat{m}', 2^g a') \) with \( \hat{m} \in \mathbb{Z}^g, a \in \Lambda_g \) we obtain \( a = 0 \) if we choose \( \lambda \) such that \( 2^{g-1}(gB^2) < 2^{2\lambda} M_1 \). Though this method works well in some applications it has several disadvantages.

(i) It is not always easy to obtain a reasonable upper bound \( B \) for the coefficients \( n_i \) of a potential non-trivial linear combination

\[ \sum_{i=1}^{g} n_i a_i = 0. \]

(ii) A similar objection holds for the determination of a lower bound \( M_1 \) for the first successive minimum of \( \Lambda_g \).

(iii) Even if \( B \) and \( M_1 \) are known the constant \( \lambda \) must usually be chosen so large that the necessary computations of the LLL algorithm have to be carried out by multi-precision arithmetics.

Therefore we choose a somewhat different approach. It is based on the observation that the choice of a huge constant \( \lambda \) is likely to speed up the computation of vectors \( \hat{c} \) by the LLL algorithm for which the last \( n \) coordinates are zero. If \( \lambda \) is chosen large enough the performance of the LLL algorithm will only hinge on the last \( n \) coordinates of all vectors involved. Since they have a weight \( 2^\lambda \) attached to them we can carry out the computations as well without that weight. Thus we obtain an algorithm operating on the vectors \( a_1, \ldots, a_g \) alone. Since they will be linearly dependent in general, we need to change the original algorithm in case a linear dependency occurs. If we keep in mind that the new algorithm is obtained from the original one for \( \lambda \rightarrow \infty \), the necessary changes become quite obvious, however.

We roughly recall how the original LLL algorithm operates on \( g \) linearly independent vectors \( b_1, \ldots, b_g \). In the first step the Gram–Schmidt orthogonalisation procedure is used to compute

\[ b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \quad \mu_{ij} = \frac{b_i^* b_j^*}{b_j^* b_j^*} \quad (1 \leq j \leq i-1, 1 \leq i \leq g). \]

At each level \( m \) (\( 2 \leq m \leq g \)) the vector \( b_m \) is reduced modulo \( b_j \) to obtain

\[ |\mu_{mj}| \leq \frac{1}{2} (j = m - 1, m - 2, \ldots, 1), \]

and in case

\[ \|b_m^* + \mu_{m,m-1} b_{m-1}^*\|^2 < \frac{3}{4} \|b_{m-1}^*\|^2 \]

the vectors \( b_m \) and \( b_{m-1} \) are interchanged (and \( m \) is decreased by 1 for \( m > 2 \)).

Considering an application to arbitrary vectors \( a_1, \ldots, a_g \) of \( \Lambda \) we note that \( a_i^* \) becomes 0 in case there is a non-trivial linear combination

\[ \sum_{i=1}^{l} n_i a_i = 0 \]

such that \( n_l \neq 0 \). The initial computations therefore remain valid for \( j \leq i-1 < l \) if \( a_1, \ldots, a_{l-1} \) are linearly independent. The application of the LLL algorithm to \( a_1, \ldots, a_l \) then provides a solution \( m_1, \ldots, m_l \) if we carry out the exchange steps \( a_i \leftrightarrow a_{i-1} \) in an appropriate way in case \( a_i^* = 0 \). A desired non-trivial linear combination

\[ \sum_{i=1}^{l} n_i a_i = 0 \]

is obtained by finally reducing \( b_l \) modulo \( b_{l-1} \). We state the modified algorithm and add
some explanatory remarks. The notations are closely related to those of the original reduction algorithm of Lenstra et al. (1982).

Modified Reduction Algorithm MLLL

INPUT. Non-zero vectors $\mathbf{a}_1, \ldots, \mathbf{a}_g$ of a lattice $\Lambda \subset \mathbb{R}^n$.

OUTPUT. Linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_\rho$ subject to

$$\sum_{i=1}^g Z \mathbf{a}_i = \bigoplus_{j=1}^{\rho} Z \mathbf{b}_j$$

and $g - \rho$ relation vectors $\mathbf{m}_1, \ldots, \mathbf{m}_{g - \rho} \in \mathbb{Z}^g$ such that

$$\sum_{i=1}^g m_{\sigma i} \mathbf{a}_i = \mathbf{0} \quad \text{for} \quad \mathbf{m}_\sigma = (m_{\sigma 1}, \ldots, m_{\sigma g})^t \quad (1 \leq \sigma \leq g - \rho).$$

STEP 1. (Initialisation) Set $\alpha \leftarrow 0$, $\beta \leftarrow 0$, $\sigma \leftarrow 0$, $\rho \leftarrow 0$, $\tau \leftarrow 2$.

STEP 2. (Orthogonalisation) Set

$$\alpha \leftarrow \alpha + 1, \quad \beta \leftarrow \beta + 1, \quad \mathbf{h}_\rho \leftarrow \mathbf{e}_2, \quad \mathbf{b}_\beta \leftarrow \mathbf{a}_x,$$

$$\mu_{\beta j} \leftarrow \frac{\mathbf{b}_{\beta j} \mathbf{b}_j^*/B_j}{(1 \leq j \leq \beta - 1)}, \quad \mathbf{b}_\beta^* \leftarrow \mathbf{b}_\beta - \sum_{j=1}^{\beta-1} \mu_{\beta j} \mathbf{b}_j^*, \quad B_\beta \leftarrow \mathbf{b}_\beta^* \mathbf{b}_\beta^*.$$

If $B_\beta \neq 0$ and $\alpha < g$ return to the beginning of Step. 2. Otherwise set $m \leftarrow \tau$, $s \leftarrow \beta$ for $B_\beta \neq 0$, $\beta - 1$ else.

STEP 3. (Set l) Set $l \leftarrow m - 1$.

STEP 4. (Reduce $\mu_{ml}$) For $|\mu_{ml}| > \frac{1}{2}$ set

$$t \leftarrow \{\mu_{ml}\}^t, \quad \mathbf{b}_m \leftarrow \mathbf{b}_m - t \mathbf{b}_l, \quad \mathbf{h}_m \leftarrow \mathbf{h}_m - t \mathbf{h}_l,$$

$$\mu_{ml} \leftarrow -\mu_{ml}, \quad \mu_{mj} \leftarrow -\mu_{mj} - t \mu_{lj} \quad (1 \leq j \leq l - 1).$$

In case $\mathbf{b}_m = \mathbf{0}$ go to 9. For $\mathbf{b}_m \neq \mathbf{0}$ and $l < m - 1$ go to 6.

STEP 5. (LLL condition violated on level $m$?) For

$$B_m < \frac{(3}{4} - \mu_{m,m-1}^2)B_{m-1}$$

go to 7.

STEP 6. (Decrease l) Set $l \leftarrow l - 1$. For $l > 0$ go to 4. Else set $m \leftarrow m + 1$. For $m > \beta$ terminate, otherwise go to 3.

STEP 7. ($B_m = \mu = 0$?) Set $\mu \leftarrow \mu_{m,m-1}$, $B \leftarrow B_m + \mu^2 B_{m-1}$. For $B = 0$ go to 8. Else set

$$\mu_{m,m-1} \leftarrow \mu(B_{m-1}/B), \quad B_m \leftarrow B_m(B_{m-1}/B),$$

$$\left(\begin{array}{c} \mu_{i,m-1} \\ \mu_{i,m} \end{array}\right) \left(\begin{array}{cc} 1 & \mu_{m,m-1} \\ 0 & 1 \end{array}\right) \left(\begin{array}{cc} 1 & 0 \\ 1 & -\mu \end{array}\right) \left(\begin{array}{c} \mu_{i,m-1} \\ \mu_{i,m} \end{array}\right) \quad (m + 1 \leq i \leq \beta).$$

STEP 8. (Interchange $\mathbf{b}_{m-1}, \mathbf{b}_m$) Set $B_{m-1} \leftarrow B$,

$$\left(\begin{array}{c} \mathbf{h}_{m-1} \\ \mathbf{h}_m \end{array}\right) \left(\begin{array}{c} \mathbf{b}_{m-1} \\ \mathbf{b}_m \end{array}\right) \left(\begin{array}{cc} \mu_{m-1,j} & \mu_{m,j} \\ \mu_{m-1,j} & \mu_{m-1,j} \end{array}\right) \quad (1 \leq j \leq m - 2).$$

For $m > 2$ decrease $m$ by 1. Then go to 3.

$\dagger \{x\}$ denotes the nearest integer to the real number $x$. 
STEP 9. (Relation vector $\mathbf{m}_j$ found) Set $\sigma \leftarrow \sigma + 1$, $\beta \leftarrow \beta - 1$, $\mathbf{m}_\sigma \leftarrow \mathbf{h}_m$; set

$$
\mathbf{b}_i \leftarrow \mathbf{b}_{i+1}, \quad \mathbf{h}_i \leftarrow \mathbf{h}_{i+1} \quad (m \leq i \leq \beta).
$$

For $\alpha \geq g$ terminate. Else update

$$
\mathbf{b}_i^*, \mu_{ij}, B_i \quad (m \leq i \leq \beta; 1 \leq j < i), \quad \text{set } \tau \leftarrow m + 1
$$

and go to 2.

REMARKS. (i) $\alpha$ denotes the number of input vectors $\mathbf{a}_i$ which have been tested already; $\beta$ denotes the number of vectors $\mathbf{b}_i$ which are momentarily tested for linear independence; $\sigma$ is the number of relation vectors found by the algorithm; $\rho$ is the (non-decreasing) number of linearly independent vectors already determined.

(ii) The output consists of $\rho$ linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_\rho$ subject to

$$
\mathbf{b}_i = \sum_{j=1}^{g} h_{ij} \mathbf{a}_j \quad (1 \leq i \leq \rho; \mathbf{h}_i = (h_{i1}, \ldots, h_{ig})^t \in \mathbb{Z}^g)
$$

forming a basis of $\Lambda_\rho = \mathbb{Z} \mathbf{a}_1 + \ldots + \mathbb{Z} \mathbf{a}_\rho$ and $0 \leq g - \rho$ relation vectors $\mathbf{m}_\sigma$. If $\mathbf{a}_1, \ldots, \mathbf{a}_\rho$ are linearly independent MLLL performs exactly as the original LLL algorithm.

(iii) If $\mathbf{a}_1, \ldots, \mathbf{a}_\rho$ are linearly dependent the algorithm operates on linearly dependent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_\rho$, hence $B_\rho = 0$. In Step 7 the possibility $B_m = 0$, $\mu \neq 0$ can occur only a finite number of times since in that case $B_{m-1}$ is multiplied by a factor of $1/4$, which is even better than the factor $3/4$ occurring in the original algorithm. The possibility $B_m = \mu = 0$, however, implies that after Steps 7, 8 have been carried out we even have a linear dependency between $\mathbf{b}_1, \ldots, \mathbf{b}_{m-1}$ (where we had one between $\mathbf{b}_1, \ldots, \mathbf{b}_m$ before). In this way we obtain a linear dependency among fewer and fewer vectors; finally, $\mathbf{b}_m$ must become zero in Step 4.

(iv) Each time a relation $\mathbf{m}_\sigma$ between vectors $\mathbf{b}_1, \ldots, \mathbf{b}_\beta$ is found it is removed from the data and we extend the remaining vectors to a new system $\mathbf{b}_1, \ldots, \mathbf{b}_\rho$ from the $\mathbf{a}_i$. We note that $\mathbf{b}_1, \ldots, \mathbf{b}_{\rho-1}$ are always linearly independent after Step 2. If we especially search for a short relation vector $\mathbf{m}$, it can be obtained by applying the original LLL algorithm to the (linearly independent) output vectors $\mathbf{m}_1, \ldots, \mathbf{m}_{\rho-\rho}$.

(v) If the lattice under consideration is not integral, roundoff errors can cause a problem in the decision whether $\mathbf{b}_\rho^*$ is actually zero. Let

$$
\Lambda_\beta = \sum_{j=1}^{\rho} \mathbb{Z} \mathbf{b}_j \subseteq \Lambda.
$$

In case of $\mathbf{b}_\rho^* \neq 0$ we obtain for its discriminant

$$
d(\Lambda_\beta) = \prod_{j=1}^{\rho} ||\mathbf{b}_j^*||
$$

and for the shortest vector of $\Lambda_\beta$, say $\mathbf{y} \neq \mathbf{0}$, the estimate

$$
||\mathbf{y}||^2 \leq (\gamma_\beta^\alpha d(\Lambda_\beta)^2)^{1/\beta},
$$

where $\gamma_\beta^\alpha$ denotes Hermite's constant. If we know a lower bound $M_1$ for the minimum of any lattice containing $\Lambda_\beta$, then we find

$$
M_1^\beta \leq \gamma_\beta^\alpha d(\Lambda_\beta)^2 = \gamma_\beta^\alpha \prod_{j=1}^{\rho} ||\mathbf{b}_j^*||^2
$$
so that $\|b^*_\beta\|$ cannot be too small. For example, if we know a basis, say $c_1, \ldots, c_r$, of some lattice $\Lambda$ containing $\Lambda_\beta$, then we can obtain a lower bound $M_1$ as follows. Let $\|c_1\| \leq \ldots \leq \|c_r\|$. From Hadamard's inequality we conclude

$$d(\Lambda') \leq \sqrt{M_1} \prod_{j=2}^r \|c_j\|,$$

hence,

$$M_1 \geq \left( \frac{d(\Lambda')}{\prod_{j=2}^r \|c_j\|} \right)^2.$$

This method is very important for unit computations in algebraic number fields.

The following simple example is to illustrate the mode of operation of the algorithm.

**Example.** Let $g = 4$ and

$$a_1 = (1, 1, 0)', \quad a_2 = (1, 0, 1)', \quad a_3 = (1, 3, -2)', \quad a_4 = (2, 3, -1)'.$$

After the initialisation we obtain

$$\alpha = \beta = 3, \quad b_i = a_i \quad (i = 1, 2, 3), \quad m = 2$$

in Step 2. Since $a_1, a_2$ are already LLL reduced the algorithm proceeds to Step 6 where $m$ is increased by 1. Now we obtain $b_3 = (3, 3, 0)'$ and $h_3 = (0, 2, 1, 0)'$ in Step 4. Via Steps 5, 7, 8 (where $b_2$ and $b_3$ are interchanged) and Step 3 we get to Step 4 and obtain $b_2 = 0$ as well as the relation vector $m_1 = (-3, 2, 1, 0)'$ in Step 9 afterwards. Next we get to Step 2 again and obtain the updated values $\alpha = 4, \beta = 3, \ b_1 = a_1, \ b_2 = a_2, \ b_3 = a_4, \ m = 2.$

Similarly, as before the second relation vector $m_2 = (-3, 1, 0, 1)'$ is determined.

A detailed discussion of the MLLL algorithm for non-integral lattices in connection with unit computations will appear as a joint paper by J. Buchmann and the author in the near future. We finally note that the complexity analysis of the original LLL algorithm can be almost literally transferred to the new algorithm.

After the author had finished this paper he learnt from J. Buchmann that there is a preprint by Hastad et al. (1986) in which an algorithm similar to MLLL is presented. Since the essential transformations (see Steps 7, 8 of MLLL) are not carried out in detail there it was impossible to decide how related both algorithms are. In any case, there is no factor of 1/4 speeding up the algorithm and their approach is totally different.

**References**


