A Delayed Size-reduction Technique for Speeding Up the LLL Algorithm [*]

Wen Zhang,
Institute of Mathematics, School of Mathematical Science
Fudan University, Shanghai, 200433 P.R. China
Yimin Wei
School of Mathematical Sciences and
Shanghai Key Laboratory of Contemporary Applied Mathematics
Fudan University, Shanghai, 200433, P.R. China
Sanzheng Qiao,
Department of Computing and Software, McMaster University
Hamilton, Ontario L8S4K1 Canada

Abstract

In this paper, we present a delayed size-reduction technique for speeding up the LLL algorithm. It can significantly speed up the LLL algorithm without sacrificing the quality of the results. Our experiments have shown that for problems of size 80, our algorithm can be twice as fast as the LLL algorithm. For larger size problems, the speed up is greater. Moreover, our algorithm provides a starting point for parallel LLL algorithms.

Keywords Lattice, lattice basis reduction, LLL algorithm.

1 Introduction

The LLL algorithm authored by Lenstra, Lenstra, and Lovász [11] is a lattice basis reduction method. The problem of reducing a lattice basis has wide applications: factorization of polynomials with rational coefficients [11], cryptography [3], integer programming [12], number theory [12], and digital communications [1, 9], just to name a few. Although it does not guarantee a best reduced lattice basis, the LLL algorithm can efficiently produce reasonably good results. Thus the LLL algorithm is widely used in wireless communications to improve the performance. For example, it is used in lattice-reduction-aided MIMO detection [4, 5, 6, 16, 18], where lattice basis reduction is used as a preprocessing. In time critical situations such as delay-constrained MIMO detection [8], speed is crucial. Thus the LLL algorithm has been modified to improve its

[*]Wen Zhang is partially supported by the National Natural Science Foundation of China under grant 10871051, Doctoral Program of the Ministry of Education under grant 20090071110003 and KLMM0901. Sanzheng Qiao is partially supported by Key Laboratory of Mathematics for Nonlinear Sciences (Fudan University) and NSERC of Canada.

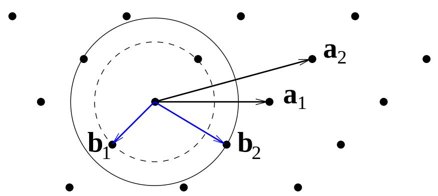

Figure 1: The columns of A (1) and B (2), two bases for the lattice L.

speed without sacrificing the quality of the reduced basis produced by the algorithm [7, 8, 17]. In this paper, we propose a technique, called delayed size-reduction, to speed up the LLL algorithm. The modified algorithm can significantly speed up the LLL algorithm without sacrificing the quality of the results. Our experiment results have shown the for problems of size 80, our algorithm can be twice as fast as the LLL algorithm. For larger size problems, the speed up is greater.

This paper is organized as follows. In Section 2, we introduce the concepts of lattice bases and reduced bases. Then we briefly describe the LLL algorithm in Section 3. Our delayed size-reduction technique and the modified algorithm is presented in Section 4. A complexity analysis of the modified algorithm is provided in Section 5. Finally, in Section 6, we show our experiment results.

2 Lattice Bases

Let A be an m-by-n, $m \geq n$, real matrix of full column rank. Thus the columns of A are linearly independent. Each integer linear combination of the columns of A represents a lattice point, or lattice vector. In other words, a lattice generated by A is defined by the set:

$$L = \{Az \quad | \quad z \in \mathbb{Z}^n\},$$

where $\mathbb{Z}^n$ denotes the set of integer n-vectors. The columns of A form a basis for the lattice L. For example, the points in Figure 1 form a lattice generated by

$$A := [a_1 \;\; a_2] = \begin{bmatrix} 2.0 & 2.7 \\ 0 & 0.7 \end{bmatrix}. \tag{1}$$

A lattice can have more than one basis. For example, the columns of

$$B := [b_1 \;\; b_2] = \begin{bmatrix} -0.7 & 1.3 \\ -0.7 & -0.7 \end{bmatrix} \tag{2}$$

also form a basis for the lattice L generated by A in (1), as shown in Figure 1. In this example, the basis $\{b_1, \; b_2\}$ (2) is more desirable than $\{a_1, \; a_2\}$ (1), since it consists of shorter and "more orthogonal" vectors than the basis $\{a_1, \; a_2\}$. We say that $\{b_1, \; b_2\}$ is a reduced basis for L.

Given a basis for a lattice, a lattice reduction method produces a reduced basis for the lattice. For details of lattices, bases, and reduced bases, see [2, 10].

3 The LLL Algorithm

The LLL algorithm is a lattice basis reduction method. It consists of two stages. Given an m-by-n lattice generator matrix A, the first stage triangularizes it using the Gram-Schmidt process. In terms of matrices, it computes the D and U matrices in the decomposition

$$A = QD^{1/2}U, \tag{3}$$

where Q is m-by-n with orthonormal columns, $D = \mathrm{diag}(d_1, d_2, ..., d_n)$ with positive diagonal, and $U = [u_{i,j}]$ upper triangular with a unit diagonal, that is $u_{i,i} = 1, i = 1, 2, \cdots, n$. In [11], a basis formed by the columns of A is called reduced if D and U in the decomposition (3) satisfy the conditions

$$|u_{i,j}| \leq \frac{1}{2}, \quad 1 \leq i < j \leq n, \tag{4}$$

and

$$d_i + d_{i-1}u_{i-1,i}^2 \geq \omega d_{i-1}, \tag{5}$$

where $1/4 < \omega < 1$ is a parameter that controls the rate of convergence of the algorithm. The condition (4) requires small off-diagonal elements, thus short column lengths. This condition is often called size-reduced condition [15]. The second condition (5) imposes a loosely increasing order on the diagonal elements d_i. By pushing up small diagonal elements and enforcing the size-reduced condition (4), the column lengths are expected to be shortened.

When A is an integer or rational matrix, the LLL algorithm can be performed in exact arithmetic in polynomial time [11]. In 2008, Luk and Tracy [14] presented a floating-point version of the algorithm for real lattice basis matrices. In this paper, we consider the real case and adopt the floating-point version in [14]. However, the technique proposed in this paper can be readily applied to the integer case. Corresponding to the conditions (4) and (5), a basis formed by the columns of a real lattice basis matrix A is called reduced if the upper triangular factor R in the QR decomposition

$$A = QR \tag{6}$$

of A, where Q has orthonormal columns and R is upper triangular, satisfies the conditions

$$2|r_{i,j}| \leq |r_{i,i}|, \quad 1 \leq i < j \leq n, \tag{7}$$

and

$$r_{i,i}^2 + r_{i-1,i}^2 \geq \omega r_{i-1,i-1}^2. \tag{8}$$

The QR decomposition (6) can be obtained in a number of ways [9]. If the Gram-Schmidt orthogonalization is used, then $D^{1/2}U$ in (3) equals R in (6).

Given a lattice basis matrix, the LLL algorithm reduces it to another lattice basis matrix that satisfies the conditions (7) and (8). To enforce the condition (7), if some $|r_{i,j}| > |r_{i,i}|/2$, the following size-reduction procedure is applied.

Procedure 1 (`Reduce`(i, j)) *Given R in (6), calculate $\gamma = \lceil r_{i,j}/r_{i,i} \rfloor$ ($\lceil a \rfloor$ denotes an integer that is closest to a), form $Z_{ij} = I_n - \gamma \mathbf{e}_i \mathbf{e}_j^{\mathrm{T}}$, where $\mathbf{e}_i$ is the i-th unit vector, and apply Z_{ij} to both R and A:*

$$R \leftarrow RZ_{ij} \quad \textit{and} \quad A \leftarrow AZ_{ij}.$$

Thus if $|r_{i,j}| > |r_{i,i}|/2$ in the current R, then in the updated R, we have $|r_{i,j}| \leq |r_{i,i}|/2$.

When the condition (8) is not satisfied, the following procedure is invoked.

Procedure 2 (`SwapRestore`(i)) *Given R in (6), where $2|r_{i-1,i}| \leq |r_{i-1,i-1}|$, swap the columns $i-1$ and i of R and A, find a Givens reflection G_i such that*

$$G_i^T \begin{bmatrix} r_{i-1,i} & r_{i-1,i-1} \\ r_{i,i} & 0 \end{bmatrix} = \begin{bmatrix} \hat{r}_{i-1,i-1} & \hat{r}_{i-1,i} \\ 0 & \hat{r}_{i,i} \end{bmatrix},$$

and apply G_i to R:

$$R \leftarrow \begin{bmatrix} I_{i-2} & & \\ & G_i & \\ & & I_{n-i} \end{bmatrix}^T R.$$

Thus the above procedure swaps the columns i and $i-1$ and then restores the upper triangular structure of R. It can be verified that if the current R does not satisfy the condition (8), then after the application of Procedure 2, the updated R satisfies the condition.

Now, we present the floating-point LLL algorithm.

Algorithm 1 (LLL) *Given an m-by-n, $m \geq n$, lattice basis matrix A, this algorithm overwrites it with a reduced one.*

```
QRD   Compute the $R$ in the decomposition (6);
      $k \leftarrow 2$;
      while $k \leq n$
C1a       if $2|r_{k-1,k}| > |r_{k-1,k-1}|$
R1            Reduce$(k-1, k)$;
          endif
C2        if $r_{k,k}^2 + r_{k-1,k}^2 < \omega r_{k-1,k-1}^2$
SR            SwapRestore$(k)$;
              $k \leftarrow \max(k-1, 2)$;
          else
              for $i = k-2$ down to 1
C1                if $2|r_{i,k}| > r_{i,i}|$
R2                    Reduce$(i, k)$;
                  endif
              endfor
              $k \leftarrow k+1$;
          endif
      endwhile
```

Note that by the definition in [11], a reduced basis satisfies both conditions (7) and (8), thus in the algorithm, before checking the condition (8) for $i = k$, the condition (7) for $i = k-1$ and $j = k$ is enforced.

4 Delaying Size-reductions

In this section, we present our new algorithm. First, we give a simple example to illustrate our ideas by tracing Algorithm 1. Let

$$A = \begin{bmatrix} 4 & 6 & 5 \\ & 2 & 10 \\ & & 1/\sqrt{2} \end{bmatrix}$$

be the lattice basis matrix. In Algorithm 1, since A is upper triangular, the QR decomposition in line QRD produces $R = A$. Then k is initialized to 2. Since the condition in line C1a is true, `Reduce`$(k-1, k)$ in line R1 is called, producing

$$R = \begin{bmatrix} 4 & -2 & 5 \\ & 2 & 10 \\ & & 1/\sqrt{2} \end{bmatrix}.$$

Now, the condition C2 is true, `SwapRestore` in line SR updates R

$$R = \begin{bmatrix} 2\sqrt{2} & -2\sqrt{2} & 5/\sqrt{2} \\ & 2\sqrt{2} & 15/\sqrt{2} \\ & & 1/\sqrt{2} \end{bmatrix}.$$

Then k is reset to 2. Again, C1a is true and R1 gives

$$R = \begin{bmatrix} 2\sqrt{2} & 0 & 5/\sqrt{2} \\ & 2\sqrt{2} & 15/\sqrt{2} \\ & & 1/\sqrt{2} \end{bmatrix}.$$

Note that in this step, the $(1, 2)$-entry of U is reduced. Now, both $C2$ and $C1$ are false, so k is incremented by 1 to 3. The condition C1a is true and R1 reduces R:

$$R = \begin{bmatrix} 2\sqrt{2} & 0 & 5/\sqrt{2} \\ & 2\sqrt{2} & -1/\sqrt{2} \\ & & 1/\sqrt{2} \end{bmatrix}.$$

Now, since C2 is true, SR updates R:

$$R = \begin{bmatrix} 2\sqrt{2} & 5/\sqrt{2} & 0 \\ & 1 & -2 \\ & & 2 \end{bmatrix}.$$

Note that in this step, the previously reduced $(1, 2)$-entry, whose value is 0, of R is swapped to $(1, 3)$. Then, k is decremented by 1 to 2. Now, C1a is true and R1 gives

$$R = \begin{bmatrix} 2\sqrt{2} & 1/\sqrt{2} & 0 \\ & 1 & -2 \\ & & 2 \end{bmatrix}.$$

Again, C2 is true. After SR, we have

$$R = \begin{bmatrix} \sqrt{3}/\sqrt{2} & 2\sqrt{2}/\sqrt{3} & -2\sqrt{2}/\sqrt{3} \\ & 4/\sqrt{3} & 2/\sqrt{3} \\ & & 2 \end{bmatrix}.$$

Note that in this step the value of the $(1,3)$-entry, which was moved from the previously reduced $(1,2)$-entry, is modified from 0 to $-2\sqrt{2}/\sqrt{3}$, whose absolute value is larger than $|r_{1,1}|/2 = \sqrt{3}/(2\sqrt{2})$. Now k is decremented by 1 to 2. The condition C1a is true and R1 produces

$$R = \begin{bmatrix} \sqrt{3}/\sqrt{2} & 1/\sqrt{6} & -2\sqrt{2}/\sqrt{3} \\ & 4/\sqrt{3} & 2\sqrt{3} \\ & & 2 \end{bmatrix}.$$

Then both C2 and C1 are false and k is incremented to 3. Finally, both C1a and C2 are false, but C1 is true when $i = 1$ and $k = 3$, thus R2 reduces the $(1,3)$-entry of R:

$$R = \begin{bmatrix} \sqrt{3}/\sqrt{2} & 1/\sqrt{6} & -1/\sqrt{6} \\ & 4/\sqrt{3} & 2/\sqrt{3} \\ & & 2 \end{bmatrix}.$$

Recalling that this $(1,3)$-entry was moved from the previously reduced $(1,2)$-entry. If it was not previously reduced, it would be reduced by now anyway. This means that some size-reductions can be saved by delaying them until a later stage. However, not all the size-reductions can be delayed. In particular, the condition (8) assumes that $r_{i-1,i}$ is size-reduced, that is $2|r_{i-1,i}| \leq |r_{i-1,i-1}|$. Thus we propose the following condition by integrating the two conditions:

$$r_{i,i}^2 + (r_{i-1,i} - \gamma r_{i-1,i-1})^2 \geq \omega r_{i-1,i-1}^2, \quad \text{where} \quad \gamma = \lceil r_{i-1,i}/r_{i-1,i-1} \rfloor. \tag{9}$$

Note that $r_{i-1,i} - \gamma r_{i-1,i-1}$ is the reduced $r_{i-1,i}$. Accordingly, we integrate the procedures `Reduce`$(i-1,i)$ and `SwapRestore`(i) into the following procedure.

Procedure 3 (`ReduceSwapRestore`(i)) *Given R in (6), calculate $\gamma = \lceil r_{i-1,i}/r_{i-1,i-1} \rfloor$, construct*

$$Z_i = \begin{bmatrix} 1 & -\gamma \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -\gamma & 1 \\ 1 & 0 \end{bmatrix}, \tag{10}$$

and update R and A:

$$R \leftarrow R \begin{bmatrix} I_{i-2} & & \\ & Z_i & \\ & & I_{n-i} \end{bmatrix} \quad \textit{and} \quad A \leftarrow A \begin{bmatrix} I_{i-2} & & \\ & Z_i & \\ & & I_{n-i} \end{bmatrix}.$$

Then find a Givens reflection G_i such that

$$G_i^T \begin{bmatrix} r_{i-1,i} - \gamma r_{i-1,i-1} & r_{i-1,i-1} \\ r_{i,i} & 0 \end{bmatrix} = \begin{bmatrix} \hat{r}_{i-1,i-1} & \hat{r}_{i-1,i} \\ 0 & \hat{r}_{i,i} \end{bmatrix}, \tag{11}$$

and apply G_i to R:

$$R \leftarrow \begin{bmatrix} I_{i-2} & & \\ & G_i & \\ & & I_{n-i} \end{bmatrix}^T R.$$

Note that the matrix Z_i is a size-reduction followed by a swap.

We delay all the other size-reductions until the end. Thus we have the following LLL algorithm with delayed size-reduction.

Algorithm 2 (LLL with delayed size-reduction) *Given an m-by-n, $m \geq n$, lattice generator matrix A, this algorithm overwrites it with a reduced one.*

```
QRD   Compute the $R$ in the decomposition (6);
      $k \leftarrow 2$;
      while $k \leq n$
          $\gamma = \lceil r_{k-1,k}/r_{k-1,k-1} \rfloor$;
C3        if $r_{k,k}^2 + (r_{k-1,k} - \gamma r_{k-1,k-1})^2 < \omega r_{k-1,k-1}^2$
RSR           ReduceSwapRestore($k$);
              $k \leftarrow \max(k-1, 2)$;
          else
              $k \leftarrow k+1$
          endif
      endwhile
      for $k = 2$ to $n$
          for $i = k-1$ down to 1
C1            if $2|r_{i,k}| > |r_{i,i}|$
R2                Reduce($i, k$);
              endif
          endfor
      endfor
```

The number of executions of `ReduceSwapRestore` in this algorithm is the same as the number of executions of `SwapRestore` in Algorithm 1, because the `Reduce` in the line R2 of Algorithm 1 has no effect on the two-by-two diagonal blocks, which are operated by `SwapReduce`. This will be verified in our numerical experiments.

The above algorithm can improve the performance of Algorithm 1 in three ways: First, by integrating the two conditions C1a and C2 in Algorithm 1 into one condition C3 in Algorithm 2, the number of conditional statements is reduced; Second, by integrating the two procedures R1 and SR in Algorithm 1 into one procedure RSR in Algorithm 2, it is more efficient; Third, by delaying some size-reductions, the total number of size-reductions can be reduced.

Let us trace Algorithm 2 using the above example. Again, since A is upper triangular, after the QR decomposition, we have $R = A$. Then k is initialized to 2. The condition C3 is true, after RSR, we have

$$R = \begin{bmatrix} 2\sqrt{2} & -2\sqrt{2} & 5/\sqrt{2} \\ & 2\sqrt{2} & 15/\sqrt{2} \\ & & 1/\sqrt{2} \end{bmatrix}.$$

Now that C3 is false for $k = 2$, k is incremented to 3 and C3 is true and RSR gives

$$R = \begin{bmatrix} 2\sqrt{2} & 21\sqrt{2} & -2/\sqrt{2} \\ & 1 & -2 \\ & & 2 \end{bmatrix}.$$

Then k is decremented to 2. Since C3 is true, RSR produces

$$R = \begin{bmatrix} \sqrt{3}/\sqrt{2} & 2\sqrt{2}/\sqrt{3} & -4\sqrt{2}/\sqrt{3} \\ & 4/\sqrt{3} & -2/\sqrt{3} \\ & & 2 \end{bmatrix}.$$

Now, all the diagonal 2-by-2 blocks satisfy the condition (9), we start the delayed size-reductions. Specifically, the last part of the algorithm reduces the $(1,2)$- and $(1,3)$-entries of R, resulting a reduced R:

$$R = \begin{bmatrix} \sqrt{3}/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{6} \\ & 4/\sqrt{3} & -2/\sqrt{3} \\ & & 2 \end{bmatrix}.$$

To compare the two algorithms, the number of calls to RSR in Algorithm 2 is three and the number of calls to R2 is two. If we count one call to RSR in Algorithm 2 as one R1 and one SR in Algorithm 1, then it is equivalent to three calls to SR, three calls to R1, and two calls to R2. The trace of Algorithm 1 shows that Algorithm 1 makes three calls to SR, four calls to R1, and two calls to R2. Moreover, Algorithm 2 checks the condition C3 three times and the condition C1 six times, a total of nine times. Whereas, Algorithm 1 checks C1a six times, C2 six times, and C1 one time, total of 13 times. Also note that the final reduced R produced by both algorithms are essentially the same up to some signs.

5 Complexity

In this section, we analyze the complexity of Algorithm 2. As we know, the complexity of the QR decomposition in the first stage is $O(mn^2)$ [9]. The rest of the algorithm consists of two parts: A while-loop and a nested for-loop. In the second part, each `Reduce`(i,k) in the nested for-loop costs $O(n)$. Thus the complexity of the second part is $O(n^3)$. In the first part, in each iteration of the while-loop, k is either incremented by 1 or decremented by at most 1 after the procedure call `ReduceSwapRestore`(k) (RSR). Thus, if RSR is executed K times, then the number of iterations of the while-loop is at most $K+n-1$, since k is initialized to 2 and the while-loop terminates when $k>n$. We claim that $K = O(n^2 \log_{\omega^{-1}} \|A\|_2)$. Thus the complexity of the first part is $O(n^3 \log_{\omega^{-1}} \|A\|_2)$, since the cost of the execution of RSR is $O(n)$. Therefore, the complexity of Algorithm 2 is $O(n^3 \log_{\omega^{-1}} \|A\|_2)$, assuming m is of the magnitude order of at most $n \log_{\omega^{-1}} \|A\|_2$.

In the following, we show that the number of executions of RSR is $K = O(n^2 \log_{\omega^{-1}} \|A\|_2)$.

Consider the products

$$p_i = \prod_{j=1}^{i} r_{j,j}^2, \qquad i = 1, 2, ..., n-1$$

and

$$P = \prod_{i=1}^{n-1} p_i.$$

Initially, after the QR decomposition, P_{init} is a product of $n(n-1)/2$ squared diagonal elements $r_{i,i}^2$ of R in the QR decomposition (3) of A. Let $\rho(A)$ denote the spectral radius of A, then

$$P_{\text{init}} \le (\rho(A)^2)^{n(n-1)/2} = \|A\|_2^{n(n-1)}.$$

On the one hand, during the execution of the while-loop in Algorithm 2, when the condition C3 is true, that is, $r_{k,k}^2 + (r_{k-1,k} - \gamma r_{k-1,k-1})^2 < \omega r_{k-1,k-1}^2$, Procedure 3 is called, in which the Givens reflection G_i (11) updates $r_{k-1,k-1}$:

$$r_{k-1,k-1}^2 \leftarrow \hat{r}_{k-1,k-1}^2 = r_{k,k}^2 + (r_{k-1,k} - \gamma r_{k-1,k-1})^2, \tag{12}$$

implying that

$$\hat{r}_{k-1,k-1}^2 < \omega r_{k-1,k-1}^2, \tag{13}$$

that is, d_{k-1} is at least reduced by a factor of ω. Also, from Procedure 3, the two-by-two diagonal block

$$\begin{bmatrix} r_{k-1,k-1} & r_{k-1,k} \\ 0 & r_{k,k} \end{bmatrix}$$

is postmultiplied by Z_k (10) and premultiplied by a Given reflection. Since $|\det(Z_k)| = 1$ and $|\det(G_k)| = 1$, we have

$$\hat{r}_{k-1,k-1}^2 \hat{r}_{k,k}^2 = r_{k-1,k-1}^2 r_{k,k}^2. \tag{14}$$

That is the product $r_{k-1,k-1}^2 r_{k,k}^2$ remains unchanged. It then follows that each execution of `ReduceSwapRestore`(k) reduces p_{k-1} by at least a factor of ω while the other products $p_1, ..., p_{k-2}, p_k, ..., p_{n-1}$ remain unchanged. Consequently, each execution of RSR reduces P at least by a factor of ω. Thus, after RSR is executed K times

$$P \leq \omega^K P_{\text{init}} \leq \omega^K \rho(A)^{n(n-1)} = \omega^K \|A\|_2^{n(n-1)}. \tag{15}$$

On the other hand, denoting A_i as the submatrix of A consisting of its first i columns and R_i the ith leading principal submatrix of R, we have

$$p_i = \prod_{j=1}^{i} r_{j,j}^2 = \det(R_i^T R_i).$$

Again, Procedure 3 applies Z_i (10) and the Givens reflection. Since $|\det(Z_i)| = 1$, we have $\det(R_i^T R_i) = \det(A_i^T A_i)$. When A is an integer matrix of full column rank, $\det(A_i^T A_i)$ is a positive integer. It then follows that $\det(R_i^T R_i) \geq 1$. Therefore, through out the execution of the while-loop, $P \geq 1$, which, from (15), shows that

$$1 \leq \omega^K \|A\|_2^{n(n-1)}.$$

It then follows that RSR is executed at most $K \leq \log_{\omega^{-1}} \|A\|_2^{n(n-1)}$ times.

In conclusion, we have the following proposition.

Proposition 1 *Let A be an m-by-n integer lattice basis matrix, then the complexity of Algorithm 2 is $O(n^3 \log_{\omega^{-1}} \|A\|_2)$, assuming m is of the magnitude order of at most $n \log_{\omega^{-1}} \|A\|_2$.*

For real lattice basis matrices, we have the following results.

Proposition 2 *Let A be an m-by-n real lattice basis matrix, then the complexity of Algorithm 2 is $O(n^3 \log_{\omega^{-1}} \text{cond}_2(A))$, assuming m is of the magnitude order of at most $n \log_{\omega^{-1}} \text{cond}_2(A)$.*

Proof. Let

$$m_i = \min_i(r_{i,i}^2) \quad \text{and} \quad M_a = \max_i(r_{i,i}^2).$$

From (13) and $\omega < 1$, we have $\hat{r}_{i-1,i-1}^2 < r_{i-1,i-1}^2$. On the one hand, from (12), we have $\hat{r}_{i-1,i-1}^2 \geq r_{i,i}^2$. Then

$$m_1 \leq r_{i,i}^2 \leq \hat{r}_{i-1,i-1}^2 < r_{i-1,i-1}^2 \leq m_2.$$

Also, from (14) and $\hat{r}_{i-1,i-1}^2 < r_{i-1,i-1}^2$, we have $\hat{r}_{i,i}^2 = r_{i,i}^2(r_{i-1,i-1}^2/\hat{r}_{i-1,i-1}^2) > r_{i,i}^2$. On the other hand, from (11), $\hat{r}_{i,i}^2 = r_{i-1,i-1}^2 - \hat{r}_{i-1,i}^2 \leq r_{i-1,i-1}^2$. Thus

$$m_1 < r_{i,i}^2 < \hat{r}_{i,i}^2 \leq r_{i-1,i-1}^2 \leq m_2.$$

It then follows that through out the execution of the while-loop in Algorithm 2, the squared diagonal elements of R remain in the range $[m_i, M_a]$. Thus the product P in (15) always satisfies

$$P \geq m_i^{n(n-1)/2}.$$

Recalling that

$$P_{\text{init}} \leq M_a^{n(n-1)/2},$$

if Procedure 3 is called K times, then we have we have

$$\omega^{-K} < \frac{P_{\text{init}}}{P} \leq \left(\frac{M_a}{m_i}\right)^{n(n-1)/2} \leq \text{cond}_2(R)^{n(n-1)} = \text{cond}_2(A)^{n(n-1)},$$

which completes the proof.

In particular, the expectation of the 2-norm condition number of a matrix A of order n with normally distributed entries is

$$E(\log(\text{cond}_2(A)) = O(\log n),$$

which leads to the following corollary.

Corollary 1 *Let A be an m-by-n random lattice basis matrix with normally distributed entries, then the complexity of Algorithm 2 is $O(n^3 \log_{\omega^{-1}} n)$, assuming m is of the magnitude order of at most $n \log_{\omega^{-1}} n$.*

In summary, the complexity of Algorithm 2 depends on the parameter ω and the condition number $\text{cond}_2(A)$ in addition to the problem size n, if m is not much larger than n.

6 Experiment Results

We programed Algorithms 1 and 2 in Octave and ran our tests in Mac OS X. The QR decomposition in both algorithms can be implemented in several ways, for example, the norm-induced ordering [7], the sorted-QR decomposition [17], and the joint sorting and reduction [8], to speed up the LLL algorithm. To exclude the effect of different implementations of the QR decomposition, in our test, we generated upper triangular lattice basis matrices whose entries are uniformly distributed between 0 and 1. Thus the QR decomposition was not performed in both

n	20	40	80	160
LLL	0.07123	0.2328	0.9795	4.483
Delay	0.05108	0.1319	0.4564	1.744

Table 1: The cpu times of the LLL algorithm and the LLL algorithm with delayed size-reduction when $\omega = 0.75$. Each entry is an average of five matrices of order n.

n	20	40	80	160
LLL	0.06774	0.3346	1.520	7.689
Delay	0.04836	0.1781	0.5992	2.123

Table 2: The cpu times of the LLL algorithm and the LLL algorithm with delayed size-reduction when $\omega = 0.99$. Each entry is an average of five matrices of order n.

algorithms. Tables 1 and 2 show the cpu times when the parameter ω was set to 0.75 and 0.99 respectively. Each entry in the tables is an average of five matrices of order n. We can see that for problems of size 80, our algorithm is twice as fast as the LLL algorithm. For larger problems, the speed-up is greater.

Tables 3 and 4 list the numbers of executions of condition checks and procedures when the parameter ω was set to 0.75 and 0.99 respectively. Again, each entry is an average of five matrices of order n. The tables show that

- the number nC2 of the condition C2 checks in Algorithm 1 is the same as the number nC3 of the condition C3 checks in Algorithm 2
- the number nSR of the executions of the procedure SR in Algorithm 1 is the same as the number nRSR of the executions of the procedure RSR in Algorithm 2, as pointed out earlier
- the number nC1 of the condition C1 checks is greatly reduced in Algorithm 2 comparing with nC1 in Algorithm 1
- if we count one RSR in Algorithm 2 as one SR and one R1 in Algorithm 1, the sum of nR2 and nRSR in Algorithm 2 is much smaller than the sum of nR1 and nR2 in Algorithm 1. That means the number of size-reductions is significantly reduced.

Conclusion

In this paper, we present a techique called delayed size-reduction to improve the performance of the LLL algorithm. The speed-up is achieved by reducing the number of conditional instructions, integrating size-reduction and swap-restore procedures, and eliminating redundant size-reduction operations. For problems of size as small as 80, our algorithm can be twice as fast as the original LLL algorithm. For larger size problems, the speed-up is greater. Further speed-up can be achieved by incorporating other techniques [7, 8, 17] in the QR decomposition stage.

	LLL					Delay			
n	nC1	nC2	nR1	nR2	nSR	nC1	nC3	nR2	nRSR
20	559	124	56.6	214.6	53.4	190	124	145	53.4
40	2696.8	251.8	115.2	1145.8	106.6	780	251.8	646.8	106.6
80	12018	532.8	244.2	5825.6	227.4	3160	532.8	2769.8	227.4
160	61545	1335.4	599	34168	588.8	12720	1335.4	11939	588.8

Table 3: The numbers of the condition checks and executions of the procedures in the LLL algorithm and the LLL algorithm with delayed size-reduction when $\omega = 0.75$. Each entry is an average of five matrices of order n.

	LLL					Delay			
n	nC1	nC2	nR1	nR2	nSR	nC1	nC3	nR2	nRSR
20	1151	264.8	91.8	272.6	123.4	190	264.8	143.6	123.4
40	4242.6	437.2	171.4	1304.8	200	780	437.2	637.2	200
80	22790	1136.2	393.8	7986	530.4	3160	1136.2	2806.2	530.4
160	100636	2463.2	847.6	38396	1153	12720	2463.2	11916	1153

Table 4: The numbers of the condition checks and executions of the procedures in the LLL algorithm and the LLL algorithm with delayed size-reduction when $\omega = 0.99$. Each entry is an average of five matrices of order n.

Moreover, our algorithm produces same reduced basis matrices as the LLL algorithm up to signs. Our analysis show that the complexity of our algorithm depends on the parameter ω and the condition number of the original lattice basis matrix in addition to the problem size. Our new algorithm also provides a starting point for parallel LLL algorithms [13].

References

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, **48**, 2002, 2201–2214.

[2] J.W.S. Cassels. *An Introduction to the Geometry of Numbers, Second Printing.* Springer-Verlag, Berlin Heidelberg, 1971.

[3] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, **10**, 1997, 233–260.

[4] J. Chun and J. Park. An efficient algorithm for clustered integer least squares problems. *Numer. Linear Algebra Appl.*, 2011.
DOI: 10.1002/nla.768

[5] M.O. Damen, H.E. Gamal, and G. Caire. On maximum-likelihood detection and the search for the closest lattice point. *IEEE Trans. Inf. Theory*, **49**, Oct. 2003, 2389–2402.

[6] Y. Gan, C. Ling, and W.H. Mow. Complex lattice reduction algorithm for low-complexity full-diversity MIMO detection. *IEEE Transactions on Signal Processing*, **57**, 2009, 2701–2710.

[7] Y.H. Gan and W.H. Mow. Complex lattice reduction algorithms for low-complexity MIMO detection. *Proc. IEEE Global Telecommunications Conf. 2005 (GLOBECOM'05)*, **5**, 2005, 5.

[8] Y.H. Gan and W.H. Mow. Novel joint sorting and reduction technique for delay-constrained LLL-aided MIMO detection. *IEEE Signal Processing Letters*, **15**, 2008, 194–197.

[9] G.H. Golub and C.F. Van Loan. *Matrix Computations, Third Edition*. The Johns Hopkins University Press, Baltimore, MD, 1996.

[10] Lattices. *Mathematical Sciences Research Institute Publications*, **44**, Cambridge University Press, Cambridge, UK, 2008, 127–181.

[11] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factorizing polynomials with rational coefficients. *Mathematicsche Annalen*, **261**, 1982, 515–534.

[12] L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*. SIAM, Philadelphia, PA, 1986.

[13] Y. Luo and S. Qiao. A parallel LLL algorithm. *Technical Report CAS-10-03-SQ*, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, 2010.

[14] F.T. Luk and D.M. Tracy. An improved LLL algorithm. *Linear Algebra and Its Applications*, **428**, 2008, 441–452.

[15] The LLL Algorithm: Survey and Applications. *Information Security and Cryptography, Texts and Monographs*. Editors P.Q. Nguyen and B. Vallée. Springer Heidelberg Dordrecht London New York, 2010.

[16] D. Wübben, R. Böhnke, V. Kühn, and K.-D. Kammeyer. Near-maximum-likelihood detection of MIMO systems using MMSE-based lattice reduction. *Proc. 2004 Int. Conf. Communications (ICC04)*, June 2004, 798–802.

[17] D. Wübben, R. Böhnke, J. Rinas, V. Kühn, and K.D. Kammeyer. Efficient algorithm for decoding layered space-time codes. *Electron. Lett.*, **37**, 2001, 1348–1350.

[18] H. Yao and G.W. Wornell. Lattice-reduction-aided detectors for MIMO communication systems. *Proc. 2002 Global Telecommunications Conf. (GLOBECOM'02)*, **1**, 2002, 424–428.