Int. J. Contemp. Math. Sciences, Vol. 7, 2012, no. 13, 599 - 613

Numerical Experiments with the LLL-Based Hermite Normal Form Algorithm for Solving Linear Diophantine Systems

M. Khorramizadeh

Department of Mathematical Sciences
Shiraz University of Technology
Shiraz 71555-313, Iran
m.khorrami@sutech.ac.ir

Abstract

In this paper we are concerned with the practical performance of the LLL-based Hermite normal form algorithm, proposed by Havas [13] (*LLLHMM*) when applied to solve linear Diophantine systems. We first compare the efficiency of the *LLLHMM* with the algorithm, named *LDSSBR*, based on the Rosser's idea. The numerical results show that for small problems, the bit length of the longest integer occurring during the execution of the *LDSSBR* (Bmax) and the computing time are less than those of the *LLLHMM*. As the size of the problem increases the gap between Bmax and the computing time of both algorithms reduces and for large problems Bmax and the computing time of the *LLLHMM* are less than those of the *LDSSBR*. Therefore, in sense of controlling the growth of intermediate results and the computing time for small problems the *LDSSBR* is more efficient while, for large problems the *LLLHMM* is more efficient. However, for all test problems the bit length of the maximum absolute value of the components of the particular solution and basis, obtained by using the *LLLHMM* is significantly less than those of the *LDSSBR*. Then, by presenting some tables of empirical results we justify the experimental conjecture that if the bit length of the components of the coefficient matrix and the right hand side vector of the Diophantine system are equal to k then the nearest integer to the bit length the maximum absolute value of the components of the particular solution and the basis obtained after an application of the *LLLHMM* is approximately $mk/(n-m)$, where m denotes the number of equations and n denotes the number of variables of the system.

Mathematics Subject Classification: 11Y50, 11D04, 15A06, 65F05, 65F20

Keywords: Lattice basis reduction algorithm, Linear Diophantine systems, Rosser's algorithm

1 Introduction

Solving linear Diophantine systems arise from many applications in mathematics and engineering such as integer programming [15], Frobenius problems [10], market split problem [25] and complex chemical reactions [23] and algorithms for computing the solution of these systems are highly desired. Several algorithms have bee proposed for solving linear Diophantine systems. Blankinship [3] introduced a procedure for triangulation of a matrix with integer components, and used the procedure for solving simultaneous linear Diophantine equations [4]. Bradley [5] made some modifications to Blankinship's algorithm and introduced a new algorithm for solving systems of linear Diophantine equations. While algorithms introduced by Blankinship and Bradley were based on explicit calculation of greatest common divisor (gcd), Barnette and Pace [2] presented a new algorithm based on the implicit calculation of gcd. Frumkin [12] showed that none of these algorithms is polynomial. Kannan and Bachem [14] introduced a polynomial algorithm for computing the Hermite and Smith normal forms of an integer matrix, and later Chou and Collins [6] used Kannan and Bachem's ideas to present a polynomial algorithm named *LDSMKB* for solving systems of linear Diophantine equations. They also developed another algorithm based on Rosser's idea [24], the so called *LDSSBR*, and showed numerically that the *LDSSBR* is more successful than *LDSMKB* in controlling the growth of intermediate results, justifying the efficiency of the *LDSSBR* (here we use this algorithm for comparison). In 1994, Contejean and Devie [7] generalized Fortenbacher's [8] algorithm for solving systems of linear Diophantine equations. Linear Diophantine systems can also be solved by using the lattice basis reduction algorithms. Phost [22] used the ideas of lattice basis reduction algorithms to propose an algorithm for solving linear Diophantine systems. Ardal et al. [1] developed an algorithm for solving linear Diophantine equations based on lattice basis reduction algorithms. Havas et al. [13] used the lattice basis reduction algorithms to present an algorithm for computing the Hermite normal form of an integer matrix [13] and Matthews [18] used this algorithm to find short integer solution of linear Diophantine systems.

The main difficulty with the algorithms for solving linear Diophantine systems is the rapid growth of intermediate results, making many algorithms impractical even for large computers. Here, by presenting some tables of numerical results we examine the empirical performance of the *LLL*-based Hermite normal form algorithm of Havas et al. [13] (*LLLHMM*), for solving linear Diophantine systems. We compare the efficiency of the *LLLHMM* with the *LDSSBR* of Cou and Collins [6], being an efficient algorithm based on the

Rosser's idea. The numerical results show that for small problems, the bit length of the longest integer occurring during the execution (Bmax) and the computing time of the *LDSSBR* are less than those of the *LLLHMM*. Therefore, for small problems the *LDSSBR* is more efficient than the *LLLHMM* in the sense of controlling the growth of intermediate results and the computing time. However, as the size of the Diophantine systems increases the gap between Bmax and the computing time of both algorithms reduces and for large problems the *LLLHMM* is more efficient. For all test problems the bit length of the maximum absolute value of the components of the particular solution and basis, obtained by using the *LLLHMM* is significantly less than those of the *LDSSBR*.

By presenting some tables of empirical results we also justify the experimental conjecture that if the bit length of the components of the coefficient matrix and the right hand side vector of the Diophantine system are equal to k then the nearest integer to the bit length the maximum absolute value of the components of the particular solution and the basis obtained after an application of the *LLLHMM* is approximately $mk/(n-m)$, where m denotes the number of equations and n denotes the number of variables of the system. This practical estimate indicates that when n is large but m is small the bit length of the components of the particular solution and the basis obtained after the application of the *LLLHMM* are small.

Let $A = (a_1, \cdots, a_m)^T$, $a_j \in Z^n$, $1 \leq j \leq m$. Consider the following system of linear Diophantine equations:

$$Ax = b, \qquad A \in Z^{m \times n},\ x \in Z^n,\ b \in Z^m. \tag{1}$$

System (1) can be written as:

$$a_i^T x = b_i, \quad 1 \leq i \leq m,$$

where a_i^T is the *i–th* row of the coefficient matrix A, and b_i is the *i–th* component of the right hand side vector b. By the integer null space of A we mean the largest subset of the null space, the elements of which are integer vectors. We say that an integer matrix N spans the integer null space of an integer matrix A, if the space generated by integer combinations of the columns of N is the integer null space of A. A vector $x_p \in Z^n$ satisfying $a_i^T x_p = b_i$, $1 \leq i \leq m$, is called a particular solution of (1). Let $v \in Z^n$, $N \in Z^{n,n_1}$, $y \in Z^{n_1}$, $n_1 \in Z$. Note that if v is a particular solution of (1) and N spans the integer null space of A then

$$x = v + Ny, \tag{2}$$

is the general solution of (1). This means that x_p is a particular solution of (1) if and only if $x_p = v + Ny_p$, for some $y_p \in Z^{n_1}$. If the columns of N are linearly independent then N is called a basis for the integer null space of A.

By the integer null space of A we mean the subspace containing integer vectors of the null space. We say that an integer matrix N spans the integer null space of an integer matrix A, if the space generated by integer combinations of the columns of N is the integer null space of A. A vector $x_p \in Z^n$ satisfying $a_i^T x_p = b_i$, $1 \leq i \leq m$, is called a particular solution of (1). Let $v \in Z^n$, $N \in Z^{n \times n_1}$, $y \in Z^{n_1}$, $n_1 \in Z$. Note that if v is a particular solution of (1) and N spans the integer null space of A then

$$x = v + Ny, \tag{3}$$

is the general solution of (1). This means that x_p is a particular solution of (1) if and only if $x_p = v + Ny_p$, for some $y_p \in Z^{n_1}$. If the columns of N are linearly independent then N is called an integer basis for the null space of A.

In section 2 we consider the LLL-based Hermite normal form algorithm of Havas et al. [13] and explain how linear Diophantine systems can be solved by using this algorithm. In section 3 we describe the *LDSSBR* of Chou and Collins. In section 4 we compare the efficiency of the *LDSSBR* with the *LLL*-based algorithm for solving linear Diophantine systems. In section 5 we present a practical estimate for the bit length of the particular solution and basis obtained after an application of the *LLLHMM*. Conclusions are given in Section 6.

2 LLL-based Hermite normal form algorithm

In this section we consider some basic ideas of the LLL-based Hermite normal form algorithm. We first need to explain lattice basis reduction algorithms. A set $L \subseteq R^n$ is called a lattice if there exists a set of integer vectors $\{c_1, c_2, \cdots, c_m\}$ such that

$$L = \{\sum_{j=1}^{m} \alpha_j c_j | \alpha_j \in Z, 1 \leq j \leq m\}.$$

If c_j, $1 \leq j \leq m$, are linearly independent then $\{c_1, c_2, \cdots, c_m\}$ is called a basis for L. Lattice basis reduction algorithms are designed to transform a given lattice basis into a basis that consists of short and pairwise nearly orthogonal vectors.

2.1 *LLL* Lattice basis reduction algorithms

In the following we only consider integer input bases. Let $\{b_1, b_2, \cdots, b_m\}$ be an ordered lattice basis, then the *Gram-Schmidt vectors* b_i^*, $1 \leq i \leq m$, are defined by recursion

$$b_1^* = b_1, \quad b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \quad 2 \leq i \leq m,$$

where the *Gram-Schmidt coefficients* μ_{ij} are defined by

$$\mu_{ij} = \frac{{b_i}^T b_j^*}{{b_j^*}^T b_j^*}, \quad \mu_{ii} = 1, \quad 1 \le j < i \le m,$$

and $\mu_{ij} = 0$ for $i < j$. An individual basis vector is *size–reduced* if

$$|\mu_{ij}| < 1/2, \quad 1 \le j < i.$$

Let $1/4 < \alpha \le 1$ be a constant. An ordered basis $\{b_1, b_2, \cdots, b_m\}$ is called *LLL–reduced* with α if b_i is *size–reduced*, for $1 \le i \le m$, and

$$\alpha \|b_{i-1}^*\|^2 \le \|b_i^* + \mu_{i,i-1} b_{i-1}^*\|^2, \quad 2 \le i \le m.$$

For a real number x, let $[x] \in Z$ denotes the nearest integer to x($[x] = x - 0.5$ for half integers). The following procedure is used for size–reduction of the individual basis vector b_k, where $\{b_1, b_2, \cdots, b_m\}$ is a lattice basis and μ_{ij}, $1 \le j < i \le m$, are the Gram–Schmith coefficients of $b_1, b_2, \cdots, b_m$.

For $j = k - 1$ **to** 1
  If $|\mu_{k,j}| > 1/2$
    $b_k \leftarrow b_k - [\mu_{kj}] b_j$
    For $i = 1$ **to** m
      $\mu_{k,j} \leftarrow [\mu_{k,j}] \mu_{j,i}$

By size–reducing each vector individually, by using the above procedure, we can obtain a size–reduced basis $\{b_1, b_2, \cdots, b_m\}$. Note that size–reducing the vector b_k does not affect the size–reducing of the other vectors. Next we describe the *LLL–reduction* algorithm (see [13, 9, 16]). Let $\{b_1, b_2, \cdots, b_m\}$ be a lattice basis, and $1/4 < \alpha < 1$.

Algorithm 2.1 ***Algorithm for*** **LLL–reduction**

Step 1: *(initiation) Set $k = 2$ and compute the Gram–Schmith coefficients μ_{ij} for $1 \le j < i \le m$ and $\|b_i^*\|^2$ for $i = 1, \cdots, m$.*

Step 2: ***While*** $k \le m$ ***do***

Size reduce the vector b_k and update μ_{kj} for $j = 1, \cdots, k-1$.

if $\alpha \|b_{k-1}^*\|^2 > \|b_k^*\|^2 + \mu_{k,k-1}^2 \|b_{k-1}^*\|^2$, ***then*** *swap b_k and b_{k-1}, $k = max(k - 1, 2)$*

else $k \leftarrow k + 1$.

The output of the above algorithm is a *LLL–reduced* basis $\{b_1, b_2, \cdots, b_k\}$.

Remark 2.2 *In 1987 de Weger [9] proposed a reformulation of the LLL–reduction algorithm, modified to work only with integer variables and avoids round of error that may occur in computation of Gram–Schmith coefficients.*

Several approaches are proposed for solving linear Diophantine systems via lattice basis reduction algorithms [1, 13, 16, 21, 22]. In the following three subsections we consider the one based on the so called LLL–based Hermite normal form algorithm of Havas, Majewski and Mattews [13]. We first need to consider the LLL–based extended gcd algorithm and LLL–based Hermite normal form algorithm.

2.2 LLL–based extended gcd algorithm

In this section we explain the LLL–based extended gcd algorithm for computing $\delta = gcd(a)$, the greatest common divisor of the elements of the integer vector $a = (a_1, \cdots, a_m)^T \in Z^m$. The extended gcd algorithm is designed to find integer vector $x = (x_1, \cdots, x_m)^T \in Z^m$ so that $\delta = x^T a$ and the bit length of the components of x are as small as possible. One approach to this problem is to apply the LLL–reduction algorithm to the lattice L spanned by the rows of the matrix $F = [I_m, \gamma a]$, where γ is a positive integer. It can be shown that if $\gamma > \theta^{(m-2)/2}\|a\|$, where $\theta = 4/(4\alpha - 1)$ and $1/4 < \alpha \leq 1$, then the reduced basis for F must satisfy $F_{j,m+1} = 0$, $1 \leq j \leq m-1$, and $F_{m,m+1} = \pm\gamma\delta$. Then F_{mj}, $1 \leq j \leq m$, can be considered as the components of x, see [13]. In practice when γ is large, this algorithm seems to settle down to the same sequence of row operations. It is not difficult to identify these operations and perform them instead on the matrix $[I_m, a]$ (see [13]). Let b_i denotes the *i–th* row of B. Then the resulting algorithm which uses the idea of de Weger [9] and is simplified in that the initial construction of the *Gram–Schmith* basis is not needed as we start with the identity matrix I_m, is called the LLL–based extended gcd algorithm. The input of the algorithm is the positive integer vector a and the outputs of the algorithm are δ and small multiplier integer vector x, such that $x^T a = \delta$. See [13] for details.

2.3 LLL–based Hermite normal form algorithm

An $m \times n$ integer matrix H is said to be in Hermite normal form if

i. The first r rows of H are nonzero.

ii. For $1 \leq i \leq r$ if h_{ij_i} is the first nonzero entry in row i of H, then $j_1 < j_2 < \cdots < j_r$.

iii. $h_{ij_i} > 0$ for $1 \leq i \leq r$.

iv. If $1 \leq k < i \leq r$, then $0 \leq h_{kj_i} < h_{ij_i}$.

Let A be an $m \times n$ integer matrix. The goal of the LLL-based Hermite normal form algorithm is to find a unimodular matrix T such that $HNF(A) = TA$, where $HNF(A)$ denotes the Hermite normal form of the integer matrix A, (see [9]). The LLL-based Hermite normal form algorithm is based on the ideas of the LLL–based extended gcd algorithm. Let C^T be the submatrix of $HNF(A)$ formed by the r nonzero rows of $HNF(A)$. Then, we have $T = [Q, N]^T$. Therefore, we can write $Q^T A = C^T$, $N^T A = 0$ and the columns of N will form an integer basis for the null space of A. The input of the LLL-based Hermite normal form algorithm is an integer matrix A, and the outputs are the $HNF(A)$ and the transformation matrix T, so that, $TA = HNF(A)$. See [13] for details. The following remark is concerned with the complexity of the LLL-based Hermite normal form algorithm, See [20] for the proof.

Remark 2.3 *Let $L \geq 2$ be such that the rows of the input matrix A have squared length at most L. Then all through the algorithm all entries have bit length $O(mlog(mL))$.*

Next, we describe how to solve a linear Diophantine system by using the LLL-based Hermite normal form algorithm.

2.4 Linear Diophantine systems

The LLL–based Hermite normal form algorithm can be used to find short solutions of linear Diophantine systems efficiently, when the Diophantine system $Ax = 0$ is nonempty. Note that in this case we have $m < n$ and therefore in the following we do not consider the case $m = n$. Let $HNF(A^T) = (C^T, 0^T)^T$, where C consists of nonzero rows. Then the linear diophantine systems $Ax = b$ has integer solutions if and only if the Hermite normal form of the matrix $G = \begin{pmatrix} A^T & 0 \\ b^T & 1 \end{pmatrix}$ has the form $HNF(G) = \begin{pmatrix} C^T & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$, and the corresponding unimodular transformation matrix P, which will have small entries has the form $T = \begin{pmatrix} Q^T & 0 \\ -p^T & 1 \\ N^T & 0 \end{pmatrix}$. See [13] for the proof. Note that since T is unimodular, then Q has full column rank. Moreover, from the identity $TG = HNF(G)$, we deduce that p is a short solution of $Ax = b$ and the columns of N form a short integer basis for the null space of A. Indeed, we have $Q^T A^T = C^T$. See [19] for details. Let $\widehat{a}_i$ be the $i - th$ column of A. The following Remark considers the complexity of the *LLL*-based Hermite normal form algorithm for solving linear Diophantine systems.

Remark 2.4 *By Remark 2.3 all through the the* LLL*-based Hermite normal form algorithm all entries have bit length* $O((n+1)log((n+1)L))$*, where*

$$L \geq Max\{Max\{\widehat{a}_i^T \widehat{a}_i,\ \ 1 \leq i \leq m\}, b^T b + 1\}.$$

3 The *LDSSBR*

In 1982, Chu and Collins [6] presented two algorithms for solving systems of linear Diophantine equations, named *LDSMKB* and *LDSSBR*, which control the growth of intermediate results. The basic ideas for the *LDSMKB* come from Kannan and Bachem [14]. They used these ideas in computing Smith and Hermite normal forms for a nonsingular integral matrix A and showed that the length of intermediate results are bounded by a polynomial function of the components of A. The *LDSSBR* (Linear Diophantine System Solver Based on Rosser's idea) is based on the Rosser's algorithm [24], for finding a general solution to a linear Diophantine equation with much smaller norm of the particular solution and columns of the integer null space generator.

Next we describe the Rosser's algorithm. consider the following single linear Diophantine equation:

$$a_1^T x = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \quad a_1, x \in Z^n, \quad b_1 \in Z. \tag{4}$$

Using the same notation as in [6], let

$$C = \begin{bmatrix} a_1^T \\ I \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} -b_1 \\ 0 \end{bmatrix} = \begin{bmatrix} -b_1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Assume $a_1 \neq 0$. Let c_{ij} be the element in the *i–th* row and *j–th* column of C, C_j be the *j–th* column of C and B_j be the *j–th* component of B. Moreover, let a and b be two integer numbers. We say a divides b, and write $a \mid b$ if and only if $b = aq$ for some $q \in Z$. Below $\lfloor x \rfloor$ is the greatest integer number less than or equal to x. Rosser's algorithm (RA) for solving (4) follows.

Algorithm 3.1 *Rosser's Algorithm (RA).*

Step 1 **:** ***For*** $j = 1, \cdots, n$ ***if*** *any leading element of* C_j *is negative* ***then*** *replace* C_j *by* $-C_j$*. Sort* $C_1, C_2, \cdots, C_n$ *in descending order with respect to their first component.*

Step 2 **:** ***While*** $c_{12} \neq 0$ ***do*** $\{B \leftarrow B - \lfloor \frac{B_1}{c_{11}} \rfloor C_1, C_1 \leftarrow C_1 - \lfloor \frac{c_{11}}{c_{12}} \rfloor C_2$*. Sort the columns* $C_1, C_2, \cdots, C_n$ *in descending order with respect to their first component.*$\}$

Step 3 **:** *At this point the matrix* C *and the vector* B *have the forms:*

$$C = \begin{bmatrix} \delta & 0 \\ p & U \end{bmatrix} = \begin{bmatrix} \delta & 0 & \cdots & 0 \\ p & u_1 & \cdots & u_{n-1} \end{bmatrix}, \qquad B = \begin{bmatrix} \theta \\ v \end{bmatrix},$$

where v*,* p *and* $u_i \in Z^n$*,* $1 \le i \le n-1$*, and* $\delta = gcd(a_{11}, a_{12}, \cdots, a_{1n})$*.* ***If*** $\theta \neq 0$ ***then*** *(4) has no integer solution* ***else*** *the general integer solution of (4) is:*

$$x = v + y_1u_1 + y_2u_2 + \cdots + y_{n-1}u_{n-1} = v + Uy, \qquad y \in Z^{n-1}. \quad (5)$$

The numerical experiments in [6] shows that in all cases the *LDSSBR*, based on the Rosser's algorithm, find smaller particular solutions and bases than the *LDSMKB*. Assume that the system (1) has an integer solution and $rank(A) = m$, so that the m rows of A are linearly independent. The *LDSSBR* for solving (1), given in [6], can be written as follows.

Algorithm 3.2 ***:*** *The* LDSSBR*.*

Step 1**:** {*Initialize*} n *= Number of Columns of* A*,* m *= Number of Rows of* A*.*

Step 2**:** {*Adjoin identity matrix to* A *and zero vector to* $-b$}

$$C \leftarrow \begin{bmatrix} A \\ I \end{bmatrix}, \quad B \leftarrow \begin{bmatrix} -b \\ 0 \end{bmatrix}.$$

Step 3**:** {*Make pivot row zero*} ***If*** *the first row of* C *is zero vector* ***then go to Step 4 else*** *apply RA to* C *and* B*.*

Step 4**:** {*Checking for inconsistency*} ***If*** $B_1 \neq 0$ ***then stop*** *(the system is inconsistent)* ***else*** *replace* C *with the matrix obtained by deleting the first row and the first column of* C *and replace* B *by the vector obtained by deleting the first element of* B *and set* $m \leftarrow m - 1$*.*

Step 5**:** ***If*** $m > 0$ ***then go to Step 3 else*** *let* $x^* \leftarrow B$*,* $N \leftarrow C$*, and* ***stop****.*

In the following section we compare the efficiency of the *LDSSBR* with the algorithm for solving linear Diophantine systems based on the *LLL*-based Hermite normal form algorithm of Havas, Majewski and Mathews [13].

4 Numerical results

In the following by the infinity norm of a vector $a = (a_1, \cdots, a_n) \in Z^n$ we mean $max\{|a_j|,\ 1 \leq j \leq n\}$ and the infinity norm of a matrix $M \in Z^{m\times n}$ is $max\{|m_{ij}| :\ 1 \leq j \leq m, 1 \leq i \leq n\}$, where m_{ij}'s are the components of M. By the bit length of an integer number x, we mean the number of binary bits necessary to represent x.

The implementation was done in the Mathematica 7.0 software environment, working on an Intel Pentium 4 processor of 1.8 GHz and 1 GB of memory.We tested the algorithms on some randomly generated systems of equations. To generate random linear systems we used the ideas of Chou and Collins [6]. Coefficients of the sample systems are randomly chosen so that the bit length of the input data are equal to k. Given the number of variables (n), the number of equations (m) and the maximum bit length of the components of the coefficient matrix and the right-hand side vector (Bl), for each sample problem we generated random systems successively until a consistent one is found (this is a reasonable approach for generation of random test problems that keeps the size of the right hand sides in control; see [6]). After an application of the prescribed algorithms to the randomly generated systems we recorded the numerical results in Tables 1, 2, 3 and 4. In these tables, m, n, Bs, Bb, $Bmax$, T and PE denote the number of equations, the number of variables, the bit length of the maximum absolute value of the components of the particular solution obtained, the maximum absolute value of the components of the basis obtained, the bit length of the longest integer accruing during the algorithm, the computing time of the algorithm (in seconds) and the practical estimate, that is, the nearest integer to $mk/(n-m)$ respectively.

Tables 1, 2, 3 and 4 are concerned with the comparison of the *LDSSBR* with the LLL-based Hermite normal form algorithm of Havas, Majewski and Mathews (*LLLHMM*), for solving systems of linear Diophantine equations. The numerical results show that for small problems the computing time of the *LDSSBR* is less than that of the *LLLHMM*. For example when $k = 10$, $m = 32$ and $n = 43$ the computing time of the *LDSSBR* is 4.97 seconds and the computing time of the *LLLHMM* is 16.22 seconds. When $k = 40$, $m = 40$ and $n = 45$ the *LDSSBR* needs 35.64 seconds, while the *LLLHMM* needs 108.67 seconds. As the size of the problem increases the gap between the computing time of both algorithms reduces until we reach at point where the computing time of both algorithms are almost equal. For example when $k = 10$, $m = 64$ and $n = 67$ the computing time of the *LDSSBR* is 90.62 seconds and the computing time of the *LLLHMM* is 93.95 seconds.

For large problems *LLLHMM* is faster. For example, in Table **??**, when $k = 10$, $m = 94$ and $n = 100$ the computing time of the *LDSSBR* is 8252.25 seconds and the computing time of the *LLLHMM* is 426.76 seconds. In Table

??, when $k = 20$, $m = 83$ and $n = 89$ the *LDSSBR* needs 4144.73 seconds, while the *LLLHMM* needs 613.67 seconds.

In all cases the bit length of the solutions and bases obtained by using the *LLLHMM* is considerably less than or nearly equal to those of the *LDSSBR*. As in table 1, for $k = 10$, $m = 82$ and $n = 90$ the bit length of the solution obtained after the application of the *LDSSBR* and *LLLHMM* are 12688 and 120 respectively and bit length of the basis obtained after the application of the *LDSSBR* and *LLLHMM* are 12689 and 121 respectively.

For small problems, the bit length of the longest integer, occurring during the computations (Bmax) of the *LDSSBR* is considerably less than those of the *LLLHMM*. For example for $k = 60$, $m = 7$ and $n = 12$ the value of Bmax for the *LDSSBR* is 152 and the value of Bmax for the *LLLHMM* is 1807. In a way similar to the computing time, as the Diophantine systems grow larger the gap between the value of Bmax for both algorithms becomes smaller and for large problems the value of Bmax for the *LLLHMM* is smaller than that of the *LDSSBR*. As in table 2 for $k = 20$, $m = 91$ and $n = 95$ the value of Bmax for the *LDSSBR* is 29525 and the value of Bmax for the *LLLHMM* is 9821. Moreover, the larger the $n - m$ is, the smaller the bit length of the solution and basis obtained by using the *LLLHMM*.

Table 1: k=10

		LDSSBR				LLL				
m	n	Bs	Bb	Bmax	T	Bs	Bb	Bmax	T	PE
6	10	20	24	27	0.094	15	17	275	0.25	15
8	13	27	29	35	0.141	17	17	375	0.375	16
21	25	103	105	107	0.765	57	58	1086	2.86	52
32	43	239	241	241	4.968	33	33	1724	16.219	29
27	52	152	156	156	5.532	13	13	1445	28.188	10
35	40	279	280	292	5.078	78	79	1910	14.125	70
38	45	356	357	357	7.938	61	62	2089	20.25	54
41	66	526	528	528	27.468	19	20	2275	70.593	16
44	45	489	490	490	9.735	489	490	2427	23.688	440
53	60	1190	1192	1192	40.765	86	87	2978	58.922	75
64	67	1682	1685	2104	90.62	245	248	3648	93.95	213
73	80	5749	5752	5752	330.75	121	122	4186	177.515	104
76	84	7956	7960	7690	513.281	111	112	4380	206.719	95
82	90	12688	12689	12689	1432.3	120	121	4750	277.032	102
94	100	28318	28319	28332	8252.25	184	186	5504	426.766	156

Table 2: k=20

		LDSSBR				LLL				
m	n	Bs	Bb	Bmax	T	Bs	Bb	Bmax	T	PE
4	8	29	31	39	0.094	20	21	349	0.203	20
7	10	68	70	70	0.172	46	48	665	0.36	46
12	18	80	84	89	0.485	40	42	1171	1.688	40
15	24	103	107	108	1.016	34	35	1475	3.859	33
22	37	184	187	199	3.828	31	32	2240	17.797	29
27	33	291	295	295	4.531	94	95	2770	13.531	90
30	37	350	352	364	7.000	91	92	3099	19.891	85
35	42	536	537	537	12.719	105	106	3635	32.797	100
41	47	789	791	802	22.281	145	146	4282	49.843	136
45	50	1046	1049	1050	33.375	190	192	4735	66.578	180
53	67	2661	2661	2661	113.531	81	82	5599	165.829	75
65	75	6311	6313	6313	334.766	140	141	6916	278.656	130
76	82	12101	12107	12107	1407.88	274	274	8113	432.86	253
83	89	20987	20988	21101	4144.73	300	301	8929	613.672	276
91	95	29525	29525	21101	14537.6	491	495	9821	829.546	455

Table 3: k=40

		LDSSBR				LLL				
m	n	Bs	Bb	Bmax	T	Bs	Bb	Bmax	T	PE
9	16	101	104	124	0.562	52	53	1716	2.015	51
11	22	114	116	143	1.109	40	41	2109	5.437	40
15	18	296	300	300	1.266	203	203	2937	3.672	200
22	38	357	360	368	8.000	57	57	4400	39.203	55
25	30	482	483	483	6.14	204	205	5014	21.328	200
27	33	570	572	572	8.437	183	184	5401	29.25	180
30	35	705	707	707	11.36	245	247	6033	39.171	240
35	42	1023	1024	1024	23.265	205	206	7089	75.719	200
40	45	1409	1411	1411	35.64	329	329	8311	108.671	320
44	50	1934	1935	1939	59.984	302	303	8945	163.516	293
53	57	3137	3141	3304	134.235	548	549	10854	290.438	530
56	62	4752	4755	4755	216.25	386	387	11483	406.093	373
67	75	12766	12770	12770	979.70	347	348	14636	905.734	335
71	77	15290	15292	15292	1702.59	491	492	14636	1035.7	473
81	86	29918	29919	29932	7880.53	673	675	16746	1814.25	648

Table 4: k=60

		LDSSBR				LLL				
m	n	Bs	Bb	Bmax	T	Bs	Bb	Bmax	T	PE
7	12	127	127	152	0.438	78	79	1807	1.156	84
13	19	253	258	266	1.625	125	126	3582	5.546	130
15	16	852	853	853	1.422	852	853	4146	4.469	900
16	24	316	318	341	3.281	117	117	4441	12.765	120
18	30	343	345	360	5.438	87	88	5014	26.469	90
21	32	473	474	490	7.953	112	112	5915	36.422	114
24	27	752	754	754	6.703	461	462	6778	25.984	480
30	35	981	983	995	16.797	348	349	8570	69.094	360
32	41	1213	1220	1220	26.188	209	210	9198	111.813	213
34	38	1298	1301	1301	25.094	498	500	9799	102.953	510
40	45	1949	1952	1952	53.937	470	473	11609	205.109	480
47	53	3434	3436	3446	133.672	462	463	13648	410.031	470
55	60	5688	5688	5929	286.875	651	651	16096	750.359	660
71	78	23886	23890	23911	4438.14	606	607	21039	2373.73	608
81	85	36554	36558	36664	22739.1	1214	1217	24137	3615.97	1215

5 Practical estimate

In this section we practically justify that if the bit length of input data is k and $n > m$, then the bit length of the maximum absolute value of the components of the particular solution and the basis obtained after an application of the *LLLHMM*, is approximately $mk/(n-m)$.

First note that according to the numerical results of the Tables of the previous section, the rate of the growth of Bs and Bb are almost the same. Therefore, it is sufficient to only consider the growth of Bs. During the implementations we observed that for example when $k = 40$ and $n - m = 1$ the rate of the growth of Bs is about 40, which is the bit length of input data and for $n - m = 5, 10$ and 20 the rate of the growth of Bs is approximately $8, 4$ and 2 respectively. This motivated us to test the conjecture that the rate of the growth of Bs is approximately $k/(n-m)$, where k denotes the bit length of input data. With the rate of the growth of Bs at hand, we expect that the value of Bs for given n, m and k to be approximately $mk/(n-m)$. To test

this conjecture we numerically compared the practical value of Bs and Bb to $mk/(n-m)$ for some random values of m, n and k, as recorded in tables 1, 2, 3 and 4. In these tables the symbol PE represents $mk/(n-m)$. The numerical results of tables 1, 2, 3 and 4 justifies that the obtained values of Bs are very close to $mk/(n-m)$. So we may come to believe that if the bit length of the input data are equal to k then the bit length of the maximum absolute value of the components of the particular solution and the basis obtained after applying the *LLLHMM* Hermite normal form algorithm for solving $Ax = b$, is approximately $mk/(n-m)$.

When $n > m$ then the numerical results of tables 1, 2, 3 and 4 shows that in all cases the value of Bs and Bb are less than or equal to $3/2mk/(n-m)$. This justifies that the value of Bs and Bb are $O(mk/(n-m))$. Note that the practical estimate indicates that when n is large, but m is small with respect to n, that is $n-m$ is large, then we expect that the value of Bs and Bb remain small. For example we observed that when $k = 40$, $n = 80$, and $m = 3$ then we have $Bs = 2$ and $Bb = 1$. Therefore, the practical estimate of this monograph, depending on three parameters m, n and k describes the behavior of the *LLLHMM* well.

6 Conclusions

We examined the practical performance of the *LLL*–based Hermite normal form algorithm for solving linear Diophantine systems. The numerical results showed that for small problems the *LDSSBR* is more efficient than the *LLLHMM* in sense of controlling the growth of intermediate results and the computing time while, for large problems the *LLLHMM* needs less computing time and controls the growth of intermediate results better. However, for all test problems the bit length of the maximum absolute value of the components of the particular solutions and bases obtained by the *LLLHMM* are less than those of the *LDSSBR*. Moreover, we proposed a practical estimate for the bit length of the generated particular solutions and bases computed by the *LLLHMM*. The practical estimate enables us to investigate the practical behavior of the *LLLHMM*.

Acknowledgments

The author thank The Research Council of Shiraz University of Technology, for its support.

References

[1] K. Aardal, C. A. J. Hurkans and A. K. Lenstra, Solving a system of linear Diophantine equations with lower and upper bounds on the variables, *Mathematics of Operations Research* **3** (2000), 427 - 442.

[2] S. Barnette and I. S. Pace, Efficient algorithms for linear systems calculations: Part I-Smith form and common divisor of polynomial matrices, *Internat. J. of Systems Sci.*, **5** (1974), 403 - 411.

[3] W. A. Blankinship, Algorithm 287, matrix triangulation with integer arithmetic [F1], *comm. ACM*, **9** (1966), 513.

[4] W. A. Blankinship, Algorithm 288, solution of simultaneous linear Diophantine equations, *Comm. ACM*, **9** (1966), 514.

[5] G. H. Bradley, Algorithms for Hermite and Smith normal matrices and linear Diophantine equations, *Math. Comp.*, **25** (1971), 897 - 907.

[6] T. J. Chou and E. E. Collins, Algorithms for the solutions of systems of linear Diophantine equations, *SIAM J. Comput.*, **11** (1982), 686-708.

[7] E. Contejean and H. Devie, An efficient algorithm for solving systems of Diophantine equations, *Information and Computation*, **1** (1994), 143 - 172.

[8] M. Clausen and A. Fortenbacher, Efficient solution of Diophantine equations, *Journal of Symbolic Computation*, **8(1&2)** (1989), 201-216.

[9] B. M. M. de Weger, An Solving exponential Diophantine equations using lattice basis reduction algorithms, *Journal of Number Theory*, **26** (1987), 25 - 367.

[10] P. Erdos and R.L. Graham, On a linear Diophantine problem of Frobenius, *Acta Arithm.*, **21** (1972), 399 - 408.

[11] H. Esmaeili , N. Mahdavi-Amiri and E. Spedicato, A class of ABS algorithms for Diophantine linear systems, *Numeriche Mathematik*, **90** (2001), 101 - 115.

[12] M. A. Frumkin, Polynomial time algorithms in the theory of linear Diophantine equations, *M. Karpinski, ed., Fundamentals of Computation Theory, Lecture Notes in Computer Science*, Vol. 56, Springer, New York, 1977, pp. 386-392.

[13] G. Havas, B.S. Majewski, K. R. Mathews, Extended GCD and Hermite normal form algorithms via lattice basis reduction, *Experimental mathematics*, **7:2** (1998), 125 - 135.

[14] R. Kannan and A. Bachem, Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix, *SIAM J. Comput.*, **8** (1979), 499 - 507.

[15] H.W. Lenstra, Jr, Integer programming with a fixed number of variables, *Mathematics of Operations Research*, **8** (1983), 538 - 548.

[16] A. K. Lenstra, H. W. Lenstra and L. Lovăsz, Factoring polynomials with rational coefficients, *Mathematische Annalen*, **261** (1982), 515 - 534.

[17] J. C. Lagarias, H. W. Lenstra and C. P. Schnorr, Korkine-zolotareff bases and successive minima of a lattice and its reciprocal lattice, *Combinatorica*, **10(4)** (1990). 333 - 348.

[18] Minkowski, Geometrie der Zahlen, Teubner 1896; Diophantische Approximation. Teubner 1907; Gessamelte Abhandlungen I,II.

[19] K. Matthews, Short solutions of $Ax = b$ using LLL-based Hermite normal form algorithm, August 18, 1998, preprint.

[20] W. V. D. Kallen, Complexity of the Havas, Majewski,Matthews LLL Hermite normal form algorithm, *J. symbolic computation* **11** (2000), 1 - 10.

[21] M. Newman, *Integral Matrices*, Academic Press, 1972.

[22] P. Pohst, *Computational algebraic Number Theory*, DMV seminar Band 21, Birkhäuser, 1994.

[23] D. Papp and B. Vizvari, Effective solution of linear Diophantine equation systems with an application in chemistry, *Journal of Mathematical Chemistry* **39** (2005) 15-31.

[24] J.B. Rosser, *A note on the linear Diophantine equation*, Amer. Math. Monthly 48(1941) 662-666.

[25] A. Wassermann, Attacking the market split problem with lattice point enumeration, *Journal of Combinatorial Optimization* **6** (2002) 5-16.

Received: August, 2011