Balanced Solutions of Linear Diophantine Equations
AMIMS 2013

K.A. Draziotis
Department of Informatics
Aristotle University of Thessaloniki

11-12 April, 2013
AMIMS
Athens

Solutions of linear diophantine equations

- Let $a_j \in \mathbb{Z} - \{0\},\ (0 \leq j \leq n)$. We consider the linear diophantine equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0 \tag{1}$$

Solutions of linear diophantine equations

- Let $a_j \in \mathbb{Z} - \{0\},\ (0 \leq j \leq n)$. We consider the linear diophantine equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0 \qquad (1)$$

- Without the bound constraints can be solved in polynomial time. For instance see
  - *H.Esmaeili, Lecturas Matemáticas Volumen 27 (2006).*

Solutions of linear diophantine equations

- Let $a_j \in \mathbb{Z} - \{0\},\ (0 \leq j \leq n)$. We consider the linear diophantine equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0 \qquad (1)$$

- Without the bound constraints can be solved in polynomial time. For instance see
  - *H.Esmaeili, Lecturas Matemáticas Volumen 27 (2006).*
- If with $|x_j| \leq X_j$ then the problem is NP-complete.

Some Applications.

- Several problems are related with integer solutions of a linear equation, under the previous bound constraints.
  for the gcd, is called extended gcd problem.

Some Applications.

- Several problems are related with integer solutions of a linear equation, under the previous bound constraints.
  for the gcd, is called extended gcd problem.

Some Applications.

- Several problems are related with integer solutions of a linear equation, under the previous bound constraints.
  for the gcd, is called extended gcd problem.

Some Applications.

- Several problems are related with integer solutions of a linear equation, under the previous bound constraints.
  for the gcd, is called extended gcd problem.
- The decidability problem of the existence of a solution of Extended gcd under the bound $|x_j| < X_j$ is proved to be NP-complete.

Some Applications.

- Several problems are related with integer solutions of a linear equation, under the previous bound constraints.
  for the gcd, is called extended gcd problem.
- The decidability problem of the existence of a solution of Extended gcd under the bound $|x_j| < X_j$ is proved to be NP-complete.
- We shall study this problem and we develop an algorithm which give us a solution for the Egcd problem (under some assumptions)

Some Applications.

- If we restrict the solutions $x_j \in \{0, 1\}$, then we have the $0 - 1$ Knapsack or subset sum problem. That is, given a set $I = \{a_1, a_2, ..., a_n\}$ and s (all positive integers) find a subset of the set I which have sum s.

Some Applications.

- If we restrict the solutions $x_j \in \{0, 1\}$, then we have the $0 - 1$ Knapsack or subset sum problem. That is, given a set $I = \{a_1, a_2, ..., a_n\}$ and s (all positive integers) find a subset of the set I which have sum s.
- The decisional form of the Knapsack problem is NP-complete, so it is thought to be quite hard.

Some Applications.

- If we restrict the solutions $x_j \in \{0, 1\}$, then we have the $0 - 1$ Knapsack or subset sum problem. That is, given a set $I = \{a_1, a_2, ..., a_n\}$ and s (all positive integers) find a subset of the set I which have sum s.
- The decisional form of the Knapsack problem is NP-complete, so it is thought to be quite hard.
- NP-complete means, if there is a polynomial algorithm which solves the problem, then there is also a polynomial algorithm for the problems in the NP-class.

Some Applications.

- If we restrict the solutions $x_j \in \{0, 1\}$, then we have the $0 - 1$ Knapsack or subset sum problem. That is, given a set $I = \{a_1, a_2, ..., a_n\}$ and s (all positive integers) find a subset of the set I which have sum s.
- The decisional form of the Knapsack problem is NP-complete, so it is thought to be quite hard.
- NP-complete means, if there is a polynomial algorithm which solves the problem, then there is also a polynomial algorithm for the problems in the NP-class.
- If we have an oracle that solves the desicional knapsack problem, then after n calls to the oracle we can solve the computational knapsack problem.

Some Applications.

- The knapsack cryptography uses a public set of weights $a_1, ..., a_n$ and in order Bob to encode a message consisting from the bits $(x_1, ..., x_n)$ he calculate $\sum_{j=1}^{n} a_j x_j = s$. Then Bob transmits s. An enemy would see s, so he has to solve a knapsack instance in order to decode the message.

Some Applications.

- The knapsack cryptography uses a public set of weights $a_1, ..., a_n$ and in order Bob to encode a message consisting from the bits $(x_1, ..., x_n)$ he calculate $\sum_{j=1}^{n} a_j x_j = s$. Then Bob transmits s. An enemy would see s, so he has to solve a knapsack instance in order to decode the message.
- The first cryptographic system based on the Knapsack problem is given by Merkle and Hellman in 1978 and was broken by Shamir after three years. Shamir uses Lattice reduction methods in order to break the scheme.

LLL reduction algorithm

- In 1982, A.Lenstra, H.Lenstra and Lovasz published in their landmark paper the LLL-algorithm, which is a basis reduction algorithm for lattices, based on Hermite's inequality. The aim of LLL algorithm is to find a short vector that approximates the shortest nonzero vector of the lattice. The LLL algorithm runs in polynomial time as a function of the rank of the lattice.

LLL reduction algorithm.

- Let a basis $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$ of a lattice $L = \mathbf{Z}\mathbf{b}_1 + \cdots + \mathbf{Z}\mathbf{b}_n \subset \mathbf{R}^k$. We associate the Gramm-Schmidt orthogonalization vectors $\{\mathbf{g}_1, ..., \mathbf{g}_n\}$ defined by the relations

$$\mathbf{g}_1 = \mathbf{b}_1, \ \mathbf{g}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij}\mathbf{b}_j,$$

where the GSO coefficients

$$\mu_{ij} = \frac{\mathbf{b}_i \cdot \mathbf{g}_j}{B_j^2}, \quad B_j = ||\mathbf{g}_j||.$$

The GSO process produce vectors that form an orthogonal basis of $\mathbf{R}^k$.

LLL reduction algorithm.

- An LLL-basis has two characteristics.
  i. Is size reduced, that is $|\mu_{ij}| < 1/2,\ 1 \leq j < i < n$ and
  ii. The vectors of the basis are almost orthogonal to each other. This, is translated to the following (Lovasz) relation $\delta||\mathbf{g}_i||^2 \leq ||\mathbf{g}_{i+1} + \mu_{i+1,i}\mathbf{g}_i||^2$ for some $\delta \in (1/4, 1)$.

LLL reduction algorithm.

- An LLL-basis has two characteristics.
  i. Is size reduced, that is $|\mu_{ij}| < 1/2,\ 1 \leq j < i < n$ and
  ii. The vectors of the basis are almost orthogonal to each other. This, is translated to the following (Lovasz) relation $\delta||\mathbf{g}_i||^2 \leq ||\mathbf{g}_{i+1} + \mu_{i+1,i}\mathbf{g}_i||^2$ for some $\delta \in (1/4, 1)$.
- The LLL-algorithm achieve to give us a small length vector, more specific
$$||\mathbf{b}_1|| \leq 2^{n-1} d(L)^{1/n},$$
where $d(L)$ is the discriminant of the latttice.

LLL reduction algorithm.

- An LLL-basis has two characteristics.
  i. Is size reduced, that is $|\mu_{ij}| < 1/2,\ 1 \leq j < i < n$ and
  ii. The vectors of the basis are almost orthogonal to each other. This, is translated to the following (Lovasz) relation $\delta||\mathbf{g}_i||^2 \leq ||\mathbf{g}_{i+1} + \mu_{i+1,i}\mathbf{g}_i||^2$ for some $\delta \in (1/4, 1)$.
- The LLL-algorithm achieve to give us a small length vector, more specific
$$||\mathbf{b}_1|| \leq 2^{n-1} d(L)^{1/n},$$
where $d(L)$ is the discriminant of the latttice.
- Some apllications are
  • to the factorization of polynomials over $\mathbf{Z}[x]$
  • to find integer relations between some real numbers $k_1, ..., k_n$.

LLL reduction algorithm.

- LLL algorithm consists from two basic subroutines $red(k, \ell)$ and $swap(k)$.

LLL reduction algorithm.

- LLL algorithm consists from two basic subroutines $red(k, \ell)$ and $swap(k)$.
- Let $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$ be a basis of our Lattice and $\{\mathbf{b}_1^*, ..., \mathbf{b}_n^*\}$ the Gramm-Schimdt Orhogonal vectors.

LLL reduction algorithm.

- LLL algorithm consists from two basic subroutines $red(k, \ell)$ and $swap(k)$.
- Let $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$ be a basis of our Lattice and $\{\mathbf{b}_1^*, ..., \mathbf{b}_n^*\}$ the Gramm-Schimdt Orhogonal vectors.
- $red(k, \ell)$ consists from the following steps.

  for $k = 1, 2, ..., \ell - 1$ do

  if $|\mu_{k\ell}| = |\frac{\mathbf{b}_k \cdot \mathbf{b}_\ell^*}{B_k}| < 1/2$ where $B_k = \mathbf{b}_k^{*2}$ then set

  $r = \lceil \mu_{k\ell} \rfloor, \quad \mathbf{b}_k \leftarrow \mathbf{b}_k - r\mathbf{b}_\ell$

  $\mu_{kj} \leftarrow \mu_{kj} - r\mu_{\ell j}$

LLL reduction algorithm.

- The second subroutine is called $swap(k)$

LLL reduction algorithm.

- The second subroutine is called *swap*(k)
- this make a swap between two vectors $\mathbf{b}_k, \mathbf{b}_{k+1}$ if the Lovasz condition (for $\delta = 3/4$) does not hold, that is :

$$\frac{3}{4}|\mathbf{b}^*_{k-1}|^2 > |\mathbf{b}^*_k + \mu_{k,k-1}\mathbf{b}^*_{k-1}|^2$$

LLL reduction algorithm.

- The second subroutine is called $swap(k)$
- this make a swap between two vectors $\mathbf{b}_k, \mathbf{b}_{k+1}$ if the Lovasz condition (for $\delta = 3/4$) does not hold, that is :

$$\frac{3}{4}|\mathbf{b}^*_{k-1}|^2 > |\mathbf{b}^*_k + \mu_{k,k-1}\mathbf{b}^*_{k-1}|^2$$

- Now LLL alogirthm excute the following loop
  set $k = 2$
  while $k \leq n$ do
    $red(k, \ell)$ $\ell = k - 2, ..., 1$ and then call $swap(k)$.

Solutions of linear diophantine equations.

- The method which we usually apply in order to find small solutions of the linear equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0 \tag{2}$$

is the closest vector problem (CVP).

Solutions of linear diophantine equations.

- The method which we usually apply in order to find small solutions of the linear equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0 \tag{2}$$

  is the closest vector problem (CVP).
- Let L be the lattice generated by integer solutions of the homogeneous linear equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = 0.$$

Solutions of linear diophantine equations.

- The method which we usually apply in order to find small solutions of the linear equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0 \tag{2}$$

  is the closest vector problem (CVP).
- Let L be the lattice generated by integer solutions of the homogeneous linear equation

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = 0.$$

- This is a lattice of dimension $n-1$ since $L = \mathbf{Z}\mathbf{b}_1 + \cdots \mathbf{Z}\mathbf{b}_{n-1}$ where $\mathbf{b}_j$ are the solutions.

Solutions of linear diophantine equations.

- Let **y** be a solution of

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0.$$

Solutions of linear diophantine equations.

- Let $\mathbf{y}$ be a solution of

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0.$$

- We solve the CVP instance $CVP(L, \mathbf{y})$, that is we apply algorithms which try to find the closest vector of L, say $\mathbf{t}$, to the vector $\mathbf{y}$

Solutions of linear diophantine equations.

- Let $\mathbf{y}$ be a solution of

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0.$$

- We solve the CVP instance $CVP(L, \mathbf{y})$, that is we apply algorithms which try to find the closest vector of L, say $\mathbf{t}$, to the vector $\mathbf{y}$
- Then $\mathbf{x} = \mathbf{y} - \mathbf{t}$ is an integer solution of (2) and has small absolute value $||\mathbf{x}||$.

Solutions of linear diophantine equations.

- Let $\mathbf{y}$ be a solution of

$$f(x_1, ..., x_n) = \sum_{j=1}^{n} a_j x_j = a_0.$$

- We solve the CVP instance $CVP(L, \mathbf{y})$, that is we apply algorithms which try to find the closest vector of L, say $\mathbf{t}$, to the vector $\mathbf{y}$
- Then $\mathbf{x} = \mathbf{y} - \mathbf{t}$ is an integer solution of (2) and has small absolute value $||\mathbf{x}||$.
- Implementations of CVP can be found in fplll and in Magma.

Solutions of linear diophantine equations.

- We set up some notation.

Solutions of linear diophantine equations.

- We set up some notation.
- Let $X_j,\ j = 1, 2, ..., n$ be positive integers and B be the lattice generated by the following vectors.

$$\{\mathbf{b}_j : \mathbf{b}_j = (0, ..., \frac{1}{X_j}, 0, ..., 0, a_j) \in \mathbf{R}^{n+2}, j = 1, 2, ..., n\}$$

Solutions of linear diophantine equations.

- We set up some notation.
- Let $X_j,\ j = 1, 2, ..., n$ be positive integers and B be the lattice generated by the following vectors.

$$\{\mathbf{b}_j : \mathbf{b}_j = (0, ..., \frac{1}{X_j}, 0, ..., 0, a_j) \in \mathbf{R}^{n+2}, j = 1, 2, ..., n\}$$

- We apply LLL and we get the following vectors

$$B' = \{\mathbf{b}_1^{'}, \mathbf{b}_2^{'}, ..., \mathbf{b}_n^{'}\}.$$

Solutions of linear diophantine equations.

- We set up some notation.
- Let $X_j,\ j = 1, 2, ..., n$ be positive integers and B be the lattice generated by the following vectors.

$$\{\mathbf{b}_j : \mathbf{b}_j = (0, ..., \frac{1}{X_j}, 0, ..., 0, a_j) \in \mathbf{R}^{n+2}, j = 1, 2, ..., n\}$$

- We apply LLL and we get the following vectors

$$B' = \{\mathbf{b}_1^{'}, \mathbf{b}_2^{'}, ..., \mathbf{b}_n^{'}\}.$$

- Finally, using the Gramm-Schimdt orthogonalization process to the LLL reduced basis B', we get the set $G = \{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_n\}$. We define $B_j = ||\mathbf{g}_j||$.

Solutions of linear diophantine equations.

- We shall prove the following Theorem.
  Let $\gcd(a_1, ..., a_n) = 1$. If the following two assumptions hold
  $\mathrm{A}_1.\ (a_n X_n)^2 + (a_j X_j)^2 < \frac{1}{2^{n+1}}(X_n X_j)^2,\ j = 1, 2, ..., n-1,$
  $\mathrm{A}_2.\ \left\lceil \frac{a_0}{B_n^2} \right\rceil = a_0,$
  then we can find in polynomial time, an integer solution $(x_1, ..., x_n)$ of the equation $\sum_{j=1}^{n} a_j x_j = a_0$, such that $|x_j| < c(n) X_j \prod_{i=1}^{n} X_i,\ \ j = 1, 2, ..., n,\ \ c(n) = \sqrt{3}(1.25)^{(n-1)/2}$

Solutions of linear diophantine equations.

- We shall prove the following Theorem.
  Let $\gcd(a_1, ..., a_n) = 1$. If the following two assumptions hold
  $\mathrm{A}_1.\ (a_n X_n)^2 + (a_j X_j)^2 < \frac{1}{2^{n+1}}(X_n X_j)^2,\ j = 1, 2, ..., n-1,$
  $\mathrm{A}_2.\ \left\lceil \frac{a_0}{B_n^2} \right\rceil = a_0,$
  then we can find in polynomial time, an integer solution $(x_1, ..., x_n)$ of the equation $\sum_{j=1}^{n} a_j x_j = a_0$, such that $|x_j| < c(n) X_j \prod_{i=1}^{n} X_i,\ \ j = 1, 2, ..., n,\ \ c(n) = \sqrt{3}(1.25)^{(n-1)/2}$
- Also, the proof of the Theorem provide us with an algorithm.

The Algorithm

- There are two basic steps, first LLL to the lattice B, and second size reduction to a new lattice of the form $M = B + \mathbf{Zb}$.

The Algorithm

- There are two basic steps, first LLL to the lattice B, and second size reduction to a new lattice of the form $M = B + \mathbf{Zb}$.
- Say that we have the linear diophantine equation

$$a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5 = a_0.$$

The Algorithm

- There are two basic steps, first LLL to the lattice B, and second size reduction to a new lattice of the form $M = B + \mathbf{Zb}$.
- Say that we have the linear diophantine equation

$$a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5 = a_0.$$

- We apply LLL to the rows of the Lattice given by the matrix

$$M = \begin{bmatrix} \frac{1}{X_1} & 0 & 0 & 0 & 0 & 0 & a_1 \\ 0 & \frac{1}{X_2} & 0 & 0 & 0 & 0 & a_2 \\ 0 & 0 & \frac{1}{X_3} & 0 & 0 & 0 & a_3 \\ 0 & 0 & 0 & \frac{1}{X_4} & 0 & 0 & a_4 \\ 0 & 0 & 0 & 0 & \frac{1}{X_5} & 0 & a_5 \end{bmatrix}$$

The Algorithm

- Then to the new reduced basis we add the row $(0, 0, 0, 0, 0, 1, -a_0)$.

The Algorithm

- Then to the new reduced basis we add the row $(0, 0, 0, 0, 0, 1, -a_0)$.
- Then apply size reduction to the rows, that is (here $n = 5$)

$$row(n+1) \leftarrow row(n+1) - \lceil \mu_{n+1,j} \rfloor row(j)$$

$$\mu_{ij} = \frac{\mathbf{b}_i' \cdot \mathbf{g}_j}{B_j^2}, \quad B_j = ||\mathbf{g}_j||.$$

first for $j = n$ and then $j = 1, 2, ..., n-1$.

The Algorithm

- Finally, we multiply the $i-$ entry of the last vector with X_i.

The Algorithm

- Finally, we multiply the $i-$ entry of the last vector with X_i.
- The resulting vector gives a solution of the diophantine equation which satisfies the bound of our theorem.

The Algorithm

- Finally, we multiply the $i-$ entry of the last vector with X_i.
- The resulting vector gives a solution of the diophantine equation which satisfies the bound of our theorem.
- But, at least experimentally satisfies the better bound $|x_j| < X_j$.

Examples

- Let

$$84 \cdot 10^5 x_1 + 4 \cdot 10^6 x_2 + 15688 x_3 + 6720 x_4 + 15 x_5 = 371065262.$$

This is example 1 of Aardal, Hurkens, Lenstra

Ref : Aardal, K.; Hurkens, C.; Lenstra, A..; Solving a linear Diophantine equation with lower and upper bounds on the variables. Integer programming and combinatorial optimization, LNCS, **1412**.

and they get the solution $\mathbf{x} = (36, 17, 39, 8, -22)$, with $||\mathbf{x}|| \simeq 60.44$. Assumption A_1 is fulfilled if

$$\max_{1 \le j \le 4} |a_j| < \frac{1}{8} X_5, \ |a_5| < \frac{1}{8} \max_{1 \le j \le 5} X_j.$$

So it is enough to choose

$$X = X_1 = \cdots = X_5 = 8 \cdot 84 \cdot 10^5 + 1.$$

Examples

- We consider the matrix

$$M = \begin{bmatrix} \frac{1}{67200001} & 0 & 0 & 0 & 0 & 0 & 8400000 \\ 0 & \frac{1}{67200001} & 0 & 0 & 0 & 0 & 4000000 \\ 0 & 0 & \frac{1}{67200001} & 0 & 0 & 0 & 15688 \\ 0 & 0 & 0 & \frac{1}{67200001} & 0 & 0 & 6720 \\ 0 & 0 & 0 & 0 & \frac{1}{67200001} & 0 & 15 \end{bmatrix}$$

Examples

- Applying LLL to the rows of M we get $M_{LLL} =$

$$\begin{bmatrix} -\frac{10}{67200001} & \frac{21}{67200001} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{15}{67200001} & -\frac{35}{67200001} & -\frac{8}{67200001} & 0 & 0 \\ \frac{1}{67200001} & -\frac{2}{67200001} & -\frac{25}{67200001} & -\frac{1}{67200001} & -\frac{72}{67200001} & 0 & 0 \\ \frac{5}{67200001} & -\frac{10}{67200001} & -\frac{95}{67200001} & -\frac{76}{67200001} & \frac{72}{67200001} & 0 & 0 \\ \frac{2}{67200001} & -\frac{4}{67200001} & -\frac{42}{67200001} & -\frac{21}{67200001} & \frac{1}{67200001} & 0 & -1 \end{bmatrix}$$

Examples

- We add a new row $\mathbf{b}_6 = (0, 0, 0, 0, 0, 1, -a_0)$.

$$\begin{bmatrix} -\frac{10}{67200001} & \frac{21}{67200001} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{15}{67200001} & -\frac{35}{67200001} & -\frac{8}{67200001} & 0 & 0 \\ \frac{1}{67200001} & -\frac{2}{67200001} & -\frac{25}{67200001} & -\frac{1}{67200001} & -\frac{72}{67200001} & 0 & 0 \\ \frac{5}{67200001} & -\frac{10}{67200001} & -\frac{95}{67200001} & -\frac{76}{67200001} & \frac{72}{67200001} & 0 & 0 \\ \frac{2}{67200001} & -\frac{4}{67200001} & -\frac{42}{67200001} & -\frac{21}{67200001} & \frac{1}{67200001} & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -a_0 \end{bmatrix}$$

Examples

- Then applying size reduction to the previous lattice i.e.

$$row(6) \leftarrow row(6) - \lceil \mu_{6,j} \rfloor row(j),$$

first for $j = 6$ and then for $j = 1, 2, 3, 4, 5$, we shall get

Examples

- $\hat{M}_{LLL} =$

$$\begin{bmatrix} -\frac{10}{67200001} & \frac{21}{67200001} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{15}{67200001} & -\frac{35}{67200001} & -\frac{8}{67200001} & 0 & 0 \\ \frac{1}{67200001} & -\frac{2}{67200001} & -\frac{25}{67200001} & -\frac{1}{67200001} & -\frac{72}{67200001} & 0 & 0 \\ \frac{5}{67200001} & -\frac{10}{67200001} & -\frac{95}{67200001} & -\frac{76}{67200001} & \frac{72}{67200001} & 0 & 0 \\ \frac{2}{67200001} & -\frac{4}{67200001} & -\frac{42}{67200001} & -\frac{21}{67200001} & \frac{1}{67200001} & 0 & -1 \\ \frac{36}{67200001} & \frac{17}{67200001} & \frac{39}{67200001} & \frac{8}{67200001} & -\frac{2}{6109091} & 1 & 0 \end{bmatrix}$$

Examples

- $\hat{M}_{LLL} =$

$$\begin{bmatrix} -\frac{10}{67200001} & \frac{21}{67200001} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{15}{67200001} & -\frac{35}{67200001} & -\frac{8}{67200001} & 0 & 0 \\ \frac{1}{67200001} & -\frac{2}{67200001} & -\frac{25}{67200001} & -\frac{1}{67200001} & -\frac{72}{67200001} & 0 & 0 \\ \frac{5}{67200001} & -\frac{10}{67200001} & -\frac{95}{67200001} & -\frac{76}{67200001} & \frac{72}{67200001} & 0 & 0 \\ \frac{2}{67200001} & -\frac{4}{67200001} & -\frac{42}{67200001} & -\frac{21}{67200001} & \frac{1}{67200001} & 0 & -1 \\ \frac{36}{67200001} & \frac{17}{67200001} & \frac{39}{67200001} & \frac{8}{67200001} & -\frac{2}{6109091} & 1 & 0 \end{bmatrix}$$

- We take the 6th row and multiply each entry with X, then we shall get the vector $\mathbf{x} = (36, 17, 39, 8, -22)$. Which is the same as was found in Lenstra's paper.

Examples

- Now if we choose $X_1 = 30, X_2 = X_3 = X_4 = X_5 = 50$ and we repeat the previous procedure we shall get the solution $\mathbf{y} = (26, 38, 39, 8, -22)$. This solution has euclidean length $\simeq 64.72$. Note that is larger than the previous solution $\mathbf{x} = (36, 17, 39, 8, -22)$, with $||x|| \simeq 60.44$. It has the advantage that satisfy our constraints (the solution $\mathbf{x}$ does not).

Examples

- We consider now $n = 50$

$\{a_j\}_j = [872934629013064, 362643350651979, 231593889792433, 1084529488472651, 152647947850799,$
$739407904067188, 1078361055147110, 522287723336618, 1048278073142822, 71464720981315,$
$1026144865997912, 401128969656441, 1104125375426692, 223040948030783, 259134135114376,$
$477165086702863, 693696459173357, 956101007737750, 1076391779531258, 887808907972169,$
$154289043341408, 1123813906929138, 100640784930380, 1028038257417354, 126747913149526,$
$345001039716371, 173180910604612, 376756743710801, 462057825850822, 105084485099476,$
$193285152829384, 663950233902816, 1005024177016821, 350981819196027, 1049577315489835,$
$455051495653072, 1014366278972062, 905067265314795, 972603957926899, 1110054606397627,$
$768533772552959, 798515502008744, 705587377794293, 64248048456242, 771519628719865,$
$190006526706907, 481482852515889, 916067763534188, 768875611228651, 666640039086558]$

$a_0 = 17297404087862459$

Examples

- We consider now $n = 50$

$\{a_j\}_j = [872934629013064, 362643350651979, 231593889792433, 1084529488472651, 152647947850799, 739407904067188, 1078361055147110, 522287723336618, 1048278073142822, 71464720981315, 1026144865997912, 401128969656441, 1104125375426692, 223040948030783, 259134135114376, 477165086702863, 693696459173357, 956101007737750, 1076391779531258, 887808907972169, 154289043341408, 1123813906929138, 100640784930380, 1028038257417354, 126747913149526, 345001039716371, 173180910604612, 376756743710801, 462057825850822, 105084485099476, 193285152829384, 663950233902816, 1005024177016821, 350981819196027, 1049577315489835, 455051495653072, 1014366278972062, 905067265314795, 972603957926899, 1110054606397627, 768533772552959, 798515502008744, 705587377794293, 64248048456242, 771519628719865, 190006526706907, 481482852515889, 916067763534188, 768875611228651, 666640039086558]$

$a_0 = 17297404087862459$

- Using our algorithm with $X_j = 3$ $(1 \leq j \leq n)$ we get

$\mathbf{x}_1 = [1, 0, 1, 2, 0, 0, 1, -1, 0, 0, 1, 2, -1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 2, 2, 0, 0, -1, 1, 1, -1, -1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0]$

with norm 6.3.

Examples

- We consider now $n = 50$

$\{a_j\}_j = [872934629013064, 362643350651979, 231593889792433, 1084529488472651, 152647947850799, 739407904067188, 1078361055147110, 522287723336618, 1048278073142822, 71464720981315, 1026144865997912, 401128969656441, 1104125375426692, 223040948030783, 259134135114376, 477165086702863, 693696459173357, 956101007737750, 1076391779531258, 887808907972169, 154289043341408, 1123813906929138, 100640784930380, 1028038257417354, 126747913149526, 345001039716371, 173180910604612, 376756743710801, 462057825850822, 105084485099476, 193285152829384, 663950233902816, 1005024177016821, 350981819196027, 1049577315489835, 455051495653072, 1014366278972062, 905067265314795, 972603957926899, 1110054606397627, 768533772552959, 798515502008744, 705587377794293, 64248048456242, 771519628719865, 190006526706907, 481482852515889, 916067763534188, 768875611228651, 666640039086558]$

$a_0 = 17297404087862459$

- Using our algorithm with $X_j = 3$ $(1 \leq j \leq n)$ we get

$\mathbf{x}_1 = [1, 0, 1, 2, 0, 0, 1, -1, 0, 0, 1, 2, -1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 2, 2, 0, 0, -1, 1, 1, -1, -1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0]$

with norm 6.3.

- Now using CVP method (as implemented in fplll) we manage to get the vector

$\mathbf{x}_4 = [0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 2, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, -1, 1, 0, 0, 1, 0, 0, 0, 2, 1, 0, 0, 0, 1, 0, 0, 1]$

which has norm 5.

Examples

- We noticed that, if the target vector has large length then fplll provide us with a large solution $\mathbf{x}$ (but smaller than the length of target vector).

Examples

- We noticed that, if the target vector has large length then fplll provide us with a large solution $\mathbf{x}$ (but smaller than the length of target vector).
- That is the CVP solvers are "sensitive" to the choice of the target vector. In order to get the previous solution $\mathbf{x}_4$ we used as target vector, the vector $\mathbf{x}_1$ from the application of our method.

Examples

- We noticed that, if the target vector has large length then fplll provide us with a large solution $\mathbf{x}$ (but smaller than the length of target vector).
- That is the CVP solvers are "sensitive" to the choice of the target vector. In order to get the previous solution $\mathbf{x}_4$ we used as target vector, the vector $\mathbf{x}_1$ from the application of our method.
- So it seems that a nice strategy is to combine the CVP solvers and our algorithm (which shall give us the target vector) in order to get a small solution.

Examples

- We noticed that, if the target vector has large length then fplll provide us with a large solution $\mathbf{x}$ (but smaller than the length of target vector).
- That is the CVP solvers are "sensitive" to the choice of the target vector. In order to get the previous solution $\mathbf{x}_4$ we used as target vector, the vector $\mathbf{x}_1$ from the application of our method.
- So it seems that a nice strategy is to combine the CVP solvers and our algorithm (which shall give us the target vector) in order to get a small solution.
- In case we have large n, say $n \geq 80$ then the CVP solver of fpLLL(and Magma) is slow.

Examples

- We noticed that, if the target vector has large length then fplll provide us with a large solution $\mathbf{x}$ (but smaller than the length of target vector).
- That is the CVP solvers are "sensitive" to the choice of the target vector. In order to get the previous solution $\mathbf{x}_4$ we used as target vector, the vector $\mathbf{x}_1$ from the application of our method.
- So it seems that a nice strategy is to combine the CVP solvers and our algorithm (which shall give us the target vector) in order to get a small solution.
- In case we have large n, say $n \geq 80$ then the CVP solver of fpLLL(and Magma) is slow.
- Thank you!