Multidimensional Subset Sum Problem

Vladimir Kolesnikov

Abstract

This thesis explores new modifications to the successful *LLL* approach to solving the *Subset Sum problem*. This work is an optimization of the matrix representation of an instance. Traditionally, the basis matrix contained only one column with set elements and the sum. In this thesis we suggest having several data columns (thus introducing multidimensionality). This allows us to reduce the size of coulmn entries which changes the complexity of the problem. Splitting the data into multiple columns greatly simplifies the task of solving the *Subset Sum problem*. However, other problems arise when we try to generate multiple columns. Here we try to find the optimal way to do the split and present the results. Our main goal was to try to solve the current hardest *Subset Sum problem* instances: the ones with density slightly greater than 1. Dramatic improvement in the rate of success was observed (up to 1500 %) compared to one-dimensional implementations.

1 Overview

Subset Sum is a well-known NP-complete problem [5].

It is is defined as follows:

Definition 1 *Given a vector of positive integers $\vec{a} = (a_1, a_2, ..., a_n)$ and a positive integer M, find a {0,1}-vector $\vec{x} = (x_1, x_2, ..., x_n)$, such that*

$$\sum_{i=1}^{n} a_i \cdot x_i = M \tag{1}$$

In other words, the *Subset Sum problem* is:

Given a set of positive integers S and a positive integer M, find a subset of S, the sum of the elements of which is equal to M.

The *Subset Sum problem* is the basis for several public key cryptography systems. They are based on the intractability of finding a solution to (1) even when the solution is known to exist. In such systems, each user publishes a vector $\vec{a}$ of a_i. A plaintext message consisting of an integer vector $\vec{v} = \{x_0, x_1, ..., x_{n-1}\}$ where $x_i = \{0, 1\}$ is encrypted by setting

$E(\vec{v}) = \sum x_i \cdot a_i$.

The elements a_i are chosen in such a way that the equation is easily solved if certain secret *trapdoor* information is known. The exact nature of this information depends on the particular system. A general property of *Subset Sum* (or *knapsack*) based public key cryptosystems is that encryption is easy: all you have to do is add. [3]

The *LLL* algorithm is based on the lattice basis reduction algorithm, proposed by Lenstra, Lenstra and Lovasz. It also uses techniques of linear algebra, specifically the Gram-Schmidt orthogonalization process. The authors of the original algorithm developed a notion of *LLL* reduction and an algorithm to *LLL* reduce a basis of integer lattice. The idea of the algorithm is to transform a vector basis given by an initial integer matrix into another integer vector basis, such that vectors in the latter are as short as possible. The latter basis must also be equivalent to the original one (i.e., span the same space). It appears that many NP complete problems, including the

Subset Sum problem, are representable as problems of finding short integer vectors in a vector space, called integer lattices. Here the LLL algorithm comes into play: it looks for short vectors by shortening the basis vectors.

The LLL-based approach is the most famous and well-known approach to solving the *Subset Sum problem*. In fact it was one of the reasons for the fall of Subset Sum based public-key cryptosystems. (See [9] and [1] for surveys in this field). Almost all of these cryptosystems have been shown to be insecure. The majority of the attacks exploited specific constructions of the relevant cryptosystems. In addition, two independent algorithms have been proposed, one by Brickell [2] and the other by Lagarias and Odlyzko [6] which show that almost all *Subset Sum problems* of low density can be solved in polynomial time, where density is defined as the ratio between the size of the set, and the bit size of the largest of the set elements. The Brickell and Lagarias-Odlyzko attacks reduce the *Subset Sum problem* to the problem of finding the Euclidian-norm shortest nonzero vector in a lattice (the job of the LLL algorithm). It is a very hard problem in general. The theoretical worst case bounds for the LLL algorithm and its variants are not encouraging. However, these techniques tend to perform much better in practice than in theory [7].

A lot of work has been done in this area, and many different approaches tried. One can point out two main research directions in applying LLL to solving *Subset Sum problems*. One is the improvement of the LLL algorithm itself. Many different heuristics were invented, and different definitions of a property being reduced were introduced (e.g. Korkine- Zolotarev reduction, presented in [11]). The other approach is to find better ways to represent a *Subset Sum problem* instance as a lattice basis reduction instance. This work definitely belongs to the second direction. There also were attempts to combine improved ways to represent the lattice with heuristic variants of lattice basis reduction implementations, for example [11]. In this thesis, a standard reduction algorithm was used, and the results observed are compared to ones obtained from experiments with similar quality LLL implementations (mainly to performance of $LLLFP$, presented in [11]).

Part I

The application of LLL algorithm to solving Subset Sum problem

2 Definitions

Let R^n and Z^n denote n-dimensional spaces over real numbers and integers, respectively. Consider space R^n.

Definition 2 *A vector is an ordered pair of points (P, Q) which we call a vector from P to Q, and denote $\vec{PQ}$. Define $\vec{PQ} = \vec{RS}$ if they have the same length and direction.*

Definition 3 *A vector space is a set, whose elements are called vectors, together with two operations. The first operation, called vector addition, assigns to each pair of vectors $\vec{A}$ and $\vec{B}$ a vector denoted $\vec{A} + \vec{B}$, called their sum. The second operation, called scalar multiplication, assigns to each vector $\vec{A}$ and each number r a vector denoted by $r\vec{A}$. The two operations are required to have the following properties:*

1. $\vec{A} + \vec{B} = \vec{B} + \vec{A}$
2. $(\vec{A} + \vec{B}) + \vec{C} = \vec{A} + (\vec{B} + \vec{C})$
3. $\exists! \vec{0}$, *such that* $\vec{A} + \vec{0} = \vec{A}, \forall \vec{A}$
4. $\forall \vec{A} \exists (\vec{-A})$, *such that* $\vec{A} + (\vec{-A}) = \vec{0}$
5. $r(\vec{A} + \vec{B}) = r\vec{A} + r\vec{B}$
6. $(r + s)\vec{A} = r\vec{A} + s\vec{A}$
7. $(rs)\vec{A} = r(s\vec{A})$
8. $\forall \vec{A}, 1\vec{A} = \vec{A}$

Let R^n be the n-dimensional real vector space with the ordinary inner product $\langle \cdot, \cdot \rangle$ and Euclidean length $|y| = \sqrt[2]{\langle y, y \rangle}$.

Definition 4 *A discrete, additive subgroup $L \in R^n$ is called a* lattice*.*

Every lattice is generated by some set of linearly independent vectors $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m \in L$ that is called a basis of L,

$$L = \{t_1\vec{b}_1 + t_2\vec{b}_2 + ... + t_m\vec{b}_m \ \mid \ t_1, t_2, ..., t_m \in Z\}$$

Let $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m \in R^n$ be a basis of lattice L. Then $\bar{b}_1, \bar{b}_2, ..., \bar{b}_m$ is also a basis of L if and only if there exists a matrix T (of size $n \times n$), such that

$$[\vec{b}_1, \vec{b}_2, ..., \vec{b}_m] = [\bar{b}_1, \bar{b}_2, ..., \bar{b}_m]T$$

Here $[\vec{b}_1, \vec{b}_2, ..., \vec{b}_m]$ denotes the $n \cdot m$ matrix in $M_{n,m}(R)$ with column vectors $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m$.

3 How to solve the problem

There are several ways to solve a *Subset Sum problem* using the *LLL* algorithm (the original approach by Lagarias and Odlyzko in [6], Radziszowski's in [10], Schnorr's in [11], etc). They all have the same idea. We need to represent an instance of the problem as a basis of integer vectors in an integer lattice in a way that the 0,1-solution vector is present in the vector space defined by the basis (it may be somewhat transformed). The vector we are looking for is very short. By applying *LLL* algorithm to the lattice representing the *Subset Sum problem* instance, we shorten the vectors. The probability that the short vector we are looking for will appear among the basis vectors is high.

Let us consider a lattice, suggested by Claus Schnorr and Michael Euchner in [11]. Consider a *Subset Sum problem* according to (1). Let us associate with integers a_i and M the following basis

$$\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1} \in Z^{n+2}$$

$$\vec{b}_1 = (2, 0, 0, ..., 0, na_1, 0)$$

$$
\begin{aligned}
\vec{b}_2 &= (0, 2, 0, ..., 0, na_2, 0) \\
\vec{b}_3 &= (0, 0, 2, ..., 0, na_3, 0) \\
&\vdots \\
\vec{b}_n &= (0, 0, 0, ..., 2, na_n, 0) \\
\vec{b}_{n+1} &= (1, 1, 1, ..., 1, nM, 1)
\end{aligned} \tag{2}
$$

Every lattice vector $\vec{z} = (z_1, z_2, ..., z_{n+2}) \in L(\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1})$ that satisfies

$$
|z_{n+2}| = 1, z_{n+1} = 0, z_1, z_2, ..., z_n \in \{1, -1\} \tag{3}
$$

yields the following solution for the *Subset Sum problem*

$$
x_i = \frac{|z_i - z_{n+2}|}{2}, for\ i = 1, 2, ...n \tag{4}
$$

All we need is an oracle that could transform the given basis into an equivalent basis of short vectors.

4 Lattice basis reduction

The goal of lattice basis reduction is to transform a given lattice basis into a basis that consists of short vectors or, equivalently, into a basis consisting of vectors that are pairwise nearly orthogonal.

With an ordered lattice basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m$ we associate the Gram-Schmidt orthogonalization $\hat{b}_1, \hat{b}_2, ..., \hat{b}_m$ together with the Gram-Schmidt coefficients

$$
\mu_{i,j} = \frac{\langle \vec{b}_i, \hat{b}_j \rangle}{\langle \hat{b}_j, \hat{b}_j \rangle}
$$

by the recursion

$$
\hat{b}_1 = \vec{b}_1
$$

$$
\hat{b}_i = \vec{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \hat{b}_j
$$

We have $\mu_{i,j} = 0$ and $\mu_{i,i} = 1$ for $i < j$. The vectors $\hat{b}_1, \hat{b}_2, ..., \hat{b}_m$ are linearly independent, but they are not necessarily in the lattice. If the basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m$ is integral, i.e. $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m \in Z^n$, then the vectors $\hat{b}_1, \hat{b}_2, ..., \hat{b}_m$ and the coefficients $\mu_{i,j}$ are rational. We can write the above as

$$[\vec{b}_1, \vec{b}_2, ..., \vec{b}_m] = [\hat{b}_1, \hat{b}_2, ..., \hat{b}_m][\mu_{i,j}]^T$$

Definition 5 *An ordered basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m \in R^n$ is called size-reduced if*

$$|\mu_{i,j}| \leq \frac{1}{2} \quad for\ 1 \leq j < i \leq m\ .$$

Definition 6 *An individual vector $\vec{b}_i$ is called size-reduced if*

$$|\mu_{i,j}| \leq \frac{1}{2} \quad for\ 1 \leq j < i.$$

Definition 7 *An ordered basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m \in R^n$ is called LLL-reduced with d, where d is a constant $\frac{1}{4} < d < 1$, if it is size-reduced and if*

$$d|\hat{b}_{k-1}|^2 \leq |\hat{b}_k + \mu_{k,k-1}\hat{b}_{k-1}|^2 \ for\ k = 2, 3, ..., m$$

For practical purposes we are interested in a constant d that is close to 1, e.g. $d = 0.99$.

4.1 Meaning of size and LLL reduction

4.1.1 Meaning of size reduction

Recall that

$$\mu_{i,j} = \frac{\langle \vec{b}_i, \hat{b}_j \rangle}{\langle \hat{b}_j, \hat{b}_j \rangle}$$

It is known that $\cos(\widehat{\vec{A}, \vec{B}}) = \frac{\langle \vec{A}, \vec{B} \rangle}{|\vec{A}||\vec{B}|}$ Thus

$$\mu_{i,j} = \frac{\langle \vec{b}_i, \hat{b}_j \rangle}{|\vec{b}_i||\hat{b}_j|} \cdot \frac{|\vec{b}_i||\hat{b}_j|}{\langle \hat{b}_j, \hat{b}_j \rangle} = \cos\theta \cdot \frac{|\vec{b}_i|}{|\hat{b}_j|}$$

Thus, size reduction imposes the condition that resulting vectors are short or nearly orthogonal.

4.1.2 Meaning of LLL reduction

Let $\lambda_1, \ldots, \lambda_m$ denote successive minima of lattice L, $\lambda_i = \lambda_i(L)$ is defined as the smallest radius r of a ball that is centered at the origin and which contains r linearly independent lattice vectors. Any LLL-reduced basis consists of relatively short lattice vectors.

Theorem [8] 1 *Every basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m$ that is LLL-reduced with d satisfies*

$$\alpha^{1-i} \leq \frac{|\vec{b}_i|^2}{\lambda_i^2} \leq \alpha^{m-1} \ , \ i = 1, 2, ..., m, \alpha = (d - \frac{1}{4})^{-1}$$

Thus, the property of a basis being LLL-reduced imposes theoretical limits on the lengths of vectors. In practice, the performance of the lattice basis reduction algorithm based on LLL-reduction is much better that in theory.

4.2 Lattice reduction algorithms

4.2.1 Algorithm for size reduction of the basis vector b_k

```
INPUT                 b_1, b_2, ..., b_m ∈ Z^n (a lattice basis)
                      μ_{i,j} for 1 ≤ j < i ≤ m (Gram-Schmidt coefficients)
   FOR j = k-1, ..., 1 DO
      IF |μ_{k,j}| > 1/2 THEN
            b_k := b_k − ⌈μ_{k,j}⌋ b_j
            FOR i = 1, 2, ..., m DO
                  μ_{k,i} := μ_{k,i} − ⌈μ_{k,j}⌋ μ_{k,i}
            END
      END IF
END
```

OUTPUT $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m$ (basis where $\vec{b}_k$ is size-reduced)
$\mu_{i,j}$ for $1 \leq j < i \leq m$

Here $\lceil \quad \rfloor$ denotes the nearest integer. We obtain a size-reduced basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m$ by size-reducing each vector individually. Size-reducing the vector $\vec{b}_k$ does not affect the size-reduction of other vectors [11].

4.2.2 Algorithm for LLL reduction

The following is the original algorithm, presented in [8].
INPUT $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m \in Z^n$(a lattice basis), d with $\frac{1}{4} < d < 1$.
(initialization) k:=2 (k is the stage)
compute the Gram-Schmidt coefficients $\mu_{k,j}$ for
$1 \leq j < i \leq m$ and $|\hat{b}_i|^2$ for $i = 1, ..., m$

WHILE $k \leq m$ DO
size-reduce the vector $\vec{b}_k$ and update $\mu_{k,j}$ for $j = 1, ..., k-1$
IF $d|\hat{b}_{k-1}|^2 > |\hat{b}_k|^2 + \mu_{k,k-1}^2|\hat{b}_{k-1}|^2$
THEN
swap $\vec{b}_k$ and $\vec{b}_{k-1}$, k := max(k-1, 2)
ELSE
$k := k+1$
END IF
END while
OUTPUT $\vec{b}_1, \vec{b}_2, ..., \vec{b}_m$ (a basis that is LLL-reduced with d).

REMARKS [8]:

1. Upon entry of stage k the basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_{k-1}$ is LLL-reduced with d.
2. For every SWAP $\vec{b}_k \leftrightarrow \vec{b}_{k-1}$ we must update $|\hat{b}_k|^2$, $|\hat{b}_{k-1}|^2$ and the Gram-Schmidt coefficients.
3. Let B = max $(|\vec{b}_1|^2, ..., |\vec{b}_m|^2)$. The algorithm terminates after at most O($m^2 \log B$) iterations. It performs at most O($m^3 n \log B$) arithmetic operations on integers that are O($m \log B$) bits long.

5 Practical algorithms to solve the Subset Sum problem

The following algorithm was suggested by Schnorr and Euchner in [11].
INPUT $a_1, a_2, ..., a_n, M \in N$

1. Compute the basis (2)

2. Randomly permute $\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1}$ so that the permuted basis starts with the vectors $\vec{b}_i$, satisfying $b_{i,n+2} \neq 0$

3. LLL-reduce the basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1}$

4. IF some vector $(z_1, z_2, ..., z_{n+2})$ in the reduced basis satisfies (3) THEN OUTPUT $x_i = \frac{|z_i - z_{n+2}|}{2}$ for $i = 1, ..., n$ and STOP

5. (reduce pairs of basis vectors)

```
F := FALSE
Sort b_1, b_2, ..., b_{n+1} so that |b_1| <= |b_2| <= ... <= |b_2|
FOR j = 1, ..., n DO
    FOR k = 0, ..., j - 1 DO
        IF |b_j ± b_k| < |b_j|
            THEN [ b_j := b_j ± b_k,  F := TRUE]
        ENDIF
    END
END
IF F = TRUE THEN GOTO 5
```

6. Repeat steps 2-5 15-100 times.

Experiments show that sometimes, depending on the instance of the problem, step 5 may take a lot of CPU time. The GOTO operation may get called thousands of times. It is effective to introduce some top line, the upper limit on the number of times allowed to execute the operation GOTO 5. According to experiments, the optimum limit is 10. Even if all pairs of vectors were not reduced after 10 iterations of step 5, after calling LLL and returning to step 5, it will reduce all pairs in less then 10 iterations.

6 Performance of the LLL algorithm

The performance of the algorithm mostly depends on the density of the instance of the problem. Density is the ratio between the size of the set and the number of bits in the largest of its elements. Precisely, the density of a

Subset Sum problem (1) is defined as

$$density = \frac{n}{\log_2(max\ \{a_i : 1 \le i \le n\})} \tag{5}$$

Currently, the most difficult instances have density slightly larger than 1, i.e. density about $1 + \frac{\log_2(\frac{n}{2})}{n}$. It has been proven rigorously (for example in [4]) that for almost all subset sum problems with density less than 0.9408 the shortest non-zero vector in the associated lattice basis yields a solution of the *Subset Sum problem* (1).

The *LLL* based algorithm calculates almost immediately *Subset Sum problem* instances that are still possible to compute by brute force (backtracking) algorithms. For example, an algorithm, presented in [11] solves 100% of the instances of the problem with a set size 26 and the number of bits being 27 and 28. On average it takes 0.05 seconds of CPU time and 2 *LLL* iterations to solve such a problem. (Timing produced on 195 MHz SGI R10000)

Other results in the area are presented in [4],where the authors present two other matrix representations of the *Subset Sum problem* that yield the following results:

n	b	success rate
50	50	1.00
50	60	1.00
66	76	0.25
66	84	0.80
66	92	0.95
66	100	1.00
66	104	1.00
66	108	1.00
66	112	1.00

The results obtained by C. P. Schnorr and M. Euchner, presented in [11] are the following for the experiments with the original LLL implementation:

n	b	successes out of 20 attempts
50	42	10
50	46	6
50	50	12
50	54	15
50	58	17
50	62	20
58	41	15
58	47	3
58	53	1
58	58	3
58	63	6
58	69	12

The improved LLL algorithm implementation proposed in [11] combined with the same matrix representation strategy produced the following rates of success:

n	b	successes out of 20 attempts
50	42	19
50	46	20
50	50	19
50	54	20
50	58	20
50	62	20
58	41	20
58	47	17
58	53	15
58	58	16
58	63	20
58	69	20

This matrix representation allowed to solve almost all *Subset Sum problems* with densities < 0.9408.

Part II
Multidimensional Subset Sum algorithm

7 Definitions and purpose of having several dimensions

In previous approaches, in the matrix representation of a *Subset Sum problem* instance, only one column carried actual data. The rest of the matrix only ensured that the solutions supplied by the *LLL* algorithm that fit the solution criteria were indeed solutions of the problem. Only one column (dimension) was used. In this thesis, we suggest having several columns (dimensions) carry the data. Thus the word 'multidimensional' emerges in the context of the *LLL* algorithm.

The purpose of having several dimensions is simple. In the context of the *Subset Sum problem*, it allows us to have less information per column, thus changing the complexity of the problem. In our most successful implementation of the *Subset Sum problem*, we have two columns of data, each of which contains only part of the information contained in the set and the number M. The two columns together, however, represent it all.

8 Algorithm performance estimates for N dimensions

A number of experiments was run to determine the optimal number of dimensions. We generated random matrices similar to the one in (2) with the exception that we generated not one but n columns. For every size/bits combination, 20 instances were attempted, and the tables below represent the rate of success for different numbers of dimensions (up to 100 LLL calls were allowed). It is important to note that in this experiment all data columns were independent.

Dimensions = 1:

size/bits	30	35	40	45
30	100%	100%	100%	94%
40	100%	100%	100%	94%
50	100%	75%	72%	50%
60	72%	90%	72%	20%

Dimensions = 2:

size/bits	30	35	40	45
30	100%	100%	100%	100%
40	100%	100%	100%	100%
50	93%	93%	93%	93 %
60	27%	13%	13%	7 %

Dimensions = 3:

size/bits	30	35	40	45
30	100%	100%	100%	100%
40	100%	100%	100%	50%
50	93%	100%	100%	87%
60	33%	0%	20%	50%

Dimensions = 5:

size/bits	30	35	40	45
30	100%	93%	50%	0%
40	100%	73%	13%	100%
50	100%	7%	0%	0%
60	60%	0%	0%	0%

Dimensions = 10:

size/bits	30	35	40	45
30	0%	0%	0%	0%
40	0%	0%	0%	0%
50	0%	0%	0%	0%
60	0%	0%	0%	0%

Dimensions = 20:

size/bits	30	35	40	45
30	80%	13%	0%	0%
40	0%	0%	0%	0%
50	0%	0%	0%	0%
60	0%	0%	0%	0%

Dimensions = 30:

size/bits	30	35	40	45
30	100%	100%	100%	100%
40	60%	0%	0%	0%
50	0%	0%	0%	0%
60	0%	0%	0%	0%

Dimensions = 50:

size/bits	30	35	40	45
30	100%	100%	100%	100%
40	100%	100%	100%	100%
50	100%	100%	100%	100%
60	27%	0%	0%	7%

Dimensions = 100, 200:

size/bits	30	35	40	45
30	100%	100%	100%	100%
40	100%	100%	100%	100%
50	100%	100%	100%	100%
60	100%	100%	100%	100%

These experiments were run on a 195MHz SGI R10000. Timings varied from up to 15 seconds per instance in 1 dimensional case to less than 0.02 seconds per instance for all 100- and 200-dimensional bit/size combinations.

Unfortunately, the condition of the columns being independent is essential to the excellent performance of the 200-dimensional case. Simple duplication of columns of data will not work, nor will different manipulations of data that keep data columns linearly dependent.

9 Use of remainders

A good way to generate linearly independent columns of data given one column of data $\{a_i\}$ is the use of remainders and modulo arithmetic. We can select n different natural numbers $r_1, \ldots, r_n$, and construct n independent columns of data, such that each of the new columns contains remainders of division of the original column elements by r_i.

original column	new coulmn
a_1	$s_1 = a_1 \bmod r_j$
a_2	$s_2 = a_2 \bmod r_j$
a_3	$s_3 = a_3 \bmod r_j$
...	...
a_m	$s_m = a_m \bmod r_j$
M	$M^* = M \bmod r_j$

Obviously, if there exists a (0,1) vector $\vec{x}$, such that

$$\sum_{i=1}^{n} \vec{x}_i \cdot a_i = M, \tag{6}$$

it is true that

$$\sum_{i=1}^{n} \vec{x}_i \cdot s_i = M^* \pmod{r_j}, \tag{7}$$

The reverse implication takes place if r_j are relatively prime and $\prod_{i=1}^{n} r_i \geq M$ (Chinese Remainder Theorem).

However, using modulo arithmetic doesn't work well with the *LLL* algorithm because there is no big difference between vectors' sizes. Therefore we had to use "traditional" arithmetic. But in this case serious problems arise. Instead of M^* we must use

$$\hat{M} = M \; mod \; r_j + k_j \cdot r_j \tag{8}$$

where k_j are unknown. We have to guess them.

The most difficult instances of the *Subset Sum problem* have a ratio of ones to zeros equal to 1. These are the problems that are addressed in [4], and [11]. Notice that this is another kind of density, not the one defined by (5). In this thesis we also try to solve problems with zero-to-one ratio equal to 1, and we use this information for constructing $\hat{M}$. Luckily, according to our large number of experiments, there is a high (around 97%) probablilily of guessing k within 2 if we know the approximate ratio of ones and zeros in vector $\vec{x}$ and r_j is sufficiently large. We use the following formula to guess k_j:

$$k_j = \frac{\sum_{i=1}^{n} s_i \cdot ratio}{r_j} \tag{9}$$

where *ratio* is the ratio between the approximate number of ones in the solution vector and the length of the solution vector (as mentioned above, we consider the ratio to equal $\frac{1}{2}$).

The necessity of guessing eliminates the possibility of having a large number of columns in the representation matrix. Indeed, guessing k's for n columns is exponential, and thus unacceptable because good performance of the algorithm is only achieved at $n > 100$.

Preliminary runs showed that the performance of representations with two and three columns are approximately the same, but there is more to guess in the three dimensional case. Thus, we concentrated on two-column representation matrices only.

10 The way to construct two columns

Since the optimal number of columns is two, a slightly different approach was found to be very efficient. We are still using remainders. But we have only one for the two columns. The first column contains remainders of division of set elements by the selected number. The second column contains results of integer division of set elements by the selected number. Let r be the number we select as the divisor. Let

$$\begin{gathered}
s_1 = a_1 \bmod r \\
s_2 = a_2 \bmod r \\
s_3 = a_3 \bmod r \\
\dots\dots\dots\dots\dots \\
s_m = a_m \bmod r \\
m = M \bmod r
\end{gathered}$$

Then

$$\begin{gathered}
a_1 = s_1 + p_1 \cdot r \\
a_2 = s_2 + p_2 \cdot r \\
a_3 = s_3 + p_3 \cdot r \\
\dots\dots\dots\dots\dots \\
a_m = s_m + p_m \cdot r \\
M = m + p_0 \cdot r
\end{gathered}$$

Let vector $\vec{x}$ be the problem solution vector, vector $\vec{a}$ be the set vector ($\vec{a} = \{a_1, a_2, ..., a_m\}$), $\vec{s}$ be the remainders vector ($\vec{s} = \{s_1, s_2, ..., s_m\}$), and $\vec{p}$ be the vector of the results of division ($\vec{p} = \{p_1, p_2, ..., p_m\}$). Then since

$$\vec{a} \cdot \vec{x} = M, \tag{10}$$

it is also true that

$$\vec{s} \cdot \vec{x} + \vec{p} \cdot \vec{x} \cdot r = m + p_0 \cdot r \tag{11}$$

Now, if we guess the coefficient k, such that

$$\vec{s} \cdot \vec{x} = k \cdot r + m \tag{12}$$

it is easy to see from (11) and (12) that

$$\vec{p} \cdot \vec{x} = p_0 - k \tag{13}$$

(simply subtract (12) from (11))
Thus, ((11) $\bigcup$ (12)) $\Leftrightarrow$ ((12) $\bigcup$ (13)), if k is guessed right. Thus the following two columns represent the *Subset Sum problem* instance:

column 1	coulmn 2
s_1	p_1
s_2	p_2
s_3	p_3
$\cdots$	$\cdots$
s_m	p_m
$k \cdot r + m$	$p_0 - k$

Guessing k is easy. As mentioned above, we consider the most difficult instances of the problem. They contain approximately equal number of ones and zeros, so using this knowledge, we guess k according to (9) where $j = 1$ and $ratio = \frac{1}{2}$.

11 Implementation details

We use the matrix representation suggested by C. P. Schnorr in [11], as described in section 3 with slight but important modifications. Firstly, instead of one column of data we have two as described above. That change forces us to make small changes in the solution criteria (3). Overall, the representation and solution criteria are the following:

Consider a *Subset Sum problem* according to (1). Let us associate with integers a_i, $i = \{1, ..., n\}$ and M the following basis

$$\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1} \in Z^{n+3}$$

$$\begin{aligned}
\vec{b}_1 &= (2, 0, 0, ..., 0, c \cdot s_1, c \cdot p_1, 0) \\
\vec{b}_2 &= (0, 2, 0, ..., 0, c \cdot s_2, c \cdot p_2, 0) \\
\vec{b}_3 &= (0, 0, 2, ..., 0, c \cdot s_3, c \cdot p_3, 0) \\
&\vdots \\
\vec{b}_n &= (0, 0, 0, ..., 2, c \cdot s_n, c \cdot p_n, 0) \\
\vec{b}_{n+1} &= (1, 1, 1, ..., 1, c \cdot (k \cdot r + m), c \cdot (p_0 - k), 1)
\end{aligned} \tag{14}$$

Here c is a multiplier (a 10-bit number proved to perform best for our tests) used to introduce longer vectors to make LLL perform more efficiently.

Every lattice vector $\vec{z} = (z_1, z_2, ..., z_{n+3}) \in L(\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1})$ that satisfies

$$|z_{n+3}| = 1, z_{n+1} = 0, z_{n+2} = 0, z_1, z_2, ..., z_n \in \{1, -1\} \tag{15}$$

yields the following solution for the *Subset Sum problem*

$$x_i = \frac{|z_i - z_{n+3}|}{2}, \text{for i = 1, 2, \ldots, n} \tag{16}$$

In the program, we try to guess the coefficient k, and try to solve the matrices (14) for ks within 3 of the original guess. Also, variations are possible in choosing the number r. Best performance of the algorithm is achieved when the bit length of r is about 65% of that of the set elements.

12 Sample data matrices

In this section we present a really simple run with a lot of extra output that lets us peek at internal data structures and data representation. The following run attempts to solve the problem with $size = 5$, $bits = 7$:

```
[sgi64:/home/vladimir/backup_thesis]21 % ss 1 5 7 123 1 65
4
15
Density = 0.714286
0:68
0:43
0:85
0:53
0:111
10010the sum: 0:121
. The matrix created is:
2 0 0 0 0 8888 4444 0
0 2 0 0 0 14443 2222 0
0 0 2 0 0 11110 5555 0
0 0 0 2 0 8888 3333 0
0 0 0 0 2 6666 7777 0
1 1 1 1 1 17776 7777 1

Checking solution on the matrix:
1  -1 -1   1  -1  0     0    -1
6   0  2  -2   0  0     0     4
1   3 -7  -1   3  0     0    -1
-3 -1  3  -5  -3  0     0    -5
1  -1  3  -1  -1 -1111  0     1
-2  0  0   2   0  0    -1111  0


10010res1
0:121
CPU TIME WAS 0.002271
solution Found 1

Solved 1 instances out of 1
Number of LLL calls: 1
Spent 0.002745CPU time

[sgi64:/home/vladimir/backup_thesis]22 %
```

The first line here is the call of the program to solve one-dimensional Subset Sum problem with the size of the set 5, the bit size of the set elements 7, seed 123. We are asking the program to attempt one instance, and set the bit size of the divisor r to be 65% of that of the set elements.

The second line shows the calculated bit size of r, the third line shows r itself. The density of the problem is reported on the next line. The next five lines represent the generated set. The output format of the long integers implemented for the thesis is the following: $\langle high.part \rangle : \langle low.part \rangle$, where both high. and low. parts are long integers, and the number that they represent is $high.part \cdot 2^{55} + low.part$. In this case high part is equal to zero, and thus the generated set is $\{68, 43, 85, 53, 111\}$.

The next line contains the generated solution (10010) and the generated sum (121). The hardcoded multiplier that was used in this run was 1111. Thus, the program split the set column into the following two columns:

```
8  4
13 2
10 5
8  3
6  7
```

Indeed, $68 = 4 \cdot 15 + 8$, $43 = 2 \cdot 15 + 13$, etc. The first (and as it turned out, correctly) guessed k was equal to zero, which defined the generated bottom row. This matrix was very easy to LLL-reduce, and the first round of reduction returned the next displayed matrix. Obviously, the first row of the matrix conforms to the solution criteria (15). The program processed that and returned vector 10010 as the solution. The following line shows the sum, calculated based on the obtained solution vector. The instance statistics printed show that 0.002271 seconds of CPU time were spent on solving this instance. The next statistics represent the whole run, but because the run consisted of only one attempt, the data is the same. (The reported CPU time is slightly larger, but that is because instance generation is included in the latter statistics). This experiment was run on a 195MHz SGI R10000.

13 Results and analysis

The following table represents the results we obtained. Twenty instances were attempted for every size/bits combination. Instances were generated by randomly setting each of the bits of set elements. Then the solution vector was generated that contained 50% ones. The sum was generated based on the set and solution vector. Then the set and the sum were passed to the algorithm, and correctness of the solution obtained by the algorithm was verified by comparing the two sums (the one generated as part of the instance of the problem, and the other generated based on the produced solution.) The entries in the table represent the number of instances solved out of 20.

size/bits	46	50	53	58	61	66	76
50	20	19	20	19	20	20	19
58	20	20	16	10	14	18	11
66	20	20	18	5	7	1	2
70	19	19	16	8	2	1	0

Significant improvement over matrix (3) was observed. The following results were obtained by Schnorr in [11] using the simplest version of *LLL* algorithm (the one used in this thesis):

size/bits	46	47	50	53	58	61	66	76
50	6	–	12	–	17	–	20	–
58	–	3	–	1	2	–	11	–
66	–	–	0	–	1	–	0	–

The most difficult problems, by Schnorr, are the ones with density slightly greater than 1, for example instances when the size of the set is 58, and the number of bits is 53. In this particular instance, the improvement was 16 times! The new representation also performs better for many other densities, both greater and smaller than 1.

An important feature of our implementation of the algorithm is single precision computations. No floating point or multiprecision arithmetic is used. The generated set elements and sums are represented as 2 long integers on a 64-bit machine. A mini-package was implemented to manipulate data that exceeds 64 bits in size. However, we do slow multiprecision computations only at the matrix generation and solution verification stages of the algorithm.

Splitting the information carried by the data column into two columns let us create matrices with single precision long integer entries. Avoiding slow multiprecision computations substantially reduced execution time. Most of the implementations don't have it.

14 Conclusion and future work

The results obtained are very encouraging. The performance of the algorithm increased very substantially with the introduction of multidimensionality. And there is still room for experiments. The variations of the parameters that we fixed (such as multiplier coefficient c, and especially the divisor r from (11)) allow to achieve better results for some size/bits combinations. For example, the following table represents the results of running the same algorithm with bit size of r being 70% of that of the set elements:

size/bits	46	50	53	58	61	66
50	19	19	19	20	20	20
58	20	20	11	9	12	18
66	20	20	17	7	5	0
70	18	19	17	4	–	–

As one can see, the latter configuration performs better for some bit/size combinations. One of the directions of future work would be theoretical explanation of the influence of this parameter on the performance of the algorithm. That would probably lead to a theoretically optimal value of r, which would improve the performance. Also, theoretical evaluation of the difficulty of multidimensional Subset Sum problem and its comparison to that of the equivalent single-dimensional problem needs to be done to find the right direction in the future research.

No matter how encouraging the results seem, similar results were achieved in [11] by using heuristic lattice basis reduction algorithms (*LLL* with deep insertions and Korkine-Zolotarev reduction). It seems very useful to integrate the two approaches and analyze the results because this thesis and [11] exploit completely different and independent ways to improve performance.

References

[1] E. F. Brickell, "The cryptanalysis of knapsack cryptosystems", in *Applications of Discrete Mathematics*, SIAM, 1988, pp. 3-23

[2] E. F. Brickell, "Solving low density knapsacks", in *Advances in Cryptology, Proceedings of Crypto '83*, Plenum Press, New York, 1984, 25-37

[3] Benny Chor and Ronald L. Rivest. "A knapsack type cryptosystem based on arithmetic finite fields", *IEEE Trans. Information Theory 34,5* (Sep. 1988), 901-909. (Also in CRYPTO 84)

[4] Coster M. et al, "Improved Low-Density Subset Sum Algorithms". *Computational Complexity 2* (1992), pp. 111-128.

[5] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, New York, 1979

[6] J.C. Lagarias and A. M. Odlyzko, "Solving low-density Subset Sum problems", *J. Assoc. Comp. Mach*, 1985, 229-246

[7] Brian A. LaMacchia, "Basis Reduction Algorithms and Subset Sum problems", SM Thesis, Dept. of Elect. Eng. and Comp. Sci., Massachusetts Institute of Technology, Cambridge, MA

[8] A. K. Lenstra, H.W. Lenstra, L. Lovasz "Factoring polynomials with rational coefficients" *Math. Ann. 261* (1982), 515-534

[9] A.M. Odlyzko, "The rise and fall of knapsack cryptosystems", in *Cryptology and Computational Number Theory, Proc. Symp. Appl. Math.* 42, Amer. Math. Soc., 1990, 75-88

[10] Radziszowski S.P. and Kreher D. L., "Solving Subset Sum Problems with the LLL Algorithm", *JCMCC 3(1988)*, pp. 49-63

[11] Schnorr C.P. Euchner M., "Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems", *Foundations of Computation Theory*, 1991, 1993