Multidimensional Subset Sum Problem

Vladimir Kolesnikov
M.Sc. Thesis Overview
Rochester Institute of Technology, 1997

1 Introduction

This document is an informal description of our main results presented in the thesis [2]. We propose heuristic modifications to the successful use of Lenstra, Lenstra and Lovasz's LLL algorithm [4] to solving the *Subset Sum* (or *knapsack*) problem.

Informally, the knapsack problem is as follows. Given a vector of n 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 $\sum_{i=1}^{n} a_i \cdot x_i = M$.

The LLL algorithm is the most recognized and successful tool for finding short vectors in an integer lattice. Although its formal performance guarantees are very weak, very often LLL performs much better than the theoretical bound. A common way to solve combinatorial problems, such as knapsack, is to represent the instance as a lattice, such that the solution to the instance is a very short vector in the lattice. Running LLL will often quickly find the right short vector and thus solve the problem. Recall, it was this approach that caused demise of knapsack based public-key cryptosystems [1, 5].

Our work concerns representation of the knapsack instance as a lattice with the above properties.

1.1 The traditional reduction

Consider the following reduction [3, 7, 6]. Associate with the n knapsack weights $a_1, ...a_n$ and knapsack volume M the following basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1} \in Z^{n+2}$:

$$
\begin{aligned}
\vec{b}_1 &= (2, 0, 0, ..., 0, na_1, 0) \\
\vec{b}_2 &= (0, 2, 0, ..., 0, na_2, 0) \\
\vec{b}_3 &= (0, 0, 2, ..., 0, na_3, 0) \\
&\vdots \\
\vec{b}_n &= (0, 0, 0, ..., 2, na_n, 0) \\
\vec{b}_{n+1} &= (1, 1, 1, ..., 1, nM, 1)
\end{aligned} \tag{1}
$$

Let $L(\vec{b}_1, ..., \vec{b}_{n+1})$ be the integer lattice generated by the basis vectors $\vec{b}_1, ..., \vec{b}_{n+1}$. Every lattice vector $\vec{z} = (z_1, z_2, ..., z_{n+2}) \in L(\vec{b}_1, ..., \vec{b}_{n+1})$ that satisfies

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

yields the following solution for the *Subset Sum problem*

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

We stress that such vectors are very short, and are likely to be found by LLL.

2 Our reduction

We note that the most interesting knapsack problems are the ones where approximately half of the items perfectly fit the knapsack and the weights are chosen randomly. We optimize our approach to solving such problems.

Our main idea is to split the "data column", defining the knapsack constraints, into several such constraints satisfied by the same solution. As our experiments show, LLL performs *much* better when it is given several *linearly independent* data columns, rather than one. The difficulty is in obtaining several linearly independent knapsack weights/volume relationships from the given one.

The first idea that comes to mind is to use remainders and modulo arithmetic. We could choose (several) random r's from an approporiate domain and set (several) new constraint(s): $\forall i = 1..n, a'_i = a_i \mod r$ and $M' = M \mod r$. Unfortunately, LLL does not seem to perform well in finite fields. Further, it seems highly desirable to have basis vectors of higly varying magnitude, as in (1).

Therefore, we propose *guessing* an integer k and setting $\forall i = 1..n, a'_i = a_i \mod r$ and $M' = M \mod r + k \cdot r$. If k is guessed correctly, a'_i, M' form a linearly independent instance of knapsack, with the solution vector coinciding with that of the original instance.

Now we make use of our assumption that the solution vector consists of roughly half zeros and half ones, and the weights are chosen uniformly from a given domain. This allows us to get a good estimate on k. According to our 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 is sufficiently large.

The necessity of guessing prevents us from using a large number of such linearly independent constraints. It turns out that having two constraints is optimal.

We now show a way to construct two constraints, while having to make only one guess of k. Let $\vec{x}$ be the knapsack instance solution vector, $\vec{a}$ be the weights vector. Suppose r is chosen. Set the remainders vector $\vec{s} = \{a_1 \mod r, ..., a_n \mod r\}$. Let $\vec{p} = \{a_1 \text{ div } r, ..., a_n \text{ div } r\}$, where div is the integer division operation. Let $m = M \mod r$ and $p_0 = M \text{ div } r$.

Then since $\vec{a} \cdot \vec{x} = M$, it is also true that $\vec{s} \cdot \vec{x} + \vec{p} \cdot \vec{x} \cdot r = m + p_0 \cdot r$. Now, if we guess the coefficient k, such that $\vec{s} \cdot \vec{x} = k \cdot r + m$, then it also holds that $\vec{p} \cdot \vec{x} = p_0 - k$. The above two constraints constitute two linearly independent constraints, simultaneously satisfied by the solution to the original knapsack instance.

2.1 Putting it all together

Consider a knapsack instance as discussed above. Associate with weights a_i, $i = \{1, ..., n\}$ and volume M the following basis $\vec{b}_1, \vec{b}_2, ..., \vec{b}_{n+1} \in Z^{n+3}$.

$$\begin{aligned}
\vec{b}_1 &= (2, 0, 0, ..., 0, c \cdot s_1, c \cdot p_1, 0) \\
\vec{b}_2 &= (0, 2, 0, ..., 0, c \cdot s_2, c \cdot p_2, 0) \\
\vec{b}_3 &= (0, 0, 2, ..., 0, c \cdot s_3, c \cdot p_3, 0) \\
&\vdots \\
\vec{b}_n &= (0, 0, 0, ..., 2, c \cdot s_n, c \cdot p_n, 0) \\
\vec{b}_{n+1} &= (1, 1, 1, ..., 1, c \cdot (k \cdot r + m), c \cdot (p_0 - k), 1)
\end{aligned} \tag{4}$$

Here $\vec{s}, \vec{p}$ are chosen as described above. Best performance is observed when r is of bit length 65% of that of a_i. Set $k = \sum_{i=1}^{n} s_i/2r$. Here c is a multiplier (a random 10-bit number proved to perform best for our tests) used to introduce longer basis 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

$$|z_{n+3}| = 1, z_{n+1} = 0, z_{n+2} = 0, z_1, z_2, ..., z_n \in \{1, -1\} \tag{5}$$

yields the following solution for the *Subset Sum problem*

$$x_i = \frac{|z_i - z_{n+3}|}{2}, \text{for i = 1, 2, ..., n} \tag{6}$$

Since we are not sure on k, we run five instances of the above algorithm, in parallel, with $k_1 = k, k_2 = k + 1, k_3 = k + 2, k_3 = k - 1, k_4 = k - 2$.

2.2 Results and Analysis

The following table represents the results we obtained. Twenty instances were attempted for every size/bits combination (size is the number of weights n, bits is the bit-size of each weight). Instances were generated by randomly setting each of the bits of set elements. Then the solution vector was generated that contained 50% zeros. 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 previous reductions was observed. The following results were obtained by Schnorr in [7] 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	–

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 two 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 previous implementations reply on the (inefficient) multiprecision arithmetic.

Efficiency of the application of LLL to knapsack increased substantially with our representation. We note that the variations of the parameters are possible. This allows to achieve better results for some size/bits combinations.

We note that a lot of work is done on improving the LLL algorithm itself, e.g. LLL with deep insertions and Korkine-Zolotarev reduction. It seems interesting to evaluate performance of our representation with new versions of LLL.

References

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

[2] Vladimir Kolesnikov, “Multidimensional Subset Sum Problem“, *M.Sc. Thesis, Rochester Institute of Technology*, 1997

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

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

[5] 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

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

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