Solving Subset Sum Problems with the L^3 Algorithm

Stanisław p. Radziszowski and Donald L. Kreher

School of Computer Science
Rochester Institute of Technology

ABSTRACT

Ideas are described that speed up the lattice basis reduction algorithm of Lenstra, Lenstra and Lovász [11] in practice. The resulting lattice basis reduction algorithm reduces the multiprecision operations needed in previous approaches. This paper describes these ideas in detail for lattices of the particular form arising from the subset sum (exact knapsack) problem. The idea of applying the L^3 algorithm to the subset sum problem is due to Lagarias and Odlyzko [8]. The algorithm of this paper also uses a direct search for short vectors simultaneously with the basis reduction algorithm. Extensive computational tests show that this algorithm solves, with high probability, instances of low density subset sum problems and has two major advantages over the method of Lagarias and Odlyzko: running time is an order of magnitude smaller and higher density subset sum problems are solved.

1. Introduction

The subset sum problem is:

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

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

The existence of a solution to (1) is in general an NP-complete problem, similar to the related knapsack problem, Garey and Johnson [4]. The subset sum problem was one of the first problems discovered to be NP-complete and is consequently very well known. Its importance increased dramatically when several public-key cryptosystems, whose security are based on the practical intractability of solving (1), were proposed and discussed [2,13].

The problem of solving (1) can be reformulated as finding a particular short vector x in an integer lattice $L = L(a,M)$. Although one could use any known method for finding short vectors in integer lattices [1,10,11], in this paper we use the so called the L^3 basis reduction algorithm of Lenstra, Lenstra and Lovász [11], with some additional features. The initial idea of our algorithm is taken from the paper of Lagarias and Odlyzko [8]. In this paper they gave a method based on the L^3 algorithm, called the SV (short vector) algorithm, for solving, with high probability, low density instances of (1). The density of a subset sum problem (1) is defined as

JCMCC 3(1988), pp. 49–63

$$d(a) = \frac{n}{\log_2(\max\{a_i : 1 \le i \le n\})} \qquad (2)$$

and represents the information rate of the corresponding knapsack code.

They suggest on the basis of extensive computational tests, that their method works for densities $d < d_c(n)$, where $d_c(n)$ is a cutoff value that is substantially larger than the best up to now theoretical estimation of $1/n$. They observed, that some of the practical upper bounds for the cutoff value $d_c(n)$, using their method, are $d_c(30) \le 0.60$ and $d_c(40) \le 0.50$. Our method, as will be shown in section 5, significantly improves these bounds (for example, for our algorithm $d_c(42) \approx 0.62$). A nice analysis of the method of Lagarias and Odlyzko can be found in [3].

Another algorithm, constructed by Brickell [1], for solving the same problem can be expected to have smaller cutoff values for similar data. Schnorr [12] presents a hierarchy of polynomial time basis reduction algorithms, which are refinements of the L^3 algorithm. Although, each algorithm in this hierarchy runs asymptotically in the same time as the L^3 algorithm and theoretically finds shorter vectors, it was not shown in practice that they are superior.

Our LS (L^3 applied to Subset sum) algorithm uses the method of Lagarias and Odlyzko as one of its basic steps. Among the most important new features we have added to the SV algorithm are:

F1: Avoidance of the many of the multiprecision operations that are required in previous approaches as in [1,8,11],

F2: The use of the L^3 algorithm jointly with other methods for finding short vectors in integer lattices.

The above improvements produce an algorithm that finds shorter vectors than L^3 does, solves higher density knapsack problems as used in cryptography, and furthermore, it is faster than the SV algorithm. We were unable to give a rigorous proof of the fact that the LS algorithm is better (faster and more powerful). However, extensive computational tests described in section 5 confirm the advantages stated above. We give some intuitive consideration which at least partially explain why the LS algorithm performs so well in practice.

Section 2 describes briefly the idea of the L^3 and SV algorithms, section 3 presents the mechanisms of the new features F1 and F2 mentioned above and the LS algorithm is stated in section 4. Next, in section 5 we describe in detail experiments done and compare it to previous work. Finally, section 6 contains concluding remarks.

2. Tools

Before describing the LS algorithm, we introduce the basic concepts about integer lattices and the L^3 algorithm we use.

Let n be a positive integer. A subset L of the n-dimensional real vector space R^n is called a *lattice* iff there is a basis $B = \{b_1, b_2, ..., b_n\}$ of R^n such that every member of L is an *integer* linear combination of the vectors in B. Recall that given a basis $B = \{b_1, b_2, ..., b_n\}$ of R^n an orthogonal basis $B^* = \{b_1^*, b_2^*, ..., b_n^*\}$ of R^n

may be obtained inductively via the Gram-Schmidt process of orthogonalization as follows:

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*, \quad \text{for } 1 \le i \le n, \tag{3}$$

$$\mu_{ij} = (b_i, b_j^*)/(b_j^*, b_j^*), \quad \text{for } 1 \le j < i \le n, \tag{4}$$

where $(\cdot,\cdot)$ denotes the ordinary inner product on R^n. An ordered basis $B = [b_1, b_2, ..., b_n]$ for a lattice L will be said to be *y-reduced (or reduced)* if the following two conditions hold:

(i) $|\mu_{ij}| \le \frac{1}{2}$ for $1 \le j < i \le n$,

(ii) $|b_i^* + \mu_{ii-1} b_{i-1}^*|^2 \ge y \cdot |b_{i-1}^*|^2$ for $1 < i \le n$,

where y, $\frac{1}{4} < y < 1$ is a constant and $|\cdot|$ denotes ordinary Euclidian length (or absolute value). Lenstra et al. [11] describe an algorithm, which when presented with y, $\frac{1}{4} < y < 1$ and an ordered basis $B = [b_1, b_2, ..., b_n]$ for a lattice L as input, produces a reduced basis $B' = [b_1', b_2', ..., b_n']$ as output. The L^3 algorithm consists of applying a finite number of two kinds of linear transformations:

T1: Interchange vectors b_i and b_{i-1} if $|b_i^* + \mu_{ii-1} b_{i-1}^*|^2 \ge y |b_{i-1}^*|^2$ does not hold, for some $1 < i \le n$, and the global constant $y \in (\frac{1}{4}, 1)$.

T2: Replace b_i by $b_i - r b_j$, where $r = \text{round}(\mu_{ij})$ is the integer nearest to μ_{ij}, and $|\mu_{ij}| > \frac{1}{2}$, for some $1 \le j < i \le n$.

The efficient implementation of a sequence of transformations T1 and T2 in the L^3 algorithm relies mainly on the fact, that old values of μ_{ij} and $|b_i^*|^2$ can be easily updated after each transformation without using the full process of orthogonalization. The L^3 algorithm performs the transformations T1 and T2 using a strategy resembling somewhat the bubble-sort, however as H.W.Lenstra [10] remarks, *any* sequence of the these transformations will lead to the reduced basis.

The L^3 algorithm terminates when neither T1 nor T2 can be applied and such a situation implies that conditions (i) and (ii) are satisfied. The resulting reduced basis B' is an integer approximation to the basis B^* defined by the Gram-Shmidt orthogonalization process and as a consequence contains short vectors, as can be seen in the following proposition of Lenstra et al. [11, prop. 1.11]:

PROPOSITION 1: *Let* $B' = [b_1', b_2', ..., b_n']$ *be a reduced basis of a lattice L. Then*

$$|b_1'|^2 \le 2^{n-1} \cdot \min\{ |b|^2 : b \in L \text{ and } b \ne 0 \}.$$

They also give the following polynomial worst-case running time for its performance [11, prop 12.6]:

PROPOSITION 2: *Let* $B = [b_1, b_2, ..., b_n]$ *be an ordered basis for an integer lattice* L *such that* $|b_i|^2 \leq \text{Max}$ *for* $1 \leq i \leq n$. *Then the* L^3 *algorithm produces a reduced basis* $B' = [b'_1, b'_2, ..., b'_n]$ *for* L *using at most* $O(n^4 \log \text{Max})$ *arithmetic operations, and the integers on which these operations are performed have length at most* $O(n \log \text{Max})$.

Finally we describe the SV algorithm of Lagarias and Odlyzko. In the remaining sections to simplify notation lattices will have dimension n+1. Given an integer vector $a = (a_1, a_2, ..., a_n)$ and target integer M as in (1), define the basis B to be the n+1 row vectors of the matrix that follows:

$$B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ 0 & 0 & 1 & \cdots & 0 & -a_3 \\ & & & \cdot & & \\ & & & \cdot & & \\ 0 & 0 & 0 & \cdots & 1 & -a_n \\ 0 & 0 & 0 & \cdots & 0 & M \end{bmatrix} \tag{5}$$

We can assume that $1 < M < \sum_{i=1}^{n} a_i$, otherwise if there is a solution to (1) it is trivial.

The SV algorithm

SV1: Define the lattice L=L(a,M) by the basis B in (5);

SV2: Find a reduced basis $B' = [b'_1, b'_2, ..., b'_{n+1}]$ of L using the L^3 algorithm;

SV3: Check if any $b'_i = (b'_{i1}, \ldots, b'_{i,n+1})$ has all $b'_{ij} = 0$ or λ for fixed λ equal to 1 or -1 and for all j, $1 \leq j \leq n$. For any such b'_i, check whether $x = (x_1, ..., x_n)$, $x_j = |b'_{ij}|$ for $1 \leq j \leq n$, gives a solution to (1), and if so, terminate;

SV4: Repeat steps SV1 - SV3 with M replaced by $M' = \sum_{i=1}^{n} a_i - M$. Then terminate.

If algorithm SV produces a solution to (1) then it *succeeds*, otherwise it *fails*. The SV algorithm has the same guaranteed running time as the L^3 algorithm. In practice, however, Lagarias and Odlyzko [8] observed that it behaves much better then the worst case estimation of $O(n^6 (\log \text{Max})^3)$ obtained from proposition 2, if a standard implementation of arithmetic operations is assumed. They claimed that it uses instead about $O(n(\log \text{Max})^3)$ bit operations. The SV algorithm also finds much shorter vectors than could be concluded from proposition 1. In section 5 we

give a comparison of the success rate for the SV algorithm and the LS algorithm for some densities and different dimensions. This comparison shows that the LS algorithm behaves even better.

3. New Features

3.1. Handling Multiprecision

In the case of an initial basis of the form in (5), we have $B = [b_1, b_2, \cdots, b_{n+1}]$, where

$$b_{ij} = \begin{cases} 0 & \text{if } 1 \le i \ne j \le n, \\ 1 & \text{if } 1 \le i = j \le n, \\ -a_i & \text{if } 1 \le i < j = n+1, \end{cases}$$

$$b_{n+1,i} = 0, \quad \text{for } 1 \le i \le n,$$

$$b_{n+1,n+1} = M.$$

Note that in this case the equality

$$(a, (b_{i1}, \cdots, b_{in})) = k_i \cdot M + b_{i,n+1}, \tag{6}$$

holds for $k_i = 0$ if $1 \le i \le n$, and $k_{n+1} = -1$. The only operations done in the L^3 algorithm over the basis B are the transformations T1 and T2, and condition (6) is an invariant of both of them. Thus, after the execution of the L^3 algorithm there will exist integers k_i, $1 \le i \le n+1$, such that equality (6) remains true for all i, $1 \le i \le n+1$. Let $S = L^3(B) = [s_1, s_2, \cdots, s_{n+1}]$ be the resulting reduced basis after running L^3 with input B. If for some i_o, $1 \le i_o \le n+1$, $s_{i_o,j}$ is 0 or 1 (or $s_{i_o,j}$ is 0 or -1) for all j, $1 \le j \le n$, $s_{i_o,n+1} = 0$ and $k_{i_o} = \pm 1$, then the equality (6) becomes

$$\sum_{j \in I} a_j = M, \quad \text{where } I = \{j : 1 \le j \le n \text{ and } s_{i_o,j} \ne 0\}, \tag{7}$$

i.e. the i_o-th row of the basis S represents a 0-1 solution to the subset sum problem given by vector a and target integer M.

The integer M and the coordinates of a are usually big numbers and a standard approach would require, that most calculations in the L^3 algorithm be done in multiprecision. Hence, our main attack in this direction consists of applying a divide and conquer technique with respect to the binary length of M and the coordinates of a. Roughly speaking, we run the L^3 algorithm some number, say s, times for the n+1 dimensional basis, but with numbers in the column n+1 of B of binary length 1/s of the original length.

The above idea is achieved by substitution for step SV2 in the SV algorithm the MP (multiprecision) algorithm.

The MP Algorithm

MP1: Let t be an integer somewhat smaller than the machine word size (say t=28 for a 32-bit computer).

$\text{Max} \leftarrow \max\{a_i : 1 \le i \le n\}$; $\text{Max} \leftarrow \max\{\text{Max}, M\}$;

$\text{top} \leftarrow \lfloor \log_2 \text{Max} \rfloor + 1$; $k \leftarrow \max\{\text{top}-t, 0\}$;

Set $S = [s_{ij}]_{1 \le i,j \le n+1}$ to be a basis of an integer lattice of the form found in (5), but with the single precision approximation to the last column of B.

$$S \leftarrow B;\ s_{i,n+1} \leftarrow \text{round}(b_{i,n+1}/2^k),\ \text{for } 1 \le i \le n+1;$$

MP2: Before each iteration of this step, S will satisfy (6) for the integer $M' = \text{round}(M/2^k)$ and the vector $a' = (\text{round}(a_1/2^k), \ldots, \text{round}(a_n/2^k))$, i.e. for the original data rounded after dropping k least significant bits. Finally, for k=0 we will obtain the same situation as after step SV2 in the SV algorithm, since in this case S will span the same lattice as B. The step MP2 will be iterated $s \approx \text{top}/t$ times.

```
while k>0 do
 begin
    S ← L³(S);
    { adjust basis S to span the same lattice as basis B }
    for i←1 to n+1 do
      begin
         s_{i,n+1} ← (a,(s_{i1},...,s_{in}));
         { represent s_{i,n+1} as in (6) }
         calculate integers c_i, k_i such that s_{i,n+1} = k_i·M + c_i, and |c_i| ≤ M/2;
         s_{i,n+1} ← c_i
      end;
    { calculate maximal size of numbers in the new last column of S }
    Max ← max{ |s_{i,n+1}| : 1≤i≤n+1};
    top ← ⌊log₂Max⌋ + 1;
    k   ← max{top−t, 0};
    { take the first block of t bits of the last column of S }
    for i←1 to n+1 do s_{i,n+1} ← round(s_{i,n+1}/2^k)
 end;
```

MP3: Rearrange the order of rows in the matrix S by shifting up to the first positions those vectors s_i for which the value k_i in the last iteration of step MP2 was nonzero. As we noted in the comment to (7) the i_o-th row of S representing a solution to (1) will have $k_{i_o} = \pm 1$. This property is strongly propagated by the L^3 algorithm to rows with higher indices. Consequently, the above arrangement of rows will increase the algorithm chance of success.

MP4: $B \leftarrow L^3(S)$.

If the MP algorithm terminates, then B will be a basis for the original lattice, hence the correctness of the updated SV algorithm follows directly from the correctness of the SV algorithm.

In the implementation of the L^3 algorithm we use (as did the authors of the SV algorithm) floating-point approximations to the rational values of μ_{ij} and $|b_i^*|^2$. The accumulation of errors in floating-point μ's in the MP algorithm is negligible, since at each iteration of step MP2, μ's are initialized directly from the definition and all integer variables defining μ_{ij} fit in single precision. All the calls to the L^3 algorithm in steps MP2 and MP4 are executed entirely in single precision arithmetic. Only a small number of multiprecision operations are done in step MP1 and in updating the basis S in the for-loops in the step MP2.

This approach appears to give the same, or even better, results than the original SV algorithm, but for a wide range of data in time approximately s^2 times smaller, where s is the number of iterations of step MP2. This technique enables us also to perform the majority of calculations in single precision arithmetic, dropping most of the time consuming multiprecision operations. A similar method for handling multiprecision calculations was used by Lehmer [9] in his greatest common divisor algorithm.

The theoretical estimation given in proposition 2 does not explain this reduction in time. On the other hand, the experimental time estimation obtained by Lagarias and Odlyzko, $O(n(\log \mathrm{Max})^3)$, at least confirms our experiment, since the divide and conquer technique with a partition into s groups should give running time proportional to

$$s \cdot n \cdot \left(\frac{\log \mathrm{Max}}{s} \right)^3 = \frac{n \cdot (\log \mathrm{Max})^3}{s^2}.$$

3.2. Weight Reduction

If B is the reduced basis produced by the L^3 algorithm, then there will often exist pairs of indices i and j, $1 \le i,j \le n+1$, $i \neq j$, and a choice of ϵ such that

$$v = b_i + \epsilon b_j, \quad \epsilon = \pm 1, \quad \text{and} \quad |v| < \max\{ |b_i|, |b_j| \}. \tag{8}$$

A pair (i,j), $i \neq j$, satisfies the last condition if and only if $\max\{ |b_i|^2, |b_j|^2 \} < 2 \cdot |(b_i,b_j)|$. In such a case we can choose ϵ to have a different sign from (b_i,b_j) and substitute the longer of b_i and b_j by v, obtaining a new basis with decreased *total weight*

$$w(B) = \sum_{p=1}^{n+1} |b_p|^2.$$

In the process of finding successive pairs of indices (i,j) satisfying (8) it is not necessary to recalculate $|v|^2$ and (v,b_k) from the definitions, instead we can keep track of the integers $|b_i|^2$ and $inn_{ij} = (b_i,b_j)$, for $1 \le j < i \le n+1$, using formulas:

$$|v|^2 = |b_i|^2 + |b_j|^2 - 2|inn_{ij}|,$$

$$(v,b_k) = inn_{ik} + \epsilon \cdot inn_{jk}, \text{ for } 1 \le k \le n+1,\ k \ne i \text{ and } k \ne j.$$

A simple algorithm for finding all such pairs can be designed and implemented in time $O(n^2)$ for each reduction, producing as output a basis with smaller weight. This algorithm, let us call it *Weight-Reduction*, is a useful complement to the L^3 algorithm. When used as follows, the algorithms L^3 and *Weight-Reduction* jointly tend to produce much shorter vectors than using L^3 or *Weight-Reduction* alone:

$B \leftarrow L^3(B)$;
repeat
  Weight-Reduction;
  sort basis with respect to $|b_i|^2$;
  $B \leftarrow L^3(B)$;
until (w(B) does not decrease);
Weight-Reduction.

The L^3 algorithm can remove the vector b_1 from the basis B only by replacing it with a shorter vector, since for $i=2$ if the transformation T1 can be applied then $|b_i|^2 < |b_{i-1}|^2$ (note that this is not true when $i > 2$). Hence sorting the basis with respect to $|b_i|^2$ guarantees that the shortest vector in the basis B will not disappear in the next iteration, unless a new shorter vector is found.

Following the above approach one can try in general to find a k-tuple of distinct vectors $b_{i_1}, \cdots, b_{i_k}$, for some $k \ge 2$, in the basis B, such that the vector

$$v = \sum_{p=1}^{k} \epsilon_p b_{i_p}, \text{ for some choice of } \epsilon_p = \pm 1,\ 1 \le p \le k,$$

is shorter than b_{i_k}, where b_{i_k} is the longest vector in the k-tuple. In the latter case the weight of basis B can be decreased by substituting b_{i_k} by v. Note that

$$|v| < |b_{i_k}| \text{ iff } |v|^2 = \sum_{j=1}^{k} |b_{i_j}|^2 + \sum_{h \ne j} \epsilon_{i_h} \epsilon_{i_j} (b_{i_h}, b_{i_j}) < |b_{i_k}|^2 \qquad (9)$$

and a necessary condition for (9) is

$$\sum_{j=1}^{k-1} |b_{i_j}|^2 < \sum_{h \ne j} |(b_{i_h}, b_{i_j})|.$$

Consequently, our approach is to search for such k-tuples of vectors by considering the complete graph G, whose vertices are the basis vectors b_i and whose edges are labeled by edge weight $|(b_i,b_j)|$. The endpoints of edges with large weight are "less" orthogonal, hence they are good candidates for the desired k-tuple. We can try to construct it by finding subgraphs of G with large edge weight.

Obviously, the complete analysis of all subgraphs in the graph G would be too expensive, however we are satisfied with heuristic search for just a few of them of relatively small size. They are used to decrease the weight of basis B similarly as before. This technique leads to the generalization of the *Weight-Reduction* algorithm and improves further the behavior of the **LS** algorithm.

3.3. Other Modifications

Let us mention two more improvements that increase the chances of success of the **LS** algorithm. Both of them consist of changing the original lattice L=L(a,M) to some other lattice L', such that L' contains the same solution vector to (1) as L (if any), but at the same time L' does not contain some of the short vectors belonging to L, that do not solve (1).

OM1: Take $L' = L(c \cdot a, c \cdot M)$, for some integer $c>1$, so if $\mathbf{x}$ is a solution to (1) then the vector $(\mathbf{x},0)$ belongs both to L and L'. Also, to each vector $b=(b_1,...,b_{n+1})$ with $b_{n+1} \neq 0$, $b \in L$ there corresponds a longer vector $b'=(b_1,...,b_n,cb_{n+1}) \in L'$. Experiments show that for random instances of (1) of dimension n in the range between 26 and 66, a good choice for c is $10<c<30$.

OM2: If we can estimate the length $len \approx |\mathbf{x}|^2$ of the solution, then we consider n+1 dimensional lattice $L' \subseteq R^{n+2}$ whose basis is given by row vectors of the following matrix:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & -a_1 & -1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 & -1 \\ & & & \cdot & & & \\ & & & \cdot & & & \\ 0 & 0 & 0 & \cdots & 1 & -a_n & -1 \\ 0 & 0 & 0 & \cdots & 0 & M & len \end{bmatrix}$$

If the estimation len is exact then the lattice L' has similar properties as in OM1. If there is only a small error in the estimation of len we will still obtain an improvement in the performance of the **LS** algorithm. In OM2 we can also use two different integer multipliers $c_1>1$ and $c_2>1$ for columns n+1 and n+2, as in OM1.

4. The LS Algorithm

The LS algorithm comprises the facts and procedures described in section 3 together with the SV algorithm.

LS1: Initialize with step MP1 (section 3.1).

In this step one can include the improvements OM1 and OM2 described in section 3.3. If the modification OM2 is used then in all subsequent calculations the (n+2)-nd coordinate has to be handled additionally in all vector operations.

LS2: Execute the remaining steps of the **MP** algorithm: MP2, MP3 and MP4 (section 3.1).

LS3: Let **B** be the reduced basis produced by step LS2. The basis **B** now contains only relatively short vectors and thus all calculations in this step can be done in single precision. Apply *Weight-Reduction* together with the $\mathbf{L^3}$ algorithm (section 3.2).

```
repeat

    { check for solution (section 2) }
    execute step SV3 of the SV algorithm;

    calculate w(B), the weight of B;

    Weight-Reduction;
    execute step SV3;

    sort basis with respect to |b_i|^2;
    B ← L^3(B);

until (w(B) does not decrease);

Weight-Reduction;
execute step SV3;
```

LS4: Similarly as in the **SV** algorithm, we can assume that $1 < M < \sum_{i=1}^{n} a_i$. Repeat steps LS1 - LS3 with M replaced by $M' = \sum_{i=1}^{n} a_i - M$ (if modification OM2 is used, change appropriately len, the solution length estimation). Then terminate.

5. Experiments

To perform multiprecision calculations a special package of routines was developed, generally following the algorithms described in Knuth [5]. In tests on a super-microcomputer MASSCOMP MC-500 (Motorola 68000 based system) we obtained on average running times of 21 minutes CPU time, for n=50, s=5, $a_p \leq 2^{100}$, $1 \leq p \leq n$, with a random (0,1)-solution vector of length 25 (in this section by the length of vector **b** we mean **(b,b)**). A sample of 20 instances of subset sum problem with these parameters was randomly generated and all of them were

solved, giving a success rate of 100%. This is an improvement in performance and in speed over the SV algorithm. Lagarias and Odlyzko do mention, that for their SV algorithm on data of this size, a CRAY-1 took about 14 minutes of CPU time with a success rate of only 33%. We claim on the base of experiments described below, that a machine like a CRAY-1 using the **LS** algorithm, should solve almost all instances of the subset sum problem of size 100 with density 0.3 in just a few minutes of CPU time.

In the implementation of the **LS** algorithm the following technical modifications were used:

1) An explicit bound on the number of iterations of the loop in step LS3 was enforced. The execution of the **LS** algorithm was aborted after 15 unsuccessful iterations or after 9 iterations when the basis contained vectors of length smaller than n/2.
2) Step MP3 was also executed inside loop LS3 if the vectors with $k_i = \pm 1$, the candidates for solution, appeared only at higher indices of the basis. The performance of the algorithm is sensitive to this modification.
3) Inside the loop LS3, the *Weight-Reduction* algorithm searched for all possible reductions based on pairs and triplets of vectors, and used heuristics to find almost all reductions based on 4-tuples of vectors (see the end of section 3.2).
4) Step LS4 was not executed, for although this step increases the probability of finding the solution, it makes the execution time about twice as long.
5) Modifications OM1 and OM2 were tested only occasionally and no evident improvement in the performance of the algorithm was observed. They were not included in the experiments reported below.

In systematic tests on a MASSCOMP MC-500 we ran the **LS** algorithm for n = 26, 34, 42, 50, 58, 66, 74, 82, 90 and 98. For each choice of n the experiment was done for different values of density d lying close to the cutoff value $d_c(n)$ of the **LS** algorithm. For each fixed pair (n,d), 20 (for $n<82$) or 10 (for $n \geq 82$) random instances of the subset sum problem together with a randomly distributed solution of length n/2 were created. For all experiments the value of t (in the **MP** algorithm) was 28 and the value of y (in the $\mathbf{L^3}$ algorithm) was 0.99.

The results and execution times are gathered in table I. For example, the first line of data in table I means: for n=26, 20 random instances of the subset sum problem with $a_i < 2^{27}$ (i.e. problems with density 26/27=0.963, see also equation (2)), each with a random 0-1 solution vector of length n/2=13 were generated. For all of them the step MP2 (multiprecision phase) was executed once and the time spent by the **MP** algorithm was on average 0.41 minutes CPU (24.6 seconds). The total time spent by the **LS** algorithm was on average 4.35 minutes CPU time (including unsuccessful runs). The overall average of lengths of vectors in the final basis was 10.3. Finally, 1 out of 20 instances were solved giving a success rate of 5 percent.

Test Results Using the LS Algorithm 20 trials for each density d for each size n								
size n	density d	bits n/d	MP2 times	MP CPU min.	LS CPU min.	average length	number of successes	% success rate
26	0.963	27	1	0.41	4.35	10.3	1	5
	0.926	28	1	0.42	4.41	10.3	2	10
	0.897	29	2	0.66	3.19	13.3	9	45
	0.867	30	2	0.70	2.60	14.5	14	70
	0.839	31	2	0.71	2.74	15.3	13	65
	0.813	32	2	0.74	2.66	15.7	14	70
	0.788	33	2	0.76	1.76	17.0	18	90
	0.765	34	2	0.77	1.79	18.0	18	90
	0.743	35	2	0.79	1.42	19.1	20	100
34	0.895	38	2	1.63	9.26	16.7	6	30
	0.872	39	2	1.68	9.93	17.3	7	35
	0.850	40	2	1.72	8.96	18.5	10	50
	0.829	41	2	1.77	9.17	18.8	8	40
	0.810	42	2	1.79	9.43	19.4	7	35
	0.791	43	2	1.82	8.41	20.7	8	40
	0.773	44	2	1.85	8.09	21.6	11	55
	0.756	45	2	1.91	6.80	23.5	13	65
	0.739	46	2	1.96	7.42	24.4	14	70
	0.723	47	2	1.97	7.49	25.3	14	70
	0.708	48	2	2.02	7.18	26.9	18	90
	0.694	49	2	2.04	6.94	28.0	16	80
	0.680	50	2	2.13	5.59	29.6	19	95
	0.667	51	2/3	2.41	6.21	31.1	18	90
	0.654	52	3	2.64	4.53	32.9	20	100
	0.642	53	3	2.71	5.35	33.1	20	100
42	0.778	54	3	4.80	19.54	27.0	6	30
	0.750	56	3	4.41	20.25	28.9	4	20
	0.724	58	3	4.45	25.35	31.0	6	30
	0.700	60	3	4.71	19.22	34.6	15	75
	0.677	62	3	4.80	20.90	37.3	14	70
	0.656	64	3	4.95	15.97	40.9	17	85
	0.636	66	3	5.16	15.96	44.9	18	90
	0.618	68	3	5.31	12.00	49.1	20	100
	0.600	70	3	5.44	12.81	51.3	20	100
50	0.714	70	3	7.84	54.65	40.5	5	25
	0.685	73	3	8.20	54.34	44.7	5	25
	0.667	75	3/4	9.08	48.14	49.1	9	45
	0.641	78	4	9.54	46.48	54.2	9	45
	0.617	81	4	10.04	40.28	58.5	14	70
	0.595	84	4	10.53	45.44	62.6	13	65
	0.574	87	4	11.09	33.03	71.8	18	90
	0.556	90	4	11.33	27.16	82.5	19	95
	0.538	93	4	11.76	24.46	89.1	20	100
	0.521	96	4	12.05	24.05	98.7	20	100
	0.500	100	4/5	14.07	21.73	112.9	20	100

Table I

Test Results Using the LS Algorithm 20 trials for each density d for each size n								
size n	density d	bits n/d	MP2 times	MP CPU min.	LS CPU min.	average length	number of successes	% success rate
58	0.624	93	4	16.2	95.0	70.6	4	20
	0.604	96	4	16.7	90.9	76.4	6	30
	0.586	99	4/5	20.1	95.4	81.3	6	30
	0.569	102	5	20.8	86.1	88.7	7	35
	0.552	105	5	19.6	74.3	98.8	15	75
	0.537	108	5	20.3	82.7	104.9	12	60
	0.523	111	5	20.9	68.7	118.9	16	80
	0.509	114	5	21.8	49.0	133.1	19	95
	0.496	117	5	22.2	46.6	144.6	20	100
66	0.569	116	5	28.4	144.3	117.5	8	40
	0.550	120	5	29.5	150.1	125.2	4	20
	0.532	124	6	38.4	142.8	138.5	8	40
	0.516	128	6	33.7	151.0	148.4	5	25
	0.500	132	6	35.7	132.6	168.3	12	60
	0.485	136	6	36.4	122.1	187.0	15	75
	0.471	140	6	37.9	121.4	203.8	16	80
	0.458	144	6/7	40.0	91.9	239.9	20	100
	0.446	148	7	46.7	83.1	276.7	20	100
74	0.463	160	7	58.8	241.4	270.9	3	15
	0.446	166	7/8	63.1	222.7	313.8	8	40
	0.430	172	8	70.3	198.8	360.2	17	85
	0.416	180	8	69.4	193.9	434.1	16	80
	0.402	184	8	72.6	139.6	538.9	19	95
	0.389	190	8/9	83.4	133.4	609.5	20	100
10 trials for each density d for each size n								
82	0.445	184	8	89.9	361.3	412.6	2	20
	0.427	192	9	110.7	372.4	471.5	4	40
	0.410	200	9	100.4	307.8	568.5	6	60
	0.402	204	9	104.9	304.7	624.1	5	50
	0.394	208	9	107.7	281.1	693.1	8	80
	0.387	212	9/10	127.7	238.6	789.8	10	100
	0.380	216	10	121.9	323.5	762.0	8	80
	0.369	222	10	116.8	231.4	960.2	10	100
90	0.360	250	11	171.2	472.9	1319.2	6	60
	0.349	258	11/12	196.6	413.2	1627.8	8	80
	0.338	266	12/13	197.1	398.9	1789.1	10	100
98	0.301	326	15/16	320.0	529.7	4548.9	10	100

Table I (cont.)

For $n \geq 50$ and all densities the solution vector was always the shortest one in the constructed basis. Moreover, when the LS algorithm failed all of the vectors in the basis were longer than $n/2$. On the other hand for $n = 26$, $n = 34$ and all densities there were cases when the basis contained vectors of length smaller than $n/2$. When n was 42 the solution vector was always the shortest one for densities $d < 0.7$.

Several regularities in the results of experiments can be observed. For example for fixed n, as the density decreases the multiprecision phase (the MP algorithm) takes more time, the overall time spent by the LS algorithm decreases and more solutions are found. Some exceptions from these rules can be noticed around densities for which the number of iterations of the step MP2 changes. Another reason for some irregularities is the early abortion strategy of our implementation.

The total *CPU time* used in the experiments reported in table I was 11 weeks, 3 days, 14 hours and 19 minutes.

The most important observation of the results gathered in table I is that the average time of the multiprecision phase has growth rate slightly smaller than $n \cdot (\log_2 M)^2$. This is an order of magnitude better than the experimental time for the SV algorithm.

6. Closing Remarks

We believe that if more care is taken in further refinement of the *Weight-Reduction* algorithm then the LS algorithm would succeed for higher densities. *Weight-Reduction* algorithm has a common point with the method of Schnorr [12], namely looking simultaneously at more then two vectors during the reduction process. Incorporating ideas of Schnorr could also improve the performance of our algorithm.

In section 2 we have mentioned that the implementation of the L^3 algorithm resembles the sequence of comparisons in the bubble-sort algorithm. It may be worthwhile to analyze other basis reduction algorithms with the sequence of transformations T1 and T2 based on some more efficient sorting method.

Finally, let us mention one application of the LS algorithm in solving several subset sum problems simultaneously, appearing in the theory of t-designs. A version of the LS algorithm was used to solve the system of 99 Diophantine equations in 132 unknowns for a (0,1)-solution vector, leading to the discovery of a much sought after new geometry, a simple 6-(14,7,4) design [6]. This version of the LS algorithm is described in [7].

Acknowledgements

Both authors' research was supported in part by the National Science Foundation under Grant DCR-8606378.

References

1. E. Brickell, Are most low density knapsacks solvable in polynomial time? In *Proceedings of the 14-th Southeastern Conference on Combinatorics, Graph Theory and Computing,* Congressus Numerantium, Vol.39 (1983), 145-156.
2. Y.G. Desmedt, J.P. Vandewalle and R.J.M. Govaerts, A Critical Analysis of the Security of Knapsack Public-Key Algorithms, *IEEE Transactions on Information Theory,* Vol.IT-30 No.4 (1984), 601-611.

3. A. M. Frieze, On the Lagarias-Odlyzko Algorithm for the Subset Sum Problem, *SIAM Journal on Computing*, Vol. 15, No. 2 (1986).

4. 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.

5. D.E. Knuth, *The Art of Computer Programming, Seminumerical Algorithms*, Vol.2, Addison-Wesley, 1981.

6. D.L. Kreher and S.P. Radziszowski, The Existence of Simple 6-(14,7,4) Designs, *Journal of Combinatorial Theory, Series A*, 43, No. 2 (1986) 237-243.

7. D.L. Kreher and S.P. Radziszowski, Finding Simple t-Designs by Basis Reduction, *Congressus Numerantium*, 55 (1986) 235-244.

8. J.C. Lagarias and A.M. Odlyzko, Solving Low-Density Subset Sum Problem, *Journal of the ACM*, Vol.32, No.1 (1985), 229-246.

9. D.H. Lehmer, Euclid's Algorithm for Large Numbers, *American Mathematical Monthly*, Vol.45 (1938), 227-233.

10. H.W. Lenstra, Jr. Integer Programming with a Fixed Number of Variables, *Mathematics of Operations Research*, Vol.8, No.4 (1983), 538-548.

11. A.K. Lenstra, H.W. Lenstra and L. Lovász, Factoring Polynomials with Rational Coefficients, *Mathematische Annalen*, 261 (1982), 515-534.

12. C. P. Schnorr, A Hierarchy of Polynomial-Time Basis Reduction Algorithm, *preliminary version*, August 1985.

13. A. Shamir, Embedding Cryptographic Trapdoors in Arbitrary Knapsack Systems, MIT, LCS, TM-230 (1982).

14. A. Shamir, A polynomial time algorithm for breaking the Merkle-Hellman cryptosystem. In *Proceedings of the 23-rd IEEE Symposium on Foundations of Computer Science*, IEEE, New York (1982), 145-152.