Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems.

C. P. Schnorr
M. Euchner
Universität Frankfurt
Fachbereich Mathematik/Informatik
Postfach 111932
6000 Frankfurt am Main
Germany

July 1993

Abstract

We report on improved practical algorithms for lattice basis reduction. We propose a practical floating point version of the L^3–algorithm of Lenstra, Lenstra, Lovász (1982). We present a variant of the L^3–algorithm with "deep insertions" and a practical algorithm for block Korkin–Zolotarev reduction, a concept introduced by Schnorr (1987). Empirical tests show that the strongest of these algorithms solves almost all subset sum problems with up to 66 random weights of arbitrary bit length within at most a few hours on a UNISYS 6000/70 or within a couple of minutes on a SPARC 1+ computer.

1 Introduction and Survey

It is a major open problem to determine the exact complexity of finding short vectors in a lattice. On the one hand the problem of finding a non–zero lattice vector that is shortest in the sup–norm is known to be NP–complete [4] (in its feasibility recognition form). On the other hand the L^3–lattice basis reduction algorithm of Lenstra, Lenstra, Lovász [17] is a polynomial time algorithm that finds a non–zero vector in an m–dimensional lattice that is guaranteed to be at most $2^{m/2}$–times the length of the shortest non–zero vector in that lattice. The L^3–algorithm finds in practice much shorter vectors than is guaranteed by the worst case $2^{m/2}$–bound. The performance of the L^3 has been further improved by suitable modifications [5,15,22], and new algorithms are being invented [19,23,24,27]. Possibly finding reasonably short vectors in a random lattice is not so difficult on the average. This would have important consequences for solving linear and non–linear integer programming problems.

Several attempts have been made to improve on the performance of the L^3–algorithm for lattice reduction. Recently Seysen [27] and Schnorr [23,24] have invented new algorithms for basis reduction in the square norm. Seysen's method performs extremely well for lattices of dimension up to 30. It operates on small integers, the intermediate integers for Seysen's algorithm are not larger than the input integers. Schnorr [23] has extended the L^3–reduction to a hierarchy of polynomial time reduction algorithms that find a non–zero vector in an m–dimensional lattice that is guaranteed to be at most $(1 + \varepsilon)^m$–times the length of the shortest non–zero vector in that lattice. The degree of the polynomial that bounds the running time increases as ε converges to zero. A different approach to improve on lattice reduction has been made by Lovász and Scarf (1992). They propose a generalised lattice reduction algorithm that works for an arbitrary norm. This general approach is tailor–made for certain integer programming problems.

A bottleneck for the speed of the L^3–algorithm is the required exact arithmetic on large integers. Most of these arithmetic steps occur in the process of Gram–Schmidt orthogonalizing the basis vectors. It has been proposed to perform this orthogonalization in floating point arithmetic while keeping the basis vectors in exact integer representation. This however makes the L^3–algorithm unstable.

In this paper we present a practical floating point L^3–algorithm, L^3FP, having good stability according to empirical tests up to dimension 125 with integer entries of bit length up to 300. We also propose a practical algorithm for block Korkin–Zolotarev reduction and we introduce the variant of the L^3–algorithm that uses "deep insertions". These algorithms produce considerably shorter lattice vectors than the original L^3–algorithm. They perform well in practice but may be inefficient in worst case. We report on the performance of all these algorithms in solving subset sum problems. These algorithms have also been applied to solve the diophantine approximation problem that yields the factorization of a given integer [25]. However to make this approach work for large integers further progress in basis reduction is needed.

The knapsack or subset sum problem is to solve, given positive integers $a_1, \ldots, a_n$ and s, the equation

$$\sum_{i=1}^{n} a_i x_i = s \text{ with } x_1, \ldots, x_n \in \{0, 1\}.$$

The Brickell [1] and the Lagarias–Odlyzko [14] algorithms solve almost all subset sum problems for which the *density* $d = n \,/\, \log_2 \max a_i$ is sufficiently small. Radziszowski and Kreher [17] evaluate the performance of an improved variant of the Lagarias–Odlyzko algorithm. In this paper we replace in the Lagarias–Odlyzko method the L^3–algorithm, by more powerful reduction algorithms, namely by the L^3–algorithm with "deep insertions" and by block Korkin–Zolotarev reduction. We also replace the Lagarias–Odlyzko lattice by a lattice (1) – see section 7 – that is better suited to produce $0, 1$–solutions for the knapsack equation. Empirical tests show that these algorithms solve almost all subset sum problems that have either sufficiently low or sufficiently high density. The hardest subset sum problems turn out to be those that have a density that is slightly larger than 1, i.e. a density about $1 + (\log_2(n/2))/n$. The new lattice basis (1) and the stronger reduction algorithms lead to a substantially improved success rate of subset sum algorithms. Using block Korkin–Zolotarev reduction with block size 20 we can solve almost all subset sum problems of dimension up to 58 even if their density is close to 1. It has been proved rigorously that for almost all subset sum problems with density less that 0.9408 the shortest non–zero vector in the associated lattice basis (1) yields a solution of the subset sum problem [3]. In section 6 we describe a particular practical algorithm for block Korkin–Zolotarev reduction. Using the improved reduction algorithms we can solve a much larger class of subset sum problems than was previously possible. Some empirical data are given in section 7. Several alternative algorithms for block Korkin–Zolotarev reduction and more empirical data are given in the master thesis of M. Euchner [5]. Another empirical comparison of the success rates for the new lattice basis (1) versus the Lagarias–Odlyzko lattice has been done by LaMacchia [15]. His success rates roughly correspond to our success rates using the weakest of our reduction methods, L^3–reduction in floating point arithmetic (algorithm L^3FP), see the comments in section 7.

Early versions of the new practical algorithms and the improved lattice (1) have been developed during the courses on lattice basis reduction which the first author gave at Frankfurt University in summer 1990. This work has been mentioned in the talk of the first author at the workshop on cryptography at Princeton University in September 1990 and has influenced the subsequent work in [3,10,15].

2 Basic concepts, L^3–reduction

Let $\mathbb{R}^n$ be the n–dimensional real vector space with the ordinary inner product $<,>$ and Euclidean length $\|y\| = \langle y, y\rangle^{1/2}$. A discrete, additive subgroup $L \subset \mathbb{R}^n$ is called a *lattice.* Every lattice L is generated by some set of linearly independent vectors $b_1, \ldots, b_m \in L$, called a *basis* of L,

$$L = \{t_1 b_1 + \cdots + t_n b_m \mid t_1, \ldots, t_m \in \mathbb{Z}\}.$$

Let $L(b_1, \ldots, b_m)$ denote the lattice with basis $b_1, \ldots, b_m$. Its *rank* or *dimension* is m and the *determinant* is defined as $d(L) = \det[\langle b_i, b_j\rangle_{1\le i,j\le m}]^{1/2}$. The rank and the determinant of the lattice do not depend on the choice of the basis. Let $b_1, \ldots, b_m \in \mathbb{R}^n$ be a basis of lattice L then $\bar{b}_1, \ldots, \bar{b}_m$ is another basis of L if and only if there exists a matrix $T \in GL_m(\mathbb{Z})$ such that

$$[b_1, \ldots, b_m] = [\bar{b}_1, \ldots, \bar{b}_m]\ T.$$

Here $[b_1, \ldots, b_m]$ denotes the $n \times m$ matrix in $M_{n,m}(\mathbb{R})$ with column vectors $b_1, \ldots, b_m$. 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 $b_1, \ldots, b_m \in \mathbb{R}^n$ we associate the *Gram–Schmidt orthogonalization* $\hat{b}_1, \ldots, \hat{b}_m \in \mathbb{R}^n$ which can be computed from $b_1, \ldots, b_m$ together with the Gram–Schmidt coefficients $\mu_{i,j} = \langle b_i, \hat{b}_j\rangle \,/\, \langle \hat{b}_j, \hat{b}_j\rangle$ by the recursion

$$\hat{b}_1 = b_1, \quad \hat{b}_i = b_i - \sum_{j=1}^{i-1} \mu_{i,j} \hat{b}_j \quad \text{for} \quad i = 2, \ldots, m.$$

We have $\mu_{i,i} = 1$ and $\mu_{i,j} = 0$ for $i < j$. The vectors $\hat{b}_1, \ldots, \hat{b}_m$ are linearly independent, they are not necessarily in the lattice. If the basis $b_1, \ldots, b_m$ is integral, i.e. $b_1, \ldots, b_m \in \mathbb{Z}^n$, then the vectors $\hat{b}_1, \ldots, \hat{b}_m$ and the coefficients $\mu_{i,j}$ are rational. We can write the above equations in matrix notation as

$$[b_1, \ldots, b_m] = [\hat{b}_1, \ldots, \hat{b}_m] \; [\mu_{i,j}]^{\top}_{1 \le i,j \le m}.$$

An ordered basis $b_1, \ldots, b_m \in \mathbb{R}^n$ is called *size–reduced* if

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

An individual basis vector b_i is *size–reduced* if $|\mu_{i,j}| \le 1/2$ for $1 \le j < i$.

Let δ be a constant, $1/4 < \delta \le 1$. Following [17] we call a basis $b_1, \ldots, b_m \in \mathbb{R}^n$ *L^3–reduced with δ* if it is size–reduced and if

$$\delta \; \|\hat{b}_{k-1}\|^2 \le \|\hat{b}_k + \mu_{k,k-1} \hat{b}_{k-1}\|^2 \quad \text{for } k = 2, \ldots, m.$$

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

Let $\lambda_1, \ldots, \lambda_m$ denote the 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 L^3–reduced basis consists of relatively short lattice vectors.

Theorem 1 [17] *Every basis $b_1, \ldots, b_m$ that is L^3–reduced with δ satisfies*

$$\alpha^{1-i} \le \|b_i\|^2 \; \lambda_i^{-2} \le \alpha^{m-1} \; \textit{for } i = 1, \ldots, m \textit{ with } \alpha = (\delta - 1/4)^{-1}.$$

The case $\delta = 3/4$ of Theorem 1 has been settled in [17]. This proof can easily be extended to all δ, $1/4 < \delta \le 1$.

We next describe some basic reduction algorithms. We restrict ourselves to integer input bases. For a real number r let $\lceil r \rfloor \in \mathbb{Z}$ denote the nearest integer, $\lceil r \rfloor = \lceil r - 1/2 \rceil$, with $\lceil r \rfloor = r - 1/2$ for half integers $r \in (2\mathbb{Z}+1)/2$.

Algorithm for size–reduction of the basis vector b_k .

```
INPUT   $b_1, \ldots, b_m \in \mathbb{Z}^n$   (a lattice basis)
        $\mu_{i,j}$ for $1 \le j < i \le m$   (its Gram–Schmidt coefficients)
FOR  $j = k-1, \ldots, 1$  DO
    IF $|\mu_{k,j}| > 1/2$ THEN [ $b_k := b_k - \lceil \mu_{k,j} \rfloor \, b_j$ ,
        FOR $i = 1, \ldots, m$ DO $\mu_{k,i} := \mu_{k,i} - \lceil \mu_{k,j} \rfloor \, \mu_{j,i}$ ]
OUTPUT  $b_1, \ldots, b_m$  (basis where $b_k$ is size–reduced)
        $\mu_{i,j}$ for $1 \le j < i \le m$  (its Gram–Schmidt coefficients)
```

We obtain a size–reduced basis $b_1, \ldots, b_m$ by size–reducing each vector individually. Size–reducing the vector b_k does not affect the size–reduction of the other vectors.

Algorithm for L^3–reduction (according to [17])

```
INPUT   $b_1, \ldots, b_m \in \mathbb{Z}^n$   (a lattice basis), $\delta$ with $1/4 < \delta < 1$.
(initiation)   $k := 2$   ($k$ is the stage)
               compute the Gram–Schmidt coefficients $\mu_{i,j}$ for
               $1 \le j < i \le m$ and $\|\hat{b}_i\|^2$ for $i = 1, \ldots, m$.
WHILE   $k \le m$   DO
    size–reduce the vector $b_k$ and update $\mu_{k,j}$ for $j = 1, \ldots, k-1$.
    IF   $\delta\, \|\hat{b}_{k-1}\|^2 > \|\hat{b}_k\|^2 + \mu_{k,k-1}^2 \|\hat{b}_{k-1}\|^2$
        THEN  [swap $b_k$ and $b_{k-1}$,  $k := \max(k-1, 2)$]
        ELSE  $k := k + 1$
OUTPUT  $b_1, \ldots, b_m$  (a basis that is $L^3$–reduced with $\delta$).
```

REMARKS 1. Upon entry of stage k the basis $b_1, \ldots, b_{k-1}$ is L^3–reduced with δ.
2. For every swap of b_{k-1}, b_k we must update $\|\hat{b}_k\|^2$, $\|\hat{b}_{k-1}\|^2$ and $\mu_{i,\nu}, \mu_{\nu,i}$ for $\nu = k, k-1$ and $i = 1, \ldots, m$, see [17].
3. In the original L^3–algorithm only the first step $b_k := b_k - \lceil \mu_{k,k-1} \rfloor b_{k-1}$ of size–reducing b_k is done before the IF step and the size–reduction of b_k is completed before incrementing k to $k+1$.
4. Let B denote $\max(\|b_1\|^2, \ldots, \|b_m\|^2)$ for the input basis. Throughout the algorithm the bit length of the numerators and denominators of the rational numbers $\|\hat{b}_i\|^2$, $\mu_{i,j}$ is bounded as $O(m \log B)$. The bit length of the coefficients of the $b_i \in \mathbb{Z}^n$ is also bounded as $O(m \log B)$ throughout the algorithm [17].
5. The algorithm terminates after at most $\binom{m}{2} \log_{1/\delta} B$ iterations. It performs at most $O(m^3 n \log B)$ arithmetic operations on integers that are $O(m \log B)$ bits long, see [17].

In practical applications the above L^3–algorithm is suffering from the slowness of the subroutines for long integer arithmetic. To speed up the algorithm it has been proposed to operate the numbers $\mu_{i,j}$ and $\|\hat{b}_i\|^2$ in floating point arithmetic. Then however the above algorithm becomes unstable and it has to be rewritten to minimize floating point errors. This will be done in section 3.

3 L^3–reduction using floating point arithmetic

In the following algorithm for L^3–reduction we keep the basis vectors $b_1, \dots, b_m \in \mathbb{Z}^n$ in exact representation and the numbers $\mu_{i,j}$, $\|\hat{b}_i\|^2$ in floating point. The basis must be exact since errors in the basis change the lattice and cannot be corrected. All other errors can be corrected using a correct basis. The following provisions are taken to minimize the floating point errors. We let v' denote the floating point value corresponding to an exact value v. Let the integer τ denote the number of precision bits in the floating point arithmetic.

1. Whenever we enter stage k we compute from the actual basis vectors $b_1, \dots, b_k$ the numbers $\mu_{k,j}$ for $j = 1, \dots, k-1$ and also $c_k = \|\hat{b}_k\|^2$. This will correct these values since the vectors $b_1, \dots, b_k$ are exact.

2. If a large reduction coefficient, $|\lceil \mu_{k,j} \rfloor| > 2^{\tau/2}$, occurs during the size–reduction of b_k then we subsequently decrease the stage k to $k-1$. This will correct the coefficients $\mu_{k-1,j}$ and $\mu_{k,j}$ for $j = 1, \dots, k-1$ as well as $c_{k-1}, c_k, b'_{k-1}, b'_k$.

3. If $|\langle b'_k, b'_j \rangle| < 2^{-\tau/2} \; \|b'_k\| \; \|b'_j\|$ then we compute $\langle b_k, b_j \rangle'$ instead of $\langle b'_k, b'_j \rangle$. Since the leading bits in the computation of $\langle b'_k, b'_j \rangle$ cancel out the value $\langle b'_k, b'_j \rangle$ is too inexact.

Algorithm L^{3}FP, L^3–reduction in floating point arithmetic

INPUT $b_1, \dots, b_m \in \mathbb{Z}^n$ (a lattice basis), δ with $1/2 < \delta < 1$.

```
1. (initiation)  $k := 2$,  $F_c :=$ false
   ($k$ is the stage. The following values are available upon entry of stage
   $k$: $\mu_{i,j}$ for $1 \le j < i < k$ and $c_i = \|\hat{b}_i\|^2$ for $i = 1, \dots, k-1$)
   FOR $i = 1, \dots, m$ DO $b'_i := (b_i)'$

2. WHILE $k \le m$ DO
   (computation of $\mu_{k,1}, \dots, \mu_{k,k-1}$, $c_k = \|\hat{b}_k\|^2$)
         $c_k := \|b'_k\|^2$,  IF $k = 2$ THEN $c_1 := \|b'_1\|^2$
         FOR $j = 1, \dots, k-1$ DO
               IF $|\langle b'_k, b'_j \rangle| < 2^{-\tau/2} \; \|b'_k\| \; \|b'_j\|$
                  THEN $s := \langle b_k, b_j \rangle'$
                  ELSE $s := \langle b'_k, b'_j \rangle$
```

```
        $\mu_{k,j} := (s - \sum_{i=1}^{j-1} \mu_{j,i}\ \mu_{k,i}\ c_i)/c_j$
        $c_k := c_k - \mu_{k,j}^2\ c_j$
3. (size–reduction of $b_k$)
    FOR $j = k-1, \ldots, 1$ DO
        IF $|\mu_{k,j}| > 1/2$ THEN
            $\mu := \lceil \mu_{k,j} \rfloor$
            IF $|\mu| > 2^{\tau/2}$ THEN $F_c :=$ true
            FOR $i = 1, \ldots, j-1$ DO $\mu_{k,i} := \mu_{k,i} - \mu\mu_{j,i}$
            $\mu_{k,j} := \mu_{k,j} - \mu,\ b_k := b_k - \mu\ b_j,\ b'_k := (b_k)'$
        END if $|\mu_{k,j}|$
    IF $F_c$ THEN [ $F_c :=$ false, $k := \max(k-1, 2)$, GOTO 2 ]
4. (swap $b_{k-1}, b_k$ or increment $k$)
    IF $\delta\ c_{k-1} > c_k + \mu_{k,k-1}^2\ c_{k-1}$
        THEN [ swap $b_k, b_{k-1}$ swap $b'_k, b'_{k-1}$
               $k := \max(k-1, 2)$ ]
        ELSE $k := k+1$
OUTPUT $b_1, \ldots, b_m$ (a basis that is $L^3$–reduced with $\delta$).
```

COMMENTS. 1. According to our experience the algorithm L^3FP has good stability even for single precision floating point arithmetic and for very large input vectors. Double precision arithmetic results in a considerable decrease of the number of swaps and in a faster algorithm. The point is that L^3FP performs reduction with respect to the leading bits of the basis vectors handling about τ of these bits at the same time, where τ is the number of precision bits of the floating point arithmetic. Thus the number of swaps in L^3FP is proportional to $\log_2 B/\tau$ times the number of swaps in the L^3–algorithm.
2. We cannot prove that L^3FP always terminates. If the floating point precision is too small compared to the length of the input vectors L^3FP might run into cycles that are caused by floating point errors. However the algorithm was successful in several thousand applications with lattices of rank up to 125 and where the bit length of the input integers was up to 300.
3. Schnorr [24] has given an algorithm for L^3–reduction with provably negligible floating point errors. Practical versions of this algorithm are about 10% slower than the above algorithm L^3FP. The reduction algorithm in [24] uses the coefficients $\nu_{i,j}$ of the inverse matrix $[\nu_{i,j}] = [\mu_{i,j}]^{-1}_{1 \le i,j \le m}$. It

corrects floating point errors via the scalar products $\langle b_i, b_j \rangle$.
4. The flag F_c is set true if a correction step has to be performed. In this case k will be decreased to $k-1$ and the $\mu_{i,j}, \|b_i\|^2$ will be corrected for $i = k-1$ and $i = k$.
5. To offset small floating point errors one has to use δ–values that are larger than 1/4, e.g. $\delta \geq 1/2$.

The following "deep insertion" step extends the swap $b_k \leftrightarrow b_{k-1}$ of the L^3–algorithm. By replacing Step 4 of algorithm L^3FP by the "deep insertion" step we obtain a variant of L^3FP that finds shorter lattice vectors.

New Step 4 (deep insertion of b_k)

```
    $c := \|b'_k\|^2,\ i := 1$
    WHILE $i < k$ DO
       IF $\delta\ c_i \leq c$
          THEN [ $c := c - \mu_{k,i}^2\ c_i,\ i := i + 1$ ]
          ELSE [ $(b_1, \ldots, b_k) := (b_1, \ldots, b_{i-1}, b_k,\ b_i, \ldots, b_{k-1})$
                 rearrange the $b'_j$ accordingly
                 $k := \max(i-1, 2)$,  GOTO 2 ]
    $k := k + 1$
```

COMMENTS. 1. A deep insertion possibly inserts b_k at some position $i < k$ and increments the indices of the old vectors $b_i, \ldots, b_{k-1}$ by 1. The position i is chosen as the minimal i which results in decreasing $c_i = \|\widehat{b_i}\|^2$ by at least a factor δ. Throughout Step 4 c is the length square of the vector $\hat{b}_i^{\text{new}}$ in case that a deep insertion step of b_k at position i is performed.
2. Algorithm L^3FP with deep insertions may be super–polynomial time in worst case. If the deep insertions are only performed in case that either $i \leq c_o$ or $k - i \leq c_o$ for a fixed constant c_o then the deep insertion variant of L^3FP remains polynomial time.

4 L^3–reduction of a linearly dependent generator system

We can extend algorithm L^3FP so that it transforms every generator system $b_1, \ldots, b_m \in \mathbb{Z}^n$ of a lattice into an L^3–reduced basis. If the vectors $b_1, \ldots, b_m$ are linearly dependent then the associated Gram–Schmidt orthogonalization $\hat{b}_1, \ldots, \hat{b}_m$ contains at least one zero–vector, which leaves us with another problem.

We must avoid increasing the stage to $k+1$ in case that $c_k = \|\widehat{b}_k\|^2$ is zero because then a division with c_k is done on the next stage $k+1$. Fortunately, if c_k is zero the condition $\delta c_{k-1} > c_k + \mu^2_{k,k-1} c_{k-1}$ for swapping b_{k-1}, b_k is satisfied since we have $\mu^2_{k,k-1} < 1/4$ and $\delta \geq 1/2$. If $k > 2$ this swap of b_{k-1}, b_k and the decrease of k will avoid a subsequent division by zero. However if $k = 2$ and $c_2 = 0$ swapping b_2, b_1 may result in a zero–vector b_1. We can simply eliminate this zero–vector b_1 from the basis. Going one step further we check after each size–reduction of b_k in Step 3 of L^3FP whether the reduced vector b_k is zero and in this case we eliminate b_k from the basis. This will correctly deal with all cases provided that initially b_1 is not the zero–vector.

Thus we insert into Step 3 of L^3FP after the reduction of b_k and before changing k the following assignment:

Additional assignment for Step 3 of L^{3}FP

```
IF  $b_k = 0$  THEN [ eliminate $b_k$,  $m := m - 1$,  $k := 2$,  GOTO 2 ]
```

We suppose that this assignment is always included in L^3FP if the input vectors $b_1, \ldots, b_m$ are linearly dependent. We call the algorithm L^3FP with the additional assignment the *extended* L^3FP.

Remarks. 1. The initial comments and the termination of the extended L^3FP (which is proved in Theorem 2 below) show that the extended L^3FP is correct up to floating point errors. It is sufficient to note that the vectors $b_1, \ldots, b_{k-1}$ of Stage k are always L^3–reduced with δ and that the algorithm terminates on Stage $m+1$. Thus the output vectors $b_1, \ldots, b_m$ form a basis that is L^3–reduced with δ.
2. Since the vectors $b_1, \ldots, b_{k-1}$ of Stage k are L^3–reduced with δ we see from Theorem 1 that $c_j \geq \alpha^{1-j}\|b_1\|^2$ holds for $j = 1, \ldots, k-1$ where $\alpha = 1/(\delta - 1/4) \leq 4$. Therefore the divisors c_j of Step 2 are sufficiently apart from 0. This helps to minimize floating point errors.
3. Resetting the stage k to 2, in the additional assignment for step 3, is a precaution against floating point errors. The generation of a zero–vector b_k

produces some floating point errors that are due to the fairly small vectors $\widehat{b}_i$ occuring within this process.

We present an upper bound on the number of swaps $b_{k-1} \leftrightarrow b_k$ in the extended L^3FP relying on the following integer quantity D:

$$D = \prod_{\hat{b}_i \neq 0} D_i \quad \text{with} \quad D_i = \det L(b_1, \ldots, b_i)^2 \ ,$$

where i ranges over all indices $1 \leq i \leq m-1$ with $\widehat{b}_i \neq 0$. The quantity D extends the corresponding D in [17] to the case of linearly dependent input vectors $b_1, \ldots, b_m$.

Theorem 2 *If the extended L^3FP is performed in exact arithmetic we have that*

1. *every swap $b_{k-1} \leftrightarrow b_k$ achieves $D^{\text{new}} \leq \delta\, D^{\text{old}}$,*
2. *the total number of swaps $b_{k-1} \leftrightarrow b_k$ is at most $\binom{m}{2} \log_{1/\delta} B$ where B is the maximum length square $\|b_i\|^2$ of the input vectors $b_i \in \mathbb{Z}^n$ for $i = 1, \ldots, m$.*

Proof. 1. A swap of the vectors b_{k-1}, b_k leaves D_i for $i \neq k-1$ unchanged. If $\widehat{b}_{k-1}^{\text{new}} \neq 0$ then $\widehat{b}_i^{\text{new}}$, $\widehat{b}_i^{\text{old}}$ are zero for the same i. We have

$$D^{\text{new}} = D^{\text{old}} \ \|\widehat{b}_{k-1}^{\text{new}}\|^2 \ \|\widehat{b}_{k-1}^{\text{old}}\|^{-2} \leq \delta \ D^{\text{old}}$$

since the swap reduces $\|\widehat{b}_{k-1}\|^2$ at least by a factor δ. This proves the claim for the case that $\widehat{b}_{k-1}^{\text{new}} \neq 0$.
In the case $\widehat{b}_{k-1}^{\text{new}} = 0$ we have $\widehat{b}_k^{\text{old}} = 0$ and thus

$$D^{\text{old}} = \prod_{i \neq k} D_i^{\text{old}}, \quad D^{\text{new}} = \prod_{i \neq k-1} D_i^{\text{new}}$$

$$D^{\text{new}} \,/\, D^{\text{old}} = D_k^{\text{new}} \,/\, D_{k-1}^{\text{old}}.$$

Now the lattice $L(b_1, \ldots, b_{k-1}^{\text{old}})$ has the same rank as the lattice $L(b_1, \ldots, b_{k-1}^{\text{old}}, b_k^{\text{old}}) = L(b_1, \ldots, b_{k-1}^{\text{new}}, b_k^{\text{new}})$ and it is properly contained in the latter lattice. This is because $b_k^{\text{old}} \notin L(b_1, \ldots, b_{k-1}^{\text{old}})$, which holds since b_k^{old} is size–reduced and $b_k^{\text{old}} \neq 0$. The proper inclusion of the above lattice and the integrality of D_k, D_{k-1} implies that $D_k^{\text{new}} \leq D_{k-1}^{\text{old}} / 2$ and thus $D^{\text{new}} \leq D^{\text{old}}/2$.

2. This is an immediate consequence of (1) and the fact that the entity D remains a positive integer throughout the computation. □

Remarks. 1. Due to floating point errors the extended L^3FP performs more than $\binom{m}{2} \log_{1/\delta} B$ many swaps $b_{k-1} \leftrightarrow b_k$. The number of swaps is about $\tau^{-1} \log_2 B$ times this bound.

2. A somewhat different entity D has been used in [23]. There we defined $D' = \prod_{i=1}^{m-1} D'_i$ with

$$D'_i \;=\; \prod_{\substack{j=1 \\ \hat{b}_j \neq 0}}^{i} \|\widehat{b}_j\|^2.$$

A detailed analysis shows that every exchange $b_{k-1} \leftrightarrow b_k$ achieves

$$D'^{\text{new}} \leq \delta \; D'^{\text{old}} \qquad \text{if} \quad \widehat{b}_{k-1}^{\text{new}} \neq 0$$

$$D'^{\text{new}} \leq D'^{\text{old}} \qquad \text{if} \quad \widehat{b}_{k}^{\text{new}} = 0.$$

5 Block Korkin Zolotarev reduction

Let $L = L(b_1, \ldots, b_m) \subset \mathbb{R}^n$ be a lattice with ordered basis $b_1, \ldots, b_m$. Let $\pi_i : \mathbb{R}^n \to \text{span}(b_1, \ldots, b_{i-1})^{\perp}$ denote the orthogonal projection so that $b - \pi_i(b) \in \text{span}(b_1, \ldots, b_{i-1})$. We let L_i denote the lattice $\pi_i(L)$, which is a lattice of rank $m - i + 1$.

An ordered basis $b_1, \ldots, b_m$ of lattice L is a *Korkin–Zolotarev basis* if it is size–reduced and if

$$\|\hat{b}_i\| = \lambda_1(L_i) \quad \text{for} \quad i = 1, \ldots, m.$$

This definition is equivalent to the one given, in the language of quadratic forms, by Hermite in his second letter to Jacobi (1845) and by Korkin and Zolotarev (1873).

Theorem 3 shows the strength of Korkin–Zolotarev reduction compared to L^3–reduction, see Theorem 1.

Theorem 3 [13] *Every Korkin–Zolotarev basis $b_1, \ldots, b_m$ satisfies*

$$\frac{4}{(i+3)} \;\leq\; \|b_i\|^2 \,/\, \lambda_i^2 \;\leq\; \frac{i+3}{4} \quad \mathit{for} \quad i = 1, \ldots, m.$$

The fastest known algorithm for Korkin–Zolotarev reduction of a basis $b_1, \ldots, b_m \in \mathbb{Z}^n$ with $B = \max(\|b_1\|^2, \ldots, \|b_m\|^2)$ has a theoretic worst case time bound of $\sqrt{n}^{n+o(n)} + O(n^4 \log B)$ arithmetic steps on $O(n \log B)$–bit integers [23]. This algorithm is an improved version of Kannan's shortest lattice vector algorithm [11].

Schnorr [23] introduced the following notion of a block Korkin–Zolotarev reduced basis. Let β be an integer, $2 \leq \beta < m$.

A lattice basis $b_1, \ldots, b_m$ is *β–reduced* if it is size–reduced and if

$$\|\hat{b}_i\| \leq \lambda_1(L_i(b_1, \ldots, b_{\min(i+\beta-1,m)})) \quad \text{for} \quad i = 1, \ldots, m-1.$$

Let α_β denote the maximum of $\|b_1\| \,/\, \|\hat{b}_\beta\|$ taken over all Korkin–Zolotarev reduced basis $b_1, \ldots, b_\beta$. We have $\alpha_2 = \frac{4}{3}$, $\alpha_3 = \frac{3}{2}$ and $\alpha_\beta \leq \beta^{1+\ln\beta}$, where $\ln\beta$ is the natural logarithm of β [23]. The constant $\alpha_\beta^{1/(\beta-1)}$ slowly converges to 1 as β increases. The corresponding constant α in Theorem 1 is at least 4/3. The strength of β–reduced bases compared to L^3–reduced bases can be seen from the following

Theorem 4 [23] *Every β–reduced basis $b_1, \ldots, b_m$ of lattice L satisfies $\|b_1\|^2 \leq \alpha_\beta^{(m-1)/(\beta-1)} \lambda_1(L)^2$ provided that $\beta - 1$ divides $m - 1$.*

We call the basis $b_1, \ldots, b_m$ *β–reduced with δ*, $1/4 < \delta \leq 1$, if it is size–reduced and if

$$\delta \, \|\widehat{b_i}\|^2 \leq \lambda_1(L_i(b_1, \ldots, b_{\min(i+\beta-1,m)}))^2 \quad \text{for} \quad i = 1, \ldots, m-1.$$

Theorem 5 *A basis $b_1, \ldots, b_m \in \mathbb{R}^n$ is 2–reduced with δ, $1/3 \leq \delta \leq 1$, if and only if it is L^3–reduced with δ.*

Proof. " $\Rightarrow$ " If $b_1, \dots, b_m$ is 2–reduced with δ then we have

$$\delta \; \|\widehat{b}_k\|^2 \;\leq\; \|\pi_k(v_k b_k + v_{k+1} b_{k+1})\|^2$$

for all $(v_k, v_{k+1}) \in \mathbb{Z}^2 - 0$ and for $k = 1, \dots, m-1$. With $v_k = 0,\ v_{k+1} = 1$ this yields $\delta \; \|\widehat{b}_k\|^2 \leq \|\pi_k(b_{k+1})\|^2$.
" $\Leftarrow$ " We show that the inequality

$$\begin{aligned} \|\pi_k(v_k b_k + v_{k+1} b_{k+1})\|^2 &= (v_k + \mu_{k+1,k} v_{k+1})^2 \; \|\widehat{b}_k\|^2 + |v_{k+1}|^2 \|\widehat{b}_{k+1}\|^2 \\ &\geq \delta \; \|\widehat{b}_k\|^2 \end{aligned}$$

holds for all $(v_k, v_{k+1}) \in \mathbb{Z}^2 - (0,0)$.
If $v_{k+1} = 0$ this inequality clearly holds.
If $v_{k+1} = \pm 1$ the minimal value for $|v_k + \mu_{k+1,k} v_{k+1}|$ occurs at $v_k = 0$. This is because $|\mu_{k+1,k}| \leq 1/2$. From this and since the basis is L^3–reduced with δ we see that the desired lower bound $\|\pi_k(b_{k+1})\|^2 \geq \delta \; \|\widehat{b}_k\|^2$ holds.
If $|v_{k+1}| \geq 2$ the desired lower bound follows from

$$4 \; \|\widehat{b}_{k+1}\|^2 \geq 4(\delta - 1/4) \; \|\widehat{b}_k\|^2 \geq \delta \; \|\widehat{b}_k\|^2.$$

Here we use that $\delta \geq 1/3$ and that the basis is L^3–reduced with δ. □

The first part of the above proof does not require that $\delta \geq 1/3$, thus any basis that is 2–reduced with δ is also L^3–reduced with δ.

6 A practical algorithm for block Korkin Zolotarev reduction

The following algorithm BKZ performs a β–reduction with δ. It uses L^3FP and a subroutine $ENUM(j,k)$ defined below which minimizes the expression

$$c_j(u_j, \dots, u_k) := \sum_{s=j}^{k} (\sum_{i=s}^{k} u_i \mu_{i,s})^2 c_s$$

for $(u_j, \dots, u_k) \in \mathbb{Z}^{k-j+1} - 0^{k-j+1}$.

Algorithm BKZ for block Korkin–Zolotarev reduction

INPUT $b_1, \ldots, b_m \in \mathbb{Z}^n$, δ with $1/2 < \delta < 1$, β with $2 < \beta < m$.

```
1. $L^3FP(b_1, \ldots, b_m, \delta)$, $z := 0$, $j := 0$
   WHILE $z < m - 1$ DO
      $j := j + 1$, $k := \min(j + \beta - 1, m)$
      IF $j = m$ THEN [ $j := 1$, $k := \beta$ ]

2. $ENUM(j, k)$
      (this finds the minimal place $(u_j, \ldots, u_k) \in \mathbb{Z}^{k-j+1} - 0^{k-j+1}$
      and the minimal value $\bar{c}_j$ for $c_j(u_j, \ldots, u_k)$ and also
      $b_j^{\rm new} := \sum_{s=j}^{k} u_s b_s$).

3. $h := \min(k + 1, m)$
      IF $\delta c_j > \bar{c}_j$
         THEN [ $F_c :=$ true, call $L^3FP(b_1, \ldots, b_{j-1}, b_j^{\rm new}, b_j, \ldots, b_h, \delta)$
                at stage $j$, $z := 0$ ]
         ELSE [ $z := z + 1$, call $L^3FP(b_1, \ldots, b_h, 0.99)$ at stage $h - 1$ ]
```

OUTPUT $b_1, \ldots, b_m$ (a basis that is β–reduced with δ).

COMMENTS. 1. Throughout the algorithm the integer j is cyclically shifted through the integers $1, 2, \ldots, m - 1$. The variable z counts the number of positions j that satisfy the inequality $\delta \|\hat{b}_j\|^2 \leq \lambda_1(\pi_j(L(b_j, \ldots, b_k)))^2$. If this inequality does not hold for j then we insert $b_j^{\rm new}$ into the basis, we call L^3FP and we reset z to 0. The integer $j = m$ is skipped since the inequality always holds for $j = m$. Obviously a basis $b_1, \ldots, b_m$ is β–reduced with δ if it is size–reduced and $z = m - 1$. On termination the basis is size–reduced by the calls of L^3FP in Step 3 and we have $z = m - 1$. Therefore the algorithm produces, up to floating point errors, a basis that is β–reduced with δ.

2. The first call of L^3FP in Step 3 transforms the generator system $b_1, \ldots, b_{j-1}, b_j^{\rm new}$, $b_j, \ldots, b_h$ of lattice $L(b_1, \ldots, b_h)$ into a basis for $L(b_1, \ldots, b_h)$ that is L^3–reduced with δ. Alternatively we can extend $b_1, \ldots, b_{j-1}, b_j^{\rm new}$ to a basis $b_1, \ldots, b_{j-1}, b_j^{\rm new}, \ldots, b_h^{\rm new}$ of the lattice $L(b_1, \ldots, b_h)$ using the coefficients u_i in the representation $b_j^{\rm new} = \sum_{i=j}^{h} u_i b_i$. For this we compute $T \in GL_{h-j+1}(\mathbb{Z})$ with $[u_j, \ldots, u_h]T = [1, 0, \ldots, 0]$ and we set $[b_j^{\rm new}, \ldots, b_h^{\rm new}] := [b_j, \ldots, b_h]T^{-1}$.

3. Setting F_c to true in Step 3 before inserting the new vector $b_j^{\rm new}$ into the basis means that the $\mu_{j,i}$ that are generated next on stage j will be

corrected. This correction is necessary since some precision bits will be lost during the reduction in size of b_j^{new}.
4. The second call of L^3FP in Step 3 makes sure that the vectors $b_j, \dots, b_k$ are always L^3–reduced with δ when calling $ENUM(j, k)$.
5. We cannot prove that algorithm BKZ runs in polynomial time. However the algorithm behaves well in practice, see section 7.

Algorithm ENUM
INPUT j, k with $1 \le j < k \le m$
(the following parameters of BKZ are used: $b_j, \dots, b_k$, $c_i = \|\widehat{b}_i'\|^2$ for $i = j, \dots, k$ and $\mu_{i,t}$ for $j \le t < i \le k$)

```
1. $\bar{c}_j := c_j,\ \widetilde{u}_j := u_j := 1,\ y_j := \triangle_j := 0,\ s := t := j,\ \delta_j := 1$
   FOR $i = j+1, \dots, k+1$ DO $[\widetilde{c}_i := u_i := \widetilde{u}_i := y_i := \triangle_i := 0,\ \delta_i := 1]$

2. WHILE $t \le k$ DO
         $\widetilde{c}_t := \widetilde{c}_{t+1} + (y_t + \widetilde{u}_t)^2 c_t$
         IF $\widetilde{c}_t < \bar{c}_j$
              THEN IF $t > j$
                     THEN [ $t := t-1,\ y_t := \sum_{i=t+1}^{s} \widetilde{u}_i \mu_{i,t}$,
                            $\widetilde{u}_t := v_t := \lceil -y_t \rfloor,\ \triangle_t := 0$
                            IF $\widetilde{u}_t > -y_t$ THEN $\delta_t := -1$
                                                 ELSE $\delta_t := 1$ ]
                     ELSE $[\bar{c}_j := \widetilde{c}_j,\ u_i := \widetilde{u}_i$ for $i = j, \dots, k]$
              ELSE [ $t := t+1,\ s := \max(s, t)$
                     IF $t < s$ THEN $\triangle_t := -\triangle_t$
                     IF $\triangle_t \delta_t \ge 0$ THEN $\triangle_t := \triangle_t + \delta_t$
                     $\widetilde{u}_t := v_t + \triangle_t$ ]
   $b_j^{\mathrm{new}} := \sum_{i=j}^{k} u_i b_i$
```

OUTPUT the minimal place $(u_j, \dots, u_k) \in \mathbb{Z}^{k-j+1} - 0^{k-j+1}$
and the minimum $\bar{c}_j$ of $c_j(u_j, \dots, u_k)$ and b_j^{new}.

COMMENTS. 1. The algorithm ENUM enumerates in depth first search all integer vectors $(\widetilde{u}_t, \dots, \widetilde{u}_k)$ for $t = k, \dots, j$ that satisfy $c_t(\widetilde{u}_t, \dots, \widetilde{u}_k) < \bar{c}_j$

where $\bar{c}_j$ is the current minimum for the function c_j. The current minimal place is $(u_j, \ldots, u_k)$. We always have that $\widetilde{c}_t = c_t(\widetilde{u}_t, \ldots, \widetilde{u}_k)$ for the current vector $(\widetilde{u}_t, \ldots, \widetilde{u}_k)$. Redundancies have been eliminated so that the following holds throughout the enumeration. The largest i with $\widetilde{u}_i \neq 0$ satisfies $\widetilde{u}_i > 0$. This is because arriving at level t for the first time from level $t-1$ we set $\triangle_t = 1$ and $\widetilde{u}_t = 1$.

2. Throughout the enumeration s is the maximal previous value for t.

3. When initially we arrive at level t from level $t-1$ we have $y_t = \triangle_t = 0$ and $s = t$. Then we set $\triangle_t$ to 1 and $\widetilde{u}_t$ to 1. When subsequently level t is reached from level $t-1$ we take for $\triangle_t$ the next value in order $1, -1, 2, -2, 3, -3, \ldots$ as long as $\widetilde{c}_t \geq \bar{c}_j$. At this latter point we increment t to $t+1$ and s to $s+1$. When level t is reached from level $t+1$ we set $\triangle_t$ to 0 and we assign to δ_t the sign of $-y_t + \lceil -y_t \rceil$. When subsequently level t is reached from level $t-1$ we take for $\triangle_t$ the next value in either the order $1, -1, 2, -2, 3, -3 \cdots$, or in the order $-1, 1, -2, 2, -3, 3 \cdots$, as long as $\widetilde{c}_t \geq c_j$. (The choice of the order depends on δ_t and it is made so that the values $(y_t + \lceil -y_t \rfloor + \triangle_t)^2 c_t$ do not decrease for the chosen sequence $\triangle_t$.) At this latter point t is incremented to $t+1$.

4. Our original ENUM–algorithm, see [26], did not enumerate the values $(y_t + \lceil -y_t \rfloor + \triangle_t)c_t$ in increasing order. The new ENUM–algorithm is slightly better for block Korkin–Zolotarev reduction with pruning, see the end of section 7.

7 Solving subset sum problems

Given positive integers $a_1, \ldots, a_n, s$ we wish to solve the equation $\sum_{i=1}^n a_i x_i = s$ with $x_1, \ldots, x_n \in \{0, 1\}$. We associate to these integers the following basis $b_1, \ldots, b_{n+1} \in \mathbb{Z}^{n+2}$.

$$\begin{array}{rcl}
b_1 & = & (2, 0, \ldots, 0, na_1, 0) \\
b_2 & = & (0, 2, \ldots, 0, na_2, 0) \\
 & \vdots & \\
b_n & = & (0, 0, \ldots, 2, na_n, 0) \\
b_{n+1} & = & (1, 1, \ldots, 1, ns\ , 1).
\end{array} \tag{1}$$

Every lattice vector $z = (z_1, \ldots, z_{n+2}) \in L(b_1, \ldots, b_{n+1})$ that satisfies

$$|z_{n+2}| = 1, \quad z_{n+1} = 0, \quad z_1, \ldots, z_n \in \{\pm 1\} \tag{2}$$

yields the following solution for the subset sum problem

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

The following algorithm SUBSETSUM improves the Lagarias–Odlyzko algorithm [14] for solving low density subset sum problems in various ways. It uses the lattice basis (1) that is better suited than the Lagarias–Odlyzko basis. It has been proved rigorously that for almost all subset sum problems of density less than 0.9408 the shortest lattice vector yields a solution of the subset sum problem [3]. SUBSETSUM also uses superior algorithms for lattice basis reduction. Step 5 of the algorithm has already been used in [22].

Algorithm SUBSETSUM

```
INPUT   $a_1, \ldots, a_n, s \in \mathbb{N}$
  1. Compute the basis (1), let $b_i = (b_{i,1}, \ldots, b_{i,n+2})$ for $i = 1, \ldots, n+1$.
  2. Randomly permute $b_1, \ldots, b_{n+1}$ so that the permuted basis starts with
     the vectors $b_i$ satisfying $b_{i,n+2} \neq 0$.
  3. Reduce the basis $b_1, \ldots, b_{n+1}$, using modifications of
     $L^3FP(b_1, \ldots, b_{n+1}, 0.99)$ or $BKZ(b_1, \ldots, b_{n+1}, 0.99, \beta)$
  4. IF some vector $(z_1, \ldots, z_{n+2})$ in the reduced basis satisfies (2) THEN
     [OUTPUT $x_i = |z_i - z_{n+2}| / 2$ for $i = 1, \ldots, n$ and stop ]
  5. (reduce pairs of basis vectors)
     Sort $b_1, \ldots, b_{n+1}$ so that $\|b_1\| \leq \|b_2\| \leq \cdots \leq \|b_{n+1}\|$
     FOR $j = 1, \ldots, n$ FOR $k = 1, \ldots, j-1$ DO
           IF $\|b_j \pm b_k\| < \|b_j\|$ THEN [ $b_j := b_j \pm b_k$, $F :=$ true ]
     IF $F$ THEN [$F :=$ false, GOTO 5].
REPEAT steps 2 – 5  15–times.
```

M. Euchner has evaluated this algorithm as part of his master thesis. He used the following reduction subroutines in Step 3:
1) $L^3FP(b_1, \ldots, b_{n+1}, 0.99)$,
2) $L^3FP(b_1, \ldots, b_{n+1}, 0.99)$ with deep insertions,
3) $BKZ(b_1, \ldots, b_{n+1}, 0.99, 10)$,
4) $BKZ(b_1, \ldots, b_{n+1}, 0.99, 20)$.

In order to optimize the program M. Euchner has added the following features. He checks after each size–reduction whether the reduced vector b_k satisfies (2) and yields a solution. He incorporates the deep insertion rule

$$(b_1, \dots, b_k) := (b_1, \dots, b_{i-1}, b_k, b_i, \dots, b_{k-1})$$

for indices $i \leq 5$ and arbitrary k. He assumes that $\sum_{i=1}^{n} x_i = n/2$ holds for the solution $(x_1, \dots, x_n)$ and therefore extends the vectors b_i in (1) by adding the component $b_{i,n+3} = n$ for $i = 1, \dots, n$ and $b_{n+1,n+3} = n^2/2$.

Statistical evaluation of the algorithm Every row with first entries n, b in the following statistic corresponds to 20 random inputs for SUBSETSUM that are generated as follows. Pick random numbers $a_1, \dots, a_n$ in the interval $[1, 2^b]$, pick a random subset $I \subset \{1, \dots, n\}$ of size $n/2$, put $s = \sum_{i \in I} a_i$. The numbers in columns suc_1, #suc are the number of successes in round 0 of steps $2-5$ and the total number of successes in all rounds for these 20 inputs. The number in column # rou gives the total number of rounds of steps $2-5$ for the 20 inputs. There is a minimum of 20 and a maximum of $16 \cdot 20 = 320$ rounds. The column $hh : mm : ss$ gives the average CPU–time per problem on a UNISYS 6000/70. The times marked with * are on a SPARC 1+. On a SPARC 1+ computer our programs are about 6 times faster.

		L^3FP, $\delta = 0.99$				L^3FP, $\delta = 0.99$, with deep insertions			
n	b	suc$_1$	#suc	#rou	hh:mm:ss	suc$_1$	#suc	#rou	hh:mm:ss
42	24	20	20	20	0:39	20	20	20	0:51
42	28	13	20	33	1:22	17	20	25	1:59
42	32	2	19	65	3:05	14	20	51	4:00
42	36	2	20	98	4:49	13	19	52	4:42
42	40	4	17	124	6:11	17	19	47	4:18
42	44	7	20	65	3:50	17	20	27	3:23
42	48	10	20	42	2:51	19	20	21	2:50
42	52	17	20	23	1:56	20	20	20	2:34
42	56	19	20	22	1:59	20	20	20	2:31
42	60	19	20	21	1:56	20	20	20	2:39
50	26	16	20	25	1:23	20	20	20	1:42
50	30	7	20	45	3:10	17	20	24	4:07
50	34	4	20	79	6:11	10	20	39	7:25
50	38	1	17	126	10:17	8	19	68	14:43
50	42	0	10	258	22:16	11	19	68	14:50
50	46	0	6	265	23:37	8	17	91	20:53
50	50	0	12	212	19:32	5	19	72	20:11
50	54	1	15	172	16:26	13	20	34	12:17
50	58	4	17	139	14:17	18	20	22	8:57
50	62	5	20	72	8:20	19	20	21	7:13
50	66	12	20	33	5:07	20	20	20	7:00
50	70	15	20	31	4:58	20	20	20	6:09
58	29	11	20	35	3:39	18	20	22	4:03
58	35	3	20	103	13:05	13	20	48	16:37
58	41	1	15	218	30:00	4	16	120	42:34
58	47	0	3	296	42:02	1	17	117	58:15
58	53	0	1	315	46:37	3	10	218	1:47:04
58	58	0	2	309	48:38	1	12	198	1:55:35
58	63	1	6	275	44:26	7	20	83	1:04:08
58	69	2	12	204	34:18	15	20	34	32:25
58	75	1	16	122	23:13	15	20	28	27:08
58	81	3	20	79	17:09	19	20	21	16:52
58	87	11	20	42	11:40	20	20	20	12:36
58	93	13	20	30	10:22	20	20	20	15:16

		$L^3FP,\ \delta = 0.99$				$L^3FP,\ \delta = 0.99$ with deep insertions			
n	b	suc_1	# suc	# rou	hh:mm:ss	suc_1	# suc	# rou	hh:mm:ss
66	18	20	20	20	1:11	20	20	20	1:34
66	26	19	20	21	2:03	20	20	20	2:58
66	34	5	20	50	9:05	12	20	33	15:53
66	42	1	16	210	44:01	3	19	124	1:10:43
66	50	0	0	320	10:14*	0	8	250	2:43:16
66	58	0	1	319	14:05*	0	4	291	4:55:39
66	66	0	0	320	11:03*	0	9	237	5:16:29
66	72	0	0	320	11:36*	1	19	125	3:45:28
66	80	0	2	315	1:23:50	9	20	69	2:35:15
66	88	1	13	203	58:18	10	20	46	2:15:07
66	96	0	16	173	51:44	17	20	23	57:32
66	104	3	17	144	46:17	20	20	20	25:51
66	112	11	20	39	20:29	20	20	20	33:36

		BKZ, $\delta = 0.99$, $\beta = 10$				BKZ, $\delta = 0.99$, $\beta = 20$			
n	b	suc_1	#suc	#rou	hh:mm:ss	suc_1	#suc	#rou	hh:mm:ss
42	24	20	20	20	0:40	20	20	20	0:40
42	28	20	20	20	1:49	18	20	22	2:28
42	32	17	20	39	4:52	20	20	20	2:58
42	36	11	18	59	8:53	15	19	45	7:27
42	40	15	20	31	5:50	18	20	30	7:05
42	44	14	20	41	8:02	19	20	25	4:46
42	48	19	20	21	2:38	20	20	20	2:40
42	52	20	20	20	2:07	20	20	20	2:19
42	56	20	20	20	2:02	20	20	20	2:05
42	60	20	20	20	2:03	20	20	20	2:07
50	26	19	20	21	2:30	20	20	20	2:11
50	30	19	20	22	3:32	20	20	20	4:25
50	34	15	20	26	7:55	18	20	22	7:54
50	38	4	19	73	19:20	17	20	25	15:24
50	42	8	19	74	25:22	14	19	53	30:51
50	46	4	11	200	58:33	10	20	77	48:15
50	50	8	20	48	25:09	16	19	41	26:28
50	54	14	20	46	18:04	19	20	22	16:57
50	58	17	20	26	10:48	20	20	20	12:28
50	62	19	20	23	9:10	20	20	20	8:45
50	66	20	20	20	7:12	20	20	20	7:11
50	70	20	20	20	6:19	20	20	20	5:53
58	29	19	20	21	4:23	20	20	20	5:45
58	35	16	20	25	9:35	17	20	26	18:38
58	41	3	18	111	50:58	10	20	34	48:20
58	47	0	14	213	1:38:43	10	17	89	16:31*
58	53	0	8	242	2:06:24	6	15	130	31:49*
58	58	9	16	105	2:10:52	2	16	155	3:45:43
58	63	11	19	68	1:44:42	15	20	35	1:14:38
58	69	16	20	27	49:25	19	20	21	42:52
58	75	20	20	20	19:57	20	20	20	28:39
58	81	19	20	21	23:02	20	20	20	16:55
58	87	20	20	20	12:52	20	20	20	12:05
58	93	20	20	20	15:40	20	20	20	11:30

		$BKZ,\ \delta = 0.99,\ \beta = 10$				$BKZ,\ \delta = 0.99,\ \beta = 20$			
n	b	suc$_1$	# suc	# rou	hh:mm:ss	suc$_1$	# suc	# rou	hh:mm:ss
66	18	20	20	20	0:12*	20	20	20	0:12*
66	26	20	20	20	0:31*	20	20	20	0:33*
66	34	16	20	25	1:55*	20	20	20	1:59*
66	42	2	17	92	8:32*	9	20	49	12:43*
66	50	1	6	269	24:07*	2	13	215	56:50*
66	58	0	1	310	30:05*	2	10	203	1:25:14*
66	66	0	0	320	35:43*	2	8	236	1:45:11*
66	72	3	10	209	27:40*	3	16	155	1:34:07*
66	80	10	20	69	4:55:40	17	20	39	5:37:05
66	88	13	20	42	3:13:48	18	20	22	1:13:37
66	96	19	20	21	1:39:04	20	20	20	54:01
66	104	20	20	20	26:30	20	20	20	31:54
66	112	20	20	20	34:26	20	20	20	26:45

The above statistic shows that L^3FP–reduction with deep insertions is much stronger than straight L^3FP–reduction. It is even stronger than BKZ–reduction with block size 10 and nearly matches the performance of BKZ–reduction with block size 20. The success rates of BKZ–reduction improves greatly with increasing block size but the running time increases as well.

Comparison with La Macchia's results. La Macchia [15] also used the lattice basis (1) to solve subset sum problems. La Macchia minimizes floating point errors in the L^3–reduction by using initially Seysen's reduction algorithm. A comparison of La Macchia's and our success rates has to take into account that La Macchia applies 5 independent randomizations to the initial basis which increases the success rates by a factor between 1 and 5. La Macchia's success rates for a single randomization of the initial basis are consistently lower than ours for L^3FP. Our improved success rates are due to the deep insertion rule that is used for indices $i \leq 5$.

Block Korkin Zolotarev reduction with pruning. We can speed up BKZ–reduction with large block size by pruning the enumeration tree that is produced by the procedure ENUM. For example we set $\alpha_t := \min\left\{1.05\,\frac{k-t+1}{k-j}, 1\right\}$ and we replace in Step 2 of ENUM the predicate "IF

$\widetilde{c}_t < \bar{c}_j$" by "IF $\widetilde{c}_t < \alpha_t \bar{c}_j$". Note that α_t is rather small if t is close to k and which is near 1 if t is close to j. Here are some performance data for solving subset sum problems using this pruned variant of block Korkine Zolotarev reduction. This algorithm improves the success rates of BKZ–reduction with block size 20 as is shown by the first block of the table. For dimension 106 we have reduced the number of problems per row. This number is given in the last column.

$BKZ,\ \delta = 0.99,\ \ \beta = 50,\ \ \alpha_t = \min\ \left(1.05\,\frac{k-t+1}{k-j}, 1\right)$

n	b	suc$_1$	# suc	# rou	$hh : mm : ss$	# problems per row
66	26	20	20	20	0:36*	20
66	34	20	20	20	3:54*	20
66	42	20	20	20	15:55*	20
66	50	10	19	78	1:30:19*	20
66	58	9	14	119	3:40:26*	20
66	66	10	19	70	3:05:43*	20
66	72	18	20	26	1:18:22*	20
66	80	20	20	20	38:10*	20
66	88	20	20	20	36:09*	20
66	96	20	20	20	28:40*	20
72	106	20	20	20	1:11:34*	20
72	118	20	20	20	1:19:14*	20
72	130	20	20	20	1:02:20*	20
82	134	20	20	20	1:25:20*	20
82	146	20	20	20	1:34:46*	20
82	158	20	20	20	1:23:02*	20
106	180	5	5	5	19:15:55*	5
106	210	10	10	10	7:30:27*	10
106	240	10	10	10	3:14:50*	10
106	270	10	10	10	2:49:52*	10
106	300	10	10	10	3:53:18*	10

Acknowledgement The first author likes to thank the students of his class in summer 1990 for evaluating early versions of the present algorithms for lattice basis reduction and for evaluating early versions of the lattice basis (1). He also wishes to thank SIEMENS/NIXDORF and the Frankfurt

department of Computer Science for their support.

References

[1] E.F. Brickell: Solving low density knapsacks. Advances in Cryptology, Proceedings of CRYPTO'83, Plenum Press, New York (1984), 25–37.

[2] B. Chor and R. Rivest: A knapsack–type public key cryptosystem based on arithmetic in finite fields. IEEE Trans. Information Theory **IT–34** (1988), 901–909.

[3] M.J. Coster, A. Joux, B.A. LaMacchia, A.M. Odlyzko, C.P. Schnorr and J. Stern: An improved low–density subset sum algorithm. computational complexity 2, (1992), 97–186.

[4] P. van Emde Boas: Another NP–complete partition problem and the complexity of computing short vectors in a lattice. Rept. 81–04, Dept. of Mathematics, Univ. of Amsterdam, 1981.

[5] M. Euchner: Praktische Algorithmen zur Gitterreduktion und Faktorisierung. Diplomarbeit Uni. Frankfurt (1991).

[6] A. M. Frieze: On the Lagarias–Odlyzko algorithm for the subset sum problem. SIAM J. Comput. 15 (2) (1986), 536–539.

[7] M. R. Garey and D. S. Johnson: Computers and Intractability: A Guide to the Theory of NP–Completeness. W. H. Freeman and Company (1979).

[8] J. Hastad, B. Just, J. C. Lagarias and C. P. Schnorr: Polynomial time algorithms for finding integer relations among real numbers. SIAM J. Comput. 18 (5) (October 1989), 859–881.

[9] C. Hermite: Extraits de lettres de M. Ch. Hermite à M. Jacobi sur différents objects de la théorie des nombres. Deuxième lettre du 6 août 1845. J. Reine Angew. Math. **40** (1850), 279–290.

[10] A. Joux and J. Stern: Improving the critical density of the Lagarias–Odlyzko attack against subset sum problems. Proceedings of Fundamentals of Computation Theory, FCT'91, Ed. L. Budach, Springer LNCS **529** (1991), pp. 258–264.

[11] R. Kannan: Minkowski's Convex Body Theory and Integer Programming. Math. Oper. Res. 12 (1987), 415–440.

[12] A. Korkine and G. Zolotareff: Sur les formes quadratiques. Math. Ann. **6** (1873), 366–389.

[13] J.C. Lagarias, H.W. Lenstra, Jr. and C.P. Schnorr: Korkin–Zolotarev Bases and Successive Minima of a Lattice and its Reciprocal Lattice. Combinatorica **10** (1990), pp. 333–348.

[14] J. C. Lagarias and A. M. Odlyzko: Solving low–density subset sum problems. J. Assoc. Comp. Mach. 32(1) (1985), 229–246.

[15] B. A. LaMacchia: Basis Reduction Algorithms and Subset Sum Problems. SM Thesis, Dept. of Elect. Eng. and Comp. Sci., Massachusetts Institute of Technology, Cambridge, MA (1991). In preparation.

[16] H. W. Lenstra, Jr.: Integer programming with a fixed number of variables. Math. Oper. Res. **8** (1983), pp. 538–548.

[17] A.K. Lenstra, H.W. Lenstra, and L. Lovász: Factoring polynomials with rational coefficients. Math. Ann. 261 (1982), 515–534.

[18] L. Lovász: An algorithmic theory of numbers, graphs and convexity. SIAM Publications, Philadelphia (1986).

[19] L. Lovász and H. Scarf: The generalized basis reduction algorithm. Math. Oper. Res. (1992).

[20] A. M. Odlyzko: The rise and fall of knapsack cryptosystems. Cryptology and Computational Number Theory, C. Pomerance, ed., Am. Math. Soc., Proc. Symp. Appl. Math. 42 (1990), 75–88.

[21] A. Paz and C. P. Schnorr: Approximating integer lattices by lattices with cyclic factor groups. Automata, Languages, and Programming: 14th ICALP, Lecture Notes in Computer Science 267, Springer–Verlag, NY (1987), 386–393.

[22] S. Radziszowski and D. Kreher: Solving subset sum problems with the L^3 algorithm. J. Combin. Math. Combin. Comput. 3 (1988), 49–63.

[23] C. P. Schnorr: A hierarchy of polynomial time lattice basis reduction algorithms. Theoretical Computer Science 53 (1987), 201–224.

[24] C. P. Schnorr: A more efficient algorithm for lattice basis reduction. J. Algorithms 9 (1988), 47–62.

[25] C. P. Schnorr: Factoring integers and computing discrete logarithms via diophantine approximation. Proceedings EUROCRYPT'91, Brighton, May 1991, Springer LNCS **547** (1991), pp. 281–293.

[26] C. P. Schnorr and M. Euchner: Lattice basis reduction: improved algorithms and solving subset sum problems. Proceedings of Fundamentals of Computation Theory, FCT'91, Ed. L. Budach, Springer LNCS **529**, (1991), pp. 68–85.

[27] M. Seysen: Simultaneous reduction of a lattice basis and its reciprocal basis. To appear in Combinatorica.