

Heuristics on Lattice Basis Reduction in Practice

Werner Backes

Max-Planck-Institut für Informatik, Saarbrücken, Germany

and

Susanne Wetzel,

Stevens Institute of Technology, Hoboken, New Jersey, USA

In this paper we provide a survey on LLL lattice basis reduction in practice. We introduce several new heuristics as to speed up known lattice basis reduction methods and improve the quality of the computed reduced lattice basis in practice. We analyze substantial experimental data and to our knowledge, we are the first to present general heuristics for determining which variant of the reduction algorithm, for varied parameter choices, yields the most efficient reduction strategy for reducing a particular problem instance.

Categories and Subject Descriptors: ?? [??]: ??

General Terms: ???

Additional Key Words and Phrases: Lattice basis reduction, dynamic approximation, modular and iterative heuristics, general reduction heuristics

1. INTRODUCTION

A lattice is a discrete additive subgroup of \mathbb{R}^n . Its rank is the dimension of the \mathbb{R} -subspace that it spans. Each lattice of rank k has a basis, i.e., a sequence of k elements of the lattice, that generate the lattice as an abelian group. Since lattices have infinitely many bases one is interested in finding the ones that are more useful than others. The term reduced lattice basis is the one used to describe this property. There are many different notions of a reduced lattice basis in the literature, most of which relate to the lengths of the basis vectors. The underlying theory of lattice basis reduction has a long history starting with the reduction of quadratic forms and recently obtained general interest with the introduction of the LLL lattice basis reduction algorithm [Lenstra et al. 1982]. It spurred extensive research thus leading to the discovery of important connections of lattice theory with other fields in mathematics and computer science. In particular, the progress in lattice theory has revolutionized combinatorial optimization [Grötschel et al. 1993] and cryptography (e.g., [Ajtai 1996; Ajtai and Dwork 1997; Coster et al. 1992; Goldreich et al. 1997; Joux and Stern 1998]). Nevertheless, despite the manifold results in theory and the availability of implementations of lattice basis reduction algorithms in various computer algebra systems (e.g., LiDIA [LiDIA Group 1999], Magma [Magma 1999] or NTL [NTL 1999]) there is still very little known about the practical performance and strength of lattice basis reduction algorithms. Thus, they are often underestimated as recent results show (e.g., on breaking cryptosystems using lattice basis reduction methods [Nguyen 1999; Nguyen and Stern 1998]). With this paper we close this gap by providing the results of analyzing extensive test data thus

A preliminary versions of this article appeared in *Proceedings of Fourth Algorithmic Number Theory Symposium (ANTS IV), 2000* and *Proceedings of Workshop on Algorithm Engineering (WAE), 2000*.

This work was done while S. Wetzel was a member of the Graduiertenkolleg Informatik at the Universität des Saarlandes, Saarbrücken (Germany), a fellowship program of the DFG (Deutsche Forschungsgemeinschaft).

Authors' addresses: Werner Backes, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, D-66123 Saarbrücken, Germany; Susanne Wetzel, Stevens Institute of Technology, Department of Computer Science, Castle Point on Hudson, Hoboken, NJ 07030, USA.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works, requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept, ACM Inc., 1515 Broadway, New York, NY 10036 USA, fax +1 (212) 869-0481, or permissions@acm.org.

supplying useful facts on the practical performance of widely used lattice basis reduction algorithms. Moreover, we introduce newly-developed heuristics designed to improve known lattice basis reduction methods in practice and provide detailed test data on these new methods which are only implemented in LiDIA so far. We restrict the discussion in this paper to the LLL algorithm and its variants as the most well-known and most widely used lattice basis reduction methods in practice.

The outline of the paper is as follows: In Section 2 we give a brief introduction to lattice theory by covering the basic terminology and stating basic auxiliary results in particular of the LLL algorithm, which is the first known polynomial time lattice basis reduction algorithm guaranteed to compute lattice bases consisting of relatively short basis vectors. We then focus on the Schnorr-Euchner algorithm [Schnorr and Euchner 1991]. Even though it cannot be proven that the algorithm will output an LLL-reduced basis in polynomial time, this algorithm provided the first essential improvement for making the LLL-reduction algorithm efficiently applicable in practice.

In Sections 3 and 4 we introduce (newly-developed) variants of the Schnorr-Euchner reduction algorithm (e.g., based on modular techniques) designed to achieve better run times and reduction results in practice than the classical Schnorr-Euchner algorithm, especially for large lattice bases or bases with large entries. The development of these new heuristics is motivated by the fact that both the run time and the stability of the classical Schnorr-Euchner algorithm strongly depend on the precision of the approximations used. Hence due to stability reasons, for large lattice bases or bases with large entries, a high precision for the approximations has to be used in the classical algorithm thus causing a major loss in efficiency. Section 5 is devoted to the description of suitable test series for testing the different reduction algorithms (i.e., the classical Schnorr-Euchner algorithm as well as the variants presented in Sections 3 and 4) and the analysis of the corresponding substantial experimental data. Among other things, the analysis shows clearly the efficiency of the newly-developed algorithms. Furthermore, based on the data and the analysis, we present general heuristics for determining which variant of the reduction algorithm, for varied parameter choices, yields the most efficient reduction strategy (with respect to run time or quality of the basis) for reducing a particular problem instance.

2. BACKGROUND ON LATTICE BASIS REDUCTION

In this section we present the basic definitions and results which will be used in the sequel. For more details and proofs we refer to [Cohen 1993; Grötschel et al. 1993; Pohst and Zassenhaus 1989; Wetzel 1998]. In the following, let $n, k \in \mathbb{N}$ with $k \leq n$. By $\|\underline{b}\|$ we denote the Euclidean length of the column vector \underline{b} and for $z \in \mathbb{R}$, $\lceil z \rceil$ stands for the closest integer to z . The identity matrix is denoted by I , or if we need to stress its dimension, by I_n . Furthermore, the i -th unit vector in \mathbb{R}^n is denoted by \underline{e}_i ($1 \leq i \leq n$). A lattice L in \mathbb{R}^n is defined as $L = \left\{ \sum_{i=1}^k x_i \underline{b}_i \mid x_i \in \mathbb{Z}, i = 1, \dots, k \right\}$, where $\underline{b}_1, \underline{b}_2, \dots, \underline{b}_k \in \mathbb{R}^n$ are linearly independent vectors. We call $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{R}^{n \times k}$ a basis of the lattice with dimension k and with $L(B)$ we denote the lattice generated by the basis B . We call the lattice integral if $L \subseteq \mathbb{Z}^n$. Obviously, any set of all integral linear combinations of a fixed number m of rational vectors is a lattice. With $L[G]$ we denote the lattice generated by the m rational vectors $G = (\underline{g}_1, \dots, \underline{g}_m) \in \mathbb{Q}^{n \times m}$ where the vectors $\underline{g}_1, \dots, \underline{g}_m$ are linearly dependent over \mathbb{R} .

Clearly, a lattice has various bases whereas the dimension is uniquely determined. A basis of a lattice is unique up to unimodular transformations such as exchanging two basis vectors, multiplying a basis vector by -1 or adding an integral multiple of one basis vector to another one, for example. The determinant $\det(L) := \det(B^T B)^{\frac{1}{2}}$ of the lattice L in \mathbb{R}^n with basis $B \in \mathbb{R}^{n \times k}$ is independent of the choice of the basis. The Hadamard inequality $\det(L) \leq \prod_{i=1}^k \|\underline{b}_i\|$ gives an upper bound for the size of the determinant of a lattice. Equality holds iff B is an orthogonal basis. Furthermore, the defect of B is defined as $\text{dft}(B) = \frac{1}{\det(L)} \prod_{i=1}^k \|\underline{b}_i\|$. In general, $\text{dft}(B) \geq 1$ and $\text{dft}(B) = 1$ iff B is an orthogonal basis. The orthogonality-defect of a basis is one measure to evaluate the quality of a basis. Thus, one notion of lattice basis reduction is reducing (possibly minimizing) the defect of a lattice, i.e., constructing one of the many bases of a lattice such that the basis vectors are as small as possible (by means of the Euclidean length) and are as orthogonal as possible to each other.

The most well-known lattice basis reduction method in this context is the so-called LLL-reduction [Lenstra et al. 1982] which centers around the Gram-Schmidt orthogonalization: For an ordered basis

$B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{R}^{n \times k}$ the Gram-Schmidt orthogonalization $B^* = (\underline{b}_1^*, \dots, \underline{b}_k^*) \in \mathbb{R}^{n \times k}$ is computed as

$$\underline{b}_i^* = \underline{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \underline{b}_j^* \quad \text{for } 1 \leq i \leq k \quad (1)$$

with the Gram-Schmidt coefficients $\mu_{ij} = \frac{\langle \underline{b}_i, \underline{b}_j^* \rangle}{\|\underline{b}_j^*\|^2}$ ($1 \leq j < i \leq k$). In general B^* is not a basis for $L(B)$. Nevertheless, two observations relating to the Gram-Schmidt orthogonalization are key for the definition of an LLL-reduced basis. First, from equation (1) it directly follows that $\underline{b}_i = \underline{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \underline{b}_j^*$. Secondly, it can be shown [Grötschel et al. 1993] that for any non-zero vector $b \in L(B)$ with ordered basis $B = (\underline{b}_1, \dots, \underline{b}_k)$ and Gram-Schmidt orthogonalization $B^* = (\underline{b}_1^*, \dots, \underline{b}_k^*)$, $\|b\| \geq \min\{\|\underline{b}_1^*\|, \dots, \|\underline{b}_k^*\|\}$. Consequently, the lattice vectors are the shorter, the smaller the Gram-Schmidt coefficients and the lengths of the vectors in the Gram-Schmidt orthogonalization are. Thus, the underlying idea of LLL-reduction is to minimize the coefficients μ_{ij} ($1 \leq j < i \leq k$) by adding a suitable integral multiple of the base vector \underline{b}_j to \underline{b}_i , i.e., $\underline{b}_i := \underline{b}_i - \lceil \mu_{ij} \rceil \underline{b}_j$, and minimize the lengths of the vectors in the Gram-Schmidt orthogonalization by reordering the basis \underline{B} . This leads to the following definition:

Definition 1. For a lattice $L \subseteq \mathbb{Z}^n$ with basis $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, corresponding Gram-Schmidt orthogonalization $B^* = (\underline{b}_1^*, \dots, \underline{b}_k^*) \in \mathbb{Q}^{n \times k}$ and Gram-Schmidt coefficients $\mu_{i,j}$ with $1 \leq j < i \leq k$, the basis B is called LLL-reduced if the following conditions are satisfied:

$$|\mu_{i,j}| \leq \frac{1}{2} \quad \text{for } 1 \leq j < i \leq k \quad (2)$$

$$\|\underline{b}_i^* + \mu_{i,i-1} \underline{b}_{i-1}^*\|^2 \geq \frac{3}{4} \|\underline{b}_{i-1}^*\|^2 \quad \text{for } 1 < i \leq k. \quad (3)$$

The first property (2) is the criterion for size-reduction and means that the Gram-Schmidt coefficients are as small as possible, i.e., the lengths of the base vectors cannot be shortened any further by adding an integer multiple of base vector \underline{b}_j to base vector \underline{b}_i ($1 \leq j < i \leq k$). The second condition refers to the reordering of the basis and thus the lengths of the vectors in the orthogonalization. The condition calls for a re-ordering of a basis by exchanging two neighboring base vectors \underline{b}_i and \underline{b}_{i-1} ($1 < i \leq k$) as long as their exchange results in a considerable decrease in the length of the corresponding new \underline{b}_i^* .

The constant factor $\frac{3}{4}$ in (3) is the so-called reduction parameter and may be replaced with any fixed real number y with $\frac{1}{4} < y \leq 1$. Clearly, the reduction parameter both influences the run time of the reduction algorithm and the quality of the reduced basis. With increasing parameter y , the run time increases as well and the quality is improved. However, for $y = 1$ which provides the best reduction result, it cannot be guaranteed that the reduction process terminates in polynomial time. In general, the following theorem provides an estimate on the quality of an LLL-reduced basis:

THEOREM 1. *Let L be a lattice in \mathbb{Z}^n and $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$ be an LLL-reduced basis of L . Then, the following holds:*

$$\|\underline{b}_1\| \cdot \dots \cdot \|\underline{b}_k\| \leq \left(\frac{4}{4y-1}\right)^{k(k-1)/4} \det(L) \quad (4)$$

$$\|\underline{b}_1\|^2 \leq \left(\frac{4}{4y-1}\right)^{k-1} \|\underline{v}\|^2 \quad \text{for all } \underline{v} \in L, \underline{v} \neq \underline{0} \quad (5)$$

We will now describe the algorithm which transforms a given lattice basis into an LLL-reduced one. We first describe the main algorithm and then focus on the subroutines REDUCE and SWAP.

ALGORITHM 1: LLL($\underline{b}_1, \dots, \underline{b}_k$)

INPUT: Lattice basis $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$

OUTPUT: LLL-reduced lattice basis

- (1) compute $\|\underline{b}_1^*\|^2, \dots, \|\underline{b}_k^*\|^2$ and the corresponding μ_{ij} for $1 \leq j < i \leq k$
- (2) $l := 2$

```

(3) while ( $l \leq k$ ) do
(4)   REDUCE( $\mu_{l-1}$ )
(5)   if ( $\|\underline{b}_l^*\|^2 < (\frac{3}{4} - \mu_{l-1}^2)\|\underline{b}_{l-1}^*\|^2$ ) then
(6)     SWAP( $\underline{b}_{l-1}, \underline{b}_l$ )
(7)      $l := \max\{l-1, 2\}$ 
(8)   else
(9)     for ( $m := l-2; m \geq 1; m--$ ) do
(10)      REDUCE( $\mu_m$ )
(11)    od
(12)     $l := l+1$ 
(13)  fi
(14) od

```

Basically, the LLL algorithm consists of two types of operations, namely the size-reduction (translation) and the exchange of two base vectors. (Note that these are unimodular transformations of the basis.) These operations are repeated until conditions (2) and (3) are fulfilled. From [Lenstra 1983] it is known that independently of the order in which those operations are performed, this will lead to an LLL-reduced basis after a finite number of steps. Nevertheless, the algorithm works step by step in order to have as little update operations as possible and to have polynomial bounds for the binary size of the occurring integers and the run time of the algorithm. Before size-reducing base vector \underline{b}_l or interchanging it with \underline{b}_{l-1} , the vectors $\underline{b}_1, \dots, \underline{b}_{l-1}$ are LLL-reduced first. Thus, the LLL algorithm starts with the computation of the Gram-Schmidt orthogonalization of the given basis and then sets the step index l to 2. As long as the step index l is less than $k+1$, the while-loop will be executed. At the beginning of the while-loop, the following loop invariants are true:

$$|\mu_{ij}| \leq \frac{1}{2} \quad \text{for } 1 \leq j < i < l$$

$$\|\underline{b}_i^* + \mu_{i-1} \underline{b}_{i-1}^*\|^2 \geq \frac{3}{4} \|\underline{b}_{i-1}^*\|^2 \quad \text{for } 1 < i < l$$

For step index l , the LLL algorithm first achieves that $|\mu_{l-1}| \leq 1/2$ by executing REDUCE(μ_{l-1}). Then, it checks the condition $\|\underline{b}_l^*\|^2 < (\frac{3}{4} - \mu_{l-1}^2)\|\underline{b}_{l-1}^*\|^2$. If it is fulfilled, the two base vectors \underline{b}_l and \underline{b}_{l-1} will be exchanged and some of the μ_{ij} with $1 \leq i < j \leq k$ have to be updated. In addition, the step index has to be reduced by one in order to guarantee the validity of the invariants upon return to the beginning of the while-loop. If the condition does not hold (exchanging the vectors would not cause a dramatic drop of the length of \underline{b}_{l-1}^* any more), only the μ_{ij} for $1 \leq j < l$ will have to be size-reduced such that $|\mu_{ij}| \leq 1/2$ and the step index has to be incremented by one since the invariants are now valid for $l+1$. The algorithm will terminate if the new step index is equal to $k+1$. Obviously, the LLL algorithm is somewhat similar to the well-known Bubble-Sort algorithm.

With the subroutine REDUCE(μ_{ij}) one performs the size-reduction of μ_{ij} , thus achieving that $|\mu_{ij}| \leq 1/2$ by subtracting $\lceil \mu_{ij} \rceil \underline{b}_j$ from \underline{b}_i , i.e., \underline{b}_i is reduced in the direction of \underline{b}_j . Even though this does not necessarily imply that \underline{b}_i will be shorter in length afterwards, reducing one complete row of the Gram-Schmidt matrix, i.e., REDUCE(μ_m) for fixed l and $1 \leq m < l$, generally implies a reduction of the length of base vector \underline{b}_l ($1 \leq l \leq k$). With the subroutine SWAP($\underline{b}_{l-1}, \underline{b}_l$), simply the two basis vectors \underline{b}_{l-1} and \underline{b}_l are exchanged.

For any lattice $L \subseteq \mathbb{Z}^n$ with basis $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, the LLL-reduction of the basis B can be computed in polynomial time. More precisely, the number of arithmetic operations needed by the LLL algorithm is $O(k^3 n \log C)$, and the integers on which these operations are performed each have binary size $O(k \log C)$ where $C \in \mathbb{R}$, $C \geq 2$ with $\|\underline{b}_i\|^2 \leq C$ for $1 \leq i \leq k$.

Thus, from a theoretical point of view, the algorithm performs very well since it yields a reasonably good reduction result within polynomial time. In practice however, the classical algorithm [Lenstra et al. 1982] suffers from the slowness of the subroutines for the exact long integer arithmetic which has to be used in order to guarantee that no errors occur in the basis (thus changing the lattice). Speeding up the algorithm by simply doing the operations in floating point arithmetic results in an unstable algorithm due to occurring floating point errors and error propagation. In [Schnorr and Euchner 1991], Schnorr and Euchner have rewritten the original LLL algorithm in such a way that an approximation of the integer lattice with

a faster floating point arithmetic is only used for the computation of the Gram-Schmidt coefficients $\mu_{i,j}$ ($1 \leq j < i \leq k$) while all the other operations are done on the integer lattice using an exact integer arithmetic. Moreover, Schnorr and Euchner have introduced heuristics for avoiding and correcting floating point errors, thus inventing a practical floating point variant of the original algorithm with good stability, which allows a tremendous speed-up in the computation of an LLL-reduced lattice basis.

Basically, the Schnorr-Euchner algorithm works the same as the original LLL algorithm does. However, unlike the original algorithm, the Schnorr-Euchner algorithm is a heuristic algorithm for which it cannot be proven that it terminates in polynomial time or outputs an LLL-reduced basis upon termination.

In the Schnorr-Euchner algorithm, the base vectors $\underline{b}_1, \dots, \underline{b}_k \in \mathbb{Z}^n$ are kept in exact as well as in floating point arithmetic. (In the following, the floating point value of an exact value v is denoted by v' , $(v)'$ stands for the corresponding type conversion and r is the number of precision bits in the floating point arithmetic.) While the latter is done for efficiency reasons, i.e., for reducing the effort of repeatedly transforming long integers into floating point representation, the first one is necessary since otherwise, errors in the basis could not be corrected in the course of the computations which would possibly even result in a change of the lattice. The μ_{ij} ($1 \leq j < i \leq k$) as well as the $\|\underline{b}_i^*\|^2$ ($1 \leq i \leq k$) are kept in floating point arithmetic. Errors occurring in these numbers can be corrected by using the correct (exact) basis. After these introductory remarks, the Schnorr-Euchner algorithm can now be stated as follows:

ALGORITHM 2: Schnorr-Euchner($\underline{b}_1, \dots, \underline{b}_k$)

INPUT: Lattice basis $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$

OUTPUT: LLL-reduced lattice basis

```

(1)  $l := 2$ 
(2)  $F_c := false$ 
(3)  $F_r := false$ 
(4) APPROX_BASIS( $B', B$ )
(5) while ( $l \leq k$ ) do
(6)   ORTHOGONALIZE( $l$ )
(7)   for ( $m := l - 1; m \geq 1; m --$ ) do
(8)     REDUCE( $\mu_{lm}$ )
(9)   od
(10)  if ( $F_r = true$ ) then
(11)    APPROX_VECTOR( $\underline{b}', \underline{b}$ );
(12)     $F_r := false$ 
(13)  fi
(14)  if ( $F_c = true$ ) then
(15)     $l := \max\{l - 1, 2\}$ 
(16)     $F_c := false$ 
(17)  else
(18)    if ( $B_l < (\frac{3}{4} - \mu_{ll-1}^2)B_{l-1}$ ) then
(19)      SWAP( $\underline{b}_{l-1}, \underline{b}_l$ )
(20)       $l := \max\{l - 1, 2\}$ 
(21)    else
(22)       $l := l + 1$ 
(23)    fi
(24)  fi
(25) od

```

With the procedures APPROX_BASIS and APPROX_VECTOR, the exact lattice basis B respectively a basis vector b are approximated by B' , respectively b' , using a floating point arithmetic, i.e., an approximate but fast data type (see [Schnorr and Euchner 1991; Wetzel 1998] for further details).

The procedure ORTHOGONALIZE contains one of the heuristics for minimizing floating point errors: If the result of computing the scalar product of two vectors is extremely small in comparison to their length (i.e., leading bits might be cancelled out because the computation of the scalar product is not well conditioned),

then the computation using floats is not exact enough and has to be redone using exact arithmetic (see instructions (3), (4) and (6)). In the next step of the Schnorr-Euchner algorithm, all μ_{lm} with $1 \leq m \leq l-1$ are size-reduced for stage l .

ALGORITHM 3: ORTHOGONALIZE(i)

INPUT: i with $1 \leq i \leq k$ and base vectors $\underline{b}_1, \dots, \underline{b}_i$ as well as $\underline{b}'_1 \dots \underline{b}'_i$, μ_{pq}
for $1 \leq q < p < i$ and $B_j = \|\underline{b}'_j\|^2$ for $1 \leq j < i$
OUTPUT: $\mu_{i1}, \dots, \mu_{ii}$ and $B_i = \|\underline{b}'_i\|^2$

- (1) $B_i := \|\underline{b}'_i\|^2$
- (2) **for** ($j := 1; j \leq i-1; j++$) **do**
- (3) **if** ($|\langle \underline{b}'_i, \underline{b}'_j \rangle| < 2^{-\frac{\epsilon}{2}} \|\underline{b}'_i\| \|\underline{b}'_j\|$) **then**
- (4) $s := \text{APPROX_VALUE}(\langle \underline{b}_i, \underline{b}_j \rangle)$
- (5) **else**
- (6) $s := \langle \underline{b}'_i, \underline{b}'_j \rangle$
- (7) **fi**
- (8) $\mu_{ij} := (s - \sum_{p=1}^{j-1} \mu_{jp} \mu_{ip} B_p) / B_j$
- (9) $B_i := B_i - \mu_{ij}^2 B_j$
- (10) **od**
- (11) $\mu_{ii} := 1$

With the flags F_c and F_r , additional provisions are made for minimizing floating point errors and preventing their propagation. The flag F_r indicates that a size-reduction step was performed, thus requiring an update of the floating point representation of the base vectors. The flag F_c is set if a large reduction coefficient, $\lceil |\mu_{ij}| \rceil > 2^{\frac{\epsilon}{2}}$, emerged. A large reduction coefficient might result from accumulated non-corrected errors in previous stages and causes major modifications of other coefficients. Therefore, the actual stage l in the Schnorr-Euchner algorithm will be decreased to $l-1$, thus correcting the coefficients μ_{ij} with $i \in \{l-1, l\}$ and $1 \leq j \leq i-1$ as well as B_{l-1} , B_l , \underline{b}'_{l-1} and \underline{b}'_l .

After the size-reduction in the Schnorr-Euchner algorithm is done, the original LLL condition (3) is checked. Note that in order to offset small floating point errors, the reduction parameter y has to be chosen from $(\frac{1}{2}, 1)$. If the condition is not fulfilled for stage l , the two base vectors \underline{b}_l and \underline{b}_{l-1} have to be exchanged.

ALGORITHM 4: REDUCE(μ_{ij})

INPUT: μ_{ij}, μ_{iq} and μ_{jq} for $1 \leq q \leq j$ ($\mu_{jj} = 1$)
OUTPUT: \underline{b}_i such that $|\mu_{ij}| \leq \frac{1}{2}$ as well as updated μ_{iq} for $1 \leq q \leq j$ and the
boolean variables F_c and F_r

- (1) **if** ($|\mu_{ij}| > \frac{1}{2}$) **then**
- (2) $F_r := \text{true}$
- (3) **if** ($\lceil |\mu_{ij}| \rceil > 2^{\frac{\epsilon}{2}}$) **then**
- (4) $F_c := \text{true}$
- (5) **fi**
- (6) $\underline{b}_i := \underline{b}_i - \lceil \mu_{ij} \rceil \underline{b}_j$
- (7) **for** ($q := 1; q \leq j; q++$) **do**
- (8) $\mu_{iq} := \mu_{iq} - \lceil \mu_{ij} \rceil \mu_{jq}$
- (9) **od**
- (10) **fi**

Apparently, both the run time and the stability of the Schnorr-Euchner algorithm strongly depend on the precision of the approximations used. While high precision approximations imply a major loss in efficiency, the algorithm might run into cycles and thus not terminate due to (accumulated) floating point errors if the precision is too small. Consequently, the algorithm still lacks of efficiency in particular for large lattice bases or bases with large entries because high precision approximations have to be used.

Therefore, before focusing on presenting and discussing practical results achieved by using the Schnorr-Euchner reduction algorithm in different settings (Section 5), thus comparing theory and practical performance of this lattice basis reduction algorithm, we will in the following (Sections 3 and 4) first present (new) heuristics designed to further speed up the reduction (such that even larger lattice bases with bigger entries can be reduced in a reasonable amount of time) and improve the quality of the reduction results (i.e., computing reduced lattice bases consisting of shorter lattice vectors in comparison with the reduction results obtained by the classical Schnorr-Euchner algorithm).

3. HEURISTICS TO ACHIEVE AN ADDITIONAL SPEED-UP

In this section we will introduce heuristics designed to allow a speed-up of the computation in comparison to the classical Schnorr-Euchner algorithm. The first heuristic in this setting, the so-called late size-reduction heuristic, is motivated by the following observation: While at stage l ($2 \leq l \leq k$) of the LLL-reduction process the Gram-Schmidt coefficients $\mu_{l,m}$ ($1 \leq m \leq l-2$) have to be size-reduced only if the basis vectors \underline{b}_l and \underline{b}_{l-1} are not swapped [Lenstra et al. 1982], in the classical Schnorr-Euchner algorithm always all the Gram-Schmidt coefficients $\mu_{l,m}$ ($1 \leq m \leq l-1$) are size-reduced. Therefore, a heuristic to speed up the LLL-reduction process in practice can be stated as follows:

HEURISTIC 1. (LATE SIZE-REDUCTION) *Before checking the LLL condition (3) at stage l of the reduction process [Schnorr and Euchner 1991], size-reduce only the Gram-Schmidt coefficient $\mu_{l,l-1}$. Perform the size-reduction of the other coefficients $\mu_{l,m}$ with $1 \leq m \leq l-2$ only if neither the stage index has to be decreased (due to accumulated floating point errors) nor the basis vectors \underline{b}_l and \underline{b}_{l-1} have to be swapped.*

While the late size-reduction heuristic centers on the time when the size-reductions are performed, another new heuristic, called modified size-reduction heuristic, focuses on the way the size-reductions are done in order to speed up the reduction of a lattice basis:

HEURISTIC 2. (MODIFIED SIZE-REDUCTION) *At the beginning of each size-reduction step check whether $\lceil |\mu_{i,j}| \rceil$ ($1 \leq j < i \leq k$) is larger than a certain bound. If so, perform a correction step and simplify the actual size-reduction by approximating $\lceil |\mu_{i,j}| \rceil$ with 2^t where $t = \lceil \log_2(\lceil |\mu_{i,j}| \rceil) \rceil$, thus replacing the original size-reduction with a fast shift operation.*

Whereas the heuristics introduced so far simplify or eliminate unnecessary operations, a different approach to reduce the overall run time can be taken by doing the computations on shorter operands thus allowing approximations with a lower precision than in the classical Schnorr-Euchner algorithm. The first heuristic in this context is as follows:

HEURISTIC 3. (DYNAMIC APPROXIMATION) *Instead of approximating the original lattice basis $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, do the approximations for $\tilde{B} = (\tilde{\underline{b}}_1, \dots, \tilde{\underline{b}}_k) \in \mathbb{Q}^{n \times k}$ with $\tilde{B} = \frac{1}{r}B$ and $r \in \mathbb{Q}, r \neq 0$.*

This heuristic is based on the following theorem:

THEOREM 2. *For a lattice $L \subseteq \mathbb{Z}^n$ with basis $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, a lattice $\tilde{L} = \frac{1}{r} \cdot L \subseteq \mathbb{Q}^n$ with basis $\tilde{B} = (\tilde{\underline{b}}_1, \dots, \tilde{\underline{b}}_k) = \frac{1}{r} \cdot B \in \mathbb{Q}^{n \times k}$ and $r \in \mathbb{Q}, r \neq 0$ the following holds:*

$$LLL(\tilde{B}) = \frac{1}{r} \cdot LLL(B) \quad (6)$$

PROOF. Since $\tilde{B}^* = \frac{1}{r} \cdot B^*$ holds for the orthogonalization and $\tilde{\mu}_{i,j} = \mu_{i,j}$ (Gram-Schmidt coefficients) for $1 \leq j < i \leq k$, with Definition 1 it follows that $LLL(\tilde{B}) = \frac{1}{r} \cdot LLL(B)$. \square

Since the size of the elements to be approximated changes in the course of the reduction process, it is not possible to use the same value r for the whole reduction process but rather r has to be changed dynamically over time in order to prevent cancellations and inaccuracies. Thus, before approximating the basis at the beginning of the reduction algorithm (see Algorithm 2), first the maximum absolute entry $m_0 = \max \{ |\underline{b}_{i,j}| \text{ for } 1 \leq i \leq k, 1 \leq j \leq n \}$ of the lattice basis is computed which is then used to determine a suitable value for $r = 2^{fact}$. Heuristically, *fact* was chosen as $\lceil \log_2(\frac{m_0}{av}) \rceil - dv$ so that it can be fit into the data type used for the approximations without having the approximated elements become too small thus causing cancellations and inaccuracies. The values *av* (adjust_value) and *dv* (decrease_value) also depend

on the approximating data type. Using `doubles` for the approximations, av and dv have heuristically been determined as $av = 1E + 52$ and $dv = 40$.

Moreover, whereas `APPROX_BASIS`, `APPROX_VECTOR` and `APPROX_VALUE` are simple data type conversions in the original Schnorr-Euchner algorithm (see Section 2), for the purpose of dynamic approximations they will have to be adjusted as follows:

ALGORITHM 5: `APPROX_BASIS`(B', B)

INPUT: $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$ exact basis
 OUTPUT: $B' = (\underline{b}'_1, \dots, \underline{b}'_k) \in \mathbb{Q}^{n \times k}$ approximated basis

```
(1) for ( $1 \leq i \leq k$ ) do
(2)   for ( $1 \leq j \leq n$ ) do
(3)      $\underline{b}'_{ij} := (\frac{1}{2^{fact}} \underline{b}_{ij})'$ 
(4)   od
(5) od
```

As before, \underline{b}'_{ij} stands for the approximated value and $(\frac{1}{2^{fact}} \underline{b}_{ij})'$ denotes the type conversion of $\frac{1}{2^{fact}} \underline{b}_{ij}$.

ALGORITHM 6: `APPROX_VECTOR`($\underline{b}'_i, \underline{b}_i$)

INPUT: \underline{b}_i exact vector
 OUTPUT: \underline{b}'_i approximated vector

```
(1) for ( $1 \leq j \leq n$ ) do
(2)    $\underline{b}'_{ij} := (\frac{1}{2^{fact}} \underline{b}_{ij})'$ 
(3) od
(4)  $m_i := \max\{|\underline{b}_{i1}|, \dots, |\underline{b}_{in}|\}$ 
(5)  $m_0 := \max\{m_1, \dots, m_k\}$ 
(6) if ( $(m_0 < av)$  and  $(fact > 0)$ ) then
(7)    $fact := fact - dv$ 
(8)   if ( $fact < 0$ ) then
(9)      $fact := 0$ 
(10)  fi
(11)  APPROX_BASIS( $B', B$ )
(12) fi
```

ALGORITHM 7: `APPROX_VALUE`(s', s)

INPUT: s exact value
 OUTPUT: s' approximate value

```
(1)  $s' := (\frac{1}{2^{fact}} s)'$ 
```

The second heuristic in the category of performing computations on shorter operands employs an iterative technique similar to the ones used in [Knuth 1981] and [Rickert 1989] for speeding up Euclid's algorithm and the reduction of quadratic forms, respectively:

HEURISTIC 4. (ITERATIVE HEURISTIC) *For reducing a given lattice basis, first work only with the leading digits of each entry of the basis vectors. Then, apply the performed reduction steps also to the original lattice basis and compute the final LLL-reduced lattice basis.*

The following theorem is a generalization of the idea presented in [Radziszowski and Kreher 1988; de Weger 1988] to arbitrarily chosen lattice bases. It shows in detail how the described heuristic is formalized and how it can be iterated:

THEOREM 3. *Let $B = (b_{i,j}) \in \mathbb{Z}^{n \times k}$ be a basis of lattice L and $u, v \in \mathbb{N}$ be such that $\lfloor \log_2(b_{i,j}) \rfloor + 1 \leq u \cdot v$ ($1 \leq i \leq n, 1 \leq j \leq k$). Moreover, for $1 \leq i \leq n, 1 \leq j \leq k$ and $1 \leq t \leq u$ let $b_{i,j}^{(t)} = \left\lfloor \frac{b_{i,j}}{2^{v(u-t)}} \right\rfloor$ and $B^{(t)} = (b_{i,j}^{(t)})$, let $D^{(t)} = (d_{i,j}^{(t)})$ be defined by $b_{i,j}^{(t+1)} = 2^v b_{i,j}^{(t)} + d_{i,j}^{(t)}$ and let $E = 2^v I_k$. With $T^{(0)} = I_k$, $C^{(1)} = B^{(1)} T^{(0)} = B^{(1)}$, $R^{(t)} = \text{LLL}(C^{(t)}) = C^{(t)} T_t$, $T^{(t)} = T^{(t-1)} T_t$ as well as $C^{(t+1)} = R^{(t)} E + D^{(t)} T^{(t)}$, then $R^{(u)} = \text{LLL}(B) = B T^{(u)}$. (T_t is the transformation matrix computed in the course of the reduction process [Lenstra et al. 1982; Schnorr and Euchner 1991].)*

Thus, for reducing the basis B , the iterative reduction algorithm applies the classical Schnorr-Euchner algorithm u times to the generating systems $C^{(1)}, \dots, C^{(u)}$. In each iteration step, v additional digits of the original input data are included in the computation. Hence, the result of the last iteration step yields the LLL-reduction of the lattice basis B :

ALGORITHM 8: Iterative($\underline{b}_1, \dots, \underline{b}_k, u, v$)

INPUT: Lattice basis $B = (\underline{b}_1, \dots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$,
 number of iterations u , number of digits v
 OUTPUT: LLL-reduced lattice basis

```

(1)  $E := 2^v I_k$ 
(2)  $T^{(0)} := I_k$ 
(3)  $C^{(1)} := B^{(1)}$ 
(4) for ( $t := 1; t \leq u; t++$ ) do
(5)    $R^{(t)} := \text{Schnorr-Euchner}(C^{(t)}, T)$ 
(6)   if ( $t < u$ ) then
(7)      $T^{(t)} := T^{(t-1)} \cdot T$ 
(8)      $C^{(t+1)} := R^{(t)} \cdot E + D^{(t)} \cdot T^{(t)}$ 
(9)   else
(10)     $B := R^{(t)}$ 
(11)  fi
(12) od
    
```

It is noted, that the Schnorr-Euchner algorithm is applied to a set of generating vectors instead of a basis, thus requiring minor modifications of Algorithm 2 (see [Schnorr and Euchner 1991]).

The last heuristic presented in this section is based on modular computations. From other areas in algorithmic number theory we know that the application of modular techniques has been instrumental in bringing about much more efficient solutions to well-known problems such as the computation of the determinant [Cohen 1993] or the Hermite normal form [Domich et al. 1987] of integer matrices. This is due to the fact that with modular techniques most of the computations can be performed on operands which are much smaller than the ones occurring in the conventional algorithms. The following considerations show how modular techniques can also be applied to the problem of computing an LLL-reduced lattice basis, thus allowing an improvement of the run time of the classical reduction algorithm:

LEMMA 1. ([DOMICH ET AL. 1987]) *Let $L \subseteq \mathbb{Z}^n$ be an n -dimensional lattice and $\Delta = z \cdot \det(L)$ with $z \in \mathbb{Z}$ be a multiple of the lattice determinant. Then, $\Delta \underline{e}_i \in L$ for $1 \leq i \leq n$.*

LEMMA 2. *Let $L \subseteq \mathbb{Z}^n$ be an n -dimensional lattice with basis $B = (\underline{b}_1, \dots, \underline{b}_n) \in \mathbb{Z}^{n \times n}$ and $\Delta = z \cdot \det(L)$ where $z \in \mathbb{Z}$. Then, $L(\underline{b}_1, \dots, \underline{b}_n) = L[\underline{b}_1 \bmod \Delta, \dots, \underline{b}_n \bmod \Delta, \Delta \underline{e}_1, \dots, \Delta \underline{e}_n]$.*

This leads to the following heuristic for full-dimensional lattices, which can be implemented in various ways [Wetzel 1998]:

HEURISTIC 5. *At first reduce the basis of the n -dimensional lattice $L \subseteq \mathbb{Z}^n$ modulo Δ , a multiple of the determinant, thus obtaining a system of generating vectors $(\underline{b}_1 \bmod \Delta, \dots, \underline{b}_n \bmod \Delta)$. Apply to that system of generating vectors a modular variant of the Schnorr-Euchner algorithm where additional modular operations are performed during the size-reduction process. Compute the LLL-reduced basis of the lattice*

$L(\underline{b}_1, \dots, \underline{b}_n)$ by applying the Schnorr-Euchner algorithm to the system of generating vectors consisting of the resulting vectors of the reduction with the modular Schnorr-Euchner algorithm as well as the vectors $\Delta \underline{e}_1, \dots, \Delta \underline{e}_n$.

Thus, the modular reduction lattice basis reduction algorithm can be formalized as follows:

ALGORITHM 9: Modular-1($\underline{b}_1, \dots, \underline{b}_n, \Delta$)

INPUT: Lattice basis $B = (\underline{b}_1, \dots, \underline{b}_n) \in \mathbb{Z}^{n \times n}$,
 $\Delta =$ multiple of the lattice determinant $\det(L)$

OUTPUT: LLL-reduced lattice basis

```

(1)  $B := B \bmod \Delta$ 
(2)  $l_d := 2; l_u := n, l_s := n + 1$ 
(3)  $F_c := false; F_r := false$ 
(4)  $counter := 0$ 
(5) for ( $i := 1; i \leq k; i ++$ ) do
(6)    $\underline{b}'_i := (\underline{b}_i)'$ 
(7)   while ( $l_d < l_s$ ) do
(8)     ORTHOGONALIZE( $l_d$ )
(9)     for ( $m := l_d - 1; m \geq 1; m --$ ) do
(10)      REDUCE( $\mu_{l_d, m}$ )
(11)     if ( $F_r = true$ ) then
(12)        $\underline{b}'_{l_d} := (\underline{b}_{l_d})'$ 
(13)        $F_r := false$ 
(14)       if ( $\underline{b}_{l_d} = \underline{0}$ ) then
(15)         for ( $i := l_d; i < l_u; i ++$ ) do
(16)            $\underline{b}_i := \underline{b}_{i+1}; \underline{b}'_i := \underline{b}'_{i+1}$ 
(17)         od
(18)          $l_u := l_u - 1; l_s := l_s - 1$ 
(19)         goto (8)
(20)       fi
(21)     fi
(22)     if ( $F_c = true$ ) then
(23)        $l_d := \max\{l_d - 1, 2\}; F_c := false$ 
(24)        $counter := 0$ 
(25)     else
(26)       if ( $\exists i : |b_{i,l}| \geq \Delta$ ) then
(27)          $\underline{b}_l := \underline{b}_l \bmod \Delta$ 
(28)          $counter := counter + 1$ 
(29)         if ( $counter = c_1$ ) then
(30)            $\underline{b}_{temp} := \underline{b}_l$ 
(31)           for ( $i := l_d; i < l_u; i ++$ ) do
(32)              $\underline{b}_i := \underline{b}_{i+1}; \underline{b}'_i := \underline{b}'_{i+1}$ 
(33)           od
(34)            $\underline{b}_{l_u} := \underline{b}_{temp}; l_s := l_s - 1$ 
(35)         fi
(36)         goto (8)
(37)       fi
(38)       if ( $B_{l_d} < (\frac{3}{4} - \mu_{l_d, l_d-1}^2) B_{l_d-1}$ ) then
(39)         SWAP( $\underline{b}_{l_d-1}, \underline{b}_{l_d}$ )
(40)          $l_d := \max\{l_d - 1, 2\}$ 
(41)       else

```

$$(42) \quad l_d := l_d + 1$$

$$(43) \quad \mathbf{fi}$$

$$(44) \quad \text{counter} := 0$$

$$(45) \quad \mathbf{fi}$$

$$(46) \quad \mathbf{od}$$

$$(47) \quad \text{Schnorr-Euchner}(\underline{b}_1, \dots, \underline{b}_{l_s-1}, \underline{b}_{l_s}, \dots, \underline{b}_{l_u}, \Delta I_n)$$

Because of the modular reduction in (27), it is possible that the Gram-Schmidt coefficients $\mu_{l_d, m}$ for $1 \leq m < l_d$ which were just size-reduced at stage l_d are no longer size-reduced. Therefore, the computation has to be repeated for stage l_d . In order to guarantee polynomial run time of the algorithm, the repetition is only done for a fixed number of times c_1 . Then, the critical vector \underline{b}_{l_d} is moved to the end of the basis. After step (47) we know that the vectors $\underline{b}_1, \dots, \underline{b}_{l_s-1}$ are LLL-reduced and the vectors $\underline{b}_{l_s}, \dots, \underline{b}_{l_u}$ are the critical ones which were moved to the end of the basis. Since the vectors $\underline{b}_1, \dots, \underline{b}_{l_s-1}, \underline{b}_{l_s}, \dots, \underline{b}_{l_u}$ do not necessarily form a basis of the lattice L , an additional overall non-modular reduction of $(\underline{b}_1, \dots, \underline{b}_{l_s-1}, \underline{b}_{l_s}, \dots, \underline{b}_{l_u}, \Delta \underline{e}_1, \dots, \Delta \underline{e}_n)$ has to be performed in order to obtain the correct result. Since the first $l_s - 1$ vectors are already LLL-reduced, the reduction process can start at stage l_s . Note that if $l_u = n$, $l_s = n - 1$ and Δ is a multiple of $\det(\hat{B}^T \hat{B})$ with $\hat{B} = (\underline{b}_1, \dots, \underline{b}_{l_u})$, then the last overall reduction can be omitted and \hat{B} is the sought LLL-reduced basis of the lattice $L(B)$.

With a slight modification, based on the following lemma, it is possible to run the algorithm without a counter [Wetzel 1998]:

LEMMA 3. ([KALTOFEN 1983]) *After size-reducing the vector b_{l_d} at stage l_d , the following holds:*

$$|b_{i, l_d}| \leq \frac{1}{2} \sqrt{(l_d + 3) \tilde{B}_{l_d}} \quad (7)$$

with $1 \leq i \leq n$ and $\tilde{B}_{l_d} = \max\{B_1, \dots, B_{l_d}\}$.

PROOF.

$$\|\underline{b}_{l_d}\|^2 = \|\underline{b}_{l_d}^*\|^2 + \sum_{j=1}^{l_d-1} \mu_{l_d, j}^2 \|\underline{b}_j^*\|^2 \leq \tilde{B}_{l_d} + \frac{l_d-1}{4} \tilde{B}_{l_d} = \frac{l_d+3}{4} \tilde{B}_{l_d}.$$

□

In other words, the size-reduction property is no longer destroyed by the modular computations as soon as they are done modulo $\left\lceil \sqrt{(l_d + 3) \tilde{B}_{l_d}} \right\rceil$. Thus, the following theorem holds:

THEOREM 4. *At stage l_d , at most $O(\log(\sqrt{n}))$ iterations of size-reductions followed by modular reductions will occur.*

PROOF. From the analysis in [Lenstra et al. 1982] we know that if $\|\underline{b}_i\|^2 \leq C$, then $\tilde{B}_{l_d} \leq C$. In our particular case, $C \leq n\Delta^2$, thus $\tilde{B}_{l_d} \leq n\Delta^2$. Since iterations occur only as long as $\Theta < \left\lceil \sqrt{(l_d + 3) \tilde{B}_{l_d}} \right\rceil$ and $\Theta = 2^v \Delta$ with $v \in \mathbb{N}_0$, we obtain the assertion. □

4. HEURISTICS TO ACHIEVE BETTER REDUCTION RESULTS

After introducing heuristics for speeding up the computation of a reduced lattice basis, we will now present two heuristics where the improvement of the quality of the reduction result is the main objective in their development, possibly even at the expense of the run time.

The first heuristic to achieve a better reduction result is motivated by the following example:

EXAMPLE 1. Let $B = \begin{pmatrix} 10 & -9 & 18 \\ 0 & 10 & 45 \\ 10 & 11 & -12 \end{pmatrix}$ be a basis of a lattice $L \subseteq \mathbb{Z}^3$. The basis B is already LLL-reduced

with reduction parameter $y \in (\frac{1}{4}, 1)$, $\|\underline{b}_1\|^2 = 200$, $\|\underline{b}_2\|^2 = 302$ and $\|\underline{b}_3\|^2 = 2493$. However, performing an additional size-reduction step even though $|\mu_{3,2}| = 0.5$ (i.e., size-reduction does not change $|\mu_{3,2}|$) results in a shorter basis vector $\underline{b}_3 = (27, 35, -23)^T$ with $\|\underline{b}_3\|^2 = 2483$ and thus in a better LLL-reduced lattice basis.

HEURISTIC 6. (SPECIAL CASE FOR μ) *If $|\mu_{i,j}| = 0.5$ with $1 \leq j < i \leq k$ perform a size-reduction step iff $\|\underline{b}_i - \text{sign}(\mu_{i,j})\underline{b}_j\| < \|\underline{b}_i\|$.*

A well known heuristic, the so-called deep insertion heuristic, is due to Schnorr and Euchner [Schnorr and Euchner 1991]:

HEURISTIC 7. (DEEP INSERTION) *For checking the LLL-reduction condition (2) at stage l take into consideration not only the values $\|\underline{b}_{l-1}^*\|$ and $\|\underline{b}_l^*\|$ (as it was done in the LLL algorithm and the classical Schnorr-Euchner algorithm) but also the earlier $\|\underline{b}_j^*\|$'s with $1 \leq j \leq l-2$ by extending the exchange of \underline{b}_l and \underline{b}_{l-1} to a deep insertion step, i.e., inserting \underline{b}_l at the best possible position i within the index interval $[1, \dots, l-1]$.*

Applying this heuristic, short orthogonal vectors are found further left in the orthogonalization thus yielding shorter basis vectors in the reduced basis than in the case of using the classical Schnorr-Euchner algorithm. It has to be noted that only a certain amount of deep insertion steps can be performed in order to guarantee polynomial run time of the reduction process (for details see [Schnorr and Euchner 1991]).

5. TESTS AND GENERAL HEURISTIC

Based on comprehensive tests, we will now analyze the practical performance of the classical Schnorr-Euchner algorithm as well as the algorithms implementing the (new) heuristics thus providing essential new insight into the practical performance of lattice basis reduction algorithm aside from the known theoretical results. Moreover, we present a general heuristic for the use of these lattice basis reduction algorithms.

For the tests we have used three generic test classes, namely knapsack lattices, unimodular lattices, and random lattices. Knapsack lattices arise in the context of solving knapsack problems [Coster et al. 1992; Schnorr and Euchner 1991] and are of the form $L(B) \subseteq \mathbb{Z}^{(n+3)}$ where

$$B = (\underline{b}_1, \dots, \underline{b}_{n+1}) = \begin{pmatrix} 2 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 2 & 0 & \cdots & 0 & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 2 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 2 & 1 \\ a_1W & a_2W & \cdots & a_{n-1}W & a_nW & SW \\ 0 & 0 & \cdots & 0 & 0 & -1 \\ W & W & \cdots & W & W & \frac{n}{2}W \end{pmatrix} \quad (8)$$

with positive integer weights a_i ($1 \leq i \leq n$), a sum $S \in \mathbb{N}$ and $W > \sqrt{n}$. This test class has been chosen due to its representative character, i.e., the fact that lattice bases in various contexts (e.g., gcd computations [Havas et al. 1994], factoring, Diophantine equations [Lenstra et al. 1982]) have a very similar structure to the one of bases of knapsack lattices. The unimodular lattices are generated by a unimodular basis B , i.e., $B \in \mathbb{Z}^{(n \times n)}$ with $|\det(B)| = 1$. This test class has been chosen for two reasons. First, unimodular lattices require an unusually high number of size-reductions in comparison to other lattice bases [Hecker 1994]. Thus, this test class can be used to determine the influence of the various heuristics on the number of required size-reductions. And secondly they are of great interest because the reduction of unimodular lattices has proven to be of great importance in the process of cryptanalyzing the NTRU signature scheme [Szydlo 2001]. The random $(n \times n)$ -lattices $L \subseteq \mathbb{Z}^n$ are generated by choosing each entry of the basis $B \in \mathbb{Z}^{(n \times n)}$ randomly. In the sequel, for simplicity we shall use the notion “knapsack lattice bases” etc. instead of “bases of knapsack lattices” etc. even though we are aware of the fact that it is not absolutely correct in a mathematical sense.

All tests have been done using the implementations of lattice basis reduction methods available in the computer algebra system LiDIA [Biehl et al. 1995; LiDIA Group 1999]. While there are various implementations of the classical Schnorr-Euchner algorithm (e.g., in computer algebra systems such as LiDIA, Magma [Magma 1999] and NTL [NTL 1999]), so far implementations of the new heuristics presented in Sections 3 and 4 are only available in LiDIA. We therefore refrain from including test data comparing implementations of the original Schnorr-Euchner algorithm in different computer algebra systems such as LiDIA, NTL, Magma etc. in this paper, as this work focuses on the previously presented heuristics. Instead we refer the interested reader to [Backes and Wetzel 2000] where extensive data can be found. Furthermore, because of the page

limit of this publication we also refrain from including the substantial test data with respect to the heuristics and rather refer to the cited website.

5.1 Tests of the Different Variants

5.1.1 Performance and Quality. At first we focus on the general performance of the classical Schnorr-Euchner algorithm, the variant of doing deep insertions, the algorithm of applying late size-reduction, the algorithm using the modified size-reduction and the variant considering the special case $|\mu_{i,j}| = 0.5$. The computations were done with reduction parameter $y = 0.99$ using `doubles` for the approximations [Schnorr and Euchner 1991]. The tests have been performed for different choices of n (e.g., $n = 10, \dots, 150$) and various sizes of the entries (e.g., bit length $b = 20, \dots, 300$).

The test results [Backes and Wetzel 2000] show that for a fixed dimension n the run time of the algorithms increases as the density of the knapsack decreases. For a fixed density the run time for reducing knapsack lattice bases increases as the dimension increases. These characteristics are due to the fact that the run time of the algorithms depends both on the dimension of the lattice to be reduced and the size of the input data such that it increases as the dimension or the size of the entries of the lattice basis vectors grow. Furthermore, the reduction of knapsack lattice bases always results in a vast decrease of the average length of the basis vectors and the defect.

For randomly chosen lattice bases the reduction time is relatively small (in comparison with the other test instances) and depends mainly on the dimension of the lattice. This is due to the fact that any randomly chosen lattice basis is already significantly reduced, thus the reduction will result only in a small decrease of the average length of the basis vectors. This can be explained by the observation that due to the Gaussian heuristic, the expected length of a smallest vector in a random lattice of dimension n with determinant Δ lies between $\Delta^{1/n} \sqrt{\frac{n}{2\pi e}}$ and $\Delta^{1/n} \sqrt{\frac{n}{\pi e}}$. In the case of reducing unimodular lattice bases, the reduction time mainly depends on the dimension of the lattice.

Checking whether an additional size-reduction step should be performed if $|\mu_{i,j}| = 0.5$ causes an increase of the run time but in general does not yield a shorter average length or a smaller defect. However, in the case of reducing knapsack lattice bases, the increase is negligible. In general, for random lattice bases the special case $|\mu_{i,j}| = 0.5$ does not occur. This is due to the fact that only few reduction steps are performed anyway (in comparison with the other test instances) and therefore it is very unlikely that $|\mu_{i,j}| = 0.5$ occurs.

Except for random lattice bases the deep insertion mechanism causes a major increase in the run time but also results in a great improvement of the average length of the vectors of the reduced bases as well as a better defect for knapsack lattice bases. In Figure 1, the influence of the deep insertion mechanism is illustrated on example of the first five basis vectors of reduced knapsack lattice bases with $n = 150$. In the

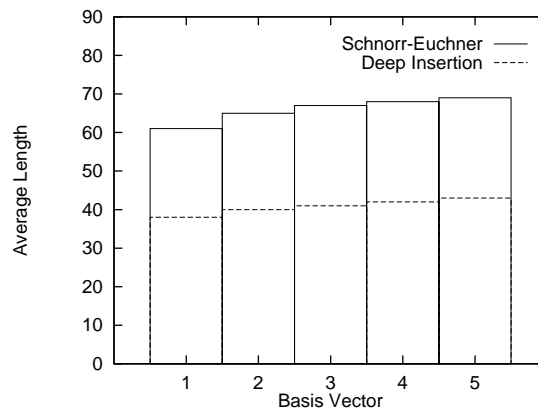


Figure 1. Different Variants: Deep Insertion Heuristic

case of random lattice bases, only few deep insertion steps are performed since these lattice bases are already quite well reduced from the beginning. Thus, there is hardly any decrease of the defect or the average length in comparison with the results of the classical algorithm. Logically, for knapsack and unimodular lattice bases, the amount of deep insertions increases with the dimension of the lattice. This is due to the also

increasing amount of reduction steps which simply implies a higher probability that deep insertions can be performed.

As for the late size-reduction variant it turns out that this algorithm performs well for small-dimensional test lattices ($n < 30$) but lacks stability otherwise, even though provisions for correcting floating point errors and preventing error propagation were already taken. For stability reasons it seems to be crucial that at stage l of the reduction process all Gram-Schmidt coefficients $\mu_{i,j}$ with $1 \leq j \leq l-1$ are size-reduced before a possible step back might occur (due to a large size-reduction coefficient), thus allowing a faster decrease in the size of the intermediate results as it is the case in the late size-reduction algorithm.

Applying the modified size-reduction algorithm for reducing the lattice bases causes no stability problems. But even though the size-reduction process was simplified, i.e., in the case of a large reduction coefficient, the original size-reductions were replaced with simple shift operations, it turned out that in most cases the heuristic does not improve the reduction time. This is due to the fact that in many cases, the shift operations are not accurate enough in a sense that after size-reducing the Gram-Schmidt coefficient $\mu_{i,j}$, $|\mu_{i,j}|$ is still far off from being less or equal to 0.5, thus causing additional operations.

In summary, one may say that for reducing unimodular lattice bases neither the application of the deep insertion mechanism nor the checks on the special case $|\mu_{i,j}| = 0.5$ are useful since already the classical Schnorr-Euchner algorithm yields a minimal basis for unimodular lattices and none of the mechanisms yields an advantage with respect to the run time. For knapsack lattice bases the additional checks in the case of $|\mu_{i,j}| = 0.5$ make no big difference in the run time but might yield shorter basis vectors in some cases, thus supporting the application of this heuristic in practice. Using the deep insertion mechanism for reducing knapsack lattice bases has the disadvantage of increasing the run time but also the advantage of a decrease of the defect and average length of the basis vectors.

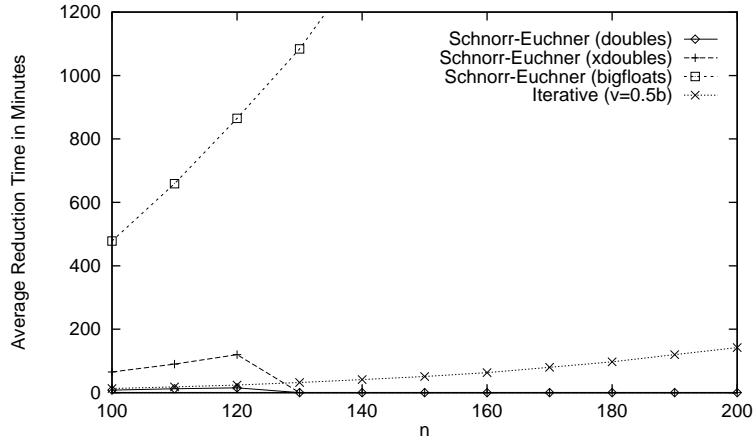
5.1.2 Limits. In the following, we will concentrate on tests of the limits of the classical Schnorr-Euchner algorithm and how improvements can be achieved by using the newly-developed dynamic approximation, modular and the iterative variant of the classical reduction algorithm. In this context, limits are either meant to be the bounds at which the other variants begin to out-perform the classical algorithm or the bounds from which on `xdoubles` (floating point arithmetic with twice the precision of `doubles`) or even `bigfloats` (multi-precision floating point arithmetic, see also [LiDIA Group 1999]) have to be used for doing the approximations in the classical Schnorr-Euchner algorithm in order to guarantee that the algorithm will terminate and yield an LLL-reduced lattice basis while in the case of the dynamic approximation, modular and iterative variants `doubles` are still sufficient for achieving the same result. It is important to know these limits since both the size of the input data as well as the approximations used affect the run time of the reduction algorithm fundamentally.

In the first test scenario, we have applied the classical Schnorr-Euchner algorithm doing the approximations with `doubles`, `xdoubles` and `bigfloats` and the iterative algorithm with $v = 0.25b$, $v = 0.33b$, $v = 0.5b$ and $v = 0.75b$ (doing the approximations with `doubles`) to knapsack lattice bases with $n = 100, 110, \dots, 200$ and bit lengths $b = n, 2n, 4n$. The reduction parameter was chosen as $y = 0.99$.

In the second set-up, we were focusing on reducing lattice bases $B \in \mathbb{Z}^{n \times n}$ with large entries where the corresponding lattice $L(B)$ has a small determinant $\det(L) = \Delta$, i.e., $n = 10, \dots, 100$, $b = 200, 400$ and $\Delta \in [1, 2^{32}]$. The tests were performed using the classical Schnorr-Euchner algorithm, the modular variant `Modular_1` and `Modular_2 = Schnorr-Euchner($\underline{b}_1 \bmod \Delta, \dots, \underline{b}_n \bmod \Delta, \Delta I_n$)` (doing the approximations by means of `doubles` and using reduction parameter $y = 0.99$).

The tests on knapsack lattice bases show (see [Backes and Wetzel 2000]) that using the Schnorr-Euchner reduction algorithm and doing the approximations by means of `doubles` works well for lattice bases with $b = n$ and $b = 2n$. In the case of $b = 4n$ and starting at $n = 130$, the approximations using `doubles` are no longer exact enough, thus resulting in a non-reduced basis. At the same time, using `xdoubles` is not sufficient either. This is due to the fact that `xdoubles` have twice the precision of `doubles` but no larger exponent [LiDIA Group 1999]. Consequently, not only the precision but also the size of the exponent of the approximation is crucial for the stability of the algorithm. Using `bigfloats` with four times the precision of `doubles` and an enlarged exponent increases the reduction time considerably but yields a correctly reduced lattice basis. A major improvement can be achieved by using the iterative algorithms (see Figure 2).

The results of the iterative variant, choosing $v = 0.25b$, $v = 0.33b$ and $v = 0.5b$ (v is the amount of additional digits of the original input data which are included in each new iteration step) show that doing the


 Figure 2. Different Variants: Knapsack Lattice Bases ($b = 4n$)

approximations with `doubles` is sufficient for all test classes. However, since several lattice basis reductions have to be done in the course of the iterative algorithm, the iterative algorithm does not out-perform the classical Schnorr-Euchner algorithm until the Schnorr-Euchner algorithm requires `xdoubles` (`bigfloats`) for the approximations while the iterative variant still works with `doubles` (`xdoubles`). These behavioral characteristics of the iterative variant and the classical Schnorr-Euchner algorithm (in combination with `doubles`, `xdoubles` and `bigfloats` for the approximations) are illustrated in Figure 2 for knapsack lattice bases with $b = 4n$.

In Figure 2 it can be seen that as long as the approximations are done with `doubles` the run time of the iterative implementation is about twice that of the Schnorr-Euchner algorithm. Furthermore, the data for

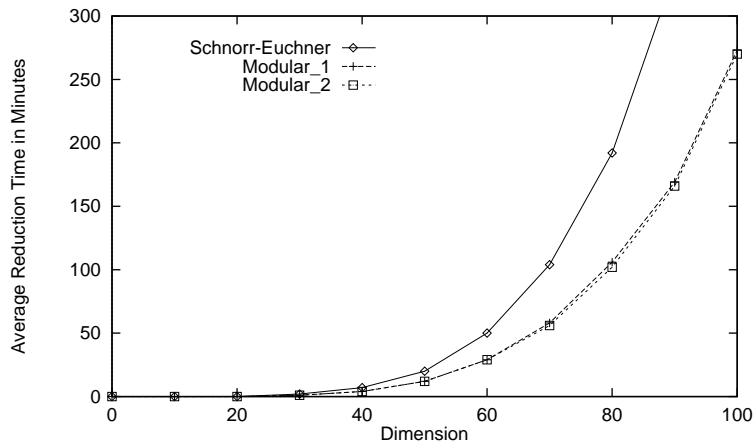


Figure 3. Different Variants: Lattices with Small Determinant

the iterative lattice basis reduction algorithms show that the reduction time decreases as v increases since a large v requires fewer iterations. At the same time, the computations have to be performed on larger operands, thus possibly causing the same problems as in the case of using the classical Schnorr-Euchner algorithm. For example, for $v = 0.75$, $n \geq 170$ and $b = 4n$ it is no longer sufficient to do the approximations by means of `doubles`. On the other hand, for large lattice bases with huge entries, the run time decreases as v decreases. In this case, the advantage that the computations can be done on small operands predominates the disadvantage that many iterations have to be performed. Hence, these observations show that the size of v and thus the amount of iterations has to be chosen skillfully in order to obtain the best possible results.

The modular variants out-perform the classical Schnorr-Euchner algorithm for increasing b . This is due to the fact that using the modular variants, the operands on which the computations are performed are

much smaller in size than in the case of applying the classical Schnorr-Euchner algorithm. For increasing b this advantage compensates the disadvantage of the additional computations in the case of the modular variants, necessary in order to guarantee that the correct LLL-reduced basis is computed in any case. The performance of the algorithms is impressively demonstrated in Figure 3 for lattices with a small determinant and $b = 400$. Figure 3 also shows that the differences in the run times of the modular variants are small. Consequently, after the initial modular reduction of the lattice only few additional modular reductions have to be performed in the course of the size-reductions in the algorithm Modular_1.

In a third scenario, we have first generated bases of knapsack lattices for $n = 10, 15, 20, \dots, 80$ and bit length $b = 50$ and random lattices for $n = 10, 15, 20, \dots, 80$, bit length $b = 6$ and $\Delta \leq 10$. Subsequently, in both cases (i.e., for knapsack lattice bases as well as random lattice bases), the bases B were then multiplied by a unimodular transformation matrix T (i.e., $\det(T) = \pm 1$ and $t_{i,j} \in \mathbb{Z}$ for $1 \leq i, j \leq n + 1$) whose entries are taken from $(0, 2^l)$ with $l = 500$ for knapsack lattices and $l = 750$ for random lattices. This scenario is motivated by the fact that recently, various public key cryptosystems based on lattice problems have been introduced (c.f. [Goldreich et al. 1997]), where the private key is an “easy to reduce lattice basis” and the public key is a (unimodular) transformation of the private key, such that the corresponding basis of the same lattice cannot be reduced efficiently using current state-of-the-art lattice reduction methods.

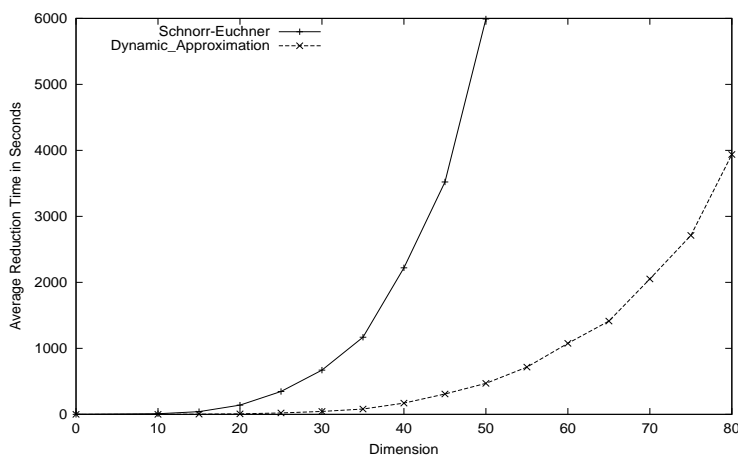


Figure 4. Dynamic Approximation: Knapsack Lattice Bases

Figures 4 and 5 demonstrate that the new algorithm implementing the dynamic approximation heuristic (thus allowing approximations with `doubles`) outperforms the standard implementation of the Schnorr-Euchner algorithm considerably.

To sum up, one may say that for large dimensions or large entries of the lattice bases, the newly-proposed dynamic approximation, modular and the iterative variants out-perform the classical Schnorr-Euchner algorithm considerably.

5.2 Additional Tests

In addition to the tests described so far, experiments with varied reduction parameters respectively series of reduction parameters and varied scalar products have been performed. Moreover, the use of performing the reductions based on the Gram matrix instead of the original basis of the lattice was tested [Wetzel 1998]. These tests have been performed in order to develop strategies for the skillful application of the reduction algorithms discussed above in order to achieve an additional speed-up in the run time or an improvement in the quality of the reduced lattice bases without changing the underlying algorithm itself.

5.2.1 Different Reduction Parameters y . At first we are interested in the influence of the reduction parameter y on the run times and the quality of the reduced lattice bases. The tests were performed using the original Schnorr-Euchner algorithm and `doubles` for the approximations. As reduction parameters we have chosen $y = 0.75$, $y = 0.875$, $y = 0.99$ as well as a sequence of reduction parameters $y = \frac{3}{4}, \frac{7}{8}, \frac{15}{16}, \frac{31}{32}, \frac{63}{64}, 0.99$ (i.e., six reductions with increasing reduction parameter y were performed).

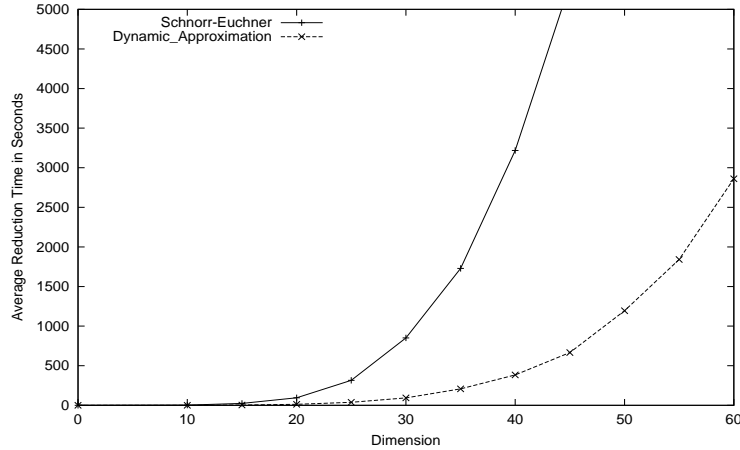
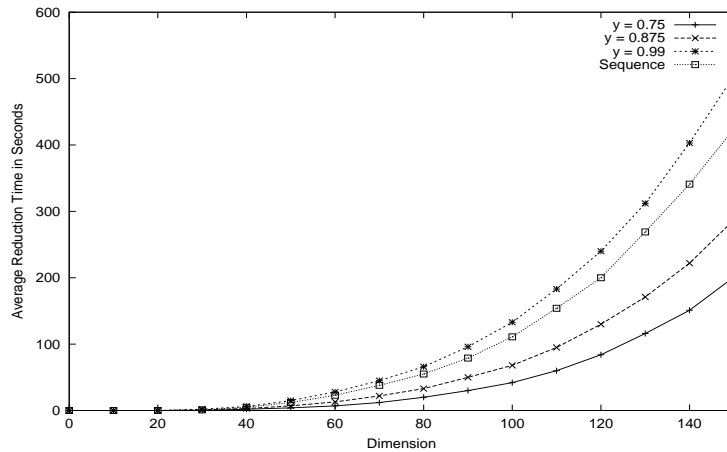


Figure 5. Dynamic Approximation: Random Lattice Bases

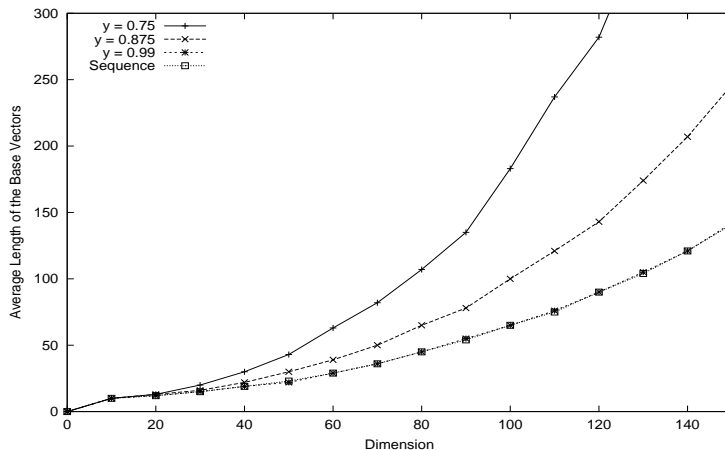
The test data in [Backes and Wetzel 2000] show that for reducing unimodular lattice bases it is sufficient to use the reduction parameter $y = 0.75$ in order to compute the LLL-reduced basis consisting of the vectors $\pm e_i$ with $1 \leq i \leq n$.

In the case of reducing random lattice bases, using $y = 0.875$ instead of $y = 0.75$ increases the run time up to 80% without implying a major decrease in the defect or the average length of the vectors in the reduced lattice basis. For $y = 0.99$ there is an additional increase of the run time by more than 200%. Not even the application of a sequence of reductions improves the quality of the bases noticeably. These observations are due to the fact that random lattice bases are already quite well reduced.


 Figure 6. Different Reduction Parameters: Knapsack Lattice Bases (Timings for $b = 2n$)

As for reducing knapsack lattice bases, it turns out that using the reduction parameter $y = 0.875$ instead of $y = 0.75$ increases the run time by approximately 40% but at the same time implies a decrease in the average length of the base vectors up to 60%. For $y = 0.99$ the reduction time is 60% higher than in the case of using $y = 0.875$ but the average length decreases by up to 30%.

5.2.2 Different Scalar Product. Depending on the application of lattice basis reduction (e.g., solving knapsack problems or doing extended gcd computations [Havas et al. 1994]) one might be interested in doing the reductions such that the row sum norm of particular rows of the lattice basis is minimized. For that purpose

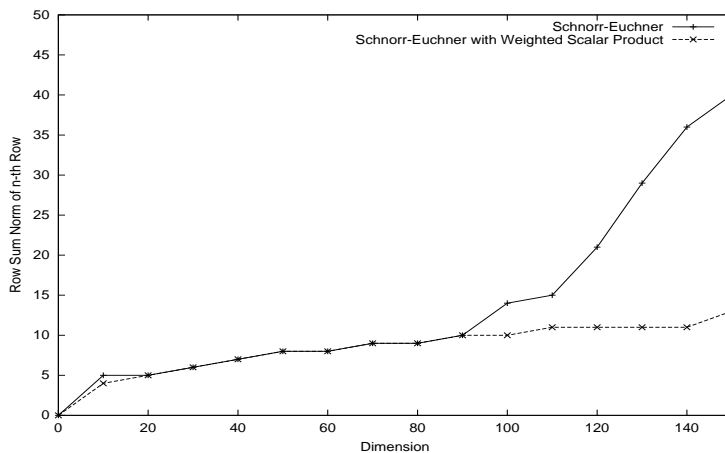
Figure 7. Different Reduction Parameters: Knapsack Lattice Bases (Av. Len. for $b = 2n$)

we have tested the effect of the use of a weighted scalar product

$$\langle \underline{u}, \underline{v} \rangle = \sum_{i=1}^{l-1} u_i v_i + \gamma^2 u_l v_l + \sum_{i=l+1}^n u_i v_i$$

($\underline{u}, \underline{v} \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}$ sufficiently large) on the row sum norm of the l -th row of the lattice basis ($1 \leq l \leq n$).

The test results in [Backes and Wetzel 2000] show that for random lattice bases the row sum norm can significantly be reduced by using the weighted norm (up to a factor 1000). The same tendency can be observed for knapsack lattice bases (see Figure 8).

Figure 8. Different Scalar Products: Knapsack Lattice Bases ($b = n$)

However, for the latter test class the improvement is less significant since with the use of the factor W one already attempts to have the corresponding row dominate the reduction process. The difference to the approach of using a weighted norm is that the effect of W is essentially confined to the beginning of the reduction process whereas the factor γ of the weighted norm basically influences the reduction process at any time.

5.2.3 Gram Matrices. The following tests have been performed to see whether it is reasonable in practice to first compute the Gram matrix $G = B^T B$ for a given lattice basis B and then use the Gram matrix for the reduction instead of the original lattice basis. This consideration is motivated by the fact that Gram

matrices consist of the scalar products which can therefore directly be used in the reduction process, thus saving the corrections of the scalar products during the orthogonalization (see [Wetzel 1998]).

The timings in [Backes and Wetzel 2000] show that for unimodular lattice bases it is possible to achieve a speed-up of the reduction by up to 40% by first computing the associated Gram matrix and then applying the corresponding reduction algorithm. In Figure 9, this behavior is pictured for unimodular lattices with $b = 100$.

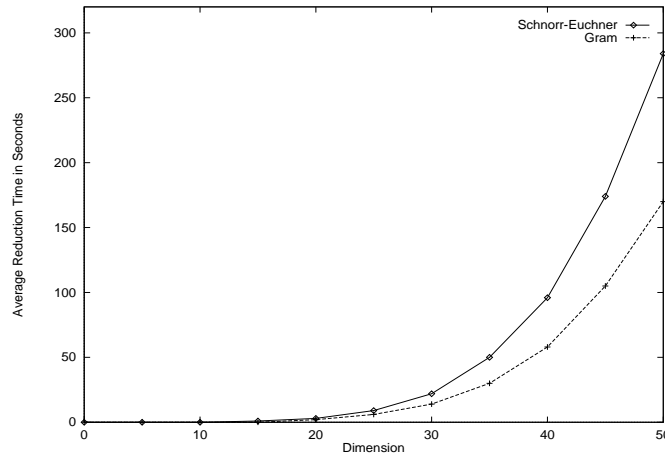


Figure 9. Gram Heuristic: Unimodular Lattice Bases

For random lattice bases the so-called Gram heuristic has advantages only if the entries in the base vectors are small, i.e., when the size of the entries is less than 50 bits.

In summary, it may be said that for achieving the best reduction results in the case of knapsack lattice bases, it is recommended to apply a sequence of reductions with varied reduction parameters. For reducing unimodular and random lattice bases in general it is sufficient to use the reduction parameter $y = 0.75$. By using a weighted scalar product to reduce a given lattice basis it is possible to decrease the row sum norm of a particular row of that lattice basis significantly. Moreover, the Gram heuristic is only advisable for reducing unimodular lattice bases.

5.3 General Heuristic

Based on the comprehensive analysis of the test results we can now deduce reduction strategies for the general use of the different variants of the Schnorr-Euchner algorithm in order to achieve the best possible reduction results or to minimize the necessary reduction time:

In the case of reducing knapsack lattice bases, it has to be noted that in general these two goals cannot be achieved at the same time. Generally, the following heuristic is suggested:

HEURISTIC 8. *In order to minimize the run time for reducing small knapsack lattice bases use the classical Schnorr-Euchner algorithm in combination with a sequence of reduction parameters doing the approximations with doubles. For large lattice bases or lattice bases where the size of the entries is large (more than 400 bits) use the iterative algorithm also in combination with a sequence of reduction parameters. In order to maximize the quality of the reduction of bases corresponding to knapsack lattices apply the deep insertion heuristic doing the approximations with doubles. For large lattice bases with huge entries (more than 400 bits) use the iterative algorithm in combination with the deep insertion mechanism.*

Since the tests have shown that randomly chosen lattice bases are already significantly reduced, it is suggested to LLL-reduce them in the following way:

HEURISTIC 9. *For reducing randomly chosen lattice bases use the classical Schnorr-Euchner algorithm with $y = 0.75$ doing the approximations by means of doubles. If the determinant is known for large $(n \times n)$ -*

lattice bases, or $(n \times n)$ -lattice bases with large entries, apply the modular reduction variant. Otherwise, use the classical Schnorr-Euchner algorithm and adjust the approximations if necessary.

In the case of unimodular lattice bases, the following heuristic should be applied:

HEURISTIC 10. *For reducing unimodular lattice bases compute the corresponding Gram matrix and apply the Schnorr-Euchner reduction algorithm with reduction parameter $\gamma = 0.75$ and `doubles` for the approximations. For large lattice bases with huge entries adjust the approximations if necessary.*

For a given lattice basis which does not necessarily belong to any of the classes discussed so far we suggest the following method for reducing the given basis:

HEURISTIC 11. *If the given lattice basis is sparse, apply the proposed heuristics for knapsack lattice bases. In the case of a dense lattice basis where no structural information is known, proceed as in the case of randomly chosen lattices. If structural information like the one in the third setting in Section 5.1.2 is known, apply the proposed heuristic for randomly chosen, replacing the modular variant with the dynamic approximation heuristic.*

If the run time is not of primary importance (e.g., when using lattice basis reduction as precomputation for the exponential enumeration step when computing the shortest vector of a lattice [Backes 1998]), the quality of the reduction of a lattice basis can be even further improved by repeatedly sorting and mixing up the reduced basis (by performing weight-reductions, permuting the basis randomly or using so-called Hadamard transformation matrices) and reducing the basis again [Backes 1998; Wetzel 1998]. For example, in Figure 10 it can be seen, that combining deep insertion with sorting, i.e., reapplying the reduction algorithm to a sorted basis results in an improved defect which directly corresponds to a better reduction result.

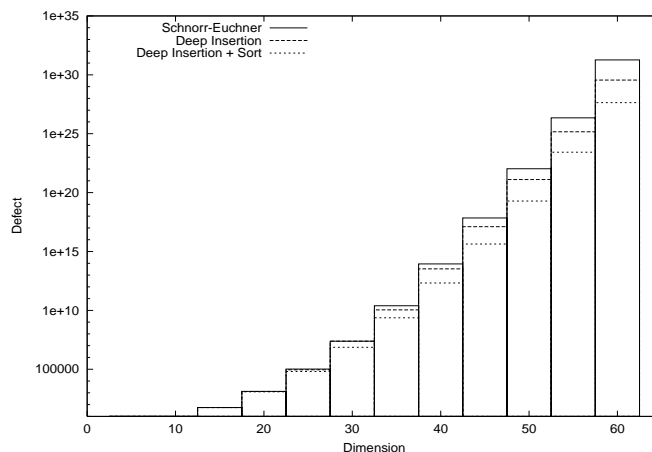


Figure 10. Combined Deep Insertion/Sort Heuristic

6. CONCLUSIONS

In this paper, we have presented various new heuristics and analyzed a comprehensive series of tests on the practical performance of the different variants of the Schnorr-Euchner algorithm. Based on these results, we have introduced heuristics for the general application of these lattice basis reduction algorithms designed to minimize the reduction time or achieve a best possible reduction result in practice. For any given reduction algorithm (e.g., Korkine-Zlotarev-reduction [Pohst and Zassenhaus 1989]), we believe that in order to draw similar conclusions about the best reduction strategy, experimentation and analysis similar to that presented herein must be performed.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their critical and valuable comments.

REFERENCES

- AJTAI, M. 1996. Generating Hard Instances of Lattice Problems. *Proceedings of the 28th ACM Symposium on Theory of Computing*, 99–108.
- AJTAI, M. AND DWORK, C. 1997. *Proceedings of the 29th ACM Symposium on Theory of Computing*, 284–293.
- BACKES, W. 1998. Berechnung kürzester Gittervektoren. *Master's Thesis, Universität des Saarlandes, Saarbrücken, Germany*
- BACKES, W., AND WETZEL, S. 2000. New Results on Lattice Basis Reduction in Practice. *Proceedings of Fourth Algorithmic Number Theory Symposium (ANTS IV)*, 135–152. Springer Lecture Notes in Computer Science LNCS 1838.
- BACKES, W., AND WETZEL, S. 2000. Lattice Basis Reduction with Dynamic Approximation. *Proceedings of Workshop on Algorithm Engineering (WAE)*, 63–73. Springer Lecture Notes in Computer Science LNCS 1982.
- BACKES, W., AND WETZEL, S. 2000. http://www.mpi-sb.mpg.de/~backes/lattice_tests.
- BIEHL, I., BUCHMANN, J., AND PAPANIKOLAOU, T. 1995. LiDIA: A Library for Computational Number Theory. *Technical Report 03/95, SFB 124, Universität des Saarlandes, Saarbrücken, Germany*
- COHEN, H. 1993. *A Course in Computational Algebraic Number Theory. Second Edition*. Springer-Verlag, Heidelberg.
- COPPERSMITH, D. 1996. Finding a Small Root of a Univariate Modular Equation. *Proceedings of EUROCRYPT '96*, 155–165. Springer Lecture Notes in Computer Science LNCS 1070.
- COSTER, M.J., JOUX, A., LAMACCHIA, B.A., ODLYZKO, A.M., SCHNORR, C.P., AND STERN, J. 1992. Improved Low-Density Subset Sum Algorithms. *Journal of Computational Complexity, Vol. 2*, 111–128.
- DOMICH, P.D., KANNAN, R., AND TROTTER, L.E. 1987. Hermite Normal Form Computation using Modulo Determinant Arithmetic. *Mathematics Operations Research, Vol. 12, No. 1*, 50–59.
- GOLDREICH, O., GOLDWASSER, S., AND HALEVI, S. Public-Key-Cryptosystems from Lattice Reduction Problems. *Proceedings of CRYPTO '97*, 112–131. Springer Lecture Notes in Computer Science LNCS 1294.
- GRÖTSCHHEL, M., LOVÁSZ, L., AND SCHRIJVER, A. 1993. *Geometric Algorithms and Combinatorial Optimization. Second Edition*. Springer-Verlag, Heidelberg.
- HAVAS, G., MAJEWSKI, B.S., AND MATTHEWS, K.R. 1994. Extended GCD Algorithms. *Technical Report TR0302, The University of Queensland, Brisbane*.
- HECKER, C. 1994. Automatische Parallelisierung und parallele Gitterbasisreduktion. *PhD Thesis, Universität des Saarlandes, Saarbrücken*.
- JOUX, A., AND STERN, J. 1998. Lattice Reduction: A Toolbox for the Cryptanalyst. *Journal of Cryptology, Vol. 11, No. 3*, 161–185.
- KALTOFEN, E. 1983. On the Complexity of Finding Short Vectors in Integer Lattices. *Proceedings of EUROCAL '83*, 236–244. Springer Lecture Notes in Computer Science LNCS 162.
- KNUTH, D.E. 1981. *The Art of Computer Programming. Volume 2: Seminumerical algorithms. Second Edition*. Addison-Wesley, Reading, Massachusetts.
- LENSTRA, A.K., LENSTRA, H.W., AND LOVÁSZ, L. 1982. Factoring Polynomials with Rational Coefficients. *Math. Ann. 261*, 515–534.
- LENSTRA, H.W. 1983. Integer Programming With a Fixed Number of Variables. *Mathematics Operations Research, Vol. 9*, 538–548.
- LiDIA GROUP 1999. LiDIA Manual. *Universität des Saarlandes/TU Darmstadt, Germany*. <http://www.informatik.tu-darmstadt.de/TI/LiDIA>.
- MAGMA 1999. <http://www.maths.usyd.edu.au:8000/comp/magma/Overview.html>.
- NGUYEN, P. 1999. Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97. *Proceedings of Crypto '99*, 288–304. Springer Lecture Notes in Computer Science LNCS 1666.
- NGUYEN, P., AND STERN, J. 1998. Cryptanalysis of a Fast Public Key Cryptosystem Presented at SAC '97. *Proceedings of Selected Areas in Cryptography '98*, 213–218. Springer Lecture Notes in Computer Science LNCS 1556.
- NLT 1999. <http://www.cs.wisc.edu/~shoup/nlt>.
- POHST, M.E., AND ZASSENHAUS, H.J. 1989. *Algorithmic Algebraic Number Theory*. Cambridge University Press.
- RADZISZOWSKI, S., AND KREHER, D.L. 1988. Solving Subset Sum Problems with the L^3 Algorithm. *J. Combin. Math. Combin. Computation, Vol. 3*, 49–63.
- RICKERT, N.W. 1989. Efficient Reduction of Quadratic Forms. *Proceedings of Computers and Mathematics '89*, 135–139.
- SCHNORR, C.P., AND EUCHNER, M. 1991. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. *Proceedings of Fundamentals of Computation Theory '91*, 68–85. Springer Lecture Notes in Computer Science LNCS 529.
- SZYDLO, M. 2001. Personal communication.
- DE WEGER, B. 1988. Algorithms for Diophantine Equations. *PhD Thesis, Centrum voor Wiskunde en Informatica, Amsterdam, Netherlands*.
- WETZEL, S. 1998. Lattice Basis Reduction Algorithms and their Applications. *PhD Thesis, Universität des Saarlandes, Saarbrücken, Germany*.