BoostReduce – A Framework For Strong Lattice Basis Reduction

Werner Backes and Susanne Wetzel

Stevens Institute of Technology
Castle Point on Hudson, Hoboken, NJ 07030 USA
Email: {wbackes,swetzel}@cs.stevens.edu

Abstract In this paper, we propose a new generic reduction framework `BoostReduce` for strong lattice basis reduction. At the core of our new framework is an iterative method which uses a newly-developed algorithm for finding short lattice vectors and integrating them efficiently into an improved lattice basis. We present `BoostBKZ` as an instance of `BoostReduce` using the Block-Korkine-Zolotarev (BKZ) reduction. `BoostBKZ` is tailored to make effective use of modern computer architectures in that it takes advantage of multiple threads. Experimental results of `BoostBKZ` show a significant reduction in running time while maintaining the quality of the reduced lattice basis in comparison to the traditional BKZ reduction algorithm.

Keywords Parallel Algorithm, Lattice Basis Reduction, Framework, Multi-Core, Cryptanalysis.

1 Introduction

Over the past few years, lattice theory has assumed an ever-increasing role in cryptography. Major milestones include the seminal work by Lenstra, Lenstra, and Lovász introducing the LLL lattice basis reduction algorithm; Schnorr and Euchner proposing a variant of the LLL algorithm [33]—marking a break-through in enabling efficient lattice basis reduction in practice; or the advances in providing for stronger lattice basis reduction in practice (e.g., Block Korkine-Zolotarev reduction [10, 34]). These works have led to lattice basis reduction becoming a prominent tool in cryptanalysis (e.g., [33, 11, 28, 29, 13, 14, 25]). Furthermore, lattice theory is gaining importance in designing cryptographic primitives. In particular, it is widely believed that lattice theory can provide the means for constructing cryptographic primitives that exhibit strong security even in the presence of quantum computers. Recent advances in constructing lattice-based cryptographic primitives include the introduction of the first fully homomorphic cryptosystem by Gentry [15] using ideal lattices, Peikert's work on constructing public key cryptosystems from the worst-case shortest vector problem [30], lattice-based hash functions (e.g., [31, 24]), lattice-based identification schemes [23], or the NTRU (e.g., [20, 37]) and GGH cryptosystems [16]—much of which was originally spurred by the work of Ajtai and Dwork [1–3] which was the first to prove security based on hardness assumptions of lattice problems.

One question that is key to lattice theory and especially its applications to cryptography concerns the efficient finding of a short or the shortest lattice vector for a given basis in practice. While considerable progress has been made in recent years (e.g., [12, 22, 26, 27, 6, 8, 40]), much needs yet to be understood better—in particular with respect to the gap between theory and practice—and further advances must be achieved. It is in this context that this paper introduces a new reduction framework `BoostReduce` that provides a novel approach to strong lattice basis reduction tailored to take advantage of today's modern multi-core computer architectures. While major advances in improving and parallelizing the Schnorr-Euchner LLL reduction in practice have been achieved for quite some time already (e.g., [6, 8, 40, 26]), there was little to no progress in parallelizing stronger lattice basis reduction algorithms, such as the Block-Korkine-Zolotarev (BKZ) reduction, until very

recently. Pujol et al. [35] developed a first parallel algorithm for finding the shortest lattice vectors based on a variant of Kannan's algorithm [21, 17]. The work of Hermans et al. [18, 19] focuses on a similar parallel algorithm for finding shortest lattice vectors using CUDA capable graphics cards. Both works constitute an important step towards a parallel BKZ reduction algorithm, but neither of these algorithms does yet support pruning techniques which are essential for enabling efficient, parallel BKZ reduction in practice—especially when considering typical problem sizes in many of today's cryptographic contexts.

Our new reduction framework pursues a different approach. The core structure of `BoostReduce` is based on the use of sequences of lattice basis reductions [4, 5]. As a main component of the novel framework we introduce a new set of intermediate steps in between the individual reductions that are tailored to further improve the lattice basis and thus the starting point for the subsequent lattice basis reduction. This is combined with a sophisticated tightening of the parameters for the intermediate steps as well as the reduction algorithm itself. In addition, the new framework supports the execution of multiple heuristics in parallel and as such provides the means to select an optimal lattice basis as the starting point for the next round of the overall reduction process.

As a second main contribution of this paper, we introduce `BoostBKZ` as an instance of our new framework `BoostReduce` which is based on the BKZ reduction algorithm. In particular, as part of `BoostBKZ` we develop a new parallel algorithm for finding short lattice vectors that makes effective use of the multi-core features of today's computer architectures.

The third main component of this paper is an extensive analysis of the performance of `BoostBKZ`. Our experiments show that `BoostBKZ` provides a significant improvement in practice over conventional BKZ reduction.

Outline: Section 2 provides the definitions and notations used in the remainder of the paper. Section 3 introduces our new reduction framework `BoostReduce` and Section 4 details `BoostBKZ` as instance of our framework `BoostReduce`. Section 5 presents our new parallel algorithm for finding short lattice vectors and Section 6 describes an algorithm for incorporating these short lattice vectors into an improved lattice basis. Section 7 provides a detailed analysis of `BoostBKZ`. The paper closes with some directions for future work.

2 Preliminaries

A *lattice* $L = \left\{ \sum_{i=1}^{k} x_i \underline{b}_i | x_i \in \mathbb{Z}, 1 \leq i \leq k \right\} \subset \mathbb{R}^n$ is an additive discrete subgroup of $\mathbb{R}^n$. The linear independent vectors $\underline{b}_1, \ldots, \underline{b}_k \in \mathbb{R}^n$ $(k \leq n)$ form a basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{R}^{n \times k}$ of dimension k of the lattice L. The basis of a lattice is not unique. A basis B may be transformed into a basis B' of the same lattice L by applying a *unimodular transformation* U, i.e., $B' = BU$ with $U \in \mathbb{Z}^{n \times k}$ and $|\det U| = 1$.

The *Gram-Schmidt orthogonalization* $B^* = (\underline{b}_1^*, \ldots, \underline{b}_k^*)$ of a lattice basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{R}^{n \times k}$ is computed as $\underline{b}_1^* = \underline{b}_1$, $\underline{b}_i^* = \underline{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \underline{b}_j^*$ for $2 \leq i \leq k$ where $\mu_{i,j} = \frac{\langle \underline{b}_i, \underline{b}_j^* \rangle}{\|\underline{b}_j^*\|}$ for $1 \leq j < i \leq k$ where $\langle ., . \rangle$ defines the scalar product of two vectors. It is important to note that B^* is not necessarily a basis for the lattice L, nor is a vector $\underline{b}_i^*$ of the orthogonalization B^* necessarily in L.

The *defect of a lattice basis* $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{R}^{n \times k}$ defined as $\mathrm{dft}(B) = \frac{\prod_{i=1}^{n} \|\underline{b}_i\|}{\det(L)}$ allows one to compare the quality of different bases. Obviously, $\mathrm{dft}(B) \geq 1$ and $\mathrm{dft}(B) = 1$ for an orthogonal basis. The goal of lattice basis reduction is to determine a basis with smaller defect. That is, for a lattice $L \subset \mathbb{R}^n$ with bases B and $B' \in \mathbb{R}^{n \times k}$, B' is better reduced than B if $\mathrm{dft}(B') < \mathrm{dft}(B)$.

For a lattice $L \subseteq \mathbb{Z}^n$ with basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, corresponding Gram-Schmidt orthogonalization $B^* = (\underline{b}_1^*, \ldots, \underline{b}_k^*) \in \mathbb{R}^{n \times k}$, and coefficients $\mu_{i,j}$ with $1 \leq j < i \leq k$, $\pi_i :\longrightarrow span(\underline{b}_1, \ldots, \underline{b}_{i-1})^{\perp}$ for $1 \leq i \leq k$ denotes the orthogonal projection with $\pi_i(\underline{b}_j) = \sum_{s=i}^{k} \mu_{js} \underline{b}_s^*$. A basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$ for lattice $L \subseteq \mathbb{Z}^n$ is *Block Korkine-Zolotarev* reduced for block size $\beta \in \mathbb{N}$ and reduction parameter $\frac{1}{4} < y < 1$, iff $|\mu_{ij}| \leq \frac{1}{2}$ for $1 \leq j < i \leq k$ (Eq. 1) and $y\|\pi_i(\underline{b}_i)\| \leq \lambda_1(L(\pi_i(\underline{b}_i), \ldots, \pi_i(\underline{b}_{min(i+\beta-1,n)})))$ (Eq. 2) where λ_1 denotes the length of the shortest non-zero vector of $L(\pi_i(\underline{b}_i), \ldots, \pi_i(\underline{b}_{min(i+\beta-1,n)}))$ [32].

3 The New Approach

In the following, we introduce and motivate the three main ideas of our new algorithmic framework tailored to significantly decrease the running time of strong lattice basis reduction algorithms in practice—thus enabling the tackling of large lattice bases.

The first idea in designing the framework is to iteratively reduce a lattice basis using a sequence of reduction parameters that increase in size. This general heuristic previously proved beneficial in the context of LLL reduction [4, 5]. The second idea is the introduction of suitable intermediate steps in between two iterations with the goal of further improving the lattice basis and thus providing for a better starting point for the subsequent reduction algorithm. In [7], for example, simple modifications—such as sorting—resulted in a decrease of the running time when LLL reducing unimodular lattice bases. The third idea for our new approach is to make effective use of today's multi-core computer architectures by exploiting the inherent parallel capabilities due to multiple cores. Specifically, our framework is designed to allow for the executing of the same basic sequential algorithm while employing differing algorithmic heuristics or using varying inputs on different cores. As such it is possible to execute multiple heuristics or treat different inputs at once with the benefit that the best one among multiple results can be selected as the starting point for subsequent steps.

These three basic ideas lead to the following structure of our new `BoostReduce` reduction framework (see Algorithm 1): At the beginning, we perform an initial reduction using a weak set of parameters. The main loop of `BoostReduce` (Lines `(2)` - `(7)`) corresponds to the overall iterative algorithmic framework of our new approach. In each iteration of the main loop, we first use an algorithm for finding a set S of linear combinations of short lattice vectors.

Algorithm 1: `BoostReduce`

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$,
P_i parameter set for *ReduceBasis*, E_i parameter set for *FindShortLV*
G_j parameter set for generating improved bases
t number of allocated threads, r number of steps
OUTPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k)$, BKZ reduced

(1) $B = ReduceBasis(B, P_0)$
(2) **for** $(1 \leq i \leq r)$ **do**
(3) $S = FindShortLV(B, E_i)$
(4) Use set S and parameters $G_1, \ldots, G_t$ to generate improved bases $B'_1, \ldots, B'_t$
(5) Run $B''_j = ReduceBasis(B'_j, P_i)$ with $1 \leq j \leq t$
in parallel until `maxred` instances have finished.
(6) Select l such that $p_l = \min\{p_j | 1 \leq j \leq t\}$ with $p_j = \prod_{i=1}^{k} \|b''_{j_i}\|$
(7) $B = B''_l$

The set S of linear combinations of short lattice vectors serves as input for generating a number of improved lattice basis B'_j with $1 \leq j \leq t$ (see Line `(4)`) where the number t corresponds to the number of threads allocated for the execution of `BoostReduce`. Subsequently, all improved lattice

bases B'_j are then reduced in parallel. In order to limit the time spent on the reductions and keeping the overall algorithm balanced, ongoing reductions as part of Line `(5)` are terminated once `maxred` number of the reductions are completed. Then, the best reduced lattice basis among the results of all completed reductions is selected as the input for the next iteration of `BoostReduce`. This selection can be done efficiently by searching for the basis with a minimal product of the lengths of its basis vectors.[1]

In the following, we detail `BoostBKZ` as a concrete instance of our new reduction framework `BoostReduce` based on BKZ reduction. In particular, as part of designing `BoostBKZ` we develop a new parallel algorithm `FindShortLV` for finding short lattice vectors, introduce a new algorithm for generating an improved lattice basis, and show how the various parameter sets need to be chosen in a suitable fashion.

4 BoostBKZ — An Instance Of BoostReduce Using BKZ

In seeking for a strong lattice basis reduction method, BKZ often is the method of choice. While the LLL lattice basis reduction algorithm generally outperforms BKZ reduction in terms of the reduction time, BKZ generally provides for better reduction results (for appropriately chosen parameters). Generally, the quality of a BKZ reduced lattice basis depends on the reduction parameter δ (see Eq. 2), the block size β, and the so-called pruning parameter for the enumeration process [10, 34].

`BoostBKZ` (see Algorithm 2) is the instantiation of the `BoostReduce` framework for the BKZ lattice basis reduction algorithm. The main component of `BoostBKZ` is the subroutine $\texttt{ThreadBoostBKZ}_\texttt{j}$ (see Algorithm 3) which allows for the reduction of the improved lattice bases in parallel. `BoostBKZ` first performs an initial BKZ reduction in Line `(1)`. The main loop (Lines `(3)` - `(9)`) implements the finding of short lattice vectors, generating, and reducing the improved lattice basis.

Algorithm 2: `BoostBKZ`

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$,
P_i parameter set for BKZ reduction,
E_i parameter set for `FindShortLV`,
G_j parameter set for `GenerateBasis`,
`maxred` number of BKZ instances,
t number of allocated threads,
r number of steps
OUTPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k)$, BKZ reduced

(1) $B = BKZ(B, P_0)$
(2) **MUTEX_INIT**$(lock)$
(3) **for** $(1 \le i \le r)$ **do**
(4) $S = \texttt{FindShortLV}(B, E_i)$
(5) `finished` $= 0$
(6) **for** $(1 \le j \le \texttt{t})$ **do**
(7) start thread with $B''_j = \texttt{ThreadBoostBKZ}_\texttt{i}(S, P_i, G_j)$
(8) find l with $\|b''_{l_1}\| \cdot \ldots \cdot \|b''_{l_k}\|$ minimal
(9) $B = B''_l$

Algorithm 3: $\texttt{ThreadBoostBKZ}_\texttt{j}$

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$,
S set of linear combinations of short vectors,
P parameter set for BKZ reduction,
G parameter set for `GenerateBasis`,
j thread number
OUTPUT: $B^*_j = (\underline{b}_1, \ldots, \underline{b}_k)$, BKZ reduced

(1) $\texttt{B}' = \texttt{GenerateBasis}(B, G, S)$ [2]
(2) $B''_j = \texttt{BKZ}(\texttt{B}', P)$
(3) **MUTEX_LOCK**$(lock)$
(4) `finished` $=$ `finished` $+ 1$
(5) **if** (`finished` $\ge$ `maxred`) **then**
(6) kill remaining instances of threads with $\texttt{ThreadBoostBKZ}_\texttt{i}$
(7) **fi**
(8) **MUTEX_UNLOCK**$(lock)$

[1] In this case the product is sufficient to determine the quality of the reduced bases because all are bases of the same lattice. Generally, determining the quality of a reduced basis requires the considering of the defect of the lattice.

[2] In outlining our multi-threaded programs, we distinguish between variables that are local for every thread and variables that are shared among all threads. Local variables are highlighted by the use of a different font (e.g., `local` vs. *shared*).

In Line (4), BoostBKZ calls algorithm FindShortLV—a newly-developed parallel algorithm for finding short lattice vectors (see Section 5 for details). Lines (5) - (7) initialize and create the t threads running algorithm $\texttt{ThreadBoostBKZ}_\texttt{j}$ with the set S of linear combinations of short lattice vectors, the parameters for generating an improved basis, and the parameters of BKZ for the current step as input parameters. In Lines (8) - (9), the best reduced basis is selected (amongst those who finished as part of the parallel threads $\texttt{ThreadBoostBKZ}_\texttt{j}$) as a starting point for the next iteration of BoostBKZ. It is necessary to adjust and tighten the parameters in each iteration of BoostBKZ in order to improve the quality of the reduced basis. In particular, the block sizes for BKZ and the size of the subsets for FindShortLV are increased in each iteration of BoostBKZ. The respective parameter sets are P_i for the BKZ reduction and E_i for FindShortLV for iteration i of BoostBKZ. Unlike the parameter sets P_i and E_i, the parameter sets G_j do not concern the tightening or updating of parameters but rather correspond to the different strategies pursued in parallel for algorithm GenerateBasis (see Section 6 for details). The number of different strategies is directly correlated to the number of allocated threads.

In algorithm $\texttt{ThreadBoostBKZ}_\texttt{j}$, we first generate the improved lattice basis using the set S provided by FindShortLV and the parameter set G which is unique for each one of the t executed threads. This ensures that GenerateBasis does create a different B' in each thread. The global variable finished is used to count the completed BKZ instances. If the required number maxred is reached, the remaining running threads $\texttt{ThreadBoostBKZ}_\texttt{j}$ (see Lines (5) - (7), Algorithm 3) are aborted. The access to finished is protected by a mutex in order to avoid a data race condition.

In the following, we first detail our newly-developed parallel enumeration algorithm FindShortLV for finding short lattice vectors. FindShortLV is based on the idea of performing multiple enumerations in parallel. The inputs for each enumeration are small subsets of the lattice basis that are distinct with high probability. In Section 6 we then introduce a new algorithm that generates an improved lattice basis using the set of short vectors found by FindShortLV.

5 Finding Short Lattice Vectors

To date, one of the best known methods for finding the shortest lattice vector in practice is the enumeration method ENUM developed by Schnorr and Hoerner [10, 34]. However, the running time of this algorithm is exponential in the size of the lattice basis and as such is not practical for higher dimensions. Nevertheless, we use this algorithm as a subroutine as part of our new parallel algorithm for finding short lattice vectors. It is in the novel way we use the subroutine that makes this approach viable in practice. In contrast to the original use of the enumeration by Schnorr and Hoerner, we are not interested in finding only the shortest lattice vectors, but we are interested in vectors that are shorter than the current basis vectors we seek to replace. It is important to note that our variant of the ENUM algorithm keeps all intermediate results and not only the shortest vector(s) thus providing a larger set of short vectors which allows for more flexibility and opportunities in improving the lattice basis as part of the subsequent steps of BoostBKZ. Another main difference of our approach to that of Schnorr and Hoerner is that the enumeration is only applied to suitable subsets of the lattice basis.

5.1 Parallel Enumeration

FindShortLV (see Algorithm 4) and ThreadFindShortLV (see Algorithm 5) implement our parallel enumeration technique for finding short lattice vectors. FindShortLV is responsible for the proper

initialization of the global variables that are later accessed and modified by the individual threads. In addition, `FindShortLV` creates a mutex to avoid data race conditions before it spins off t threads each running algorithm `ThreadFindShortLV`.

Overall, the parallel enumeration process is carried out in q sequence steps, each of which includes the parallel enumeration of p subsets of a basis B. A simple approach for distributing the work load among all threads would be to assign each thread p/t subsets in each sequence step. This approach is not optimal because the running time of the subroutine `ENUM` depends on the chosen subset and therefore varies greatly. This behavior forces us to implement a work-stealing approach [9] in order to balance the work load equally among all threads. `ThreadFindShortLV` entails q iterations and implements the idea of updating the parameters in each sequence step of the iteration process. In Line `(2)`, the p subsets of the lattice basis vectors are chosen based on a specific strategy (see Section 5.2 for details). The thread that finishes the last one of the p subsets in the current sequence step performs the transition of the parallel enumeration process to the next sequence step (Lines `(9) - (13)`) resetting the subset counter p_c and adjusting the parameters L_c and c for `ENUM` and f, w and d for $\texttt{Strategy}_\texttt{x}$[3]. It is particularly important to note that the length of the currently shortest lattice vectors found L_c is used to limit the enumeration bounds for subsequent calls of the algorithm `ENUM`. It is updated within a particular sequence step, i.e., in the course of treating the p sets in parallel (Line `(7)`), as well as during the transition from one iteration to the next (Line `(11)`).

Algorithm 4: `FindShortLV`

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$,
$f, w, d \in \mathbb{N}$ param for $\texttt{Strategy}_\texttt{x}$
c pruning param for `ENUM`,
p no subsets per sequence step,
q no sequence steps,
f_a, w_a, d_a adjustment for f, d and w,
c_a adjustment value for c
l_1, l_2 slack factors
OUTPUT: S set of linear combinations of short lattice vectors

(1) $S = \emptyset$
(2) $L_c = \min\{\|\underline{b}_1\|^2\}, \ldots \|\underline{b}_k\|^2\}$
(3) $q_c = p_c = 0$
(4) **MUTEX_INIT**$(lock)$
(5) **for** $(1 \le i \le t)$ **do**
(6) start thread $\texttt{ThreadFindShortLV}_\texttt{i}$

Algorithm 5: $\texttt{ThreadFindShortLV}_\texttt{i}$

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$,
$f, w, d \in \mathbb{N}$ param for $\texttt{Strategy}_\texttt{x}$
c pruning param for `ENUM`,
p no subsets per sequence step,
q no sequence steps,
f_a, w_a, d_a adjustment for f, d and w,
c_a adjustment value for c
l_1, l_2 slack factors
OUTPUT: S set of linear combinations of short lattice vectors

(1) **repeat**
(2) $\texttt{Strategy}_\texttt{x}(\texttt{B}', B, f, w, d)$
(3) $\texttt{L}_\texttt{e} = L_c \cdot l_1$
(4) $\texttt{S}' = \texttt{ENUM}(\texttt{B}', \texttt{L}_\texttt{e}, c, 1, d)$
(5) **MUTEX_LOCK**$(lock)$
(6) $S = S \cup S'$
(7) $L_c = \min\{\|\underline{v}\|^2\} | \underline{v} \in S\}$
(8) $p_c = p_c + 1$
(9) **if** $(p_c \ge p)$ **then**
(10) $p_c = 0$, $q_c = q_c + 1$
(11) $L_c = l_2 \cdot L_c$
(12) $f = f + f_a$, $d = d + d_a$
(13) $w = w + w_a$, $c = c \cdot c_a$
(14) **MUTEX_UNLOCK**$(lock)$
(15) **until** $(q_c \ge q)$

In order to enable the finding of sufficiently many short lattice vectors, Algorithms 4 and 5 use so-called slack or length adjustment parameters l_1 and l_2. These parameters provide an effective means for adjusting the upper bound for the length of lattice vectors ($\texttt{L}_\texttt{e}$) considered in the process of `ENUM`. To recall, we are not only interested in finding the shortest vectors but strive to also find vectors that are close in length to the shortest lattice vectors. While the introduction of the

[3] In case of $\texttt{Strategy}_1$ (see Section 5.2), parameter w is ignored.

slack parameters significantly increases the number of short vectors found, this comes at the cost of a performance overhead. Consequently, it is crucial to strike a good balance between finding sufficiently many short vectors and not suffering too much of a performance overhead.

5.2 Selecting Suitable Subsets

We have developed a randomized heuristic for selecting suitable subsets as part of the parallel enumeration process. The heuristic approach is based on Conjecture 1 which was established as a result of extensive experiments in which the Schnorr/Hoerner enumeration method was used for determining a shortest lattice vector.

Conjecture 1. It is possible to compute short basis vectors of a lattice as a linear combination of mostly short lattice basis vectors and a few longer vectors of the respective lattice basis.

In using the conjecture in practice, we start with a reduced lattice basis. Roughly speaking, Conjecture 1 is then interpreted as mainly using lattice basis vectors located in the front of the lattice basis and combining them with a few vectors selected from the remainder of the reduced lattice basis. We have developed and evaluated three different strategies. While the first two are designed in accordance with the conjecture, the third strategy was purposely designed to violate the conjecture.

For a lattice basis $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$ and $f, d \in \mathbb{N}$ with $f < d$, our first strategy ($\texttt{Strategy}_1$) selects the first f vectors of the reduced lattice basis B. The other $d - f$ vectors are chosen at random from the remaining lattice basis vectors. $\texttt{Strategy}_2$ introduces an additional parameter w with $f < w$. The parameter w defines a window of consecutive basis vectors from which the first f vectors are selected. In $\texttt{Strategy}_2$, this window is located at the front of the lattice basis in order to comply with the conjecture. The remaining vectors are chosen at random from basis vectors outside of the window (i.e., similar to $\texttt{Strategy}_1$).

Strategy$_1$

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, $f < d$
OUTPUT: $B' = (\underline{b}'_1, \ldots, \underline{b}'_d)$

(1) $M = \emptyset$
(2) **for** $(1 \leq i \leq f)$ **do**
(3) $\quad M = M \cup \{i\}$
(4) $j = f$
(5) **while** $(j < d)$ **do**
(6) $\quad$ select i randomly out of $[f + 1, k]$
(7) $\quad$ **if** $(M \cap \{i\} = \emptyset)$ **then**
(8) $\quad\quad M = M \cup \{i\},\ \ j = j + 1$
(9) $j = 1$
(10) **for** $(1 \leq i \leq k)$ **do**
(11) $\quad$ **if** $(M \cap \{i\} \neq \emptyset)$ **then**
(12) $\quad\quad \underline{b}'_j = \underline{b}_i,\ \ j = j + 1$

Strategy$_2$

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$, $f < w < d$
OUTPUT: $B' = (\underline{b}'_1, \ldots, \underline{b}'_d)$

(1) $M = \emptyset,\ \ j = 0$
(2) **while** $(j < f)$ **do**
(3) $\quad$ select i randomly out of $[1, w]$
(4) $\quad$ **if** $(M \cap \{i\} = \emptyset)$ **then**
(5) $\quad\quad M = M \cup \{i\},\ \ j = j + 1$
(6) $j = f$
(7) **while** $(j < d)$ **do**
(8) $\quad$ select i randomly out of $[w + 1, k]$
(9) $\quad$ **if** $(M \cap \{i\} = \emptyset)$ **then**
(10) $\quad\quad M = M \cup \{i\},\ \ j = j + 1$
(11) $j = 1$
(12) **for** $(1 \leq i \leq k)$ **do**
(13) $\quad$ **if** $(M \cap \{i\} \neq \emptyset)$ **then**
(14) $\quad\quad \underline{b}'_j = \underline{b}_i,\ \ j = j + 1$

The third strategy ($\texttt{Strategy}_3$) is based on $\texttt{Strategy}_2$ but allows for the position of the window of size w to be selected at random. As a consequence, $\texttt{Strategy}_3$ chooses the majority of the basis vectors from the middle or the back part of the reduced lattice basis and as such is in clear violation of Conjecture 1.

5.3 Evaluation of Selection Strategies

The effectiveness of the three strategies in selecting subsets of lattice basis vectors was evaluated using lattice bases generated as part of a lattice challenge contest conducted by the TU Darmstadt [36]. These lattice bases are assumed to be hard instances constructed to allow the testing and comparing of lattice basis reduction algorithms. Our tests use the lattice bases of the challenge for dimensions 200 to 500 (the challenge provides one lattice basis for each dimension). Each basis was first BKZ reduced with a block size of $\beta = 20$ using the NTL [40] implementation of the BKZ reduction algorithm. Then, for each BKZ reduced basis we carried out 10 runs of `FindShortLV` (using parameters $q = 1$ and $p = 20,000$) each yielding a set of short vectors. Using the length of the shortest of the short vectors of each one of the ten sets allows us to determine the average length of the shortest of the short vectors determined by the experiment.

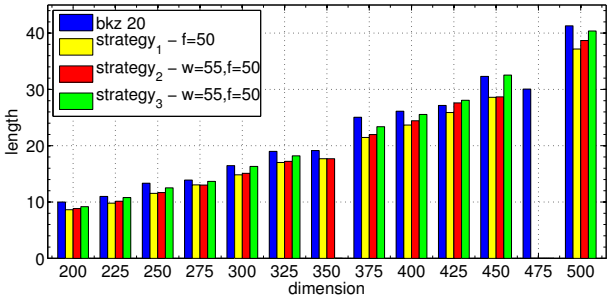


Figure 1. $\texttt{Strategy}_\texttt{x}$ for `FindShortLV` with $q = 1$ and $p = 20000$.

In a series of tests, we fix the number d of lattice vectors chosen as part of the $\texttt{Strategy}_\texttt{x}$ algorithms to $d = 75$. For $\texttt{Strategy}_1$ we chose $f = 50$. For $\texttt{Strategy}_2$ and $\texttt{Strategy}_3$ we used a window of size $w = 55$ for selecting the first $f = 50$ basis vectors. Figure 1 summarizes the results of the test. For each dimension, it shows the average length of the shortest of the short vectors averaged over the ten runs for each test instance. For the challenge lattice basis of dimension 475 none of our tested strategies were able to find short lattice vectors. $\texttt{Strategy}_3$ also failed for dimension 350. The results show that the average length of the shortest of the short vectors found by $\texttt{Strategy}_1$ is significantly smaller than that for $\texttt{Strategy}_3$. This result was expected due to the fact that $\texttt{Strategy}_3$ was designed to violate Conjecture 1 while $\texttt{Strategy}_1$ was designed in compliance with Conjecture 1. The average length for $\texttt{Strategy}_2$ is significantly better than that for $\texttt{Strategy}_3$, yet slightly inferior to $\texttt{Strategy}_1$. These results clearly support Conjecture 1 and suggest that $\texttt{Strategy}_1$ is the best among the tested strategies. We therefore use $\texttt{Strategy}_1$ exclusively for the experiments detailed in Section 7.

6 Generate An Improved Basis

In order to allow for an effective use of the short vectors found by `FindShortLV` as part of `BoostBKZ` (see Algorithm 2, Line `(4)`), some of these short vectors must first be integrated properly into a lattice basis. A straightforward approach is to add the short vectors to the lattice basis which was the starting point for `FindShortLV`, thus yielding a generating system of lattice vectors. One would then have to either transform the generating system into a basis or use a variant of the BKZ that allows the use of a generating system instead of a lattice basis. However, both of these options have a significant running time overhead. We therefore devise a new heuristic algorithm `GenerateBasis` (see Algorithm 6) which allows the efficient computing of an improved lattice basis.

Algorithm 6: `GenerateBasis`

INPUT: $B = (\underline{b}_1, \ldots, \underline{b}_k) \in \mathbb{Z}^{n \times k}$,
S set of linear combinations of short vectors,
N number of vectors to be replaced
OUTPUT: $B' = (\underline{b}'_1, \ldots \underline{b}'_k) \in \mathbb{Z}^{n \times k}$
improved basis

(1) $\underline{z} = (0, \ldots, 0)$
(2) $B' = B$
(3) $l = 0$
(4) remove trivial $\underline{s}$ with $\|\underline{s}\| = 1$ out of S
(5) **foreach** $(\underline{s} \in S)$ **do**
(6) $S = S \setminus \{\underline{s}\}$
(7) find i with $\underline{s}_i = \pm 1$ and $\underline{z}_i = 0$
(8) **if** (i exists) **then**
(9) $\underline{b}'_i = \sum_{i=1}^{k} \underline{s}_i \underline{b}_i$
(10) $l = l + 1$
(11) **for** $(1 \leq j \leq k)$ **do**
(12) **if** $(\underline{s}_j \neq 0)$ **then**
(13) $\underline{z}_j = 1$
(14) **if** $(l == N)$ **then**
(15) **break**
(16) possible repositioning (sort/random)
of newly inserted vectors

`GenerateBasis` receives as input the set S of linear combinations of short lattice vectors which was computed by `FindShortLV`. These linear combinations are key in selecting a suitable linear combination of basis vectors (which can be used to compute the actual set of short lattice vectors) that can be used to replace a longer basis vectors. Specifically, `GenerateBasis` uses a flag array, i.e., vector $\underline{z}$ that allows one to determine whether the exchange of a specific basis vector is allowed thus ensuring that one never leaves the actual lattice. At the outset, all basis vectors are possible candidates to be replaced (see Line `(1)`, Algorithm 6). The main loop of the algorithm is executed (Lines `(5) - (15)`) until a sufficiently large number N of vectors has been replaced. At the beginning of each iteration of the loop, the linear combination $\underline{s} \in S$ is chosen which represents the shortest of the short vectors left in S at that point. In order to determine whether the lattice vector corresponding to the linear combination $\underline{s}$ can be used to replace a lattice basis vector, it is necessary to check whether there is an $1 \leq i \leq k$ such that $\underline{s}_i = \pm 1$ and $\underline{z}_i = 0$ (Line `(7)`). The lattice vector corresponding to $\underline{s}$ cannot be used to replace a basis vector $\underline{b}_i$ if $\underline{s}_i \neq \pm 1$ as one would otherwise leave the lattice. It is important to note that the replacing of a lattice basis vector (Line `(9)`) must be carried out on a copy B' of input basis B as all the linear combinations in S are relative to B. Thus, a modification of B would invalidate the set of linear combinations S. The flag array $\underline{z}$ must be updated after the respective lattice basis vector has been replaced. In Lines `(11) - (13)`, every position j in $\underline{z}$ is marked where the corresponding entry in the linear combination $\underline{s}_j$ is not equal to zero—thus disallowing every basis vector which has been used as part of the linear combination $\underline{s}$ for further consideration. The final step of algorithm `GenerateBasis` allows the newly inserted vectors to be repositioned within the improved lattice basis B'.

One of challenges with this new algorithm `GenerateBasis` is to decide which vector should be replaced. The replacement is triggered by i (Line `(7)`) and each lattice vector $\underline{b}_i$ with $\underline{s}_i = \pm 1$ and $\underline{z}_i = 0$ is a proper candidate. In our experiments (see Section 7) we focus on two variants: The first variant chooses a minimal position i to allow the newly inserted vector to have an effect early on in the subsequent BKZ reduction. This variant starts with $i = 1$ and checks the $\underline{s}_i$ and $\underline{z}_i$ in a forward direction by continuously increasing i. The second variant is tailored to incorporate the length of the basis vector to be replaced. As such, the variant maximizes the position i in order to replace longer vectors. The variant starts with $i = k$ and searches for a suitable position in a backwards direction by decreasing i continuously. It, however, is important to note that positioning a short vector towards the end of the lattice basis can be a disadvantage as it takes longer for this shorter vector to have an impact on the subsequent BKZ reduction.

The repositioning step as a final step of `GenerateBasis` (see Line `(16)`) allows us to, e.g., move the inserted vectors to the front of the lattice basis and sort them according to their length. It is important to note that we do not change the order of the vectors that have not been replaced.

Some of our earlier experiments suggest that sorting the entire lattice basis may have a negative impact on the quality of the reduced basis.

7 Results

The experiments were performed on a Sun x4150 server with two quad core Intel Xeon processors at 2.83 GHz and 8 GB of main memory. We used NTL 5.5.1 [40] for the BKZ reduction, GMP 4.3.1 [38] as long integer arithmetic, and MPFR 2.3.1 [39] as multi-precision floating point arithmetic. All programs (including our own code for `GenerateBasis`, `FindShortLV` etc.) have been compiled with GCC 4.3.2 using identical optimization flags. As detailed in Section 5.3, for our experiments we use the lattice bases published as part of the lattice contest conducted by the TU Darmstadt [36].

7.1 Evaluating `FindShortLV`

A first set of experiments was carried out to determine suitable parameters for `FindShortLV`. The goal was to determine a set of parameters that yield the best cost-benefit ratio, i.e., balancing the running time, the length, and the number of short vectors found. Due to the large number of parameters, we were forced to fix some of the parameters for our tests. In particular, we fixed $p = 2,000$ and through some initial tests we determined that it proves beneficial to fix $f = d - 10$ for a given d. Table 1 summarizes the parameter sets for four different series of experiments. It is important to note that in test series 3 and 4, the size of the subset of lattice vectors is adjusted before advancing to the next iteration in `ThreadFindShortLV`$_\mathtt{i}$. Each test lattice is first BKZ reduced with block size 20. Then, for each combination of parameters and test lattice we execute `FindShortLV` ten times. Each run results in a set S of short lattice vectors. Using the length of the shortest of the short vectors of each one of the ten sets allows us to determine the average length of the shortest of the short vectors determined in each instance of the experiment. Figure 2 shows the average

series	q	c the	c_a	l_1	l_2	f	d	f_a, d_a
1	10	0.001	0.25	1.0	1.1	70	80	0
2	10	0.001	0.25	1.0	1.05	75	85	0
3	10	0.001	0.25	1.025	1.05	70	80	1
4	5	0.000001	0.25	1.0 (last seq. step 1.1)	1.0	70	80	1

Table 1. Parameters for `FindShortLV`

length of the shortest of the short lattice vector found by `FindShortLV` for a specific parameter set and lattice basis. As expected, an increase in f and d has a positive effect on the average length of the shortest of the short vectors found. Both the series 3 and 4 with their dynamically increasing f and d manage to find the shortest vectors in our experiments. It is important to note that in most cases series 4 yields a significantly higher number of short lattice vectors compared to series 1, 2, and 3. This can easily be seen in Figure 3 which shows the average number of short vectors found by `FindShortLV` (where the average is taken over the sizes of the ten sets resulting from the ten individual runs). With the exception of dimensions 475 and 500, test series 3 does not find more short vectors than test series 1 and 2 even though the slack parameter l_1 is increased (see Table 1). The shorter lattice vectors found by series 3 parameters make it more difficult to find additional lattice vectors of similar size and this therefore requires an additional increase of slack parameter l_1. The parameters for test series 4 have been chosen in order to overcome these shortcomings (see

Figure 3). To compensate for the increase in running time, test series 4 therefore uses a reduced number of sequence steps q but with a tighter initial pruning parameter c. For the first four of the $q = 5$ sequence steps test series 4 uses a slack parameter $l_1 = 1.0$. This measure helps us to find a reasonable short lattice vector as starting point for the final one of the q sequence steps. In the 5th and final sequence step we use a slack parameter $l_1 = 1.1$ in order to maximize the number of short lattice vectors found. Figure 4 shows the average running time (in hours, averaged over the ten

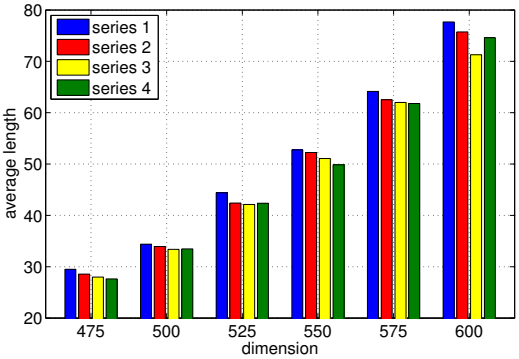


Figure 2. Average length of shortest of the short vector found by `FindShortLV`.

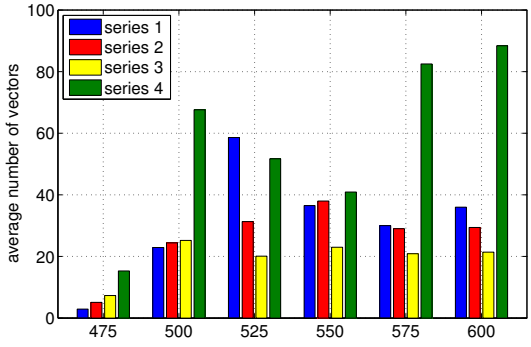


Figure 3. Average number of short vectors found by `FindShortLV`.

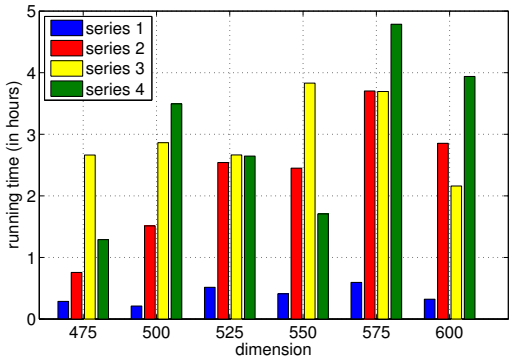


Figure 4. Average real time of `FindShortLV` with 8 threads and different parameters.

individual runs) of `FindShortLV`. An increase in d and f leads to a significant increase in running time due to the exponential character of the subroutine `ENUM`. The increase of $d = 80$ for series 1 to $d = 85$ for series 2 leads (in most cases) to more than a fourfold increase in the running time. (Recall that f is chosen as $f = d - 10$.) In addition, Figure 4 shows that we have been successful in keeping the overhead for test series 4 within reasonable limits compared to series 2 and 3. While one would hope that spending more time on finding shorter lattice vectors will pay off in the long run, i.e., in the overall generic framework, we will see later that this, unfortunately, is generally not true. The improvements of our reduction framework `BoostBKZ` strongly depend on both the composition of the set of short lattice vectors generated by `FindShortLV` and their positioning within the improved lattice basis.

7.2 Evaluating BoostBKZ

Through a set of initial experiments it was determined that replacing a large number of basis vectors often has no positive impact on the running time of BoostBKZ. Consequently, the number of lattice basis vectors to be replaced as part of GenerateBasis is limited to at most ten. As discussed earlier, the number of different parameter sets G_j for GenerateBasis is tied to the number of available threads t. The compute servers in our test setup (see above) offer 8 cores. We therefore have chosen the number of threads to be $t = 8$. Table 2 details the parameter sets G_j (N, the direction, and the repositioning) used in combination with the test series as outlined previously ($1 \leq j \leq 8$). These parameter sets have been determined based on an analysis of a set of initial experiments. (Further details will be included in an extended version of this paper.)

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
N	3	5	7	10	3	5	7	10
Direction	fwd.	fwd.	fwd.	fwd.	bwd.	bwd.	bwd.	bwd.
Repositioning	—	—	—	—	random	random	random	random

Table 2. Parameters for GenerateBasis

In analyzing BoostBKZ, we conduct an experiment that is tailored to show the effectiveness of our intermediate steps introduced as part of BoostBKZ to improve the overall reduction result. For that, we fixed the number of iterations in BoostBKZ to $r = 1$. For the initial BKZ reduction (Line (1) in Algorithm 2) we used block size 20. For the BKZ reductions in $\texttt{ThreadBoostBKZ}_\texttt{j}$ (as part of the one and only iteration in BoostBKZ) we used block size 25. The other parameters, such as the reduction parameter ($\delta = 0.99$) and the pruning parameter for the enumeration (2^{-15}), were identical in both cases. The maximum number of BKZ instances to be completed was set to maxred $\in \{1, 2, 3\}$. We compared the performance of BoostBKZ to a conventional BKZ reduction with block size 25 (referred to as BKZ 25) and a sequence of conventional BKZ reductions with first block size 20 followed by a second one with block size 25 (referred to as BKZ 20/25). Each experiment was conducted ten times and the averages were computed over the results of these ten runs. Figure 5

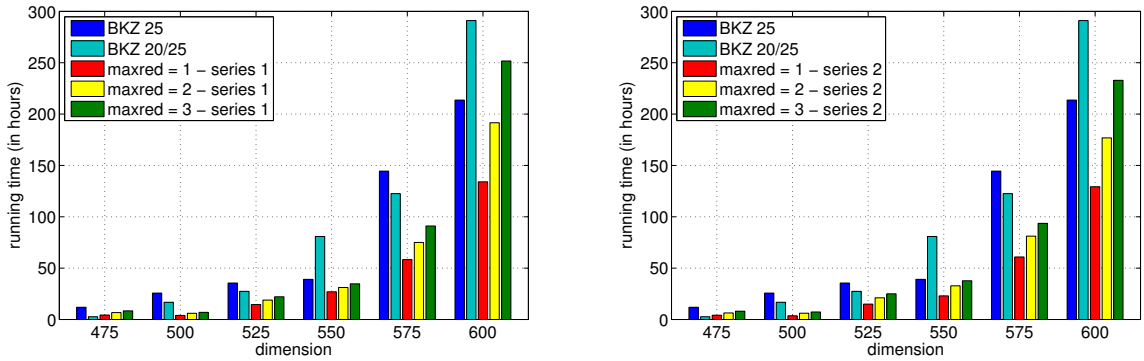


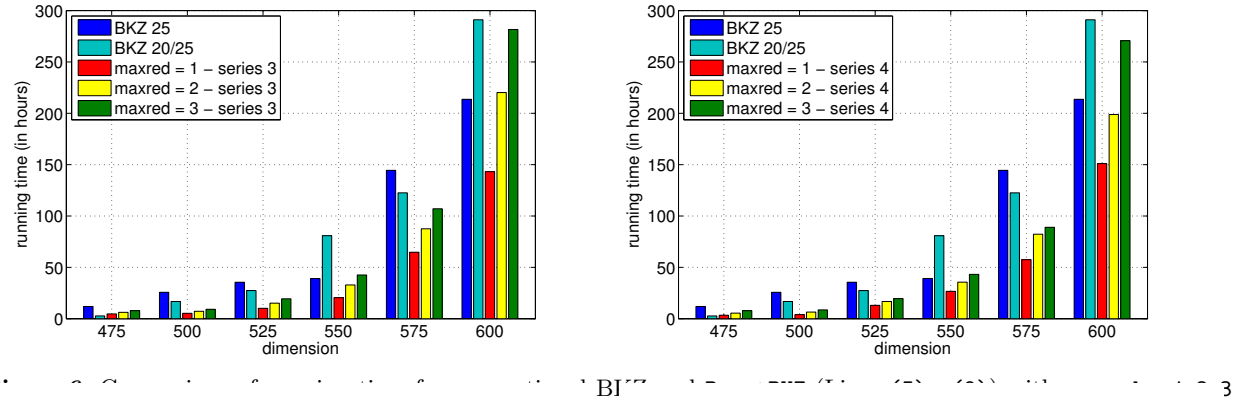
Figure 5. Comparison of running time for conventional BKZ and BoostBKZ (Lines (5) - (9)) with maxred $= 1, 2, 3$ for FindShortLV with series 1 and 2

shows the running time (in hours) of Lines (5) - (9) of BoostBKZ for series 1 and 2 in comparison to the conventional BKZ reductions. The experiments clearly show a substantial advantage in the running time of our approach with maxred $= 1$ over both BKZ 25 and BKZ 20/25. It is interesting

to note that for some test lattices BKZ 25 outperforms BKZ 20/25 and for some it is the other way around. That is, neither BKZ 25 nor BKZ 20/25 is an optimal choice for conventional BKZ. Figure 5 also shows that an increase in f and d from series 1 to series 2 has a significant impact on the running time, especially for higher dimensions. The additional time spent in `FindShortLV` for series 2 in comparison to series 1 is beneficial with respect to the overall running time. In particular, it is important to note, that the running time for `FindShortLV` (see Figure 4) is negligible when compared to the running time of Lines `(5)` - `(9)` of algorithm `BoostBKZ` in higher dimensions[4]. Figure 6 shows the running time for series 3 and 4. At first, the results seem inferior in comparison to the running times for series 1 and 2. However, when comparing the quality of the reduced basis (see Figure 7) it becomes clear that series 3 and especially series 4 produce lattice bases of better quality. Figure 7 shows that the quality for series 4 with `maxred` $= 2$ is almost equal to the quality obtained for series 1 and 2 with `maxred` $= 3$. Even for `maxred` $= 1$, `BoostBKZ` yields lattice bases of acceptable quality given test series 2 and 4 (see Figure 8) when compared to the conventional BKZ reduction.

In order to gain a better understanding for the connection between the parameter `maxred` and the quality of the reduced lattice basis we have analyzed the set of short lattice vectors found by `FindShortLV` in more detail. Figure 9 shows the average length of the first 3, 5, 7, and 10 lattice vectors that are inserted into the lattice basis by `GenerateBasis`. It is interesting to see that the difference between the first ten short vectors and the first three short vectors is noticeably higher for series 2 than for series 4. This difference is caused by the use of slack parameter l_2 for series 2 and l_1 for series 4. Slack parameter l_1 should therefore be preferred over l_2. These results indicate that

[4] By construction, `FindShortLV` scales almost perfectly with the number of dedicated CPU cores. E.g., doubling the number of CPU cores will in general cut the running time of `FindShortLV` in half.



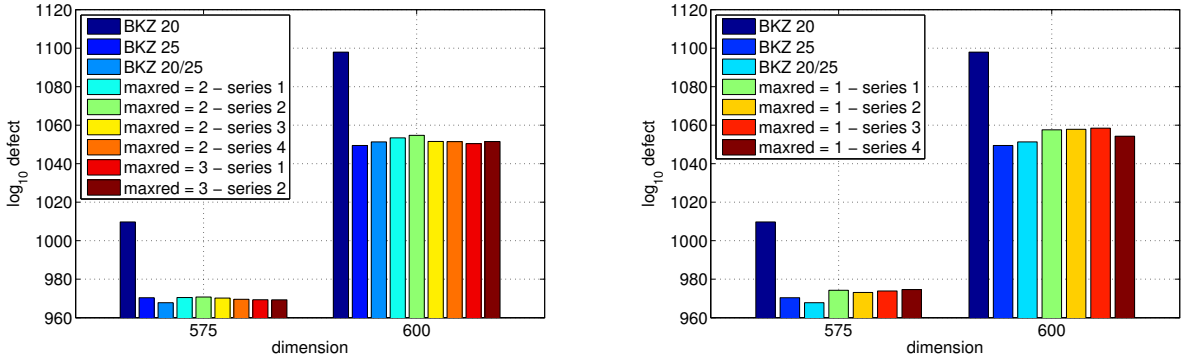


Figure 7. Quality of lattice basis for `maxred` $= 2, 3$.

Figure 8. Quality of lattice basis for `maxred` $= 1$.

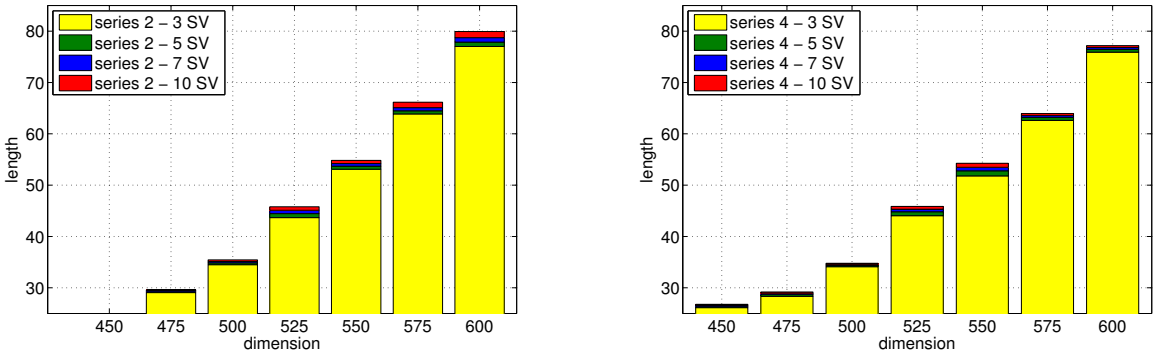


Figure 9. Average length of 3, 5, 7 and 10 vectors inserted by `GenerateBasis`.

dedicating additional time in `BoostBKZ` for `FindShortLV` is beneficial in cases where the difference in length between the inserted lattice vectors is minimal.

8 Conclusion and Future Work

The new algorithmic framework `BoostReduce` and the concrete instance `BoostBKZ` based on the BKZ reduction presented in this paper provide an effective new means to strong lattice basis reduction. Future work includes additional experiments to further fine-tune the choice of the various parameters. In addition, we will investigate how much of an additional advantage one may achieve by replacing the traditional BKZ reduction or the `ENUM` algorithm with recent developments in reducing lattice bases and finding shortest lattice vectors [35, 18].

References

1. M. Ajtai. Generating Hard Instances of Lattice Problems. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, 1996.
2. M. Ajtai. The Shortest Vector Problem in L_2 is $\mathcal{NP}$-hard for Randomized Reductions. Technical report, ECCC Electronic Colloquium on Computational Complexity, Trier, 1997.
3. M. Ajtai and C. Dwork. A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 284–293. ACM Press, 1997.
4. W. Backes and S. Wetzel. New Results on Lattice Basis Reduction in Practice. In *Algorithmic Number Theory (ANTS-00)*, volume 1838 of *LNCS*, pages 135–152. Springer, 2000.
5. W. Backes and S. Wetzel. Heuristics on Lattice Basis Reduction in Practice. *ACM Journal on Experimental Algorithms*, 7, 2002.
6. W. Backes and S. Wetzel. An Efficient LLL Gram Using Buffered Transformations. In *Proceedings of CASC 2007*, volume 4770 of *LNCS*, pages 31–44. Springer, 2007.
7. W. Backes and S. Wetzel. The Effect Of Sorting On Lattice Basis Reduction. Poster Session of LLL+25 Workshop, 2007.
8. W. Backes and S. Wetzel. Parallel Lattice Basis Reduction Using a Multi-threaded Schnorr-Euchner LLL Algorithm. In *Proceedings of Euro-Par 2009*, volume 5704 of *LNCS*, pages 960–973. Springer, 2009.
9. R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. In *SFCS '94: Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 356–368, Washington, DC, USA, 1994. IEEE Computer Society.
10. Claus-Peter Schnorr. Block Reduced Lattice Bases and Successive Minima. *Combinatorics, Probability & Computing*, 3:507–522, 1994.
11. M. Coster, A. Joux, B. LaMacchia, A. Odlyzko, C. Schnorr, and J. Stern. Improved Low-Density Subset Sum Algorithm. *Journal of Computational Complexity*, 2:111–128, 1992.
12. B. Filipovic. Implementierung der Gitterbasenreduktion in Segmenten. Master's thesis, University of Frankfurt am Main, 2002.

13. N. Gama, N. Howgrave-Graham, H. Koy, and P. Nguyen. Rankin's Constant and Blockwise Lattice Reduction. In *Advances in Cryptology (CRYPTO 2006)*, volume 4117 of *LNCS*, pages 112–130. Springer, 2006.
14. N. Gama, N. Howgrave-Graham, and P. Nguyen. Sympletic Lattice Reduction and NTRU. In *Advances in Cryptology (EUROCRYPT 2006)*, volume 4004 of *LNCS*, pages 233–253. Springer, 2006.
15. C. Gentry. Fully Homomorphic Encryption using Ideal Lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 169–178. ACM, 2009.
16. O. Goldreich, S. Goldwasser, and S. Halevi. Public-Key-Cryptosystems from Lattice Reduction Problems. In *Proceedings of CRYPTO 1997*, volume 1294 of *LNCS*, pages 112–131. Springer, 1997.
17. G. Hanrot and D. Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *CRYPTO'07: Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, pages 170–186, Berlin, Heidelberg, 2007. Springer-Verlag.
18. J. Hermans, M. Schneider, J. Buchmann, F. Vercauteren, and B. Preneel. Parallel Shortest Lattice Vector Enumeration on Graphics Cards. Cryptology ePrint Archive, Report 2009/601, 2009. `http://eprint.iacr.org/`.
19. J. Hermans, M. Schneider, J. Buchmann, F. Vercauteren, and B. Preneel. Parallel Shortest Lattice Vector Enumeration on Graphics Cards. In *Third International Conference on Cryptology in Africa (Africacrypt)*, volume 6055 of *LNCS*, pages 52–68. Springer-Verlag, 2010.
20. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In *Algorithmic Number Theory (ANTS-98)*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998.
21. R. Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 193–206. ACM Press, 1983.
22. H. Koy and C. Schnorr. Segment LLL-Reduction of Lattice Bases. In *Cryptography and Lattices*, volume 2146 of *LNCS*, pages 67 – 80. CaLC 2001, Springer, 2001.
23. V. Lyubashevsky. Lattice-Based Identification Schemes Secure Under Active Attacks. In *Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, Proceedings*, volume 4939 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2008.
24. V. Lyubashevsky and D. Micciancio. Generalized Compact Knapsacks are Collision Resistant. In *33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, volume 4052 of *LNCS*, pages 144–155. Springer, 2006.
25. P. Nguyen and O. Regev. Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures. In *Advances in Cryptology (EUROCRYPT 2006)*, volume 4004 of *LNCS*, pages 271–288. Springer, 2006.
26. P. Nguyen and D. Stehlé. Floating-Point LLL Revisited. In *Proceedings of Eurocrypt 2005*, volume 3494 of *LNCS*, pages 215–233. Springer, 2005.
27. P. Nguyen and D. Stehlé. LLL on the Average. In *Proceedings of the 7th Algorithmic Number Theory Symposium (ANTS-06)*, volume 4076 of *LNCS*, pages 238–256. Springer, 2006.
28. P. Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97. In *Advances in Cryptology (CRYPTO 1999)*, pages 288 – 304. Springer, 1999.
29. P. Q. Nguyen and J. Stern. Lattice Reduction in Cryptology: An Update. In *Algorithmic Number Theory (ANTS-00)*, volume 1838 of *LNCS*, pages 85 – 112. Springer, 2000.
30. C. Peikert. Public-key Cryptosystems from the Worst-Case Shortest Vector Problem (Extended Abstract). In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 333–342. ACM, 2009.
31. C. Peikert and A. Rosen. Efficient Collision-Resistant Hashing from Worst-case Assumptions on Cyclic Lattices. In *3rd Theory of Cryptography Conference (TCC 2006)*, volume 3876 of *LNCS*, pages 145–166. Springer, 2006.
32. H. Ritter. *Aufzählung von kurzen Gittervektoren in allgemeiner Norm*. PhD thesis, University of Frankfurt am Main, Research Group of Prof.C.P.Schnorr, 1997.
33. C. Schnorr and M. Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. In *Proceedings of FCT 91*, pages 68–85. Springer, 1991.
34. C. Schnorr and H. Hörner. Attacking the Chor-Rivest Cryptosystem by Improved Lattice Reduction. In *Proceedings of EUROCRYPT 1995*, volume 921 of *LNCS*, pages 1–12. Springer, 1995.
35. Homepage of Xavier Pujol (LatEnum/ParEnum), May 2010. `http://perso.ens-lyon.fr/xavier.pujol/`.
36. Lattice Challenge TU Darmstadt, May 2010. `http://www.latticechallenge.org/`.
37. NTRU - Hompage, May 2010. `http://www.ntru.com/`.
38. GMP - Homepage. `http://gmplib.org/`.
39. MPFR - Homepage, May 2010. `http://www.mpfr.org/`.
40. NTL - Homepage, May 2010. `http://www.shoup.net/ntl/`.