Quasi-polynomial time algorithms for block basis reduction of linear codes

Kolesnikov N.S. and Novoselov S.A.

Immanuel Kant Baltic Federal University, ul. Nevskogo 14, Kaliningrad, Russia
{nikolesnikov100,novsem}@gmail.com

Abstract. The block code basis reduction algorithm (BKZ) proposed by Ghentiyala and Stephens-Davidowitz requires an exponential number of calls to the minimum weight codeword (MWC) oracle according to the weak complexity bounds established in the original paper. In almost all cases these bounds are worse than plain enumeration of codewords, despite the fact that in practice the algorithm works much better. In this work we propose block code basis reduction algorithms with polynomial and quasi-polynomial complexity for fixed β. Our algorithms are relaxations and modifications of the original algorithm. First, we show that the truncated version of the BKZ algorithm requires heuristic quasi-polynomial time of calls to MWC-oracle with the same basis quality as in the full BKZ algorithm. Second, we add some weak restrictions on the code basis that enables the algorithm to run in polynomial number of calls to the oracle. Third, we introduce the DeepBKZ algorithm which is adapted from lattice reduction. Under some relaxations (analogs of Pot-BKZ from the lattices) it also runs in quasi-polynomial number of calls to the MWC-oracle. We provide an implementation of proposed algorithms which is capable to process block sizes $2 - 24$ and more. Based on this implementation, we provide concrete complexity estimates and an analysis of the approximation factors achieved for the low-weight codeword problem on random linear codes.

Keywords: linear codes · low-weight codeword problem · BKZ reduction · LLL reduction.

1 Introduction

The low-weight codeword and decoding problems are foundations of post-quantum cryptographic code based schemes, including McEliece and HQC schemes. The first problem occurs mainly as a useful tool in structural and dual attacks on cryptographic schemes, while solving the decoding problem breaks almost all known code-based schemes. Both problems are well known to be hard and NP-complete [5]. There is also quantum reduction between low-weight codeword and decoding problems [7]. The code basis reduction was introduced in [6], where authors proposed the LLL algorithm for binary codes as an adaptation of the well-known LLL basis reduction algorithm for lattices. Later, in [8] the

BKZ algorithm from lattices to codes was adapted together with generalizations of reduction theory from binary codes to any q-ary linear codes. The code basis reduction algorithms provide a way to solve both low-weight codeword and decoding problems. The basic tool to solve these problems is information set decoding algorithms (ISD). In ISD algorithms such as Prange, Lee-Brickell, Dumer/Stern, BJMM [4] and other, we explore a space of codewords using systematic form of matrix to reduce basis, which reduces the codewords to the average weight $\frac{n-k}{2}+1$. The code basis reduction algorithms provide better average weight reduction and can be used with some technical modifications in ISD algorithms as replacement for systematization. It was already used in such way to establish low-weight codeword computational records [2]. However, the BKZ algorithm for codes proposed in [8] has theoretic complexity bounds [8, Claim A.1] that are worse for most codes than plain brute-force enumeration of all codewords and the known bounds for approximation factor [8, §6.1.3] for the low-weight codeword problem are trivial – in most cases these bounds include all codewords. These issues call into question the practical and theoretical value of code-basis reduction. To address these issues the authors of [8] proposed slide and backward reduction, which are adaptations of slide and dual reduction from lattice theory.

Our contribution. We propose solutions for both described issues by adopting from lattice theory of truncated BKZ, PotBKZ, and DeepBKZ algorithms. More precisely:

1. We introduce a truncated version of the original BKZ algorithm. This is the original algorithm terminated after processing the N-th basis element. This algorithm has runtime $\mathcal{O}(\beta n^{N+\beta}T)$ where, n is the length of code, β is the block size, T is the cost of MWC-oracle. Heuristically, we show that for block sizes $2-24$ the original BKZ algorithm provides almost the same basis quality as the truncated BKZ algorithm with $N=\mathcal{O}(\log n)$. Thus, we can in practice achieve basis quality of the original BKZ algorithm in quasi-polynomial calls to the oracle.
2. We propose an algorithm with a polynomial number of calls to the oracle, which we call PotBKZ. This algorithm is analogous to PotBKZ from lattices. The idea is to relax the reduction conditions by forbidding certain local basis changes that would increase a suitably chosen potential function too much. The potential function method is a classical method of proving complexity of reduction algorithms in lattices.
3. We develop the third algorithm, called DeepBKZ that is also adapted from corresponding algorithm from lattices.
4. We provide an implementation of three proposed algorithms and estimate reached approximation factors and calls to the oracle to support our theoretical claims.
5. We propose a simple heuristic on the weight of the first element in the BKZ reduced basis. We support it with experimental data computed with our implementation for block sizes $2-24$ and random linear codes of cryptographic lengths 1024 and 1280.

2 Preliminaries

2.1 Notations

We use the following notation throughout the paper:

- $\mathcal{C}(B) \subset \mathbb{F}_2^n$ is a $[n,k]$-linear code defined over $\mathbb{F}_2$ by its generator matrix $B \in \mathbb{F}_2^{k\times n}$;
- $\mathbf{b}_1, \ldots, \mathbf{b}_k$ are the basis vectors of the code $\mathcal{C}$, that are written in the rows of matrix B;
- $\|\mathbf{b}\|$ is the Hamming weight of the vector $\mathbf{b} \in \mathbb{F}_2^n$;
- $(B)_i$ is the i-th row of matrix B;
- $A \oplus B := \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$

2.2 The Gilbert-Varshamov bound

For a random linear $[n,k]$-code we denote by $d_{GV} := GV(n,k)$ the Gilbert-Varshamov bound that is defined as the smallest integer d, such that the inequality holds:

$$\sum_{j=0}^{d-1} \binom{n}{j} \geq 2^{n-k}.$$

It is well-known that the Gilbert-Varshamov bound provides an approximate value of the minimum distance of a random linear code. Computing the minimum distance is a hard NP-complete problem, while the Gilbert-Varshamov bound is easy to compute. So, we can use it in practice as a reference value to estimate approximation factors reached by various basis reduction algorithm, without knowledge of exact value of the minimum distance. This is a direct analogy of estimating approximation factors of lattice basis reduction algorithms using Gaussian Heuristic.

2.3 BKZ: from lattices to linear codes

The BKZ algorithm for linear codes is an adaptation of the corresponding algorithm from the lattices. The idea of the original algorithm for lattices is to reduce the global problem of finding short lattice vectors to doing subsequent local changes inside blocks – small sub-lattices of dimension β. Each such sub-lattice is connected to previously processed sub-lattices using a special projection function π. The purpose of the projection is to exclude relevant information about short vectors from already processed sub-lattices and process the remaining part. Since blocks are connected using projections, changes in one block propagate to previously processed ones. All basis vectors in the lattice connected with projections are stored in Gram-Schmidt orthogonalization B^* of the basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$. Let $\pi_i(\mathbf{b}) := \pi_{\mathbf{b}_1,\ldots,\mathbf{b}_{i-1}}(\mathbf{b})$. Then

$$B^* = (\mathbf{b}_1, \pi_2(\mathbf{b}_2), \ldots, \pi_k(\mathbf{b}_k)).$$

The BKZ algorithm for each i-th vector finds a vector $\mathbf{c} \in \mathcal{L}(b_i, \ldots, b_{i+\beta-1})$ with shortest projection and places it at position i with recomputation of all connected sub-lattices. And it repeats this procedure while no further changes can be made. The shortest projection is found by solving the shortest vector problem (SVP) in the projected lattice called block with basis defined as $B_{[i,i+\beta-1]} := (\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_{i+\beta-1}))$.

For linear codes, an analogy to this construction was proposed for binary linear codes and block size 2 (analog of LLL for lattices) in [6] and was generalized to all q-ary codes and blocks in [8]. Instead of shortest vectors, we seek minimum-weight codewords. The projection function $\pi_{\mathbf{b}}(\mathbf{a})$ is the puncturing operation. It sets to zero all coordinates of $\mathbf{a}$ where vector $\mathbf{b}$ has non-zero coordinates (i.e. support of $\mathbf{b}$). In the BKZ for codes it is used to remove previously found minimum weight codewords in already processed blocks from the consideration and to work only with remaining part. More precisely, let $B = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$ be a basis of a linear code. Define projection $\pi_i(\mathbf{b}) := \pi_{\mathbf{b}_1,\ldots,\mathbf{b}_{i-1}}(\mathbf{b}) = \pi_{\mathbf{b}_1}(\pi_{\mathbf{b}_2}(\ldots \pi_{\mathbf{b}_{i-1}}(\mathbf{b})))$. So, $\pi_i(\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_k))$ is a punctured code obtained from $\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_k)$ by removing all coordinates corresponding to non-zero positions in $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. In the BKZ algorithm we store at positions $1, \ldots, i-1$ the minimum weight codewords from previously processed blocks. Therefore, the projection π_i removes these codewords from the consideration and we reduce the remaining part – the punctured code $\pi_i(\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_k))$. The BKZ algorithm proceeds from $i = 1$ to k by finding minimum weight codewords for blocks

$$B_{[i,i+\beta-1]} := (\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_{i+\beta-1}))$$

and placing them at the i-th position. If at the position i we already have a minimum weight codeword, then it proceeds to the position $i + 1$. Otherwise, the algorithm recomputes connected blocks, i.e. step back by β positions. The algorithm works not with the all punctured code $\pi_i(\mathcal{C}(\mathbf{b}_1, \ldots, \mathbf{b}_k))$, but only up to β basis codewords, providing only local reduction.

The matrix

$$B^+ = (\mathbf{b}_1^+, \mathbf{b}_2^+, \ldots, \mathbf{b}_k^+) := (\mathbf{b}_1, \pi_2(\mathbf{b}_2), \ldots, \pi_k(\mathbf{b}_k))$$

is called an epipodal matrix. It is a direct analog of Gram-Schmidt orthogonalization. Since we use puncturing that removes many coordinates, it can quickly remove all coordinates leaving zero projections after some i-th positions. To exclude such situation, the notion of *proper bases* was introduced in [6]. These are the bases where each i-th projection π_i is non-zero. It also was shown that basis of the code in systematic form is proper. So, in the following we will use only such bases.

The values $\ell_1 = \|\mathbf{b}_1^+\|, \ldots, \ell_k = \|\mathbf{b}_k^+\|$ are called *profile* of the basis B. It determines the quality of the basis. The first element in the profile ℓ_1 is equal to the Hamming weight of the first element in the basis B. The number k_1 of elements not equal to one determines efficiency of decoding using an analog of Babai's algorithm for codes and efficiency of hybrid ISD algorithms that use basis reduction. The ISD algorithms for decoding problem use systematic form to reduce

the basis. Block basis reduction is a natural replacement for this operation. And it already was used in establishing of some decoding records [2].

3 BKZ algorithms for codes

3.1 One block BKZ

One block BKZ algorithm (see [8, §8.8]) is just one round of BKZ-algorithm. It makes one minimal weight codeword computation for the first block, places this codeword at first position in the basis and returns it. Since the algorithm performs only one MWC-computation and it does not use relations between blocks for the basis improvement, it has on average the least-reduced basis among the BKZ-algorithms considered. However, such a basis can be obtained with fastest time. The weight of the first element of such a basis can be used as natural non-trivial and easy-to-compute upper bound for the first elements of BKZ-reduced bases. More precisely, we can estimate weight of first element of one-round reduced basis in the following way. The first block of dimension β consists of the first elements of the basis. Assuming that the basis is proper and obtained via systematizing plus some preprocessing (LLL or BKZ with small block size), we have that the first block has length $\leq n-k+\beta$ after puncturing zero-positions in the elements of the basis. Now we can just take the Gilbert-Varshamov bound for a random $[n-k+\beta, \beta]$-code.

Approximation factor reached by this algorithm can be computed as

$$\gamma_{\text{1-round}} = \frac{GV(n-k+\beta, \beta)}{GV(n,k)},$$

assuming that the first block in the basis behaves like a random linear code after pruning of zero positions and the whole input basis is also random.

Note that the current low-weight codeword computation record [2] for $n = 1280$ is $w = 204$ with $\gamma = 1.416$. To reproduce such a result with one-round BKZ we have to take $\beta \geq 140$. Fortunately, the full-rounds BKZ algorithms works much better.

3.2 Basic BKZ

Definition 1. *The basis B of linear code $\mathcal{C}$ is called β-BKZ-reduced if it is proper and the first vector in each block $B_{[i,i+\beta-1]}$ is a minimal codeword of the code $\mathcal{C}(B_{[i,i+\beta-1]})$ for each $i = 1, \ldots, k$.*

Such bases can be obtained with Algorithm 1 proposed in [8, Alg. 8].

One-block reduction is one iteration of the BasicBKZ algorithm. Therefore, it provides an upper bound for the approximation factor γ_{basic} reached by the BasicBKZ algorithm. Let us establish a heuristic lower bound. Note that most rows of epipodal matrix have Hamming weight one and the reduction in the BasicBKZ algorithm stops when we reach the tail of ones (i.e. MinCodeword

Algorithm 1: BasicBKZ(B, β). Block basis reduction [8] for a code $\mathcal{C}(B)$.

Data: A proper basis B of a linear code $\mathcal{C}$ and $\beta \in [2, k]$.
Result: A β-BKZ-reduced basis.

$i = 1$;
while $i < k$ **do**
  $j = \min(i + \beta - 1, k)$;
  $\mathbf{c} = \text{MinCodeword}(\mathcal{C}(B_{[i,j]}))$;
  if $\|\mathbf{c}\| < \|(B_{[i,j]})_1\|$ **then**
    find a matrix T s.t. $\mathbf{c} = (T \cdot B_{[i,j]})_1$ and $T \cdot B_{[i,j]}$ is proper;
    $B = (I_{i-1} \oplus T \oplus I_{k-j}) \cdot B$;
    $i = \max(1, i - \beta + 1)$;
  else
    $i = i + 1$
  end
end
return B

procedure will always return 1 for the rest of algorithm). Thus, the algorithm uses only $\approx k_1 + \beta$ first vectors of the basis, where k_1 is the number of non-ones in profile. In fact, we use only a subcode of dimension $k_1 + \beta$ of our k-dimensional code. Assuming that this subcode is a random linear code of length $n - k + k_1 + \beta$ (after removing zero-columns), we obtain the following lower bound for the length of the first element in β-BasicBKZ-reduced basis

$$\ell_1 \approx \text{GV}(n - k + k_1 + \beta, k_1 + \beta).$$

Experimentally, for small block sizes ($\beta = 2, 4, 8$) it was shown in [6] and [8, Fig. 3] that $k_1 = \mathcal{O}(\log n)$ on average. The same holds for big block sizes (see results of our experiments at Fig. 1). These experiments show that $k_1 = c_1 \log_2 n$ for $c_1 \approx 2$ for block sizes $\beta = 2, \ldots, 24$. Therefore, we propose the following heuristic upper bound for the weight of the first element in β-BKZ-reduced basis:

$$\ell_1 \approx GV(n - k + c_1 \cdot \log n + \beta, c_1 \cdot \log n + \beta).$$

Now, we have lower and upper bounds on the weight of the first element in the β-BKZ-reduced basis. Let us introduce an heuristic for this first element. From experimental data (see Fig. 2), we have that the BasicBKZ algorithm returns the average value between the lower and the upper bounds:

$$\mathbb{E}[\ell_1] \approx \frac{GV(n - k + \beta, \beta) + GV(n - k + k_1 + \beta, k_1 + \beta)}{2}.$$

This means that we can heuristically estimate approximation factor in BasicBKZ algorithm by introducing the following heuristic.

Heuristic 1 *(Approximation factor for BasicBKZ) Let B be a β-BKZ-reduced basis of a random linear $[n,k,d]$-code, and $\gamma_{\text{basic}} = \frac{\|\mathbf{b}_1\|}{d}$. Then*

$$\gamma_{\text{basic}} \approx \frac{\gamma_{\text{1-round}}}{2} + \frac{GV(n-k+\beta+c_1\cdot\log n, \beta+c_1\cdot\log n)}{2\cdot GV(n,k)}$$

for some constant c_1 such that $k_1 \leq c_1 \log_2 n$.

Here, we have that to beat the current low-weight codeword computation record we need block-size $80-110$.

3.3 Truncated BKZ

The truncated BKZ algorithm is a BKZ algorithm that stops after processing index $i = N$ of algorithm for some $N \leq k$ given as input of algorithm.

Theorem 1. *Algorithm 1 (BasicBKZ), if terminated after processing N-th block, takes time $\mathcal{O}(\beta n^{N+\beta}T)$, where T is the time to find a minimal weight codeword in $[n-k+\beta,\beta]$-code.*

Proof. Since we restrict index i in the BasicBKZ algorithm and the algorithm is doing only local changes during its work (only the block is changed when doing insertion at position i), the algorithm uses during its work only first $N+\beta$ basis vectors. Thus, the runtime of the truncated BKZ algorithm is upper bounded by runtime of the BasicBKZ algorithm at the subcode $\mathcal{C}(\mathbf{b}_1,\ldots,\mathbf{b}_{N+\beta})$ of the input code $\mathcal{C}(\mathbf{b}_1,\ldots,\mathbf{b}_k)$. It was proven in [8, Claim A.1] that the full BasicBKZ algorithm runs while loop at most $\beta n^{k-1}(n-k+2)-1+k$ times. Therefore, for the truncated algorithm we have $\beta n^{N+\beta-1}(n-N-\beta+2)-1+N+\beta$ while loops. The dominating step in each while loop is finding minimal weight codeword. This procedure is performed at the dimension β block that has length $\leq n-k+\beta$ after removing of zero columns. Thus, resulting complexity of the truncated BKZ algorithm is $\mathcal{O}(\beta n^{N+\beta}T)$. □

Experimentally, for random codes a typical profile for proper bases after the BKZ reduction consists of first $\mathcal{O}(\log n)$ elements greater than one and others are ones. If at i-th iteration we have $\ell_i = 1$, then we have already a minimal weight codeword at this position and the algorithm is doing nothing at most of positions since we typically have big tail of ones at the end of profile. So, the basic BKZ algorithm in practice behaves like truncated BKZ algorithm with $N = \mathcal{O}(\log n)$.

Corollary 1. *For random linear codes under heuristic assumption that $k_1 = \mathcal{O}(\log n)$ the Algorithm 1 has runtime $\beta n^{\mathcal{O}(\log n)+\beta}T$, where T is the time to find a minimal weight codeword in $[n-k+\beta,\beta]$-code.*

3.4 Deep BKZ

This algorithm is a direct analogue of lattice-reduction algorithms [12]. The main difference between BasicBKZ and DeepBKZ is usage of deep insertions: instead of

placing minimal weight codeword at current position i, we search for position $j < i$ with projection $\|\pi_j(\mathbf{b}_i)\| < \|\pi_j(\mathbf{b}_j)\| = \|\mathbf{b}_j^+\|$ and, if such position is found, we place here vector $\mathbf{b}_i$. This procedure reduces the number of calls to the expensive minimum-weight-codeword (MWC) computation procedure, because otherwise MWC procedure (with block size 2 in case of LLL or a full block size) is used for sorting of the basis.

Definition 2. *The basis B of a linear code $\mathcal{C}$ is called β-DeepBKZ-reduced if it is proper and for each $i = 1, \ldots, k$:*

1. *the first vector in block $B_{[i,i+\beta-1]}$ is a minimal codeword of the code $\mathcal{C}(B_{[i,i+\beta-1]})$;*
2. *$\|\pi_j(\mathbf{b}_i)\| \geq \|\mathbf{b}_j^+\|$ for all $j < i$.*

We present DeepBKZ for codes as Algorithm 2.

Algorithm 2: DeepBKZ(B, β). The block basis reduction with deep insertions for a code $\mathcal{C}(B)$.

Data: A proper basis B of a linear code $\mathcal{C}$ and $\beta \in [2, k]$.
Result: A β-DeepBKZ-reduced basis.

$i = 1$;
while $i < k$ **do**
  $j = \min(i + \beta - 1, k)$;
  $\mathbf{c} = \text{MinCodeword}(\mathcal{C}(B_{[i,j]}))$;
  if $\|\mathbf{c}\| < \|(B_{[i,j]})_1\|$ **then**
    find a matrix T s.t. $\mathbf{c} = (T \cdot B_{[i,j]})_1$ and $T \cdot B_{[i,j]}$ is proper;
    $B = (I_{i-1} \oplus T \oplus I_{k-j}) \cdot B$;
    find the minimal $m < i$ s.t. $\|\pi_m(\mathbf{b}_i)\| < \|\mathbf{b}_m^+\|$;
    if *m exists* **then**
      `// Deep insertion`
      $\mathbf{c} = \text{MakePrimitive}(B_{[m,i]}, \pi_m(\mathbf{b}_i))$;
      $T = \text{InsertPrimitive}(B_{[m,i]}, \mathbf{c})$;
      $B = (I_m \oplus T \oplus I_{k-i}) \cdot B$;
      $i = \max(1, m - \beta + 1)$;
    else
      $i = \max(1, i - \beta + 1)$;
    end
  else
    $i = i + 1$
  end
end
return B

The procedures MakePrimitive and InsertPrimitive are polynomial time (in n) procedures from [8] which, when used together, find a transformation of the basis that keeps it proper (i.e. without zeroes under projections) and makes target codewords be the first element in the block.

Theorem 2. *Algorithm 2 (DeepBKZ), if terminated after processing N-th block, takes time $\mathcal{O}(\beta n^{N+\beta}T)$, where T is the time to find a minimal weight codeword in $[n-k+\beta,\beta]$-code.*

Proof. If there are no deep insertions in the algorithm then the algorithm 2 is just BasicBKZ algorithm. Let us show that each deep insertion changes the potential function by a big amount and, so the running time in the worst case is bounded by BasicBKZ runtime.

Let $\Phi(\ell_1,\dots,\ell_k)=\sum_{j=1}^{k} n^{k-j}\ell_j$ be the potential function from the [8, Claim A.1]. In the work [8] it was proved that after each (usual, not deep) insertion in BasicBKZ algorithm the potential function Φ decreases by 1. Then since $\Phi>0$, the upper bound on Φ gives us an upper bound on the number of insertions. And from this number it is easy to find total complexity of the algorithm.

We adapt here the proof from [8, Claim A.1]. Suppose a deep insertion occurs at position $m<i$ and let $\ell'_1,\dots,\ell'_k$ be new profile elements. Our goal is to show that $\Delta\Phi=\Phi(\ell_1,\dots,\ell_k)-\Phi(\ell'_1,\dots,\ell'_k)\geq 1$. We have the following conditions.

1. $\ell_m-\ell'_m\geq 1$, since we change position m;
2. $\ell_j=\ell'_j$ for $j<m$, since the transformation leaves first $m-1$ basis elements unchanged;
3. $\sum_{j=m+1}^{k}\ell'_j\leq n-1$, since $\sum_{j=1}^{k}\ell'_j\leq n$ and $\ell_j\geq 1$.

Then

$$\Delta\Phi=\sum_{i=1}^{k} n^{k-j}(\ell_j-\ell'_j)=n^{k-m}(\ell_m-\ell'_m)+\sum_{i=m+1}^{k} n^{k-j}(\ell_j-\ell'_j)$$

by the second condition. Applying the first condition we obtain:

$$\Delta\Phi\geq n^{k-m}+\sum_{i=m+1}^{k} n^{k-j}(\ell_j-\ell'_j)\geq n^{k-m}-\sum_{j=m+1}^{k} n^{k-j}\ell'_j.$$

The sum $\sum_{i=m+1}^{k} n^{k-j}\ell'_j$ can be upper bounded using the third condition as follows:

$$\sum_{j=m+1}^{k} n^{k-j}\ell'_j\leq n^{k-m-1}\sum_{j=m+1}^{k}\ell'_j\leq n^{k-m-1}(n-1).$$

Therefore, $\Delta\Phi\geq n^{k-m}-n^{k-m-1}(n-1)=n^{k-m-1}\geq n$, since $m<i\leq N\leq k-1$. Thus, deep insertions decrease potential even faster than usual ones and the complexity of the DeepBKZ algorithm is upper bounded by the complexity of BasicBKZ algorithm. □

Using the same heuristic as in BasicBKZ, that the basis have $\ell_i=1$ for $i>c\log n$, we obtain the same quasi-polynomial number of calls to MWC-oracle. However, every deep insertion at position m during the execution of the algorithm decreases the upper bound (maximal value of potential function in the proof) for the number of insertions by the value n^{k-m-1}. Thus, DeepBKZ requires fewer oracle calls and runs faster than the BasicBKZ algorithm for big block sizes, where the MWC-oracle becomes expensive.

3.5 PotBKZ

To prove polynomial or quasi-polynomial time complexity for BKZ algorithm without using heuristics, we will impose relaxations in the algorithm. These relaxations restrict the potential function change in the analysis so that it becomes negative during insertions of minimal weight codewords. So, we can choose a potential function with polynomial bounded maximum. As a consequence, we have a polynomial number of insertions in the algorithm and a polynomial total number of calls to MWC-oracle. Algorithms with such relaxations are called PotBKZ (Pot = Potential) in lattices [10] and we use the same name for linear codes. However, since we exclude some found minimal weight codewords, the reduced basis quality can degrade. Therefore, the potential function should be selected in such a way that excluding found minimal weight codewords is a rare event and occurs in the later iterations of the algorithm, where changes have minimal influence on the overall basis quality. The PotBKZ algorithm with such a potential function is given as Algorithm 3.

Algorithm 3: PotBKZ(B, ε, β).

Data: A proper basis B of a linear code $\mathcal{C}$ and $\beta \in [2, k]$ and $\varepsilon \geq 1$.
Result: An ε, β-PotBKZ-reduced basis.

$i := 1$;
while $i < k$ **do**
  $j := \min(i + \beta - 1, k)$;
  $\mathbf{c} := \text{MinCodeword}(\mathcal{C}(B_{[i,j]}))$;
  if $\|\mathbf{c}\| < \|(B_{[i,j]})_1\|$ **then**
    find a matrix T s.t. $\mathbf{c} = (T \cdot B_{[i,j]})_1$ and $T \cdot B_{[i,j]}$ is proper;
    $B' := (I_{i-1} \oplus T \oplus I_{k-j}) \cdot B$;
    $(\ell_1, \ldots, \ell_k) := \text{profile}(B)$;
    $(\ell'_1, \ldots, \ell'_k) := \text{profile}(B')$;
    $B' := LLL(B')$;
    $(\ell''_1, \ldots, \ell''_k) := \text{profile}(B')$;
    $\Delta := \sum_{j=1}^{k} k^{\varepsilon\beta/j} \cdot (\ell_j - \ell''_j)$;
    if $\Delta \geq 1$ **then**
      $B := B'$;
      $i := \max(1, i - \beta + 1)$;
    else
      $i := i + 1$
    end
  else
    $i := i + 1$
  end
end
return B

This algorithm computes a reduced basis of a linear code satisfying the following definition.

Definition 3. *A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_k)$ of a linear code $\mathcal{C}$ is ε, β-PotBKZ-reduced if for each $i = 1, \ldots, k-1$ and $j = \min(k, i+\beta-1)$ the element $\mathbf{b}_i^+$ is either a minimum weight codeword of the code $\mathcal{C}(B_{[i,j]})$, or there is a non-singular transformation T of $B_{[i,j]}$ such that for $T \cdot (\mathbf{b}_i, \ldots, \mathbf{b}_j) = (\mathbf{b}'_i, \ldots, \mathbf{b}'_j)$ the following conditions hold*

1. *$(\mathbf{b}'_i, \ldots, \mathbf{b}'_j)$ is an LLL-reduced basis;*
2. *$\mathbf{b}_i'^+$ is a minimal weight codeword in $\mathcal{C}(B_{[i,j]})$;*
3. *$\sum_{s=i}^{j} k^{\varepsilon\beta/s} \cdot (\|\mathbf{b}_s^+\| - \|\mathbf{b}_s'^+\|) < 1$.*

Potential function and its properties. We analyze the algorithm presented above by utilizing the potential function approach suggested in [8, appendix A]. Whenever the basis changes in the BKZ algorithm, its profile (and hence, the value of its potential function defined below in Eq. (1)) changes respectively. Let

$$\Phi(\ell_1, \ldots, \ell_k) := \sum_{j=1}^{k} k^{\varepsilon\beta/j} \ell_j \tag{1}$$

be a potential function. Here the parameter $\varepsilon \geq 1$ regulates the probability of $\Delta \geq 1$ condition on line 13 and, as a consequence, the complexity of our algorithm. For $\varepsilon = \text{const}$ the algorithm is polynomial-time. We can also achieve quasi-polynomial complexity by taking $\varepsilon = \mathcal{O}(\log n)$. However, in our computations it was enough to take ε to be small integer constant. At first, we estimate the bounds on this potential function.

Lower bound. Recall that $\ell_i \geq 1$ for all $1 \leq i \leq k$, since the basis is proper, and hence potential function is always positive. Moreover, it is lower bounded by $\Phi(\ell_1, \ldots, \ell_k) \geq \ell_1 k^{\varepsilon\beta} \geq k^{\varepsilon\beta}$.

Upper bound. Note that the potential function takes on its maximum value with $\ell_1 = n-k+1$, that maximizes the first term in (1). Then $\ell_2 = \ldots = \ell_k = 1$ and $\Phi(\ell_1, \ldots, \ell_k) \leq \Phi(n-k+1, 1, \ldots, 1) = k^{\varepsilon\beta}(n-k+1) + k^{\varepsilon\beta/2} + k^{\varepsilon\beta/3} + \ldots + k^{\varepsilon\beta/k} \leq nk^{\varepsilon\beta}$.

We recall some facts that we appeal to in this section:

*1. $\ell_j - \ell'_j = 0 \; \forall j \notin [i, \min(k, i+\beta-1)]$. First $i-1$ basis vectors remain unchanged after i-th stage of while loop, and hence $\ell_1 = \ell'_1, \ldots, \ell_{i-1} = \ell'_{i-1}$. Epipodal lengths $\ell_j, j > i+\beta-1$ also remain unchanged, because InsertPrimitive() procedure (lines 6-7 in Algorithm 3) do not change the support of the subcode $B[1, \min(k, i+\beta-1)]$.

*2. $\ell_i - \ell'_i \geq 1$, that requires "if" condition on line 5.

Lemma 1. *LLL-reduction of any basis B of the code $\mathcal{C}$ does not increase the potential (1) of its profile with any $\beta \geq 2$, $\varepsilon \geq 1$. Thus, we have $\Phi(\ell'_1, \ldots, \ell'_k) - \Phi(\ell''_1, \ldots, \ell''_k) \geq 0$ after line 10 of Algorithm 3.*

Proof. Suppose we run LLL-reduction for a basis B by calling BasicBKZ($B, \beta = 2$) (Algorithm 1). Let the basis changes after inserting the minimum weight codeword to the projected block $B_{[i,i+1]}$, and the basis profile changes from $(\ell_1, \ldots, \ell_k)$ to $(\ell'_1, \ldots, \ell'_k)$. Similar reasoning as in (*1) shows that $\ell_j - \ell'_j = 0$ for all $j \notin \{i, i+1\}$, whereas (*2) says that $\ell_i - \ell'_i \geq 1$. Additionally, we have that $\ell_{i+1} - \ell'_{i+1} = -(\ell_i - \ell'_i)$, since both bases are proper. Now compute the value of the potential function (1) change for this transformation:

$$\Phi(\ell_1, \ldots, \ell_k) - \Phi(\ell'_1, \ldots, \ell'_k) = k^{\varepsilon\beta/i}(\ell_i - \ell'_i) - k^{\varepsilon\beta/(i+1)}(\ell_i - \ell'_i) \geq k^{\varepsilon\beta/i} - k^{\varepsilon\beta/(i+1)},$$

that is nonnegative for $\varepsilon\beta \geq 2$. □

Complexity of Algorithm 3. We now have enough properties of potential function to prove the complexity bound for Algorithm 3.

Theorem 3. *The overall complexity of the PotBKZ (Algorithm 3) is given by* $\mathcal{O}(n^3 k^{3+\varepsilon\beta} \cdot T_{MW})$*, where* T_{MW} *is the cost of the minimum weight codeword searching in the projected block* $B_{[i,i+\beta-1]}$ *of length* $\leq n$ *and dimension* $\leq \beta$*.*

Proof. To begin with, we estimate the number of iterations of the main while loop. Each iteration may result in either profile change $(\ell_1, \ldots, \ell_k) \to (\ell''_1, \ldots, \ell''_k)$ or just increment the iterator i. Thus, any profile change is preceded by at most $k-1$ increments of the iterator without affecting the basis. That means $N \leq (N_1 + 1)(k - \beta)$, where N is the number of main cycle iterations, N_1 is the number of profile changes. There are also at most $k - \beta$ "extra" iterations with no basis change, that are taken into account.

Next we evaluate N_1. Whenever profile changes, the value of its potential function decreases by at least one, that requires step 13 of the algorithm. Otherwise the iteration is included in N value. Thus, N_1 cannot exceed the distance between maximum and minimum value of the potential function:

$$N_1 \leq nk^{\varepsilon\beta} - k^{\varepsilon\beta} = k^{\varepsilon\beta}(n-1).$$

The overall number of main cycle iterations is at most $N \leq (N_1+1)(k-\beta) = nk^{\varepsilon\beta}(k-\beta) \leq nk^{\varepsilon\beta+1}$. Each iteration calls the MinCodeword() procedure, that has $\mathcal{O}(T_{MW})$ complexity and depends on the algorithm used, and the LLL-reduction procedure, that performs $\mathcal{O}(\frac{nk(k+1)}{2} \max_j \ell_j) = \mathcal{O}(n^2k^2)$ operations in $\mathbb{F}_2$. Other steps take negligible time. Summarizing, the total number of operations in $\mathbb{F}_2$ of Algorithm 3 is given by $\mathcal{O}(N \cdot T_{MW} \cdot T_{LLL}) = \mathcal{O}(k^{\varepsilon\beta+1} n^3 k^2 \cdot T_{MW}) = \mathcal{O}(n^3 k^{3+\varepsilon\beta} \cdot T_{MW})$. □

Theorem 4. *Let* $\mathcal{C}$ *be* $[n,k]$*-code with* $k^2 > n$ *given by its generator matrix* B*. Algorithm 3 returns basis* $\widetilde{B}$ *of the code* $\mathcal{C}$*, that is* β*-BKZ-reduced upon row* $i \leq \lfloor \frac{-1+\sqrt{1+2\varepsilon\beta}}{2} \rfloor$*.*

Proof. To prove this statement, we show that if $i \leq \lfloor \frac{-1+\sqrt{1+2\varepsilon\beta}}{2} \rfloor$, then the condition on line 13 is always true, and hence the algorithm operates the same

as the basic BKZ (Algorithm 1) with an additional LLL block reduction on step 10. However, it does not break the property of the first block $B_{[1,i]}$ being β-BKZ reduced.

Rewrite the condition on line 13 considering that the basis changes twice sequentially, by insertion and LLL reduction:

$$\Delta = \sum_{j=1}^{k} k^{\varepsilon\beta/j} \cdot (\ell_j - \ell'_j) + \sum_{j=1}^{k} k^{\varepsilon\beta/j} \cdot (\ell'_j - \ell''_j) \overset{Lemma\ 1}{>} \sum_{j=1}^{k} k^{\varepsilon\beta/j} \cdot (\ell_j - \ell'_j) \quad (2)$$

Thus, it is sufficient to show that $i \leq \lfloor \frac{-1+\sqrt{1+2\varepsilon\beta}}{2} \rfloor \Rightarrow \sum_{j=1}^{k} k^{\varepsilon\beta/j} \cdot (\ell_j - \ell'_j) \geq 1$. Consider the latter sum that represents basis potential change after BKZ insertion:

$$\begin{aligned}
\Phi(\ell_1, \ldots, \ell_k) - \Phi(\ell'_1, \ldots, \ell'_k) &= \sum_{j=1}^{k} k^{\varepsilon\beta/j}(\ell_j - \ell'_j) = \\
&= \sum_{j=1}^{i-1} k^{\varepsilon\beta/j}(\ell_j - \ell'_j) + k^{\varepsilon\beta/i}(\ell_i - \ell'_i) + \sum_{j=i+1}^{k} k^{\varepsilon\beta/j}(\ell_j - \ell'_j) \overset{(*1)}{=} \\
&= k^{\varepsilon\beta/i}(\ell_i - \ell'_i) + \sum_{j=i+1}^{\min(k,i+\beta-1)} k^{\varepsilon\beta/j}(\ell_j - \ell'_j) \geq \\
&= k^{\varepsilon\beta/i}(\ell_i - \ell'_i) - \sum_{j=i+1}^{\min(k,i+\beta-1)} k^{\varepsilon\beta/j}\ell'_j \overset{(*2)}{\geq} \\
&\geq k^{\varepsilon\beta/i} - (n-1)k^{\varepsilon\beta/(i+1)}. \qquad (3)
\end{aligned}$$

The last inequality utilizes the fact (*2) and since the basis is proper, there is at least one term in the sum $\sum_{j=i+1}^{\min(k,i+\beta-1)} k^{\varepsilon\beta/j}\ell'_j$ with $\ell'_j \geq 1$. Let $i \leq \lfloor \frac{-1+\sqrt{1+2\varepsilon\beta}}{2} \rfloor$. We show that it follows the desired property.

$$\begin{aligned}
i \leq \lfloor \frac{-1+\sqrt{1+2\varepsilon\beta}}{2} \rfloor \Rightarrow \sqrt{1+2\varepsilon\beta} > 2i+1 \Rightarrow 2i^2 + 2i - \varepsilon\beta < 0 \Rightarrow \\
\frac{\varepsilon\beta}{i(i+1)} = \frac{\varepsilon\beta}{i} - \frac{\varepsilon\beta}{i+1} > 2 \Rightarrow k^{\frac{\varepsilon\beta}{i} - \frac{\varepsilon\beta}{i+1}} > k^2 > n \Rightarrow \\
k^{\frac{\varepsilon\beta}{i}} > nk^{\frac{\varepsilon\beta}{i+1}} > (n-1)k^{\frac{\varepsilon\beta}{i+1}} + 1 \Rightarrow k^{\frac{\varepsilon\beta}{i}} - (n-1)k^{\frac{\varepsilon\beta}{i+1}} > 1 \overset{(3)}{\Rightarrow} \\
\sum_{j=1}^{k} k^{\varepsilon\beta/j}(\ell_j - \ell'_j) > 1 \overset{(2)}{\Rightarrow} \Delta > 1.
\end{aligned}$$

□

Remark 1. Condition $k^2 > n$ for the $[n, k]$-linear code stated in Theorem 4 means that the code has a non-negligible rate with growth of n. If it is not the case, $k^2 < n$ which implies $R = \frac{k}{n} \leq \frac{\sqrt{n}-\beta+2}{n} \underset{n\to\infty}{\longrightarrow} 0$.

4 Implementation

We implemented the DeepBKZ and PotBKZ reduction algorithms in SageMath 10.6 computer algebra system [11] on top of existing [9] implementation of the BasicBKZ algorithm for codes. The source code is attached to this paper[1].

Routines from the CodeRed package [6] were also used in computations. To speed up the computations, we compiled SageMath files to binary modules using Cython. The slide reduction and full backward reduction from [8] do not have publicly available implementations, so we did not include them to the comparison.

5 Experiments

5.1 Setup

To estimate approximation factors and the value of k_1 that determine efficiency of the decoding with hybrid ISD algorithms, we used the following experimental setup.

1. For each code length n we generated 10 random generating matrices in the systematic form and run the BKZ reduction with different algorithms on them. The code rate was taken 0.5.
2. Average values for estimated parameters among 10 samples were taken for resulting tables and plots.
3. Computations for $n = 1280$ were performed on a machine with Xeon W-1370 with 8 cores clocked at 2.90 GHz. For $n = 64, \ldots, 1024$ we used the machine with Xeon Silver 4210R with 20 cores clocked at 2.40 GHz. Computations were performed in parallel, each pair (n, β) in separate process using the available cores.
4. Number of calls to MWC-oracle is given in the normalized form as the approximate number of calls to the MWC-oracle for the first block. Since the projections reduce length of the code, blocks have variable length. Therefore, on different iterations of the algorithm we have different complexity of the minimal weight codeword computation. We normalized it as follows. The first block has length $n-k+\beta$ and dimension β after removing zero columns. So, the cost of call to MWC-oracle for first block is $(n-k+\beta)2^{\beta}$ assuming that we use plain enumeration of codewords. Thus, we took on i-th position the value of $\frac{n_i 2^{\beta}}{(n-k+\beta)2^{\beta}} = \frac{n_i}{(n-k+\beta)}$, where n_i is the length of i-th projected block.
5. The approximation factor γ is estimated by dividing the weight of first vector in the BKZ reduced basis by the expected minimum weight of a codeword in a random linear code (Gilbert-Varshamov bound).

[1] Available at https://github.com/n7v/codered-qpoly/

5.2 Approximation factor estimation for $n = 1280$

The code length $n = 1280$ with rate 0.5 is length of cryptographic size. It was selected in finding low-weight codewords challenge [3] to have approximate complexity 2^{128} with BJMM [4] algorithm having best known asymptotic complexity. Results of computations for $n = 1280$ with experimental setup described above are presented in the Table 1.

Complexity estimation. The depth in the table is the maximal position (index i in the algorithm) reached during computations. From the proof of Theorem 1 we have upper bound $\mathcal{O}(n^{Depth+\beta})$ for the number of insertions (= new minimal weight codewords found for a projected block) during the work of algorithm. As we can see this bound is very pessimistic, since in all cases the number of insertions in the algorithm is less than n. The number of calls to the first block MWC-oracle after normalization is less than n. In fact, in practice the number of calls to the oracle is of order $\mathcal{O}(\log_2(n))$. The PotBKZ algorithm has lowest number of insertions and MWC-calls, however due to overhead (computing projections to check the $\Delta\Phi$ condition) it runs faster only for large block sizes. The same is true for the DeepBKZ algorithm, each deep insertion reduces potential and therefore total number of the insertions in the algorithm.

Approximation factor γ. BasicBKZ and DeepBKZ algorithms provide comparable approximation factors per taken time. In cases when one algorithm has better approximation factor it takes more time to compute the basis. The PotBKZ algorithm produces slightly worse approximation factor (length of first element in the basis is $\approx 1-3$ higher), but with the theoretical polynomial-time complexity bound and with less insertions and calls to MWC-oracles.

Decoding performance estimation, k_1. The parameter k_1 is the number of elements not equal to 1 in the epipodal matrix B^+. According to the work on LLL ($\beta = 2$) [6] the hybrid Lee-Brickell-Babai algorithm, i.e. algorithm that uses LLL instead of systematic form in ISD Lee-Brickell algorithm, provides exponential in k_1 speed up for Lee-Brickell algorithm. More advanced ISD algorithm based on birthday paradox – hybrid Wagner-Babai was used to establish low-weight computation record [1]. As we can see from the Table 1 this parameter grows linearly as the block size β increases. As a result, we expect much more speed up for Lee-Brickell-Babai by comparing with Lee-Brickell algorithm when we increase the block size.

5.3 Approximation factors for $n = 1280$ (BKZ with preprocessing)

Much better and stable results can be achieved using EpiSort preprocessing as it was done for LLL (block size 2) in [6]. This procedure makes sorting of the basis in the following way: $\mathbf{b}_1$ in the sorted basis is taken as shortest weight element in the basis, $\mathbf{b}_2$ is taken to have shortest projection among elements $\pi_1(\mathbf{b}_j)$ for $j > 1$ and so on. The EpiSort procedure is cheap and it takes time $\mathcal{O}(nk^2)$. The approximation factors and other parameters reached when we apply EpiSort before performing BKZ reduction are presented in Table 2.

Table 1. Approximation factor γ reached for different block sizes and algorithms.

n	β	Algorithm	γ	ℓ_1	k_1	MWC-calls	Depth	Insertions	Time, sec.
1280	2	BasicBKZ	2.054	295.8	12.2	7.93	12.9	29.9	14 ± 6
		DeepBKZ	1.993	287.1	12.7	12.91	12.8	33.9	34 ± 6
	4	BasicBKZ	1.968	283.4	15.2	26.26	16.5	58.7	46 ± 21
		DeepBKZ	1.963	282.7	15.5	26.57	16.8	58.4	64 ± 25
		PotBKZ	1.995	287.4	14.7	7.02	15.1	11.0	73 ± 29
	6	BasicBKZ	1.945	280.1	16.7	37.72	19.3	90.7	103 ± 39
		DeepBKZ	1.945	280.1	16.7	37.15	19.5	82.7	115 ± 39
		PotBKZ	1.957	281.9	16.3	15.30	18.0	21.7	150 ± 55
	8	BasicBKZ	1.900	273.7	18.3	39.93	23.9	143.0	242 ± 126
		DeepBKZ	1.901	273.8	17.8	40.01	23.1	138.1	252 ± 147
		PotBKZ	1.924	277.2	18.0	22.92	21.6	49.2	449 ± 170
	10	BasicBKZ	1.874	269.9	19.3	36.79	25.4	240.8	511 ± 222
		DeepBKZ	1.879	270.7	19.0	34.18	24.6	192.4	427 ± 145
		PotBKZ	1.876	270.2	18.8	33.35	25.1	127.0	914 ± 335
	12	BasicBKZ	1.859	267.7	19.7	29.43	26.6	261.4	654 ± 356
		DeepBKZ	1.860	267.9	20.1	30.26	29.1	255.6	728 ± 321
		PotBKZ	1.851	266.6	19.7	30.63	27.6	180.5	1314 ± 609
	14	BasicBKZ	1.829	263.4	20.5	28.92	29.9	446.0	1436 ± 597
		DeepBKZ	1.832	263.9	20.5	28.55	28.2	450.0	1493 ± 668
		PotBKZ	1.831	263.7	20.1	27.80	29.9	238.2	1826 ± 664
	16	BasicBKZ	1.804	259.8	20.7	25.85	32.7	472.5	1726 ± 1139
		DeepBKZ	1.804	259.9	20.9	27.28	32.0	570.7	2212 ± 1373
		PotBKZ	1.811	260.8	20.4	27.65	31.5	453.1	4625 ± 1939
	18	BasicBKZ	1.786	257.2	21.2	27.10	33.6	794.5	3256 ± 914
		DeepBKZ	1.786	257.2	21.2	27.22	31.1	823.4	3412 ± 1724
		PotBKZ	1.800	259.3	21.1	22.15	34.1	333.4	2819 ± 1246
	20	BasicBKZ	1.770	255.0	21.8	26.01	34.1	904.1	3964 ± 2104
		DeepBKZ	1.777	256.0	21.5	23.11	34.8	579.9	2983 ± 1151
		PotBKZ	1.786	257.3	21.2	23.17	36.9	435.1	3911 ± 1502
	22	BasicBKZ	1.763	254.0	22.2	24.39	37.6	748.3	4445 ± 2263
		DeepBKZ	1.760	253.5	22.3	26.26	37.6	1208.7	7953 ± 6168
		PotBKZ	1.759	253.3	22.0	25.38	40.1	387.76	7332 ± 4184
	24	BasicBKZ	1.731	249.4	22.6	28.18	40.2	1417.6	14246 ± 9152
		DeepBKZ	1.738	250.3	22.7	26.18	39.8	926.2	9909 ± 3660
		PotBKZ	1.741	250.8	22.6	24.30	44.4	710.2	9311 ± 5823

Table 2. Approximation factor γ reached for different block sizes and algorithms (with EpiSort preprocessing).

n	β	Algorithm	γ	ℓ_1	k_1	MWC-calls	Depth	Insertions	Time, sec.
1280	2	BasicBKZ	1.931	278.2	22.8	2.41	323.3	3.3	34 ± 1
		DeepBKZ	1.931	278.2	22.8	2.53	323.3	2.7	35 ± 2
	4	BasicBKZ	1.925	277.3	23.3	7.67	323.3	16.7	43 ± 3
		DeepBKZ	1.925	277.3	23.3	7.67	323.3	16.4	46 ± 4
		PotBKZ	1.930	278.0	23.2	3.37	323.3	3.0	49 ± 8
	6	BasicBKZ	1.910	275.1	23.1	19.32	323.3	30.5	68 ± 13
		DeepBKZ	1.910	275.1	23.1	18.90	323.3	30.7	74 ± 16
		PotBKZ	1.916	276.0	23.3	6.85	323.3	6.6	72 ± 17
	8	BasicBKZ	1.890	272.3	23.7	29.12	323.3	69.3	158 ± 54
		DeepBKZ	1.890	272.3	23.8	28.68	323.3	61.3	150 ± 51
		PotBKZ	1.890	272.3	23.7	12.34	323.3	20.5	163 ± 75
	10	BasicBKZ	1.851	266.6	23.8	28.21	323.3	112.4	252 ± 108
		DeepBKZ	1.851	266.6	23.7	28.18	323.3	109.4	274 ± 113
		PotBKZ	1.861	268.1	23.3	21.89	323.3	50.3	371 ± 127
	12	BasicBKZ	1.845	265.7	24.3	25.12	323.3	163.1	439 ± 173
		DeepBKZ	1.845	265.7	24.2	24.45	323.3	153.7	469 ± 175
		PotBKZ	1.854	267.0	23.6	22.05	323.3	87.6	647 ± 361
	14	BasicBKZ	1.818	261.9	24.1	24.78	323.3	331.5	948 ± 485
		DeepBKZ	1.818	261.9	24.1	23.81	323.3	266.2	930 ± 566
		PotBKZ	1.832	263.9	24.3	22.65	323.3	166.4	1294 ± 950
	16	BasicBKZ	1.786	257.3	24.6	28.05	323.3	541.2	1781 ± 1325
		DeepBKZ	1.795	258.6	24.9	27.21	323.3	449.3	1599 ± 740
		PotBKZ	1.804	259.8	24.3	20.56	323.3	177.7	1429 ± 787
	18	BasicBKZ	1.779	256.2	25.7	25.95	323.3	502.5	2150 ± 719
		DeepBKZ	1.775	255.7	25.5	24.63	323.3	443.0	1782 ± 995
		PotBKZ	1.797	258.8	24.6	19.68	323.3	176.0	1447 ± 516
	20	BasicBKZ	1.763	253.9	25.4	24.64	323.3	642.9	2843 ± 2329
		DeepBKZ	1.761	253.7	25.7	24.00	323.3	489.1	2425 ± 1469
		PotBKZ	1.774	255.5	25.5	22.85	323.3	421.2	3687 ± 1843
	22	BasicBKZ	1.743	251.1	26.1	25.08	323.3	628.2	3449 ± 1299
		DeepBKZ	1.747	251.6	25.9	24.32	323.3	523.2	3483 ± 1657
		PotBKZ	1.754	252.6	25.3	21.50	323.3	492.8	4852 ± 4486
	24	BasicBKZ	1.736	250.0	26.0	24.76	323.3	813.6	7487 ± 3672
		DeepBKZ	1.737	250.2	26.6	24.57	323.3	777.9	8047 ± 6294
		PotBKZ	1.733	249.6	26.0	22.25	323.3	433.2	5852 ± 5530

Complexity estimation. The EpiSort function makes the profile to contain some number of 2 amount ones in the middle of profile. So, the Depth is the same for all cases, but the most of elements in profile are still equal to 1. Thus this does not affect much the normalized number of calls to MWC-oracle. So, even the BKZ algorithm with preprocessing has also a number of MWC-calls of order $\mathcal{O}(\log n)$. Deep insertions in the DeepBKZ algorithm reduce the overall number of insertions (=changes in the basis when new minimum weight codeword found) as expected from theory – each deep insert reduces current potential by a big value. This leads to less calls to MWC-oracle. However, this reduces the computational time only for block sizes ≥ 14, because the computation of a minimal weight codeword in block of lesser dimension is very fast and the DeepBKZ algorithm has overhead in computing projections.

Approximation factors are significantly better with EpiSort preprocessing than without it. They are almost the same for BasicBKZ and DeepBKZ. The PotBKZ algorithm has slightly worse approximation factors in comparison with BasicBKZ and DeepBKZ, however it is a polynomial algorithm (for fixed block size β).

Decoding performance is also much better with EpiSort preprocessing.

5.4 Experimental validation of Heuristic 1

We remind the reader that Heuristic 1 estimates the approximation factor of the weight of the first element in the β-BKZ-reduced basis. Since Heuristic 1 depends on the claim that $k_1 = \mathcal{O}(\log n)$, we first experimentally show that $k_1 = c_1 \cdot \log_2 n$ for a constant c_1. Fig. 1 represents experimental data supporting this claim. As we can see the constant c_1 is ≈ 2 and less than 3 for all $\beta \leq 22$.

Fig. 2 shows results of computations for 10 random linear codes and algorithms presented in this paper. The plot shows that the first vectors of the β-BKZ-reduced bases always have weights near the Heuristic 1. Preprocessing with EpiSort makes the algorithms return the value below the heuristic bound.

5.5 Running time for $\beta = 24$

In this section we consider the case of big $\beta = 24$ and $n = 64, \ldots, 1024$. Figures 3 and 4 show number of calls and running time of the algorithms described in our work. As we can see PotBKZ algorithm has the smallest number of calls to the MWC-oracle and the fastest running time while providing basis quality near the Heuristic 1 according to Fig. 2. DeepBKZ and BasicBKZ algorithms take comparable time, when one algorithm is faster it provides worse basis reduction. Preprocessing with EpiSort reduces the number of calls and runtime for all algorithms.

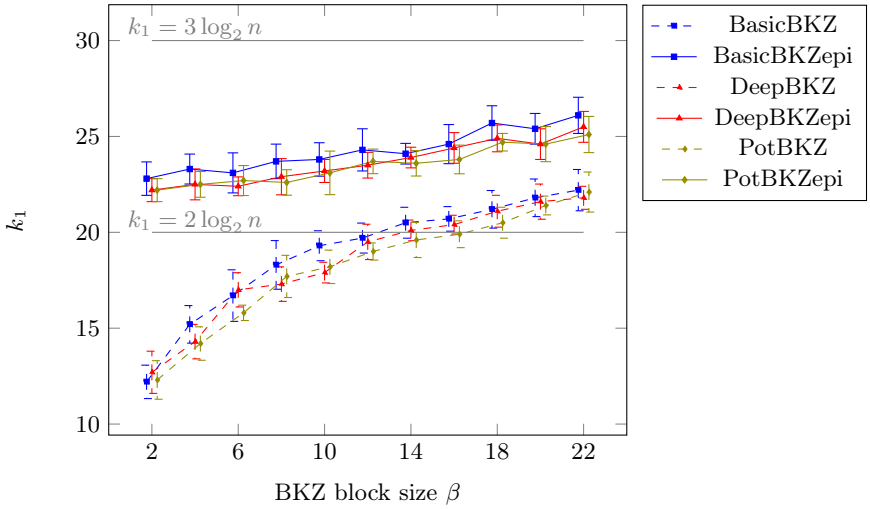

Fig. 1. Parameter k_1 (decoding performance) for BKZ algorithms for random codes of length $n = 1024$, $k = 512$

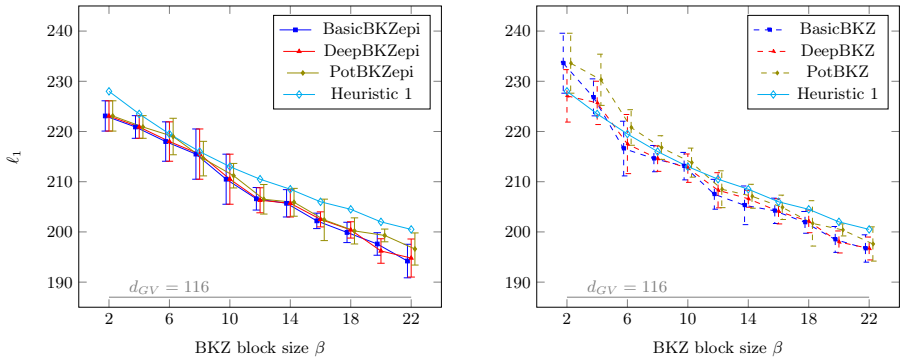

Fig. 2. Average value of ℓ_1 (weight of the first element in the reduced basis) for BKZ algorithms for 10 random codes of length $n = 1024$, $k = 512$

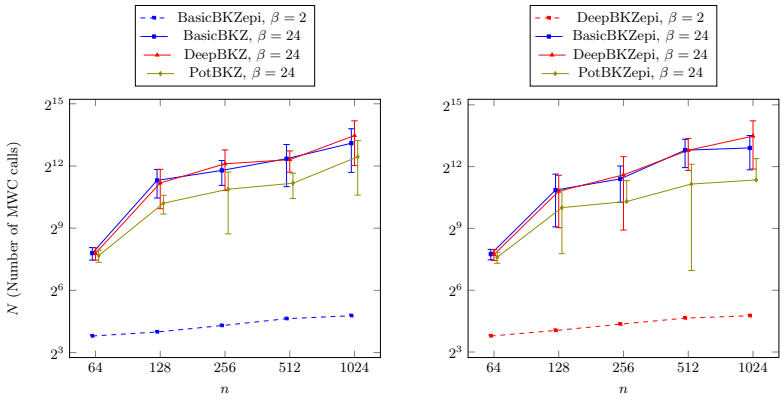

Fig. 3. BKZ on random codes of length n, dimension $k = n/2$, number of calls to MWC-oracle, mean value of $S = 10$ samples. Logarithmic scale.

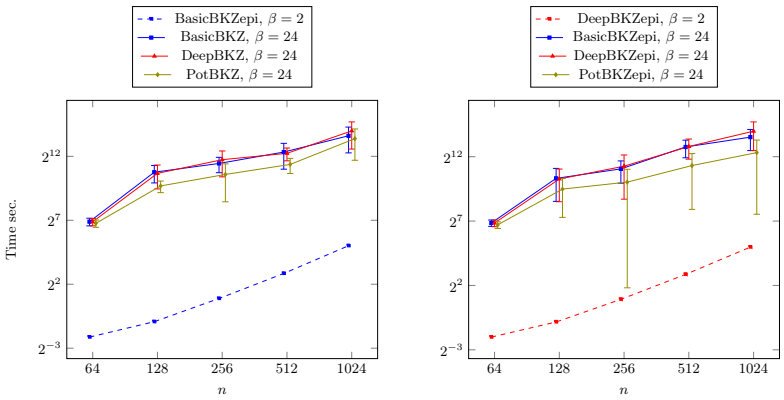

Fig. 4. BKZ on random codes of length n, dimension $k = n/2$, running time, mean value of $S = 10$ samples.

Conclusion

We proposed modifications of the BKZ algorithm for linear codes with polynomial and quasi-polynomial complexity for fixed β. The obtained modifications produce the bases of slightly lower quality, but have non-trivial complexity bounds. This opens the possibility to build hybrid ISD algorithms with non-trivial complexities. In addition, we studied more advanced modification of the BKZ algorithm, the DeepBKZ algorithm, that provides speed up of the BKZ algorithm for big block sizes.

Acknowledgments. The research was funded by the Russian Science Foundation (project No. 25-21-00668, https://rscf.ru/en/project/25-21-00668/).

References

1. Challenges for code-based problems — decodingchallenge.org. https://decodingchallenge.org/low-weight/record/17 (2024)
2. Challenges for code-based problems. low weight codeword problem: Details on record 20. https://decodingchallenge.org/low-weight/record/20 (2024)
3. Challenges for code-based problems — decodingchallenge.org. https://decodingchallenge.org/low-weight (2026)
4. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in 2 n/20: How 1+ 1= 0 improves information set decoding. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 520–536. Springer (2012)
5. Berlekamp, E., McEliece, R., Van Tilborg, H.: On the inherent intractability of certain coding problems (corresp.). IEEE Transactions on Information theory **24**(3), 384–386 (1978)
6. Debris-Alazard, T., Ducas, L., Van Woerden, W.P.: An algorithmic reduction theory for binary codes: LLL and more. IEEE Transactions on Information Theory **68**(5), 3426–3444 (2022)
7. Debris-Alazard, T., Remaud, M., Tillich, J.P.: Quantum reduction of finding short code vectors to the decoding problem. IEEE Transactions on Information Theory **70**(7), 5323–5342 (2023)
8. Ghentiyala, S., Stephens-Davidowitz, N.: More basis reduction for linear codes: backward reduction, BKZ, slide reduction, and more. arXiv preprint arXiv:2408.08507 (2024)
9. Kolesnikov, N.S., Novoselov, S.A.: Complexity estimation of decoding problem and finding of low-weight codewords using basis reduction. Prikladnaya Diskretnaya Matematika. Supplement (18), 229–233 (2025)
10. Sato, A., Yasuda, M.: A new BKZ-type reduction with provable termination and development of its self-dual variant. Japan Journal of Industrial and Applied Mathematics **43**(2), 42 (2026)
11. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 10.6) (2026), `https://www.sagemath.org`
12. Yasuda, M., Nakamura, S., Yamaguchi, J.: Analysis of DeepBKZ reduction for finding short lattice vectors. Designs, Codes and Cryptography **88**(10), 2077–2100 (2020)