Lattice Basis Reduction in Practice

A Study on real-world performance

Mar 22, 2024

Contents

1 Introduction **4**

2 Background **5**

2.1 Lattice Basis Reduction algorithms 5

2.1.1 LLL Algorithm 5

2.1.2 BKZ Algorithm 5

2.1.3 Slide Algorithm 6

2.1.4 SDBKZ Algorithm 6

2.2 Metrics for studying Lattice Reduction Algorithms 6

2.2.1 Orthogonality Defect 7

2.2.2 Root Hermite Factor 7

2.2.3 Half Volume Ratio 7

2.2.4 CPU time 7

3 Evaluations **9**

3.1 Experimental Setup 9

3.2 Results 10

3.2.1 Convergence Analysis 10

3.2.2 Basis Quality Analysis 10

3.2.3 Performance Analysis 11

3.2.4 Trade-off Analysis 11

4 Conclusion **20**

List of Figures

3.1 Convergence rate of BKZ vs Slide vs SDBKZ algorithms 10
3.2 $log(|b_i^*|)$ plot comparison 11
3.3 Orthogonality Defect (Δ) comparison 12
3.4 Root Hermite Factor (γ) comparison 13
3.5 Half Volume Ratio (ν) comparison 14
3.6 Orthogonality Defect (Δ) comparison (Normalised) 15
3.7 Root Hermite Factor (γ) comparison (Normalised) 16
3.8 Half Volume Ratio (ν) comparison (Normalised) 17
3.9 Performance comparison 18
3.10 Quality-Performance trade-off comparison 19

1 Introduction

Lattices are mathematical constructs that deal with discrete regular arrangements of points in $n-$dimensional space. Lattice occur in many areas of mathematics and computer science, and have seen widespread applications in coding theory, optimisation and cryptography. Particularly in cryptography, in recent times, lattice-based cryptographic techniques have seen a rapid surge in popularity due to their resistance to quantum attacks.

Lattice basis reduction is a process of finding a "smaller"/"nicer" representation for a lattice, that is amenable to be used in applications. A reduced lattice basis can enable and/or accelerate computations to certain NP-Hard problems that have implications for cryptographic security and cryptanalysis attacks.

Several lattice basis reduction algorithms have been developed over the years, with the most popular ones being the Lenstra-Lenstra-Lovász algorithm (LLL) [LLL82] and the Block Korkin-Zolotarev algorithm (BKZ) [SE94]. These algorithms have been widely studied, theorised and used in practice. However, the performance of these algorithms in practice often differs significantly from that of the theory and is not very well understood [MW15].

In this work, we study the performance of lattice basis reduction algorithms in practice. We evaluate the quality and performance of LLL, BKZ, Slide [GN08], Self-Dual BKZ (SDBKZ) algorithms [MW15] on modern laptop and server commodity hardware. We study in detail the impact of the various algorithm parameters on the quality and performance of these lattice reduction algorithms and also compare among them. We study the scalability of these algorithms. We also attempt to characterise the trade-offs between the quality of the reduced basis and the performance of the reduction algorithms.

2 Background

A lattice (λ) can be imagined as a vector space that operates on an integer coefficient space. It is defined as

$$\Lambda \stackrel{\triangle}{=} \{Bx, x \in Z^n\}$$

Where $B = (b_1, b_2, ..., b_n)$, $b_1, b_2, ..., b_n$ are linearly independent basis vectors.

It is to be noted that a unitary transformation U on a lattice basis B does not alter the lattice (Λ).

$$B^{'} = BU$$
$$\Lambda^{'} = \mathcal{L}(B^{'}) = \mathcal{L}(BU) = \mathcal{L}(B) = \Lambda$$

2.1 Lattice Basis Reduction algorithms

A lattice basis reduction is a process of finding a "nice" unitary transform U that would produce a lattice basis that is as close to Orthogonal as possible and has small basis vectors. A smaller basis would mean that the lattice is amenable to be used for many applications.

We will now discuss some of the popular lattice basis reduction algorithms that have been developed over the years in this section.

2.1.1 LLL Algorithm

Lenstra-Lenstra-Lovász algorithm (LLL) [LLL82] is a lattice basis reduction algorithm that is based on the Gram-Schmidt orthogonalization process. The LLL algorithm is a heuristic algorithm that is known to produce a reduced basis that is close to orthogonal and has small basis vectors, and it executes in polynomial time. The LLL algorithm is widely used in practice due to its simplicity and efficiency.

LLL algorithmic framework provides a bounded guarantee on the quality of the reduced basis. Refer to [Mic24] [Den16] for the details of the algorithms and the bounds analysis.

2.1.2 BKZ Algorithm

Block Korkin-Zolotarev algorithm (BKZ) [SE94], as the name suggests is a block reduction algorithm that operates on κ sized blocks of lattice basis vectors, i.e. a κ dimensional sublattice. BKZ algorithm performs SVP reduction on the κ sized blocks of lattice basis vectors to obtain SVP reduced sublattice blocks. BKZ enforces stronger conditions on

basis vectors than LLL, and thereby it is a generalisation of the LLL algorithm. The BKZ algorithm is known to produce a reduced basis that is of higher quality than the LLL algorithm but at the cost of higher computational complexity.

For $\kappa = 2$ the BKZ algorithm is equivalent to the LLL algorithm. For more details on the algorithm and analysis of the running time and convergence, refer to [Wal20a].

2.1.3 Slide Algorithm

The Slide algorithm [GN08], like BKZ, also belongs to the class of block reduction algorithms. However, unlike in BKZ, the Slide algorithm applies a Dual-SVP (DSVP) reduction operation on the κ dimensional sublattice.

The Slide algorithm has a higher computational cost and is not used often in practice [MW15]. For the details of the algorithm and the analysis, refer to [Wal20b].

2.1.4 SDBKZ Algorithm

Self-Dual BKZ (SDBKZ) algorithm [MW15] is a modern variant of the BKZ algorithm that leverages and combines the ideas of SVP reduction and DSVP reduction into a single algorithm. The SDBKZ algorithm is known to produce a reduced basis that is of higher quality than the BKZ algorithm. The SDBKZ algorithm is widely used in practice due to its favourable quality-performance trade-off.

SDBKZ algorithm involves a forward iteration and a backward iteration, and a pair of forward and backward iterations known as an SDBKZ "tour". In the forward iteration, similar to the BKZ algorithm, an SVP reduction is performed on the κ dimensional sublattices, and in the backward iteration the algorithm performs DSVP reduction. For the details of the algorithm and the analysis, refer to [Wal20c] [MW15].

2.2 Metrics for studying Lattice Reduction Algorithms

We would like to study the quality of the reduced basis and the compute performance of the lattice reduction algorithms.

The quality of the basis is critical for understanding the efficiency and thereby the feasibility of certain cryptanalysis attacks on cryptographically secure systems. The quality can be assessed using various metrics, each highlighting different aspects of the basis's utility and performance. Some of the metrics we consider in this report include the Orthogonality Defect (Δ), Root Hermite Factor (γ), and Half Volume Ratio (ν).

We measure the performance of the lattice reduction algorithms using the CPU time. The CPU time is a direct measure of the computational cost of the reduction process.

In practice, achieving a balance between the quality of the basis and the computational resources is crucial for certain classes of attacks to be feasible.

2.2.1 Orthogonality Defect

The Orthogonality Defect (Δ) is a measure of how far the basis is from being orthogonal. A lower Δ indicates a basis closer to orthogonality, which is a desirable property. The Orthogonality Defect is defined as follows:

$$\Delta(B) = \left(\frac{\prod_{i=1}^{n} ||b_i||}{\det(\Lambda)} \right)$$

Where b_i are the basis vectors and $\det(\Lambda)$ is the determinant of the lattice, equivalent to the volume spanned by the basis.

2.2.2 Root Hermite Factor

The Root Hermite Factor (γ) is a dimensionless quantity used to measure the relative closeness of the shortest basis vector to the Minkowski bound. The RHF is defined as follows:

$$\gamma(B) = \left(\frac{||b_0||}{\det(\Lambda)^{\frac{1}{n}}} \right)$$

Where $||b_0||$ is the length of the shortest basis vector in the reduced basis B, n is the dimension of the lattice, and $\det(\Lambda)$ is the lattice's determinant. A smaller γ suggests that the shortest basis vector is close to the actual shortest vector in the lattice, which is also a desirable property.

2.2.3 Half Volume Ratio

The Half Volume Ratio (ν) is a metric that measures the disparity in the size of basis vectors. It compares the volume of the parallelepiped spanned by the first half of the basis vectors to that of the volume spanned by the second half of the lattice. The HVR is given by:

$$\nu(B) = \left(\frac{\prod_{i=1}^{n/2} ||b_i||}{\prod_{i=n/2+1}^{n} ||b_i||} \right)^{\frac{1}{n}}$$

Where $||b_i||$ is the length of the basis vectors. A smaller ν indicates that the basis vectors are more uniformly distributed in length, which is a desirable property.

2.2.4 CPU time

CPU time is an essential metric for practical applications and to provide real-world risk assessment. It measures the computational cost of performing lattice basis reduction. CPU time alongside other metrics provides a holistic view of the algorithm.

$$\text{CPU time} = t_{\text{end}} - t_{\text{start}}$$

Where t_{start} and t_{end} are the timestamps at the beginning and end of the reduction process, respectively.

3 Evaluations

In this section, we present the results of our evaluations of the lattice basis reduction algorithms.

We begin by studying the convergence behaviour of these algorithms, in Sec 3.2.1, followed by the effect of the algorithm parameters on the quality of the reduced basis, in Sec 3.2.2.

Upon understanding the behaviour of the algorithms, we evaluate the performance and study its scaling characteristics, in Sec 3.2.3.

Further, we characterise the trade-offs between the quality of the reduced basis and the performance of the lattice reduction algorithms, in Sec 3.2.4.

3.1 Experimental Setup

We are using the floating-point LLL (fpLLL) library [dt23a] for the implementation of LLL, BKZ, Slide, and SDBKZ lattice algorithms. fpyLLL [dt23b] is a Python wrapper around the fpLLL library that provides a high-level interface to the algorithms.

The implementation of the experiments presented in this study and the testing tools is published in a GitHub repository [Kad24]. The repository also contains dockerfiles for easy replication of the experiments.

We evaluate the performance of the lattice reduction algorithms on modern commodity hardware, such as a laptop and a server. The hardware configurations for our experiments:

- Laptop: Intel Core i7-6700HQ, 8 x 2.60GHz (upto 3.50GHz), 32GB DDR4 RAM
- Server: Intel Xeon Gold 6326, 64 x 2.90GHz, 512GB DDR4 RAM

In our evaluation, we consider a range of realistic lattice sizes and dimensions, with 128-bit multi-precision integer (mpz) and 64-bit floating-point arithmetic.

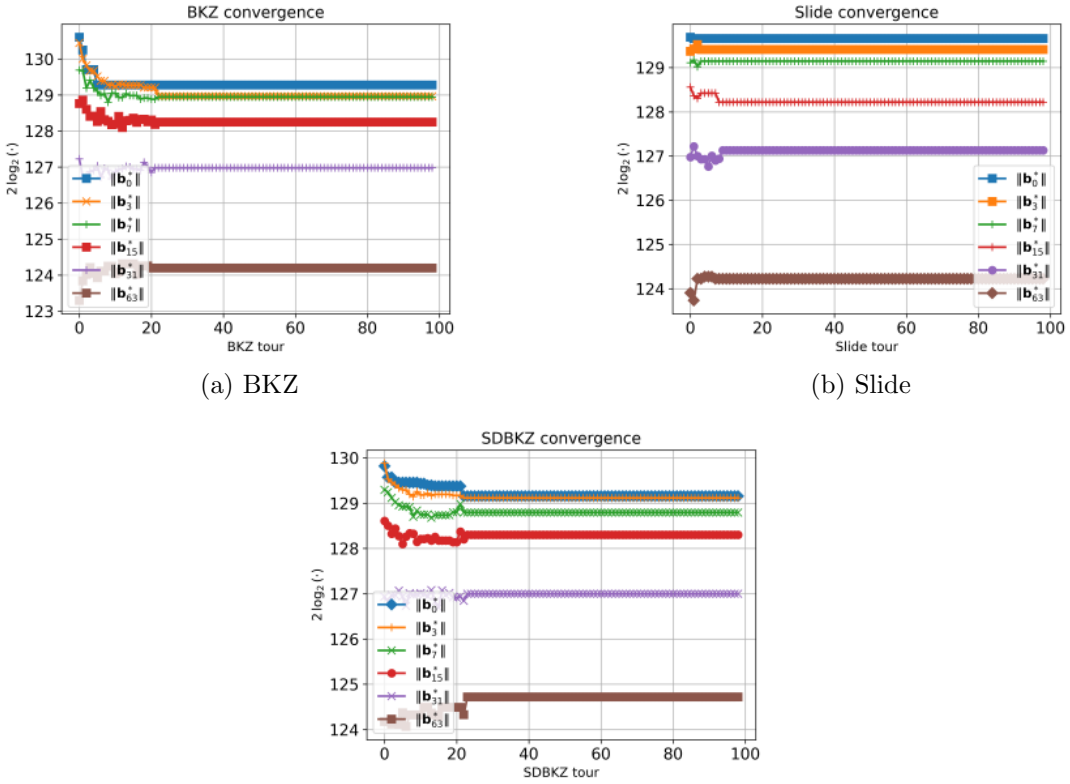
The shaded region in all the following figures depicts the 95% confidence interval.

We observe that the algorithm executes faster on the Laptop than on the Server. We believe that this is due to the fact that the laptop has a higher burst clock speeds than the server. As the fpLLL library is single threaded we do not benefit from the additional cores in the server.

3.2 Results

3.2.1 Convergence Analysis

In Figure 3.1, we plot the log magnitude of a few basis vectors. We observe that all the algorithms converge quite quickly in between 25-30 "tours".



(a) BKZ

(b) Slide

(c) SDBKZ

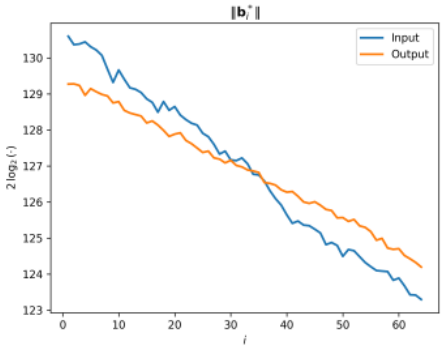
Figure 3.1: Convergence rate of BKZ vs Slide vs SDBKZ algorithms

In Figure 3.2, we plot the log magnitude of the basis vectors before and after the reduction process. We observe that the basis vectors are reduced and also the curve is flattened.

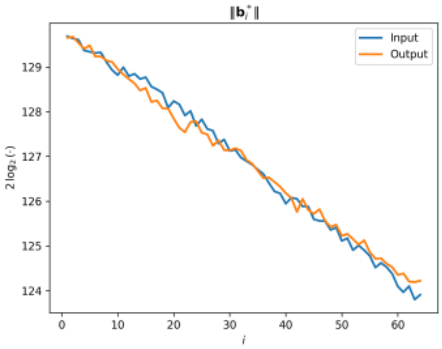
3.2.2 Basis Quality Analysis

In Figure 3.3, 3.4, 3.5, we plot the Orthogonality Defect (Δ), Root Hermite Factor (γ), and Half Volume Ratio (ν) for the BKZ, Slide, and SDBKZ algorithms against the block parameter κ for different dimension lattices.

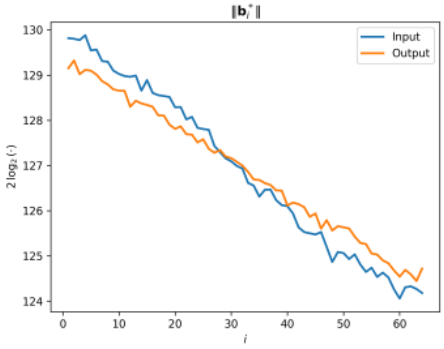
We plot the normalised quality metric to study the relative impact of κ at different dimensions in Figure 3.6, 3.7, 3.8.



(a) BKZ



(b) Slide



(c) SDBKZ

Figure 3.2: $log(|b_i^*|)$ plot comparison

As we expect the quality metric improves with increasing κ, and we also observe that the relateive impact of κ is higher at higher dimensions.

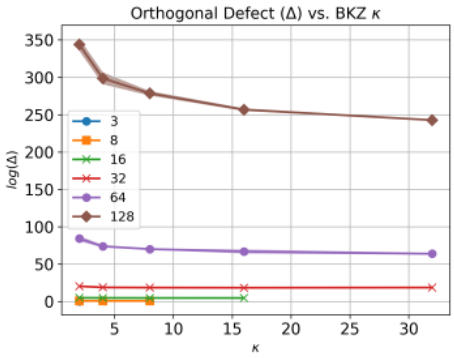
3.2.3 Performance Analysis

In Figure 3.9, we plot the CPU time for the BKZ, Slide, and SDBKZ algorithms against the block parameter κ for different dimension lattices.

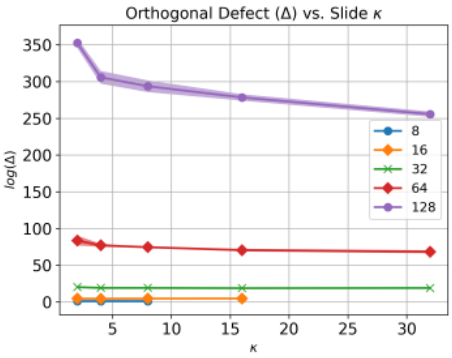
We observe that the CPU time increases exponentially with increasing κ, and as we might expect, higher dimensional lattices require CPU time. Note that the CPU time for the Slide algorithm is scaling sublinearly with κ compared to the BKZ and SDBKZ algorithms.

3.2.4 Trade-off Analysis

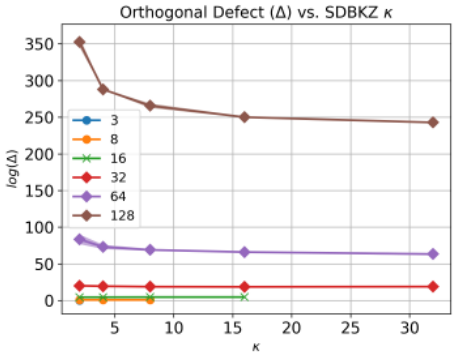
In Figure 3.10, we plotted the position of the various algorithms on a compute quality trade-off diagram. On the x-axis is the compute cost and on the y-axis is the quality metrics, for a given dimension of the lattice (128 dimensions in this case).



(a) BKZ



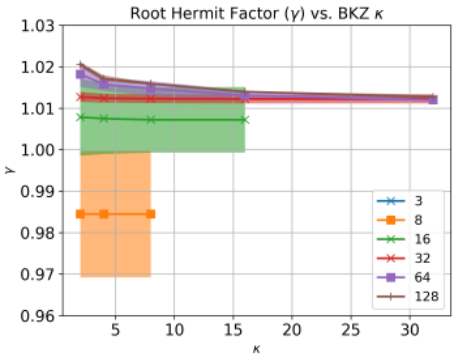
(b) Slide



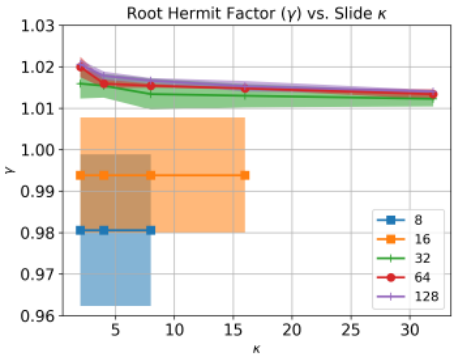
(c) SDBKZ

Figure 3.3: Orthogonality Defect (Δ) comparison

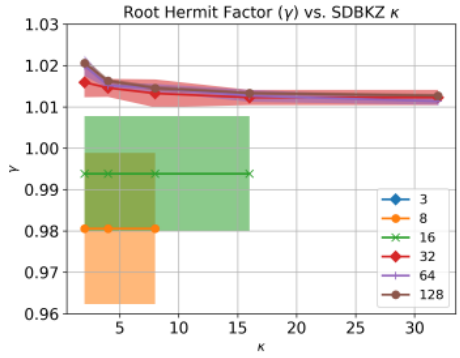
We observe that the SDBKZ algorithm achieves a better quality-performance trade-off than the other algorithms. The Slide and BKZ algorithms perform more or less similarly in terms of the quality-performance trade-off.



(a) BKZ

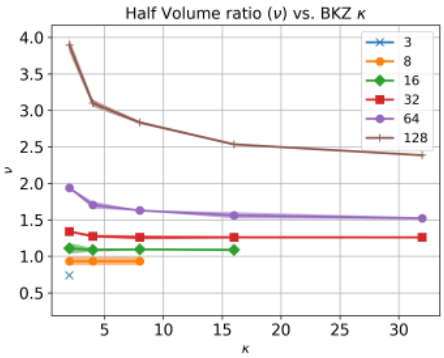


(b) Slide

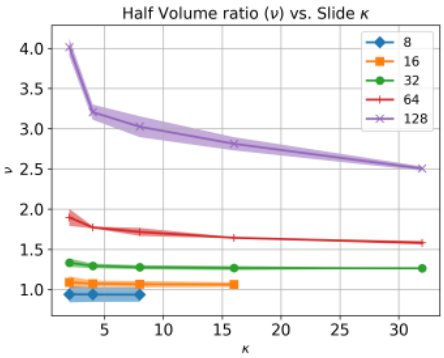


(c) SDBKZ

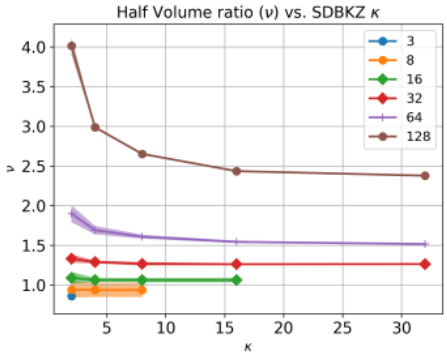
Figure 3.4: Root Hermite Factor (γ) comparison



(a) BKZ

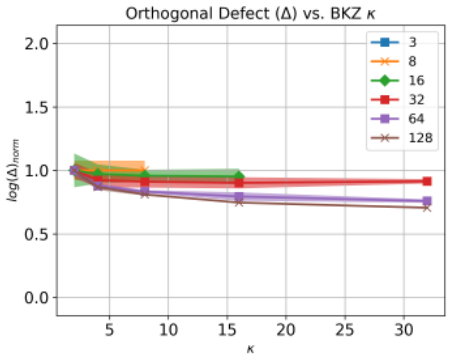


(b) Slide

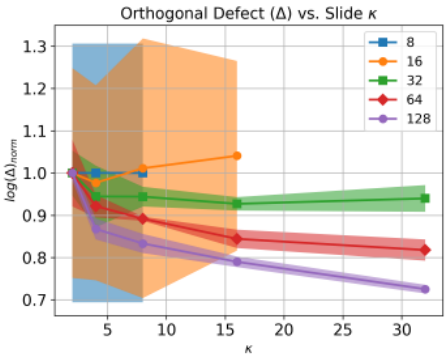


(c) SDBKZ

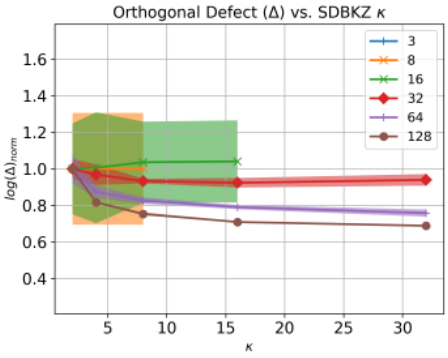
Figure 3.5: Half Volume Ratio (ν) comparison



(a) BKZ

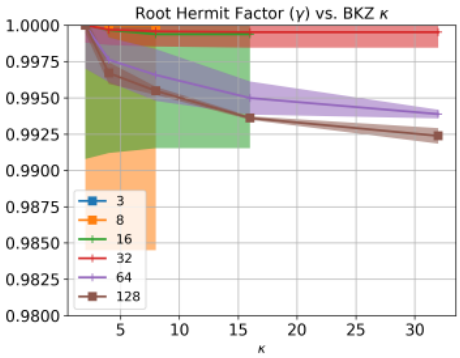


(b) Slide

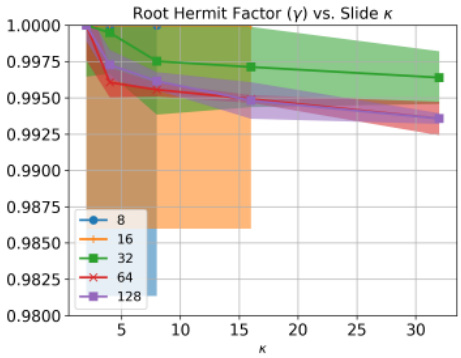


(c) SDBKZ

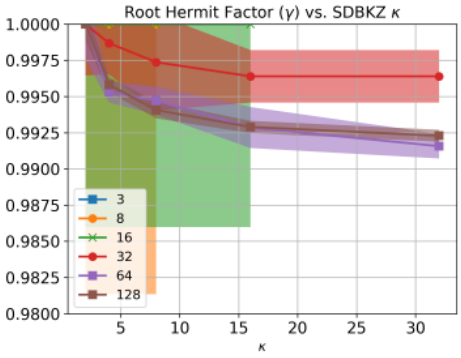
Figure 3.6: Orthogonality Defect (Δ) comparison (Normalised)



(a) BKZ

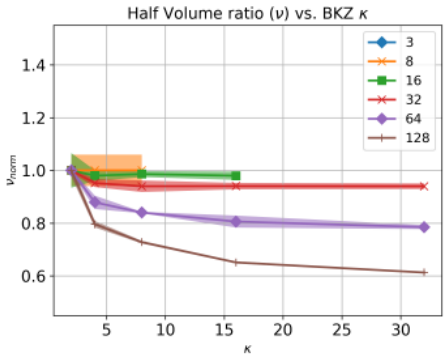


(b) Slide

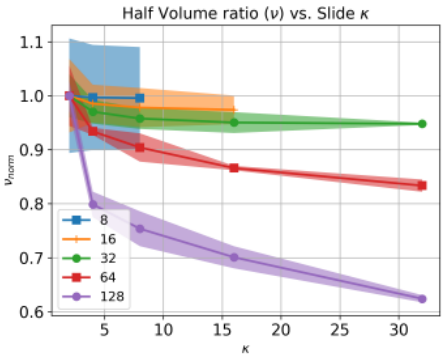


(c) SDBKZ

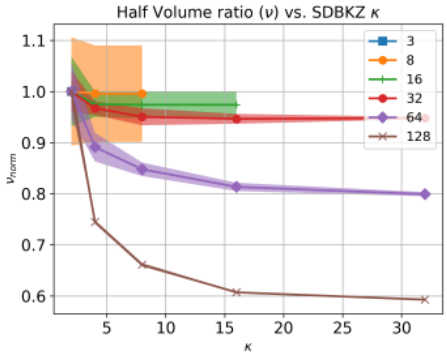
Figure 3.7: Root Hermite Factor (γ) comparison (Normalised)



(a) BKZ

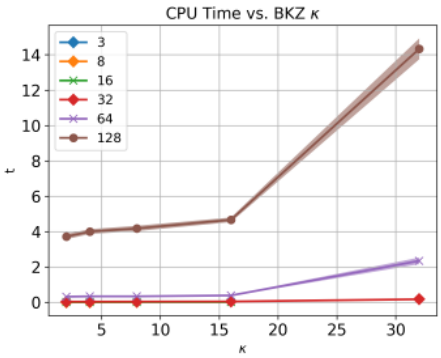


(b) Slide

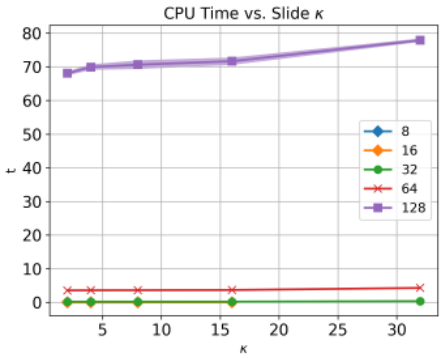


(c) SDBKZ

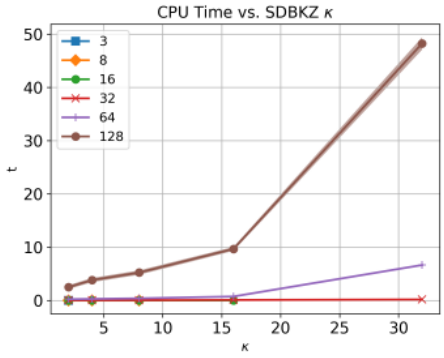
Figure 3.8: Half Volume Ratio (ν) comparison (Normalised)



(a) BKZ



(b) Slide



(c) SDBKZ

Figure 3.9: Performance comparison

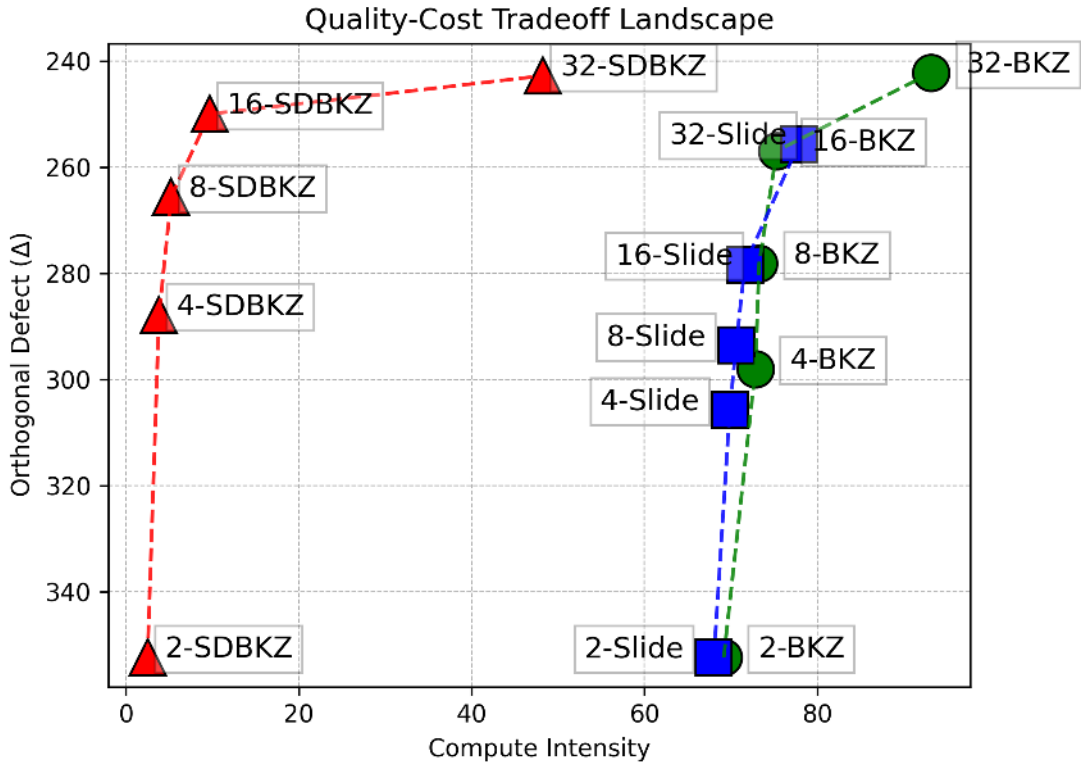


Figure 3.10: Quality-Performance trade-off comparison

4 Conclusion

In this work, we conducted an extensive study on the performance of various lattice basis reduction algorithms, including LLL, BKZ, Slide, and SDBKZ, within practical environments, focusing on modern laptop and server hardware. Through rigorous experimentation and analysis, we explored the impact of algorithm parameters on both the quality of the reduced basis and the computational efficiency of these algorithms. Our research revealed significant insights into the scalability of these algorithms and allowed us to delineate the trade-offs between basis quality and algorithmic performance.

Our findings indicate that the SDBKZ algorithm outperforms its counterparts in achieving an optimal balance between basis quality and computational requirements. This algorithm demonstrates a superior quality-performance trade-off, making it a preferred choice for applications demanding high efficiency and reliability in lattice basis reduction.

The practical implications of our study extend to the fields of cryptography, optimization, and computational mathematics, where lattice basis reduction plays a critical role. By offering a comparative analysis of the state-of-the-art algorithms' performance. This work will help bridge the gap between theory and practice, aiding in the informed selection of lattice basis reduction algorithms tailored to specific hardware and application requirements.

Bibliography

[Den16] Xinyue Deng. An introduction to lenstra-lenstra-lovasz lattice basis reduction algorithm. *Massachusetts Institute of Technology (MIT)*, 2016.

[dt23a] The FPLLL development team. fplll, a lattice reduction library, Version: 5.4.5. Available at `https://github.com/fplll/fplll`, 2023.

[dt23b] The FPLLL development team. fpylll, a Python wrapper for the fplll lattice reduction library, Version: 0.6.1. Available at `https://github.com/fplll/fpylll`, 2023.

[GN08] Nicolas Gama and Phong Q Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 207–216, 2008.

[Kad24] Sreevatsank Kadaveru. GitHub Repository for Lattice Reduction Study. `https://github.com/sreevatsank1999/LatticeReduction-Study/`, 2024. [Online].

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.

[Mic24] Daniele Micciancio. Lecture 3, CSE 206A: Lattice Algorithms and Applications Notes. `https://cseweb.ucsd.edu/classes/wi24/cse206A-a/LecLLL.pdf`, 2024. [Online].

[MW15] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. Cryptology ePrint Archive, Paper 2015/1123, 2015. `https://eprint.iacr.org/2015/1123`.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66:181–199, 1994.

[Wal20a] Michael Walter. Lattice Blog Reduction – Part I: BKZ. `https://blog.simons.berkeley.edu/2020/04/lattice-blog-reduction-part-i-bkz/`, 2020. [Online].

[Wal20b] Michael Walter. Lattice Blog Reduction – Part II: Slide Reduction. `https://blog.simons.berkeley.edu/2020/05/lattice-blog-reduction-part-ii-slide-reduction/`, 2020. [Online].

[Wal20c] Michael Walter. Lattice Blog Reduction – Part III: Self-Dual BKZ. https://blog.simons.berkeley.edu/2020/08/lattice-blog-reduction-part-iii-self-dual-bkz/, 2020. [Online].