An Experimental Analysis of Basis Reduction Algorithms

Guy Zamir

March 2024

1 Introduction

Given a matrix $\mathbf{B} = [\mathbf{b_1}, \ldots, \mathbf{b}_m] \in \mathbb{R}^{n \times m}$ where $\mathbf{b_1}, \ldots, \mathbf{b_n}$ are linearly independent vectors, the lattice generated by $\mathbf{B}$ is defined to be

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) = \left\{ \sum_{i=1}^{m} x_i \mathbf{b}_i \,\middle|\, x_i \in \mathbb{Z} \right\}.$$

We call $\mathbf{B}$ a basis for the lattice. Furthermore, we define the minimum distance of a lattice $\mathcal{L}$ to be

$$\lambda(\mathcal{L}) := \min_{\mathbf{v} \in \mathcal{L} \setminus \{0\}} \|\mathbf{v}\|.$$

We now describe the Shortest Vector Problem (SVP), which is arguably the most important problem involving lattices. Given a basis $\mathbf{B}$ for a lattice $\mathcal{L}$, SVP asks us to find a lattice point $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| = \lambda_1$.

SVP is rather difficult to solve. Under randomized reductions, it is NP-hard to solve and NP-hard to approximate within constant factor. Moreover, although it is still an open problem to show that SVP is NP-hard under deterministic reduction, it is even conjectured that SVP is NP-hard to approximate within a polynomial factor [1].

In practice, we typically approximate SVP by using basis reduction algorithms. Given a basis for a lattice, the goal of a basis reduction algorithm is to find a "good" basis that generates the same lattice, i.e. one with short vectors that are almost orthogonal. It turns out that basis reductions algorithms usually come with some guarantees on the size of the shortest vector in the reduced basis.

2 Algorithms for SVP

The first algorithm for solving SVP that we will discuss is LLL. It was the first polynomial-time algorithm to guarantee a solution within an exponential approximation factor. That is,

$$\frac{\|LLL(\mathbf{B})_1\|}{\lambda(\mathcal{L})} \leq 2^{O(n)}.$$

Indeed, omitting the details, one can show without too much difficulty that

$$\|LLL(\mathbf{B})_1\| \leq \beta^{\frac{n-1}{2}} \lambda(\mathcal{L}),$$

where $\beta = \frac{1}{\delta - \eta^2}$ depends on the LLL hyperparamters $\delta \in (\frac{1}{4}, 1]$ and $\eta \in [\frac{1}{2}, \sqrt{\delta})$. Subsequently, if we choose δ close to 1 and η close to $\frac{1}{2}$ (which is customary), we get a worst case bound of

$$\|LLL(\mathbf{B})_1\| \lesssim \left(\frac{4}{3}\right)^{\frac{n}{2}} \approx 1.15^n.$$

In fact, it turns out that these bounds are pretty much sharp. One can easily verify that this is the case for the LLL-reduced basis

$$\mathbf{B} = \begin{bmatrix} \gamma^{n-1} & 0 & \dots & 0 & 0 & 0 \\ \frac{\gamma^{n-1}}{2} & \gamma^{n-2} & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \gamma^2 & 0 & 0 \\ \frac{\gamma^{n-1}}{2} & \frac{\gamma^{n-2}}{2} & \dots & \frac{\gamma^2}{2} & \gamma & 0 \\ \frac{\gamma^{n-1}}{2} & \frac{\gamma^{n-2}}{2} & \dots & \frac{\gamma^2}{2} & \frac{\gamma}{2} & 1 \end{bmatrix},$$

where $\gamma = \sqrt{\frac{4}{3}}$ is the hermite constant for dimension 2 (page 36 in [2]).

The next algorithm we will discuss is the BKZ algorithm, also known as block reduction. While one mechanism of LLL involves swapping two basis vectors at a time, BKZ operates on blocks of multiple vectors at once. In fact, when we choose block size $k = 2$, we obtain a variant of LLL, so BKZ can be viewed as somewhat of a generalization of LLL.

The block size gives us more control over the quality and runtime. In particular, a larger block size yields a better output but requires more time to terminate. We can see this in the approximation bound

$$\|BKZ(\mathbf{B})_1\| \leq \gamma_k^{\frac{n-1}{k-1}} \lambda(\mathcal{L})$$

for BKZ with block size k, where γ_k is the hermite constant for dimension k.

Furthermore, BKZ seems to be the best and most efficient approximation algorithm in practice, especially with recent variants such as BKZ 2.0 and SD-BKZ. Despite this fact, we do not have a good, provable bound on the time complexity of BKZ like we do for LLL.

3 Experimental Results

Nguyen and Stehle [3] confirm experimentally that LLL usually outperforms its theoretical guarantees, despite the bounds being sharp in the worst case. They estimate that in sufficiently high dimension, given a random basis $\mathbf{B}$ for almost any lattice $\mathcal{L}$, we have

$$\|LLL(\mathbf{B})_1\| \approx 1.02^n (\det\, \mathcal{L})^{\frac{1}{n}} \approx 1.02^n \lambda(\mathcal{L}).$$

We would like to compare this "practical bound" to the one achieved by the supposedly superior BKZ algorithm.

To sample random bases from random lattices, we adopt the same methods as in [3]. The Ajtai-type bases in dimension n and factor α are lower triangular integer matrices with diagonal elements

$$B_{i,i} = \left\lfloor 2^{(2d-i+1)^{\alpha}} \right\rfloor$$

and below diagonal elements $B_{j,i}$ $(j > i)$ sampled uniformly at random in $\left(-\frac{B_{i,i}}{2}, \frac{B_{i,i}}{2}\right)$. Bases of these form have been used in the past to exhibit worst-case outcomes. The knapsack-type bases in dimension n are the $(n+1) \times n$ matrices

$$\begin{bmatrix} A_1 & A_2 & \dots & A_n \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix},$$

with $A_1, \dots, A_n$ sampled uniformly at random in $[-2^b, 2^b]$ for some bound on the bitsize b. Note that these bases generate an n-dimensional lattice in $\mathbb{R}^{n+1}$, i.e. a lattice that is not full rank. Bases of these form occur often in practice.

We run our experiments in python using implementations in the FPyLLL library.

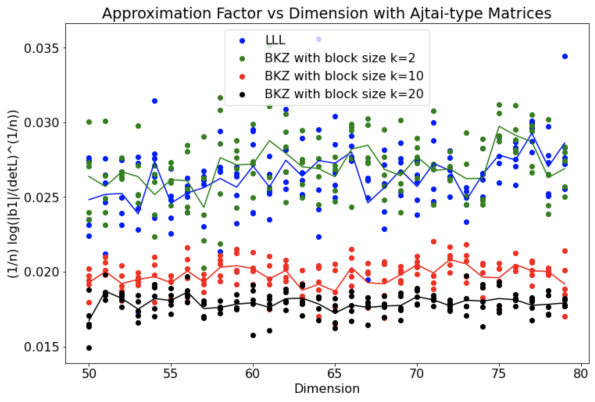

Figure 1: $\frac{1}{n}\log\frac{\|\mathbf{b_1}\|}{(\det\mathcal{L})^{\frac{1}{n}}}$ as a function of n for samples of Ajtai-type matrices with $\alpha = 1.2$.

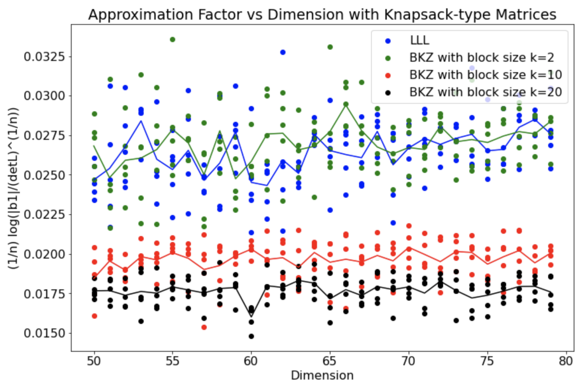

Figure 2: $\frac{1}{n}\log\frac{\|\mathbf{b_1}\|}{(\det\mathcal{L})^{\frac{1}{n}}}$ as a function of n for samples of Knapsack-type matrices with bit bound $b = 10$.

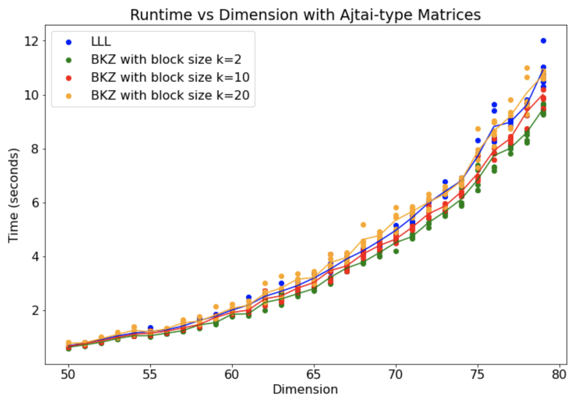

Figure 3: Time in seconds as a function of n for samples of Ajtai-type matrices with factor $\alpha = 1.2$. We omit this figure for the knapsack version because it looks identical.

In our experiments (Figures 1 and 2) we vary the dimension n, generate a random basis, run our basis reduction algorithms on it, and plot the value of $\frac{1}{n}\log\frac{\|\mathbf{b_1}\|}{(\det\mathcal{L})^{\frac{1}{n}}}$, where $\mathbf{b}_1$ is the first vector in the reduced basis. Note that all logarithms are in base 2. Our results for LLL match the experiments of [3]: $\frac{1}{n}\log\frac{\|LLL(\mathbf{B})_1\|}{(\det\mathcal{L})^{\frac{1}{n}}}$ converges slightly below 0.03 for both types of matrices. This implies that

$$\|LLL(\mathbf{B})_1\| \approx 2^{0.03n}(\det\mathcal{L})^{\frac{1}{n}} \approx 1.02^n(\det\mathcal{L})^{\frac{1}{n}} \approx 1.02^n\lambda(\mathcal{L}),$$

where the last "$\approx$" follows from the Gaussian heuristic, which is very likely to hold on random lattices [3].

Furthermore, observe that BKZ with block size $k = 2$ produces almost identical results to LLL, which supports our earlier remark about how BKZ generalizes LLL.

Notably, BKZ with larger block sizes seems to quite significantly outperform LLL. We have $\frac{1}{n}\log\frac{\|BKZ_{k=10}(\mathbf{B})_1\|}{(\det\mathcal{L})^{\frac{1}{n}}}$ is constantly 0.02ish, and $\frac{1}{n}\log\frac{\|BKZ_{k=10}(\mathbf{B})_1\|}{(\det\mathcal{L})^{\frac{1}{n}}}$ is constantly 0.0175ish, which through a similar analsis as above implies

$$\|BKZ_{k=10}(\mathbf{B})_1\| \approx 1.014^n\lambda(\mathcal{L})$$

and

$$\|BKZ_{k=20}(\mathbf{B})_1\| \approx 1.012^n\lambda(\mathcal{L}).$$

Seeing that this improvement in performance requires no sigficant increase in runtime (Figure 3), it is no wonder why BKZ is considered the better option in practice. Indeed, we observe that BKZ matches the polynomial running time of LLL.

However, the results shown in Figure 3 may be misleading, as they seemingly suggest that we can continue to increase the block size without increasing runtime by much. As we can see in Figure 4, this is certainly not the case; past a certain point, increasing the block size drastically increases the running time. Moreover, it seems that the tradeoff between performance and runtime seems to be balanced best around block size $k = 20$. Since the approximation factor can only decrease by so much, increasing the block size any further is not worth the extra runtime, at least in the setting of our experiments.

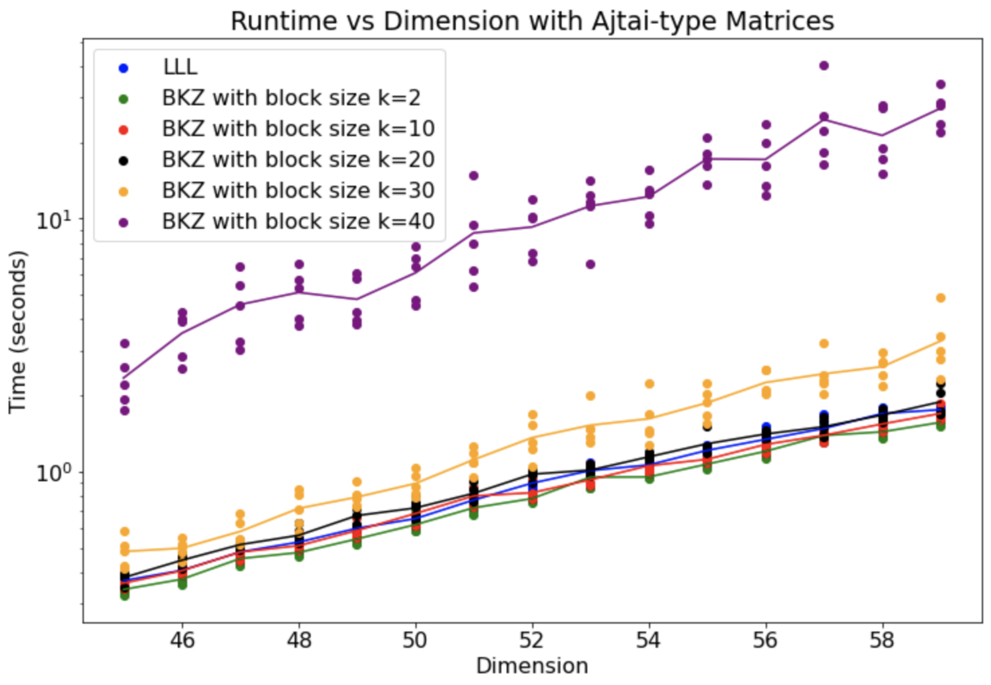

Figure 4: The cost of very large block size. Note that the y-axis is in log-scale.

References

[1] Huck Bennett. The complexity of the shortest vector problem. *SIGACT News*, 54(1):37–61, mar 2023.

[2] Nicolas Gama. *Geometrie des nombres et cryptanalyse de NTRU*. PhD thesis, 2008.

[3] Phong Q Nguyen and Damien Stehlé. Lll on the average. In *Algorithmic Number Theory: 7th International Symposium, ANTS-VII, Berlin, Germany, July 23-28, 2006. Proceedings 7*, pages 238–256. Springer, 2006.

[4] Michael Walter. Lattice blog reduction - part i: Bkz, 2020.